

DYNAMIC VISUALIZATIONS

Developing a Framework for Crowd-Based Simulations

by

Ao (Leo) Liu

A thesis

presented to the University of Waterloo

in fulfilment of the

thesis requirement for the degree of

Master of Architecture

Waterloo, Ontario, Canada, 2020

© Ao (Leo) Liu 2020

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Since its conception in the 1960s, digital computation has experienced both exponential growth in power and reduction in cost. This has allowed the production of relatively cheap electronics, which are now integrated ubiquitously in daily life. With so much computational data and an ever-increasing accessibility to intelligent objects, the potential for integrating such technologies within architectural systems becomes increasingly viable. Today, dynamic architecture is already emerging across the world; it is inevitable that one day computation will be fully integrated within the infrastructures of our cities.

However, as these new forms of dynamic architecture becomes increasingly commonplace, the standard static medium of architectural visualization is no longer satisfactory for representing and visualizing these dynamic spaces, let alone the human interactions within them. Occupancy within a space is already inherently dynamic and becomes even more so with the introduction of these new forms of architecture. This in turn challenges our conventional means of visualizing spaces both in design and communication. To fully represent dynamic architecture, the visualization must be dynamic as well. As such, current single image rendering methods within most existing architectural design pipelines becomes inadequate in portraying both the architectural dynamics of the space, as well as the interaction and influences these dynamics will have with the occupants.

This thesis aims to mitigate these shortcomings in architectural visualization by investigating the creation of a crowd simulation tool to facilitate a foundation for a visualization framework that can be continuously built upon based on project needs, which answers the question of how one can utilize current technologies to not only better represent responsive architecture but also to optimize existing visualization methodologies. By using an interdisciplinary approach that brings together architecture, computer science, and game design, it becomes possible to establish a more powerful, flexible, and efficient workflow in creating architectural visualizations.

Part One will establish a foundation to this thesis by looking at the state of the current world, its buildings in the sense of *dynamic*, and the current state of visualization technologies that are being utilized both within architectural design as well as outside of it. Part Two will investigate complex systems and simulation models, as well as investigating ways of integrating them with human behaviors to establish a methodology for creating a working crowd simulation system. Part Three will take the methodology developed within Part Two and integrate it within modern game engines, with the intent of creating an architectural visualization pipeline that can utilize the game engine for both crowd analytics as well as visualization. Part Four will look at some of the various spatial typologies that can be visualized with this tool. Finally, Part Five will speculate on various future directions to improve this tool beyond the current scope of this thesis.

Acknowledgments

To my supervisor Terri Boake, thank you for guiding me throughout this long process. This thesis would not have been possible without your insight and patience.

To my committee member David Correa, thank you for your helpful and thorough feedback. The additional resources you provided me were exceedingly helpful in finishing this thesis.

To Michelle Bullough, thank you for selflessly offering your time to edit my writing, even when you are in the process of finishing your own thesis. Your feedback and suggestions were very helpful in making this whole thing more coherent.

To Wesley Chu, thank you for putting up with my numerous questions about page layout options during my process of putting together the book. Your feedback kept me sane as I struggled to organize all of these figures.

To the rest of my friends who offered up their couches and helped me “*balance*” my thesis life by keeping me company, thank you.

To my parents, thank you for letting me mooch off your food and shelter throughout this whole process, all the while limiting your questions on when I will be done to only once every other week.

To my past self, thank you for choosing such a “*fun*” topic.

Table of Contents

Author's Declaration	iii
Abstract	v
Acknowledgments	vii
List of Figures	xi
List of Abbreviations	xxv
Technical Note	xxvii
Part 1 Introduction and Theory	1
Chapter 1.1 Emergence of Interactive and Dynamic Architecture	5
Chapter 1.2 Inadequacy of Current Visualization Methods	22
Chapter 1.3 Advent and Progression of the Gaming Engine	45
Chapter 1.4 Proposed Framework	62
Part 2 Technical Research	67
Chapter 2.1 Simulations Ideology	70
Chapter 2.2 Establishing Model Methodology	78
Chapter 2.3 Abstracting the Human Systems	92
Chapter 2.4 Spatial Functions	120
Chapter 2.5 Prototyping the System Model	128
Part 3 Tool Creation	131
Chapter 3.1 Utilizing the Gaming Engine	134
Chapter 3.2 Asset Creation	138
Chapter 3.3 Human Agents	140
Chapter 3.4 Architectural Objects	192
Chapter 3.5 Environment Context	202
Chapter 3.6 Application Methodology	206
Part 4 Tool Evaluation	223
Chapter 4.1 Spatial Conditions	226
Chapter 4.2 Nuit Blanche Cushion	238
Chapter 4.3 Riverside Gallery	248
Part 5 Next Steps	259
Chapter 5.1 Simulation Improvements	262
Chapter 5.2 Photorealism	266
Chapter 5.3 Virtual Reality	270
Chapter 5.4 Workflow Refinements	274
Bibliography	279
Appendix A Multimedia Figures	287

List of Figures

Part 1 | Introduction and Theory

- 7 Figure 1.1.1 A static living room with the Television turned off
Photographed by Author.
- 7 Figure 1.1.2 A static living room with the Television turned on feels more dynamic by comparison
Filmed by Author.
- 8 Figure 1.1.3 Virtual Rendering showing a static light source within a room
Rendered by Author.
- 8 Figure 1.1.4 Virtual Rendering showing a dynamic light source within a room.
Simulated by Author.
- 9 Figure 1.1.5 Physical Study showing a static light within a space.
Photographed by Author.
- 9 Figure 1.1.6 Physical Study showing a dynamic light within a space.
Filmed by Author.
- 10 Figure 1.1.7 Rotating light of lighthouse
From Antoni Cladera, "Milky Way Photography: The Definitive Guide (2019)," PhotoPills, accessed December 17, 2019, <https://www.photopills.com/articles/milky-way-photography-guide>.
- 11 Figure 1.1.8 Smoke Signals of the Great wall of china
From "Smoke Signals - 900 B.C.," The History of Media (The Beginning-1950 A.D.), accessed December 17, 2019, <http://thehistoryofmedia.weebly.com/smoke-signals.html>.
- 11 Figure 1.1.9 Drawbridge at the fort of Ponta da Bandeira
By Georges Jansoone, "File:Lagos48.jpg," Entrance with Drawbridge; Forte Da Ponta Da Bandeira; Lagos, Portugal, September 24, 2006, Wikimedia Commons, accessed December 17, 2019, <https://commons.wikimedia.org/wiki/File:Lagos48.jpg>.
- 12 Figure 1.1.10 The fountain of Neptune
By KatDevsGames, "File:Villa d'Este 01.Jpg," Wikimedia Commons, accessed December 18, 2019, https://commons.wikimedia.org/wiki/File:Villa_d%27Este_01.jpg.
- 12 Figure 1.1.11 One hundred fountains walkway
By Wknight94, "File:Villa d'Este fountains 6.jpg," Fountains at Villa d'Este in Tivoli, April 29, 2008, Wikimedia Commons, accessed December 18, 2019, https://commons.wikimedia.org/wiki/File:Villa_d%27Este_fountains_6.jpg.
- 12 Figure 1.1.12 Oval Fountain
By Dnalor 01, "File:Tivoli, Villa d'Este, Fontana dell'Ovato.jpg," Deutsch: Tivoli, Villa d'Este, Fontana Dell'Ovato, May 15, 2005, Wikimedia Commons, accessed December 18, 2019, https://commons.wikimedia.org/wiki/File:Tivoli,_Villa_d%27Este,_Fontana_dell%27Ovato.jpg.
- 13 Figure 1.1.13 Spectators viewing the Rock Gardens from the veranda
By Sean Pavone, from Don George, "Finding Peace in 21st-Century Kyoto." National Geographic, July 7, 2015, accessed December 18, 2019, <https://www.nationalgeographic.com/travel/intelligent-travel/2015/07/07/finding-peace-in-21st-century-kyoto/>.
- 13 Figure 1.1.14 Cherry blossoms hang over the rock garden, bringing the dynamics of nature further into the space
By Bjørn Christian Tørrissen, "File:Ryoan-ji-Garden-2018.jpg," Stones in the Zen Garden/Rock Garden at the Ryōan-ji Temple in Kyoto, Japan, May 11, 2018, Wikimedia Commons, accessed December 18, 2019, <https://commons.wikimedia.org/wiki/File:Ryoan-ji-Garden-2018.jpg>.
- 13 Figure 1.1.15 Cherry blossoms hang over the rock garden, bringing the dynamics of nature further into the space
By Didier Moïse, "File:Cherry blossom at the rock garden of Ryōan-ji Temple in Kyoto, Japan.jpg," Cherry Blossom at the Rock Garden of Ryōan-ji Temple in Kyoto, Japan, April 12, 2005, Wikimedia Commons, accessed December 18, 2019, https://commons.wikimedia.org/wiki/File:Cherry_blossom_at_the_rock_garden_of_Ry%C5%8Dan-ji_Temple_in_Kyoto,_Japan.jpg.
- 15 Figure 1.1.16 Silicon transistor progression through the years
From "Happy birthday transistor!," translated with Google Translate, accessed December 18, 2019, <http://astron.dmitryshevchenko.com/2017/12/19/transistor/>.
- 15 Figure 1.1.17 This graph shows the progression of transistor count within integrated circuit chips through the years, as described by Moore's Law
From Max Roser and Hannah Ritchie, "Technological Progress," May 11, 2013, Our World in Data, accessed December 18, 2019, <https://ourworldindata.org/technological-progress>.
- 17 Figure 1.1.18 The Al Bahar Towers Facades utilizes motorized folding louvers to control the amount of sunlight that can pass through.
From "Al Bahar Towers Responsive Facade / Aedas," September 5, 2012, ArchDaily, accessed December 18, 2019, <http://www.archdaily.com/270592/al-bahar-towers-responsive-facade-aedas/>.

- 17 Figure 1.1.19 **The Umbrellas in Medina opens and closes to open up the space as well as offer environmental protection depending on the weather and time of day.**
From “Umbrellas in the Mosque of the Prophet’s Courtyard and Surrounding Open Spaces,” Abdullatif Al Fozan Award for Mosque Architecture, accessed December 18, 2019, <https://alfozanaward.org/mosques/umbrellas-in-the-mosque-of-the-prophets-courtyard-and-surrounding-open-spaces/>.
- 17 Figure 1.1.20 **The Roof of the Rogers Center in Toronto opens and closes to provide outdoor or indoor experiences depending on the exterior conditions.**
From Laura Armstrong, “Rogers Centre Roof to Be Opened for Blue Jays Game Tonight,” The Star, accessed December 18, 2019, <https://www.thestar.com/sports/bluejays/2016/05/27/rogers-centre-roof-to-be-opened-for-blue-jays-game-tonight.html>.
- 19 Figure 1.1.21 **Starscape**
Photographed by Author.
- 19 Figure 1.1.22 **Ocean**
Photographed by Author.
- 19 Figure 1.1.23 **Cushion**
Photographed by Author.
- 20 Figure 1.1.24 **Acadia conference in Michigan**
Photographed by Author.
- 20 Figure 1.1.25 **Acadia lecture in Michigan**
Photographed by Author.
- 21 Figure 1.1.26 **Acadia 2013 poster**
From Sebastian Jordana, “Adaptive Architecture ACADIA 2013,” October 3, 2013, ArchDaily, accessed December 18, 2019, <http://www.archdaily.com/434672/adaptive-architecture-acadia-2013/>.
- 23 Figure 1.2.1 **Vanishing point, depicted in Della Pittura by Alberti**
By Leon Battista Alberti, *Della pittura e della statua di Leonbatista Alberti* (Milano : Società tipografica de’Classici italiani, 1804), <http://archive.org/details/dellapitturaedel00albe>.
- 23 Figure 1.2.2 **Perspective pillars on grid, depicted in Della Pittura by Alterbi.**
By Alberti, *Della pittura e della statua di Leonbatista Alberti*.
- 24 Figure 1.2.3 **The calling of the Apostles Peter and Andrew by Duccio, 1308-11**
By Duccio di Buoninsegna, *The Calling of the Apostles Peter and Andrew*, 1308-1311, tempera on panel, 42.7 x 45.5 cm, Samuel H. Kress Collection, National Gallery of Art, accessed December 18, 2019, <https://www.nga.gov/collection/art-object-page.282.html>.
- 25 Figure 1.2.4 **The Last Supper by Leonardo Da Vinci, 1495-96**
By Leonardo Da Vinci, from Paris Orlando, “File:Last Supper by Leonardo da Vinci.jpg,” November 10, 2019, Wikimedia Commons, accessed December 18, 2019, https://commons.wikimedia.org/wiki/File:Last_Supper_by_Leonardo_da_Vinci.jpg.
- 27 Figure 1.2.5 **Photograph of Bibliothèque Sainte-Geneviève by Bisson Frères**
By Bisson Frères, from Neil Levine, “The Template of Photography in Nineteenth-Century Architectural Representation,” *Journal of the Society of Architectural Historians* 71, no. 3 (January 2012), <https://doi.org/10.1525/jсах.2012.71.3.306>.
- 27 Figure 1.2.6 **Perspective view of Bibliothèque Sainte-Geneviève, traced from Bisson Frères’ photograph by Henri Labrousse, engraving by Jacques-Joseph Huguenet**
By Henri Labrousse, traced from photograph by Bisson Frères, and engraved by *Jacques-Joseph Huguenet*, from Levine, “The Template of Photography in Nineteenth-Century Architectural Representation.”
- 29 Figure 1.2.7 **Archigram Information Tear-off Sheet**
From “Archigram: Tear-off Information Sheets,” BALTIC Centre for Contemporary Art, accessed December 18, 2019, <http://balticplus.uk/archigram-tear-off-information-sheets-c8292/>.
- 30 Figure 1.2.8 **Plugin City concept by Peter Cook (Archigram), 1964**
By Peter Cook, “Plug-In_City, Max. Pressure Area, Long Section,” 1964, photochemical print overdrawn with ink and gouache, 1159 x 552 mm, Archigram Archives, accessed December 18, 2019, <http://archigram.net/portfolio.html>.
- 30 Figure 1.2.9 **Walking City Concept by Ron Herron (Archigram), 1964**
By Ron Herron, from Rowan Moore, “The World According to Archigram,” November 18, 2018, The Observer, accessed December 18, 2019, <https://www.theguardian.com/artanddesign/2018/nov/18/archigram-60s-architects-vision-urban-living-the-book>.
- 31 Figure 1.2.10 **Instant City concept by Peter Cook (Archigram), 1969**
By Peter Cook, from “‘Instant City’ Travelling Exhibition, Now at Collège Maximilien de Sully,” December 19, 2015, BMIAA, accessed December 18, 2019, <https://www.bmiaa.com/instant-city-travelling-exhibition-now-at-college-maximilien-de-sully/>.
- 31 Figure 1.2.11 **Computer City concept by Dennis Crompton (Archigram), 1964**
By Dennis Crompton, “Computer City,” 1964, photochemical print mounted on board, 887 x 697 mm, Archigram Archives, accessed December 18, 2019, <http://archigram.net/portfolio.html>.
- 33 Figure 1.2.12 **2D floorplans, created within Autodesk AutoCAD.**
From mtcarrillo, “Creating Basic Floor Plans From an Architectural Drawing in AutoCAD,” Instructables, accessed December 18, 2019, <https://www.instructables.com/id/Creating-Basic-Floor-Plans-from-an-Architectural-D/>.
- 33 Figure 1.2.13 **A building model rendered within 3D space on a viewport in Autodesk 3DS Max**
By Ronen Bekerman, “Making of MS House at Dusk, Part 2,” October 23, 2009, Ronen Bekerman - 3D Architectural Visualization & Rendering Blog, accessed December 18, 2019, <https://www.ronenbekerman.com/making-of-ms-house-at-dusk-part-2/>.
- 33 Figure 1.2.14 **The same building model rendered out with Vray**
By Bekerman, “Making of MS House at Dusk, Part 2.”
- 35 Figure 1.2.15 **People moving through a gallery space within the Solomon R. Guggenheim Museum in NYC**
Photographed by Author
- 35 Figure 1.2.16 **Macy’s Thanksgiving Day parade, NYC**
Photographed by Author
- 35 Figure 1.2.17 **People moving through a gallery space within the MOMA in NYC**
Photographed by Author
- 36 Figure 1.2.18 **The Horse in Motion cabinet cards by Eadweard Muybridge, 1878**
By *Eadweard Muybridge*, from *Neil Patrick*, “Filmed in 1878, ‘The Galloping Horse’ Is the First Motion Picture Ever Made,” June 27, 2016, The Vintage News (blog), accessed December 18, 2019, <https://www.thevintagenews.com/2016/06/27/46591-2/>.
- 36 Figure 1.2.19 **Animation made from Eadweard Muybridge’s cards**
From silentfilmhouse, “Race Horse First Film Ever 1878 Eadweard Muybridge,” YouTube, 0:15, accessed December 18, 2019, <https://www.youtube.com/watch?v=IEqccPhsqgA>.
- 37 Figure 1.2.20 **Lord of the rings Return of the King, 2003**
From Film Radar, trimmed by Author, “Special Effects in ‘The Lord of the Rings: The Essence of Movie Magic,’” YouTube, 12:08, accessed December 18, 2019, <https://www.youtube.com/watch?v=p6M8Yem5j0s&v=en>.
- 37 Figure 1.2.21 **Royal Ontario Museum architectural walk-through, 2003**
Obtained from supervisor, created by B+H Architects.
- 38 Figure 1.2.22 **The 3rd and the Seventh by Alex Roman, 2009**
By Alex Roman, trimmed by Author, “The Third & The Seventh,” uploaded November 24, 2009, Vimeo, 12:29, accessed December 18, 2019, <https://vimeo.com/7809605>.
- 39 Figure 1.2.23 **Architecture Walk-through by Framemakers Creative SB, 2015**
From Framemakers Creative SB, trimmed by Author, “Star Residences 3D Animation Walkthrough Video,” YouTube, 3:05, accessed December 18, 2019, <https://www.youtube.com/watch?v=8qU2xhZlsJE>.
- 39 Figure 1.2.24 **Architecture Walk-through by Momo Graphics, 2016**
By Momo Graphics, from Kenny Khoo, trimmed by Author, “3d Architecture Walkthrough Flythrough Animation Service Singapore Building Interior Exterior,” YouTube, 2:09, accessed December 18, 2019, <https://www.youtube.com/watch?v=fC1OrZ4kAJs&t=1s&pbjreload=10>.
- 41 Figure 1.2.25 **Revit Perspective Viewport**
Sample Architecture Project from Revit, screen-captured by Author.
- 41 Figure 1.2.26 **Security footage showing various frame rates**
From daksec1, trimmed by Author, “IP Video Frame Rate Demo,” YouTube, 0:50, accessed December 18, 2019, <https://www.youtube.com/watch?v=XRaDV8YADiQ>.
- 43 Figure 1.2.27 **Typical breakdown of architectural fees**
By Jorge Fontan, “Architectural Fees,” February 7, 2018, Fontan Architecture, accessed December 18, 2019, <https://jorgefontan.com/architectural-fees/>.
- 43 Figure 1.2.28 **Typical time-line of architectural design phases**
By HMM Modern Architecture, from “Architectural Phases,” Ibello Architect, accessed December 18, 2019, <https://www.ibelloarchitects.com/architectural-phases/>.
- 46 Figure 1.3.1 **Pac-Man, Namco, 1980**
From “Original Pac-Man” APKPure, accessed December 19, 2019, <https://apkpure.com/original-pac-man/com.classicretrogames.pacman>.
- 46 Figure 1.3.2 **Catacomb Abyss, Softdisk, 1992**
From “The Catacomb Abyss Review,” October 30, 2018, GameFAQs, accessed December 19, 2019, <https://gamefaqs.gamespot.com/pc/954269-the-catacomb-abyss/reviews/167153>.
- 47 Figure 1.3.3 **Hand Drafted South Elevation of Denver Library by Michael Graves, 1994**
By Michael Graves, Denver Library, South Elevation, 1994, pencil and colored pencil on yellow tracing paper, 14 x 26 inches, from Rory Stott, “Gallery of In Honor of Michael Graves, The Architectural League Revisits 200 Years of Drawing,” November 21, 2014, ArchDaily, accessed December 19, 2019, <https://www.archdaily.com/570439/in-honor-of-michael-graves-the-architectural-league-revisits-200-years-of-drawing/546fa61be58ece2295000037-denver-library-sout>.
- 47 Figure 1.3.4 **Creating 3D building walls from a 2D Building plan in virtual space**
From “How to Create a 3D Architecture Floor Plan Rendering,” TonyTextures, accessed December 19, 2019, <https://www.tonytextures.com/how-to-create-a-3d-architecture-floor-plan-rendering/>.
- 48 Figure 1.3.5 **The Division, Massive Entertainment, 2016**
From “Tom Clancy’s The Division (Preowned),” EB Games, accessed December 19, 2019, <https://www.ebgames.co.nz/product/ps4/165235-tom-clancys-the-division-preowned>.

49 Figure 1.3.6 **Hudson Yards Rendering by KPF, 2019**
By Kohn Pedersen Fox (KPF), “Hudson Yards,” accessed December 19, 2019, <https://www.kpf.com/projects/hudson-yards>.

51 Figure 1.3.7 **“An abstract model of how an engine might be put together”**
By Björn Nilson and Martin Söderberg, “Game Engine Architecture,” (May 26, 2007): 3-6, accessed December 19, 2019, <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.459.9537&rep=rep1&type=pdf>.

53 Figure 1.3.8 **Grand Theft Auto 3, DMA Design, 2001**
From AndromedaDude, trimmed by Author, “Grand Theft Auto III Gameplay (Playstation 2),” YouTube, 10:39, accessed December 19, 2019, <https://www.youtube.com/watch?v=jONTvpvj7DM>.

53 Figure 1.3.9 **The Witcher 3, CD Projekt, 2015**
From Im Qith, trimmed by Author, “The Witcher 3, Entering Novigrad (No Commentary),” YouTube, 8:09, accessed December 19, 2019, <https://www.youtube.com/watch?v=MTrxkDLi6sg>.

55 Figure 1.3.10 **Rendering a frame from Vray**
By Jordivdm, trimmed by Author, “FullHD 3D VRay Render at I7-5820k 6 Cores (12 Virtual Cores),” YouTube, 0:58, accessed December 19, 2019, <https://www.youtube.com/watch?v=rjvimjwhams>.

55 Figure 1.3.11 **Rendering Frames from Unreal Engine 4**
Screen-captured by Author.

57 Figure 1.3.12 **Ray tracing**
By Henrik, “File:Ray trace diagram.svg,” This Diagram Illustrates the Ray Tracing Algorithm for Rendering an Image, April 12, 2008, Wikimedia Commons, accessed December 19, 2019, https://commons.wikimedia.org/wiki/File:Ray_trace_diagram.svg.

57 Figure 1.3.13 **Rasterization**
From “Rasterization: A Practical Implementation,” Scratchapixel, accessed December 19, 2019, <https://www.scratchapixel.com/lessons/3d-basic-rendering/rasterization-practical-implementation>.

58 Figure 1.3.14 **CPU vs GPU Processing**
From Gino Baltazar, “CPU vs GPU in Machine Learning,” September 13, 2018, Oracle Data Science Blog, accessed December 19, 2019, <https://blogs.oracle.com/datascience/cpu-vs-gpu-in-machine-learning>.

59 Figure 1.3.15 **Texture Baking utilizes normal maps to preserve detail without the additional polygons**
By fra3point, “Total Baker - Texture Baking System,” Unity Forum, accessed December 19, 2019, <https://forum.unity.com/threads/total-baker-texture-baking-system.546341/>.

61 Figure 1.3.16 **A traditional desktop setup with a monitor and various input devices such as mouse and keyboard, game controller, and joysticks**
By WeazelBear, “I Built My Own Live Edge Desk out of Teak. I Hope You All like It. Album in Comments,” Reddit, accessed December 19, 2019, https://www.reddit.com/r/battlestations/comments/7wls6/i_built_my_own_live_edge_desk_out_of_teak_i_hope/.

61 Figure 1.3.17 **Possible VR setup with various trackers for interactive inputs**
From “Fully immersive VR Entertainment Solutions,” Cyberith, accessed December 19, 2019, <https://www.cyberith.com/entertainment/>.

61 Figure 1.3.18 **Architectural Rendering by HOK**
From Ken Pimentel, trimmed by Author, “HOK on Architectural Visualization: Aggregate, Iterate, Communicate,” Unreal Engine, accessed December 19, 2019, <https://www.unrealengine.com/en-US/spotlights/hok-architectural-visualization-aggregate-iterate-communicate>.

63 Figure 1.4.1 **Massive**
From Film Radar, trimmed by Author, “Special Effects in The Lord of the Rings: The Essence of Movie Magic,” YouTube, 12:08, accessed December 18, 2019, <https://www.youtube.com/watch?v=p6M8Yem5j0s&v=1>.

63 Figure 1.4.2 **Golaem**
By Golaem, trimmed by Author, “Golaem Crowd 4: Take Control of Your Crowds,” YouTube, 3:17, accessed December 19, 2019, <https://www.youtube.com/watch?v=rr6tDBeNEv0>.

63 Figure 1.4.3 **Miarmy**
From Basefount, trimmed by Author, “Miarmy 3 Crowd Simulation DEMO 8,” YouTube, 3:12, accessed December 19, 2019, https://www.youtube.com/watch?v=3wjCwtc_-hk.

65 Figure 1.4.4 **Oasys mass motion**
By TheOasysSoftware, trimmed by Author, “Oasys Software - MassMotion, The World’s Most Advanced Crowd Simulation Software,” YouTube, 2:30, accessed December 19, 2019, <https://www.youtube.com/watch?v=dR5G5SNI5T4>.

65 Figure 1.4.5 **A crowd Visualization Tool in Autodesk 3ds Max**
From sanvfx, trimmed by Author, “Creating Crowd Simulation in 3ds Max,” YouTube, 23:09, accessed December 19, 2019, <https://www.youtube.com/watch?v=h-PMBi8gze4&t=454s>.

Part 2 | Technical Research

71 Figure 2.1.1 **Various wave patterns seen on-top of the ocean surface**
From Alex Green, “An Aerial Birds Eye Shot Of The Ocean and Waves,” YouTube, 0:10, accessed December 25, 2019, <https://www.youtube.com/watch?v=1jUnZ4VnoD4>.

71 Figure 2.1.2 **Video showing phantom traffic jam**
From New Scientist, trimmed by Author, “Shockwave Traffic Jams Recreated for First Time,” YouTube, 0:39, accessed December 25, 2019, <https://www.youtube.com/watch?v=Suugn-p5C1M>.

71 Figure 2.1.3 **This crowded concert shows how the interaction between each individual human produces various wave patterns throughout the entire crowd.**
From Australian Concert And Entertainment Security Pty Ltd, trimmed by Author, “Crowd Wave Surge Example,” YouTube, 4:56, accessed December 25, 2019, <https://www.youtube.com/watch?v=BgpdmAtbhbE&t=8s>.

73 Figure 2.1.4 **Snowflakes**
By Wilson Bentley, “File:SnowflakesWilsonBentley.jpg,” Wikimedia Commons, accessed December 25, 2019, <https://commons.wikimedia.org/wiki/File:SnowflakesWilsonBentley.jpg>.

73 Figure 2.1.5 **Termite mound**
By Brian Voon Yee Yap, from Yewenyi, “File:Termite Cathedral DSC03570.jpg,” Wikimedia Commons, accessed December 25, 2019, https://commons.wikimedia.org/wiki/File:Termite_Cathedral_DSC03570.jpg.

73 Figure 2.1.6 **Starling murmurations**
From National Geographic, trimmed by Author, “Flight of the Starlings: Watch This Eerie but Beautiful Phenomenon | Short Film Showcase,” YouTube, 2:00, accessed December 25, 2019, https://www.youtube.com/watch?v=V4f_1_r80RY.

75 Figure 2.1.7 **Rule 30 as introduced by Stephen Wolfram, 1983**
From Eric W. Weisstein, “Rule 30,” Wolfram Math World, accessed December 25, 2019, <http://mathworld.wolfram.com/Rule30.html>.

76 Figure 2.1.8 **250 iterations of Rule 30**
From Eric W. Weisstein, “Rule 30,” Wolfram Math World.

79 Figure 2.2.1 **Simulation Study Diagram**
From Jerry Banks et al., *Discrete-Event System Simulation* (Upper Saddle River, NJ: Prentice Hall, 2001), 16.

83 Figure 2.2.2 **The Lagrangian description calculates the position and velocity of the individual particles within the fluid**
From “Descriptions of Fluid Flows,” accessed December 25, 2019, https://www.me.psu.edu/cimbala/Learning/Fluid/Introductory/descriptions_of_fluid_flows.htm.

83 Figure 2.2.3 **The Eulerian description calculates the output velocities from the input velocities, in which the space inside the control volume is assumed to be completely filled as a continuous mass**
From “Descriptions of Fluid Flows.”

85 Figure 2.2.4 **Much like how humans interact with spaces, Autonomous Agents act within the simulation through its perception of the environment**
By Stuart Russell and Peter Novig, “Intelligent Agents - Chapter 2,” from *Artificial Intelligence: A Modern Approach*, obtained from “Agents: Artificial Intelligence,” accessed December 25, 2019, <https://www.doc.ic.ac.uk/project/examples/2005/163/g0516334/>.

86 Figure 2.2.5 **2-Dimensional Cellular Automata**
From Hubert Klüpfel, “A Cellular automaton model for crowd movement and egress simulation,” (July 2003): 33-35, accessed December 26, 2019, https://www.researchgate.net/publication/29800160_A_Cellular_automaton_model_for_crowd_movement_and_egress_simulation.

86 Figure 2.2.6 **Social Forces**
From Dirk Helbing and Péter Molnár, “Social Force Model for Pedestrian Dynamics,” *Physical Review E* 51, no. 5 (1995): 15-17, doi:10.1103/PhysRevE.51.4282.

87 Figure 2.2.7 **Reciprocal Velocity Obstacles**
From Jur Van Den Berg, Ming Lin, and Dinesh Manocha, “Reciprocal Velocity Obstacles for Real-Time Multi-Agent Navigation,” 2008 IEEE International Conference on Robotics and Automation, (May 2008): 1928-1932, <https://doi.org/10.1109/robot.2008.4543489>.

87 Figure 2.2.8 **Adaptive Roadmaps**
From Avneesh Sud et al., “Real-time Navigation of Independent Agents Using Adaptive Roadmaps,” ACM SIGGRAPH 2008, (2008): doi:10.1145/1401132.1401207.

89 Figure 2.2.9 **Centroidal Particles**
From Omar Hesham and Gabriel Wainer, “Centroidal Particles for Interactive Crowd Simulation,” 2016 Summer Computer Simulation Conference (SCSC 2016), (2016): <https://doi.org/10.22360/summersim.2016.scs.012>.

89 Figure 2.2.10 **HiDAC**
From Nuria Pelechano, Jan M. Allbeck, & Norman I. Badler, “Controlling Individual Agents in High-Density Crowd Simulation,” Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, (2007): 99-108, <http://repository.upenn.edu/hms/210>.

93 Figure 2.3.1 **Three hierarchical layers of motion behaviors**
From Craig W. Reynolds, “Steering Behaviors For Autonomous Characters,” Reynolds Engineering & Design, accessed October 17, 2019, <http://www.red3d.com/cwr/steer/gdc99/>.

93 Figure 2.3.2 **Three hierarchical layers of human behaviors**
Illustrated by Author.

94 Figure 2.3.3 **Analog to Digital**
Illustrated by Author.

- 97 Figure 2.3.4 Vectors can be thought of as the difference between 2 points.
From Daniel Shiffman, "Chapter 1. Vectors," in *The Nature of Code* (United States: D. Shiffman, 2012), accessed October 17, 2019, <https://natureofcode.com/book/chapter-1-vectors/>.
- 97 Figure 2.3.5 Vectors can be described by 2 scalar variables.
From Shiffman, "Chapter 1. Vectors."
- 97 Figure 2.3.6 Velocity vector updates position
From Shiffman, "Chapter 1. Vectors."
- 98 Figure 2.3.7 A list of Vector operations that can be used within Processing.
From Shiffman, "Chapter 1. Vectors."
- 98 Figure 2.3.8 Vector Multiplication
From Shiffman, "Chapter 1. Vectors."
- 99 Figure 2.3.9 Vector Addition
From Shiffman, "Chapter 1. Vectors."
- 99 Figure 2.3.10 Vector Subtraction
From Shiffman, "Chapter 1. Vectors."
- 99 Figure 2.3.11 Vector Division
From Shiffman, "Chapter 1. Vectors."
- 101 Figure 2.3.12 The relationship between distance, velocity, and acceleration
From "Motion Graphs," accessed December 27, 2019, <http://hyperphysics.phy-astr.gsu.edu/hbase/Mechanics/motgraph.html>.
- 103 Figure 2.3.13 The calculated opposing Steering force, when added to current velocity, will bring it closer to desired velocity
From Daniel Shiffman, "Chapter 6. Autonomous Agents," in *The Nature of Code* (United States: D. Shiffman, 2012), accessed October 17, 2019, <https://natureofcode.com/book/chapter-6-autonomous-agents/>.
- 103 Figure 2.3.14 The steering force is pushing down on the vehicle to steer it towards desired velocity
From Shiffman, "Chapter 6. Autonomous Agents."
- 103 Figure 2.3.15 Desired velocity can be calculated by obtaining the Vector distance between the vehicle position and agent position
From Shiffman, "Chapter 6. Autonomous Agents."
- 103 Figure 2.3.16 We must then limit this distance vector to obtain our desired velocity so our vehical, or human, can't move too fast
From Shiffman, "Chapter 6. Autonomous Agents."
- 103 Figure 2.3.17 How Max force can affect radius
From Shiffman, "Chapter 6. Autonomous Agents."
- 105 Figure 2.3.18 The 3 rules of flocking is defined by Reynolds as Separation, Alignment, and Cohesion
From Shiffman, "Chapter 6. Autonomous Agents."
- 105 Figure 2.3.19 Arriving behavior once they get to a certain distance from target location
From Shiffman, "Chapter 6. Autonomous Agents."
- 105 Figure 2.3.20 Avoiding walking into walls
From Shiffman, "Chapter 6. Autonomous Agents."
- 105 Figure 2.3.21 These same forces are useful in other types of simulations as well, such as planetary motion
From Daniel Shiffman, "Chapter 2. Forces," in *The Nature of Code* (United States: D. Shiffman, 2012), accessed October 17, 2019, <https://natureofcode.com/book/chapter-2-forces/>.
- 107 Figure 2.3.22 Edward T Hall's Interpersonal Distances of man
By WebHamster, "File:Personal Space.svg," Diagram Representation of Personal Space Limits, According to Edward T. Hall's Interpersonal Distances of Man, March 8, 2009, Wikimedia Commons, accessed December 25, 2019, https://commons.wikimedia.org/wiki/File:Personal_Space.svg.
- 107 Figure 2.3.23 We can use these defined personal spaces to determine the area around the agent in which they will be affected
From Shiffman, "Chapter 6. Autonomous Agents."
- 107 Figure 2.3.24 Utilizing Fleeing behaviour to avoid other agents that may have entered the Agent's personal space
From Shiffman, "Chapter 6. Autonomous Agents."
- 109 Figure 2.3.25 Crowd density vs crowd flow rate graph
From Keith Still, "Static crowd density (general)," Crowd Safety and Risk Analysis, accessed December 27, 2019, <http://www.gkstill.com/Support/crowd-density/CrowdDensity-1.html>.
- 110 Figure 2.3.26 Pathfinding flowchart
Illustrated by Author.
- 113 Figure 2.3.27 Canadian census info-graphic breaking down the population into percentages
From Statistics Canada, "Journey to Work, 2016 Census of Population," November 29, 2017, Government of Canada, accessed December 27, 2019, <https://www150.statcan.gc.ca/n1/pub/11-627-m/11-627-m2017038-eng.htm>.

- 115 Figure 2.3.28 If you are in a room with a door and a chair, do you go to the chair? Or do you go to the door?
Illustrated by Author, chair and door graphics taken from "STEFAN Chair - Brown-Black," IKEA, accessed December 28, 2019, <https://www.ikea.com/ca/en/p/stefan-chair-brown-black-00211088/>, and "ReliaBilt Colonist Primed 6-Panel Hollow Core Molded Composite Pre-Hung Door (Common: 30-in x 80-in; Actual: 31.5625-in x 81.6875-in)," Lowe's, accessed December 28, 2019, <https://www.lowes.com/pd/ReliaBilt-Colonist-Primed-6-Panel-Hollow-Core-Molded-Composite-Pre-Hung-Door-Common-30-in-x-80-in-Actual-31-5625-in-x-81-6875-in/1000537851>.
- 115 Figure 2.3.29 A decision Tree based on percentages
By Chooseco, from Sarah Laskow, "These Maps Reveal the Hidden Structures of 'Choose Your Own Adventure' Books," June 13, 2017, Atlas Obscura, accessed December 27, 2019, <http://www.atlasobscura.com/articles/cyoa-choose-your-own-adventure-maps>.
- 117 Figure 2.3.30 Human Visual Limit- Top View
From "Environmental Considerations and Human Factors for Videowall Design," Extron, accessed December 28, 2019, <https://www.extron.com/article/environconhumanfact>.
- 117 Figure 2.3.31 Human Visual Limit- Side View
From "Environmental Considerations and Human Factors for Videowall Design."
- 117 Figure 2.3.32 Sensory limit within simulation
From Shiffman, "Chapter 6. Autonomous Agents."
- 119 Figure 2.3.33 Human systems flowchart
Illustrated by Author.
- 123 Figure 2.4.1 The face on mars
From "Unmasking the Face on Mars," NASA Science, accessed December 28, 2019, https://science.nasa.gov/science-news/science-at-nasa/2001/ast24may_1.
- 123 Figure 2.4.2 Google Maps
Google Maps Android application, screen-captured by Author.
- 123 Figure 2.4.3 Fearful Symmetry by Ruairi Glynn
By Ruairi Glynn, "Fearful Symmetry," accessed October 18, 2019, <http://www.ruairiglynn.co.uk/portfolio/fsymmetry/>.
- 129 Figure 2.5.1 Prototype 2D Simulation created in Processing, based on the Nuit Blanche Installation Cushion
CAD file from the Cushion group, simulated and screen-recorded by Author.
- 129 Figure 2.5.2 Cushion invites people to walk through a narrow corridor. The light filled ballons change color as people interact with them.
Filmed by Author.

Part 3 | Tool Creation

- 135 Figure 3.1.1 Parametric node system within Grasshopper
By David Rutten, "File:Grasshopper MainWindow.png," A Screen Shot of the Grasshopper Main Window, 2011, Wikimedia Commons, accessed December 28, 2019, https://commons.wikimedia.org/wiki/File:Grasshopper_MainWindow.png.
- 135 Figure 3.1.2 Material node system within 3ds Max
Screen-captured by Author.
- 135 Figure 3.1.3 Scripting node system within Unreal Engine 4
Default character asset script within UE4, screen-captured by Author.
- 139 Figure 3.2.1 Game assets within project browser
Screen-captured by Author.
- 141 Figure 3.3.1 How each asset will be utilized within this software environment
Illustrated by Author.
- 143 Figure 3.3.2 AI perception
Simulated and screen-recorded by Author.
- 143 Figure 3.3.3 EQS trace test
From "Environment Query System Overview," Unreal Engine Documentation, accessed December 28, 2019, <https://docs.unrealengine.com/en-US/Engine/ArtificialIntelligence/EQS/EQSOverview/index.html>.
- 145 Figure 3.3.4 Simplified visual scripting process within UE4
Illustrated by Author.
- 147 Figure 3.3.5 Decision Network
Screen-captured by Author.
- 148 Figure 3.3.6 Event Node begins the execution line
Screen-captured by Author.
- 148 Figure 3.3.7 Boolean percentages controls which path the line takes
Screen-captured by Author.

149 Figure 3.3.8 Once the execution line reaches a decision, a node is triggered to set the AgentState and then apply a timer to establish how long the agent might be in that state
Screen-captured by Author.

149 Figure 3.3.9 The execution line moves down and repeats all the checks for every entity type if it does not reach the end of the logic
Screen-captured by Author.

150 Figure 3.3.10 Functional Decision Logic during runtime
Simulated and screen-recorded by Author.

153 Figure 3.3.11 Navmesh at work
Simulated and screen-recorded by Author.

153 Figure 3.3.12 Flocking Behaviors recreated with Vector and World-Offset nodes within UE4
Screen-captured by Author.

155 Figure 3.3.13 The Behavior Tree allows a visual way to define AI tasks within UE4
From "Behavior Tree Overview," Unreal Engine Documentation, accessed October 18, 2019, <https://docs.unrealengine.com/en-US/Engine/ArtificialIntelligence/BehaviorTrees/BehaviorTreesOverview/index.html>.

155 Figure 3.3.14 Behavior Tree Execution Order from top to down and left to right
From "Behavior Tree Overview," Unreal Engine Documentation, accessed October 18, 2019, <https://docs.unrealengine.com/en-US/Engine/ArtificialIntelligence/BehaviorTrees/BehaviorTreesOverview/index.html>.

156 Figure 3.3.15 Explore Tasks defined within the Behaviour Tree
Screen-captured by Author.

157 Figure 3.3.16 EQS allows us to check the surrounding environment to calculate locations based on how far away they are from the agent
Screen-recorded by Author.

157 Figure 3.3.17 Agent Exploring the space by utilizing this data
Simulated and screen-recorded by Author.

159 Figure 3.3.18 Behavior Tree Object-Looking state tasks
Screen-captured by Author.

160 Figure 3.3.19 EQS also allows us to calculate a random location that is biased to how far away they are from an object
Screen-recorded by Author.

161 Figure 3.3.20 Agents 'juggling' the distances to the objects until they look right
Simulated and screen-recorded by Author.

165 Figure 3.3.21 Simplified steps for establishing object Interaction tasks
Illustrated by Author.

165 Figure 3.3.22 Object-Interact State tasks within behavior tree
Screen-captured by Author.

167 Figure 3.3.23 Agent interacting with an object
Simulated and Screen-recorded by Author.

167 Figure 3.3.24 Agents lining up before interacting with the object
Simulated and Screen-recorded by Author.

169 Figure 3.3.25 Agent Interact state tasks within Behavior Tree
Screen-captured by Author.

169 Figure 3.3.26 Agents interacting with each other
Simulated and Screen-recorded by Author.

171 Figure 3.3.27 Threshold State tasks within Behavior Tree
Screen-captured by Author.

171 Figure 3.3.28 Agent moving through threshold
Simulated and Screen-recorded by Author.

173 Figure 3.3.29 To spawn Agents, we must first make a list of all the Entrance Threshold objects at the start of the simulation
Screen-captured by Author.

173 Figure 3.3.30 Spawning Agents at Entrance Thresholds
Screen-captured by Author.

175 Figure 3.3.31 Enter State tasks within the Behavior Tree
Screen-captured by Author.

175 Figure 3.3.32 Agents entering and exiting the building
Simulated and Screen-recorded by Author.

177 Figure 3.3.33 Agents sometimes becomes stuck in the corner due to the EQS continuously perceiving the corner in front of the agent to be the furthest point within the environment
Screen-captured by Author.

177 Figure 3.3.34 Unstuck State within Behavior Tree tells the agent to query a new location behind them if they don't move for longer than a specified time interval
Screen-captured by Author.

178 Figure 3.3.35 Bug with obtaining a position vector for establishing agent global offset
Simulated and Screen-recorded by Author.

178 Figure 3.3.36 Bug with obtaining a position for the agents to form a line
Simulated and Screen-recorded by Author.

179 Figure 3.3.37 Bug with animation looping for the Agent Interact State
Simulated and Screen-recorded by Author.

179 Figure 3.3.38 Another global offset bug when establishing a position vector from the agent's personal space
Simulated and Screen-recorded by Author.

180 Figure 3.3.39 Functional Behavior Tree during runtime
Simulated and Screen-recorded by Author.

183 Figure 3.3.40 UE4 animation system breakdown
From "Animation System Overview," Unreal Engine Documentation, accessed December 28, 2019, <https://docs.unrealengine.com/en-US/Engine/Animation/Overview/index.html>.

183 Figure 3.3.41 Autodesk Character Generator
Screen-captured by Author, from "Autodesk Character Generator," accessed December 28, 2019, <https://charactergenerator.autodesk.com/>.

185 Figure 3.3.42 Agents become lifeless floating objects without animation
Simulated and Screen-recorded by Author.

185 Figure 3.3.43 Animation Sequence within UE4 drives the skeletal mesh asset.
Screen-recorded by Author.

186 Figure 3.3.44 Blend Space
Screen-recorded by Author.

186 Figure 3.3.45 State Machine
Screen-captured by Author.

187 Figure 3.3.46 Adobe Maximo
Screen-recorded by Author, from "Mixamo," Adobe, accessed December 28, 2019, <https://www.mixamo.com/#/>.

189 Figure 3.3.47 Animations controlled by the State Machine during runtime
Simulated and screen-recorded by Author.

190 Figure 3.3.48 Flowchart of updated Human Systems within UE4
Illustrated by Author

193 Figure 3.4.1 The Construction Script is visually very similar to the Blueprint
Screen-captured by Author.

193 Figure 3.4.2 The Construction Script allows us to sync model attributes within the game space with variables within the Blueprint
Screen-recorded by Author.

195 Figure 3.4.3 Command Center viewport and properties
Screen-captured by Author.

195 Figure 3.4.4 The Command Center asset contains the blueprint nodes (as defined in the Enter State on pg. 168) to spawn agents at thresholds every game tick during the simulation process
Screen-captured by Author.

197 Figure 3.4.5 Entrance/Exit viewport and properties
Screen-captured by Author.

197 Figure 3.4.6 Threshold viewport and properties
Screen-captured by Author.

199 Figure 3.4.7 Architectural Object viewport and properties
Screen-captured by Author.

199 Figure 3.4.8 The 3D mesh and materials of the object can be changed in the properties panel depending on the typology of the actual object
Screen-captured by Author.

200 Figure 3.4.9 Blueprint Interface node within the Interact node within the Object Interact State of the Behavior Tree.
Screen-captured and edited by Author.

201 Figure 3.4.10 The Interact Event within the object blueprint allows it to change the color of its texture whenever the Interact node within the behavior tree of the Agent is triggered
Screen-captured by Author.

203 Figure 3.5.1 3ds Max export window
Screen-captured and edited by Author.

- 203 Figure 3.5.2 Static Mesh collision properties
Screen-captured and edited by Author.
- 204 Figure 3.5.3 Collision lines right after Importing from Revit
By Twiz, “Unreal Engine 4 Tutorial - Export From Revit to UE4,” YouTube, 9:44, accessed December 28, 2019, https://www.youtube.com/watch?v=Ux_zJ4WJbZg.
- 204 Figure 3.5.4 Fixed collision lines within UE4 with a physical material
By Twiz, “Unreal Engine 4 Tutorial - Export From Revit to UE4.”
- 205 Figure 3.5.5 Landscape Creation in UE4
By Virtus Learning Hub / Creative Tutorials, trimmed by Author, “Populating Scenes With The Foliage Tool - #17 Unreal Engine 4 Level Design Tutorial Series,” YouTube, 14:18, accessed December 28, 2019, <https://www.youtube.com/watch?v=XYYfYDqsDA>.
- 207 Figure 3.6.1 Step 1: Import FBX model
Screen-recorded by Author.
- 207 Figure 3.6.2 Step 1 sequential frames
Frame-captured by Author.
- 208 Figure 3.6.3 Step 2: Define entrances/exits and thresholds
Screen-recorded by Author.
- 208 Figure 3.6.4 Step 2 sequential frames
Frame-captured by Author.
- 209 Figure 3.6.5 Step 3: Define Interactive Elements
Screen-recorded by Author.
- 209 Figure 3.6.6 Step 3 sequential frames
Frame-captured by Author.
- 210 Figure 3.6.7 Step 4: Initiate the simulation
Simulated and screen-recorded by Author.
- 211 Figure 3.6.8 Step 4 sequential frames
Frame-captured by Author.
- 212 Figure 3.6.9 Various forms of data can be visualized and mapped out during the simulation and toggled on or off via the GUI (Graphical User Interface) or hotkeys
Simulated and screen-recorded by Author.
- 213 Figure 3.6.10 Data visualization sequential frames
Frame-captured by Author.
- 214 Figure 3.6.11 GUI controlled Solar Studies
Simulated and screen-recorded by Author.
- 215 Figure 3.6.12 GUI controlled Solar Studies sequential frames
Frame-captured by Author.
- 216 Figure 3.6.13 The GUI can also be used to control various scripted events, such as an evacuation scenario
Simulated and screen-recorded by Author.
- 217 Figure 3.6.14 Scenario programming sequential frames
Frame-captured by Author.
- 218 Figure 3.6.15 The simulation can also be visualized and interacted from different perspectives to better visualize the space
Simulated and screen-recorded by Author.
- 219 Figure 3.6.16 Interactive perspective variation sequential frames
Frame-captured by Author.
- 220 Figure 3.6.17 The virtual camera can be utilized to simulate a real camera to produce cinematic footage
Screen-recorded by Author.
- 221 Figure 3.6.18 Virtual camera sequential frames
Frame-captured by Author.

Part 4 | Tool Evaluation

- 226 Figure 4.1.1 Open Space Condition
Simulated and screen-recorded by Author.
- 227 Figure 4.1.2 Open Space analytical frames
Frame-captured by Author.
- 228 Figure 4.1.3 Corridor Condition
Simulated and screen-recorded by Author.

- 229 Figure 4.1.4 Corridor analytical frames
Frame-captured by Author.
- 230 Figure 4.1.5 Intersection Condition
Simulated and screen-recorded by Author.
- 231 Figure 4.1.6 Intersection analytical frames
Frame-captured by Author.
- 232 Figure 4.1.7 Expansion Condition
Simulated and screen-recorded by Author.
- 233 Figure 4.1.8 Expansion analytical frames
Frame-captured by Author.
- 234 Figure 4.1.9 Open space condition sped-up 20x
Simulated and screen-recorded by Author.
- 234 Figure 4.1.10 Open space condition sped-up analytical frames
Frame-captured by Author.
- 235 Figure 4.1.11 Grand Central Station time-lapse
From Rocketboom, “Time Lapse Grand Central Station,” YouTube, 2:54, accessed December 28, 2019, <https://www.youtube.com/watch?v=eimuAboXSdo&feature=youtu.be>.
- 235 Figure 4.1.12 Grand Central Station time-lapse analytical frames
Frame-captured by Author, from Rocketboom, “Time Lapse Grand Central Station.”
- 236 Figure 4.1.13 The simulated agents will react and adapt to a changing environment in real time.
Simulated and screen-recorded by Author.
- 237 Figure 4.1.14 Adaptive agents sequential frames
Frame-captured by Author.
- 238 Figure 4.2.1 Cushion Revit model
Screen-captured by Author.
- 239 Figure 4.2.2 Cushion Floor Plan
Illustrated by Author, CAD file obtained from the Cushion team.
- 240 Figure 4.2.3 Cushion crowd simulation
Simulated and screen-recorded by Author.
- 241 Figure 4.2.4 Cushion crowd simulation analytical frames
Frame-captured by Author.
- 242 Figure 4.2.5 Cushion agent-comfort map
Simulated and screen-recorded by Author.
- 243 Figure 4.2.6 Cushion agent-comfort map analytical frames
Frame-captured by Author.
- 244 Figure 4.2.7 Cushion real world footage
Filmed by Author.
- 244 Figure 4.2.8 Cushion real world footage analytical frames
Frame-captured by Author.
- 245 Figure 4.2.9 Cushion rendered visualization
Simulated by Author.
- 245 Figure 4.2.10 Cushion rendered visualization analytical frames
Frame-captured by Author.
- 247 Figure 4.2.11 Cushion rendered visualization without simulated crowds
Simulated by Author.
- 247 Figure 4.2.12 Cushion rendered visualization analytical frames without simulated crowds
Frame-captured by Author.
- 248 Figure 4.3.1 Riverside Gallery Revit model
Screen-captured by Author.
- 249 Figure 4.3.2 Riverside Gallery Floor Plan
Illustrated by Author, PDF file obtained from “Floor Plans,” Plant Operations, accessed 28, 2019, <https://uwaterloo.ca/plant-operations/floor-plans>.
- 250 Figure 4.3.3 Riverside Gallery crowd simulation
Simulated and screen-recorded by Author.
- 251 Figure 4.3.4 Riverside Gallery crowd simulation analytical frames
Frame-captured by Author.
- 252 Figure 4.3.5 Riverside Gallery agent-comfort map
Simulated and screen-recorded by Author.

- 253 Figure 4.3.6 Riverside Gallery agent-comfort map analytical frames
Frame-captured by Author.
- 254 Figure 4.3.7 Riverside Gallery real world footage
Filmed by Author.
- 254 Figure 4.3.8 Riverside Gallery real world footage analytical frames
Frame-captured by Author.
- 255 Figure 4.3.9 Riverside Gallery rendered visualization
Simulated by Author.
- 255 Figure 4.3.10 Riverside Gallery rendered visualization analytical frames
Frame-captured by Author.
- 257 Figure 4.3.11 Riverside Gallery rendered visualization without simulated crowds
Simulated by Author.
- 257 Figure 4.3.12 Riverside Gallery rendered visualization analytical frames without simulated crowds
Frame-captured by Author.

Part 5 | Next Steps

- 263 Figure 5.1.1 Blueprint scripting vs C++ scripting
From Jayanam, frame-captured and edited by Author, “Unreal Engine 4 : C++ and Blueprints Tutorial,” YouTube, 7:36, accessed January 1, 2020, <https://www.youtube.com/watch?v=SW09W182Ws0>.
- 265 Figure 5.1.2 An airport is one example of a dynamic space that requires multiple layers of queuing, and many groupings of occupants
By John Amis, from Don Schanche Jr., “Airlines Struggle to Get Back on Schedule after Atlanta Fire,” December 18, 2017, FWBP, accessed January 1, 2020, http://www.fortworthbusiness.com/news/airlines-struggle-to-get-back-on-schedule-after-atlanta-fire/article_c1ce0496-e41b-11e7-8980-5b5f5acbd106.html.
- 267 Figure 5.2.1 Photogrammetry recreates an object by capturing multiple images of the object in various angles
From Joseph Azzam, “Everything You Need to Know about Photogrammetry I Hope,” January 10, 2017, Gamasutra, accessed January 1, 2020, https://www.gamasutra.com/blogs/JosephAzzam/20170110/288899/Everything_You_Need_to_Know_about_Photogrammetry_I_hope.php.
- 268 Figure 5.2.2 Star Wars Battlefront Photogrammetry Mod
From Martin Bergman, trimmed by Author, “STAR WARS™ Battlefront: Toddyhancer Showcase (Less Filmic Version),” YouTube, 0:52, accessed January 1, 2020, https://www.youtube.com/watch?v=a72hU_l6mKc.
- 268 Figure 5.2.3 Rebirth photorealism within UE4 demo
From Quixel, trimmed by Author, “Rebirth: Introducing Photorealism in UE4,” YouTube, 2:24, accessed January 1, 2020, <https://www.youtube.com/watch?v=9fC20NWhx4s>.
- 269 Figure 5.2.4 Star Wars realtime Ray-tracing demo
Trimmed by Author, videos from moviemaniacsDE, “Star Wars: Reflections | Official Unreal Engine Real-Time Ray-Tracing Demo (2018),” YouTube, 4:09, accessed January 1, 2020, <https://www.youtube.com/watch?v=AV279wIhmVU>, and Unreal Engine, “Reflections Real-Time Ray Tracing Demo | Project Spotlight | Unreal Engine,” YouTube, 1:04, accessed January 1, 2020, <https://www.youtube.com/watch?v=J3ue35ago3Y>.
- 271 Figure 5.3.1 Tiltbrush is one example of an VR painting application
From “Tilt Brush by Google,” accessed January 1, 2020, <https://www.tiltbrush.com/>.
- 272 Figure 5.3.2 Kinect Body Tracking within UE4
Trimmed by Author, Videos from Opaque Media Group, trimmed by Author, “Kinect 4 Unreal 1.1 - Introduction,” YouTube, 9:17, accessed January 1, 2020, <https://www.youtube.com/watch?v=WHmPvZvRyxc>, and Jiayi Wang, “Kinect for UE4 local multi rig tracking test,” YouTube, 1:51, accessed January 1, 2020, <https://youtu.be/KZuauZW8Tgg>.
- 272 Figure 5.3.3 Leap Motion Hand Tracking
From Leap Motion, trimmed by Author, “Leap Motion Blocks for Oculus Rift Playthrough,” YouTube, 6:15, accessed January 1, 2020, https://www.youtube.com/watch?v=oZ_53T2jBGg.
- 273 Figure 5.3.4 Utilizing a VR camera rig in the making of the short film Rebirth in UE4
From Quixel, trimmed by Author, “Create Photoreal Cinematics in UE4: Rebirth Tutorial,” YouTube, 44:56, accessed January 1, 2020, <https://www.youtube.com/watch?v=0iQJkSpOoOQ&feature=youtu.be&t=2215>.
- 273 Figure 5.3.5 Project Siren demonstrates real-time face and body tracking alongside photorealistic human rendering within UE4
Trimmed by Author, videos from Unreal Engine, “Siren Real-Time Performance | Project Spotlight | Unreal Engine,” YouTube, 0:41, accessed January 1, 2020, <https://www.youtube.com/watch?v=9owTAISsvwk>, and “Siren Behind The Scenes | Project Spotlight | Unreal Engine,” YouTube, 0:51, accessed January 1, 2020, <https://www.youtube.com/watch?v=NW6mYurjYZ0>.
- 274 Figure 5.4.1 Python Editor Script Plugin within UE4
From “Scripting the Editor Using Python,” Unreal Engine Documentation, accessed October 18, 2019, <https://docs.unrealengine.com/en-US/Engine/Editor/ScriptingAndAutomation/Python/index.html>.

- 275 Figure 5.4.2 Datasmith is a collection of tools and plugins that automates various tasks from traditional workflows
From Ken Pimentel, “Technology Sneak Peek: Python in Unreal Engine,” Unreal Engine, November 15, 2017, accessed November 20, 2019, <https://www.unrealengine.com/en-US/tech-blog/technology-sneak-peek-python-in-unreal-engine>.
- 275 Figure 5.4.3 An example of a Python script used to automatically generate a LOD from a higher complexity mesh
From Pimentel, “Technology Sneak Peek: Python in Unreal Engine.”

List of Abbreviations

ACADIA: Association for Computer Aided Design in Architecture

ALU: Arithmetic Logic Units

API: Application Programming Interface

CAD: Computer Aided Design

CAM: Computer Aided Manufacturing

CPU: Central Processing Unit

DIY: Do It Yourself

DOF: Depth Of Field

EQS: Environment Query System

GE: Game Engine

GPU: Graphics Processing Unit

GUI: Graphical User Interface

HiDAC: High-Density Autonomous Crowds

IDE: Integrated Development Environment

IoT: The Internet of Things

OOP: Object Oriented Programming

PBR: Physically Based Rendering

UE4: Unreal Engine 4

Technical Note

This thesis contains animated video figures.

To view this paper in its intended format, download and extract the corresponding zip folder to the same folder location of this PDF file.

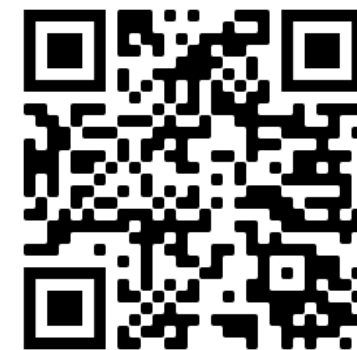
These videos can then be opened by clicking on any corresponding figure with an “▶” symbol.

Depending on if the video figure is linked to the local zip folder, or an online source, the video may be launched in a separate media player, or on the web browser.

Alternatively, these local video files can be opened manually by referring to their filenames, which will match the figure numbers in the work.

See *Appendix A: Multimedia Figures* for more details.

This thesis can also be viewed online at:
<https://leoaoliu.github.io/Thesis/>



Part 1 | Introduction and Theory

Why do we Need a New Workflow?

Why do we Need a New Workflow?

Traditionally, buildings have been very static elements within society, however, technological advancements in recent years have allowed faster, smaller, and cheaper electronics to be embedded within architectural systems. As a result, coded infrastructures are emerging, allowing the creation of dynamic architectural spaces throughout the world. Now as humanity enters the information age, new tools, knowledge, and technologies have made it possible to create and visualize new forms of architectural spaces unlike ever before.

These new spaces bring about complexities both in operation and design, which in turn demands a higher standard of visualization to fully portray the occupant interactions within the space. The inherent dynamics of human occupancy are already challenging to portray with current visualization methods—especially within the ‘fast-paced’ design phases of most architectural projects—but becomes even more so with the introduction of these increasingly complex interactions within these increasingly dynamic spaces.

A solution to these problems may lie within the game design industry. Since video games rely largely on real-time interactions, the tools for their creation require an emphasis on both rendering speed and scripting. These characteristics are in line with the requirements of simulating and visualizing occupancy interactions within dynamic architectural spaces, and as such offer a valid direction of investigation.

This section of the thesis will investigate these claims to provide a foundation for this thesis. Chapter 1.1 will first define the interpretation of *dynamic architecture* within the context of this thesis. It will then investigate the formalities of what makes a space dynamic by supporting it with various logical tests and real-world examples. Chapter 1.2 will then investigate the progression of visualization both within the world and within architectural visualization. From this, it intends to deduce the reasons behind the current inadequacies of current visualizations in portraying these new dynamic spaces, as well as the lack of crowd dynamics within them. Chapter 1.3 will then investigate the advent of the game engine and examine why it may be a suitable tool to make up for these shortcomings. Chapter 1.4 will then consider all these points and propose a methodology and framework for this thesis.

“Increasingly active, responsive, and kinetic, the material of the built environment is being animated in the truest sense of the word. Architecture imbued with autonomy, and uncanny sense of life, challenges us to look beyond design disciplines to understand the perceptual, emotional, and social effects of these pervasive technologies.”¹

¹ Michael Fox, *Interactive Architecture: Adaptive World* (New York: Princeton Architectural Press, 2016), 7.

Chapter 1.1 | Emergence of Interactive and Dynamic Architecture

Dynamic, as defined by the Oxford English Dictionary, is: “Of or pertaining to force producing motion: often opposed to static.”¹ From this definition, it is then possible to interpret *dynamic architecture* as a class of architecture pertaining to a space that is able to employ motion. This contrasts with the conventional *static* sense of architecture, where people typically associate buildings with defined walls and thresholds. The origin of this association perhaps comes from the origin of architecture itself, where its purpose was that of a shelter against the unstable nature, to allow humanity to live in a space of control and order. Abraham Akkerman briefly touched upon this in his article “Urban Void and the Deconstruction of Neo-Platonic City-Form,” in which he relates the two facets of city form—urban constructs and urban voids—with what Friedrich Nietzsche showed as the two impulses of human psyche—the Apollonian and the Dionysian.

“He called the two impulses the Apollonian and the Dionysian, respectively. The spatial attributes of the human temperament, epitomized by Apollo, the god of colonies and of city-walls, correspond to harmony, order, reason, certainty and stability. Capriciousness and turbulence, expressive of Dionysus, the bisexual god of wine, on the other hand animate euphoric and rapturous attributes of the human character, involving unpredictable outbursts tempered by intervals of quiet (Zeitlin, 1982). It is only a small conceptual step to relate the mind’s spatial disposition to a planned shelter and its temporal outlook to raw nature and open space. [...] It is from within the tension between the turbulence and uncertainty of nature’s ferocity, and the firmness and security of a human-made shell, that the intellectual quandary of uniformity amid diversity, and of permanence amid change, arose. [...] The origin of the mind city composite, thus, seems to be traceable to mutual relationship between nature’s peril and a thought about, or a mental image of, a shelter against it.”²

Akkerman contrasts the “firmness and security of a human-made shell,” to the “turbulence and uncertainty of nature’s ferocity;” describing the shelter with words such as *harmony, order, reason, certainty* and *stability*, while describing nature as *raw* and *open*, relating it to that of Dionysus, who can be expressed by words such as *capriciousness* and *turbulence*. While it is unlikely Akkerman had *dynamic architecture* in mind, it can be inferred from his passage the archetypal *static* quality of architecture and the *dynamic* quality of nature. *Dynamic architecture* then can adopt both archetypes, becoming a more fluid form of stability and security. This merger of the two archetypes presents an interesting repositioning of agency, which proposes a mirror of what

¹ “Dynamic, Adj. and n.,” in *OED Online* (Oxford University Press), accessed October 18, 2019, <http://www.oed.com/view/Entry/58818>.

² Abraham Akkerman, “Urban Void and the Deconstruction of Neo-Platonic City-Form,” *Ethics, Place & Environment* 12, no. 2 (2009): 207-208, <https://doi.org/10.1080/13668790902863416>.

Akkerman phrases as “the intellectual quandary of uniformity amid diversity, and of permanence amid change.”^[3] Instead of “uniformity amid diversity,” *dynamic architecture* would instead potentially introduce uncertainty within stability. However, while the stability gained by removing the agency of nature provided the basis of a valid shelter, the potential for controlled dynamics gained by translocating agency from the occupants back into the architecture can be just as, if not more, compelling. It can be seen within life that occupancy within a space is inherently dynamic, and as such, the demand and functions of a space can change throughout its use. People are dynamic, crowds are dynamic, and sometimes the static nature of traditional architecture cannot fully accommodate the dynamic nature of its occupants. Dynamics can disrupt the stability and order of architectural spaces, but when done right, they can actually enhance it.

Architectural space, when broken down to its components, can be categorized into various elements. When considering a typical house, one might notice elements such as furniture, thresholds, lights, windows, etc. These elements are generally what define a space; a kitchen might have elements such as a fridge, sink, microwave, etc. whereas a living room would have elements such as a couch, coffee table, and television. At present, these elements are relatively simple, but even so, they can range in complexity from static tables and chairs to more interactive devices such as televisions and computers. While these higher complexity elements are still limited by their lack of physical motion, the photonic and acoustic stimuli they can release within a space go beyond what a static piece of furniture can accomplish. A television that is turned on will introduce motion to the room by virtue of light and sound. Therefore, a television that is turned on can be considered dynamic while a television that is turned off can be considered static. Accordingly, a living room with a television that is turned on will feel more dynamic compared to a living room with a television that is turned off. (Fig. 1.1.1 - 2) From this, it can then be argued that the introduction of dynamic elements within the space can cause an initially static space to become dynamic. While natural elements such as a wind and light can also influence this space dynamic due to the time of day and the seasons, these elements are both less deliberate (unless controlled by a dynamic element) and more subtle (due to their effect over a greater timeframe when compared to the immediate stimuli delta of the television example).

In these instances, the semantics of dynamic spaces may differ based on the threshold at which the space is classified to be dynamic, but if one deems the threshold of dynamic to be anything above static within a perceivable timeframe, then it can be reasoned that it only takes one dynamic element to convert a space from static to dynamic—whether it is an element like a television that introduces dynamics, or some kind of screen element that can dynamically affect natural elements such as light and wind. This can be further demonstrated by comparing a room with a lightbulb versus a room with a fireplace. (Fig. 1.1.3 - 6) While both elements provide a light source to the room, the light fixture provides a constant static lighting whereas the fire dances, changing shape from convectional air currents within the space.

3 Akkerman, “Urban Void and the Deconstruction of Neo-Platonic City-Form,” 208.



Figure 1.1.1 A static living room with the Television turned off



Figure 1.1.2 A static living room with the Television turned on feels more dynamic by comparison



Figure 1.1.3 *Virtual Rendering showing a static light source within a room*



Figure 1.1.4 *Virtual Rendering showing a dynamic light source within a room.*

Figure 1.1.3 - 1.1.4

This is a virtual study of a room with either a static light bulb or a dynamic fire. It can be seen here that while the fire is technologically primitive compared to the light-bulb, it causes the space to feel much more dynamic.



Figure 1.1.5 *Physical Study showing a static light within a space.*



Figure 1.1.6 *Physical Study showing a dynamic light within a space.*

Figure 1.1.5 - 1.1.6

The same study, but within the Physical world. Here it can be seen that the same ideology applies to these spaces even at this smaller scale.

Whether the fire or television is sufficient to convert the room into a dynamic space is debatable, but what is clear is that the fire is more dynamic than the lightbulb, and the room with the fire will feel more dynamic than the room with the lightbulb; Conversely, the television is more dynamic when turned on then off, and the room with the television turned on is more dynamic than with the television turned off.

Both the room with fire and the room with television analogies are generic examples by logic, but more substantial examples of these elementary dynamic spaces can be dated back throughout history, such as the beacon towers of the Great Wall of China,^[4] the drawbridges of medieval castles, and lighthouses throughout the world. (Fig. 1.1.7 - 9) These forms of architecture represent the most basic uses of dynamic elements, where their utility focused on a singular function. While this is a valid use case, the potential of dynamics becomes much greater with the addition of multiple dynamic elements. Prime examples of this can be observed in the form of the Zen garden in the Ryoan-ji Shrine in Kyoto, Japan, and the multitude of fountains at the Villa d'Este in Tivoli, Italy. These spaces utilize multiple dynamic elements—deriving its dynamics from nature such as the gravel within the gardens, the leaves within the trees, and the water within the fountains—to enhance the space for the occupants. (Fig. 1.1.10 - 15)

⁴ Cheng Dalin, "The Great Wall of China," in *Borders and Border Politics in a Globalizing World*, ed. Paul Ganster and David E. Lorey (Lanham, MD: SR Books, 2005), 12-13.



Figure 1.1.7 Rotating light of lighthouse

Figure 1.1.7 - 1.1.9

These three structures are examples of simple dynamics being utilized for a specific function. The great wall of china made use of smoke signals on top of its watch towers to signal of incoming invasions. Draw bridges of medieval castles utilized a hinge system to control passage into the castle. Light houses utilized a light source such as fire or a rotating mirror to signal



Figure 1.1.8 Smoke Signals of the Great wall of china

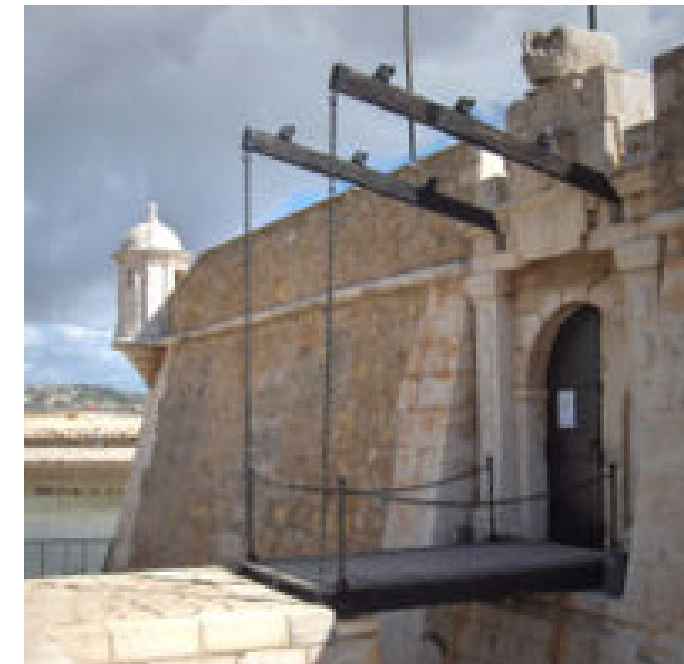


Figure 1.1.9 Drawbridge at the fort of Ponta da Bandeira



Figure 1.1.10 *The fountain of Neptune*



Figure 1.1.11 *One hundred fountains walkway*

Figure 1.1.10 - 1.1.12

Villa D'Este in Tivoli Italy, erected in the 16th century, makes use of many water features throughout its property. The use of water creates spaces that are inherent static but becomes dynamic as the fluid not only reacts to the architecture, but also the occupants. This in turn floods the senses with dynamic sounds, temperature changes, as well as providing the occupants with an interactive medium to manipulate.

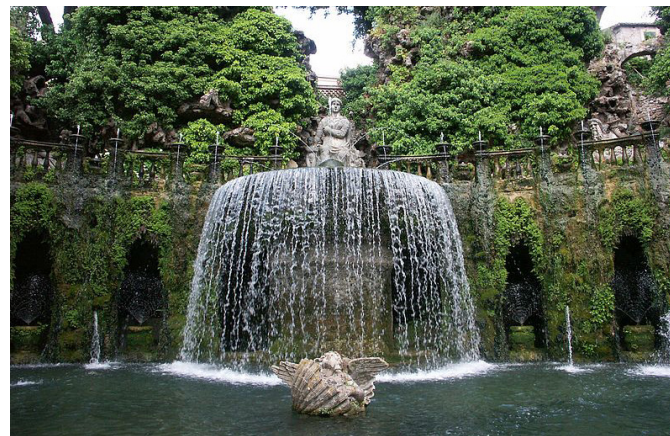


Figure 1.1.12 *Oval Fountain*



Figure 1.1.13 *Spectators viewing the Rock Gardens from the veranda*



Figure 1.1.14 *Cherry blossoms hang over the rock garden, bringing the dynamics of nature further into the space*



Figure 1.1.15 *Cherry blossoms hang over the rock garden, bringing the dynamics of nature further into the space*

Figure 1.1.13 - 1.1.15

The Ryoan-Ji shrine in Kyoto Japan, erected in the 13th century, is famous for its rock garden which provides an ever-changing space within the premise. While the garden is simply meant to be viewed by the public, its changing arrangements along with its natural provides a more dynamic environment compared to traditional architecture.

With the addition of technology, however, the potential use cases of dynamics become even greater. Digital computation of integrated circuits has continued to progress, experiencing an exponential growth in power and a reduction in cost, creating a chain reaction where advancements in one field can drive advancements in another. Gordon E. Moore states in his 1965 paper “Cramming More Components onto Integrated Circuits:”

“Integrated electronics will make electronic techniques more generally available throughout all of society, performing many functions that presently are done inadequately by other techniques or not done at all. [...] Reduced cost is one of the big attractions of integrated electronics, and the cost advantage continues to increase as the technology evolves toward the production of larger and larger circuit functions on a single semiconductor substrate. [...] The complexity for minimum component costs has increased at a rate of roughly a factor of two per year (see graph). Certainly over the short term this rate can be expected to continue, if not to increase. Over the longer term, the rate of increase is a bit more uncertain, although there is no reason to believe it will not remain nearly constant for at least ten years.”^[5]

Moore’s Law, although not a law by the traditional definition, is a perceived rate of technological growth based on these observations. According to Moore, technological progression will continue to grow where the number of transistors within an integrated circuit (therefore, computational power) will double approximately every year. He later updates this observation to every two years in 1975.^[6] While Moore’s Law is by no means certain, this observation has proven to be fairly accurate from the technological progressions of the past decades. (Fig. 1.1.16 - 17)

This has allowed the production of relatively cheap electronics, which are now integrated throughout people’s everyday lives. It can be seen as smartphones in people’s hands, appliances in various homes, computers in various offices, and as streetlights around the world. Michael Fox states in *Interactive Architecture: Adaptive World*, “The field of industrial design came to engage with tangible interaction out of necessity as appliances became progressively ‘intelligent’ containing more and more electronic and digital components,”^[7] which not only substantiates this observation of technological integration amongst the populace but also mentions the influence of this technological procession on other fields, which shows the significance of this technological revolution on the world. With so much computational data around us, and the ever-increasing accessibility of intelligent objects, the potential for integrating such technologies within architectural systems becomes increasingly powerful. Many architectural elements are already making use of these technological

5 Gordon E. Moore, “Cramming More Components onto Integrated Circuits,” *Proceedings of the IEEE* 86, no. 1 (1998): 83, <https://doi.org/10.1109/jproc.1998.658762>.
 6 Gordon E. Moore, “Progress in Digital Integrated Electronics [Technical Literature, Copyright 1975 IEEE. Reprinted, with Permission. Technical Digest. International Electron Devices Meeting, IEEE, 1975, Pp. 11-13.],” *IEEE Solid-State Circuits Society Newsletter* 11, no. 3 (2006): 37, <https://doi.org/10.1109/n-ssc.2006.4804410>.
 7 Michael Fox, *Interactive Architecture: Adaptive World* (New York: Princeton Architectural Press, 2016), 12.

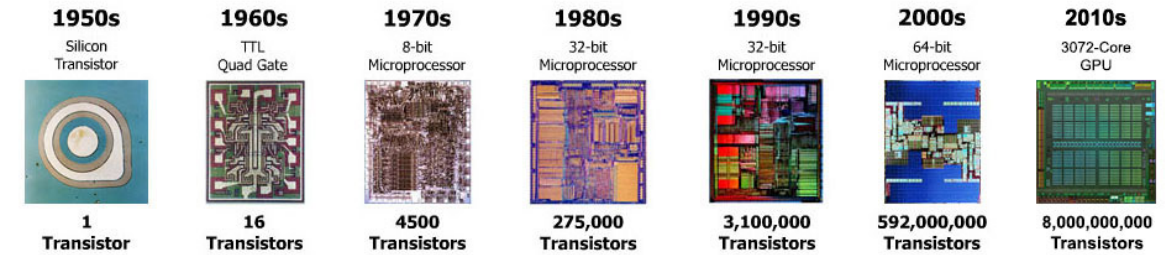
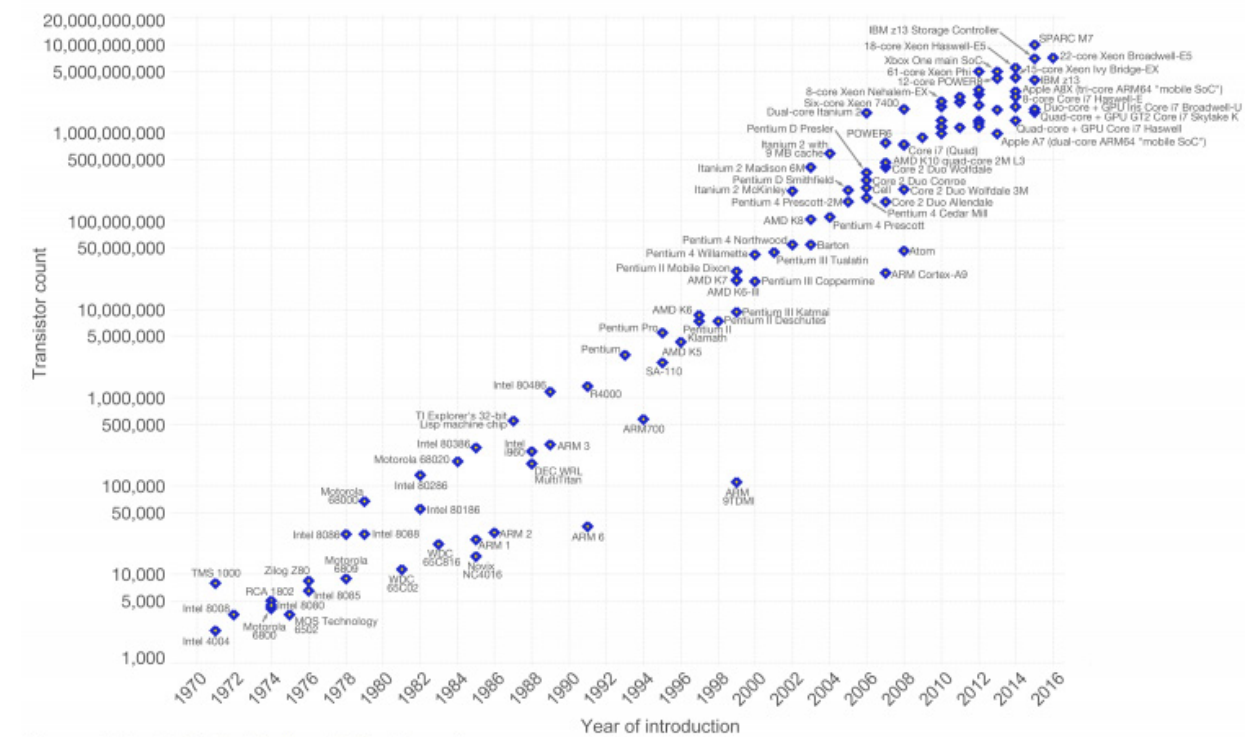


Figure 1.1.16 Silicon transistor progression through the years

Moore’s Law – The number of transistors on integrated circuit chips (1971-2016) 
 Moore’s law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important as other aspects of technological progress – such as processing speed or the price of electronic products – are strongly linked to Moore’s law.



Data source: Wikipedia (https://en.wikipedia.org/wiki/Transistor_count)
 The data visualization is available at [OurWorldinData.org](https://www.ourworldindata.org). There you find more visualizations and research on this topic. Licensed under CC-BY-SA by the author Max Roser.

Figure 1.1.17 This graph shows the progression of transistor count within integrated circuit chips through the years, as described by Moore’s Law

improvements, from standard functional elements such as elevators and automatic doors, to more complicated constructed elements such as the solar responsive façades of the Al Bahar Towers in Abu Dhabi,^[8] the shading umbrellas in Medina,^[9] and the operable roof of the Rogers Centre in Toronto.^[10] (Fig. 1.1.18 - 20) These examples are all valid ways of embedding technology within architecture and could very well be only the beginning of what will be possible in the near future. If technology continues to follow the trend observed by Moore's Law, this current level of coded infrastructure will seem minuscule to what's to come.

Even now, prefabricated Do-It-Yourself (DIY) solutions such as Sonoff smart controllers,^[11] Raspberry Pi computers,^[12] and Arduino micro-controllers^[13] are becoming increasingly accessible to the general public. This is facilitated not only by the reduction in cost of micro-controllers but also the increasing accessibility to information as a result of more people having access to high speed Internet. The world is becoming increasingly connected, and as a result of this connection, these various electronic and digital components will gain the ability to communicate with one another. The Internet of Things refers to an emerging global internet-based information architecture that is slowly developing, allowing the networked interconnection of everyday objects, vehicles, and buildings with embedded intelligence. As defined by the International Telecommunication Union, the Internet of Things is "a global infrastructure for the information society."^[14] This has the potential to transition current infrastructures into smart grids, where the integration of physical and digital systems will allow the automation of everyday tasks and open new forms of digital and physical communication. Privacy concerns aside, this new level of worldwide connectivity will allow higher potential flexibility within architectural design, allowing the possibility of altering both the occupant's perception of the space, as well as the functionalities of the space. In Fox's words:

"The influence of technological and economic feasibility within a connected world has resulted in the explosion of current exploration with the foundations of interaction design in architecture. The Internet of Things (IoT) has quite rapidly come to define the technological context of interactive design as all-inclusive, existing within this connectedness in a way that affects essentially everything, from graphics to objects to buildings to cities. [...] Interactive are no longer limited to those of people interacting with an object, environment, or

8 Russell Fortmeyer and Charles D. Linn, "Abu Dhabi Investment Council Headquarters" in *Kinetic Architecture: Designs for Active Envelopes* (Mulgrave: Images Publishing, 2014), 176-183.

9 Michael Barnes and Michael Dickson, *Widespan Roof Structures* (London: Telford, 2000), 14-16.

10 Andrew H Frazer, "Design Considerations for Retractable-roof Stadia" (Master's thesis, 2005), 8-11, accessed July 23, 2019, <https://dspace.mit.edu/handle/1721.1/31119>.

11 "DIY A Temperature Controlled Smart Lock," Sonoff, accessed July 23, 2019, <https://sonoff.ithead.cc/en/news/266-diy-a-temperature-controlled-smart-lock>.

12 "Raspberry Pi Blog - News, Announcements, and Ideas," accessed July 23, 2019, Raspberry Pi, <https://www.raspberrypi.org/blog/>.

13 "What Is Arduino?," Arduino, accessed July 23, 2019, <https://www.arduino.cc/en/Guide/Introduction>.

14 "Internet of Things Global Standards Initiative." ITU, accessed July 23, 2019, <https://www.itu.int/en/ITU-T/gsi/iot/Pages/default.aspx>.

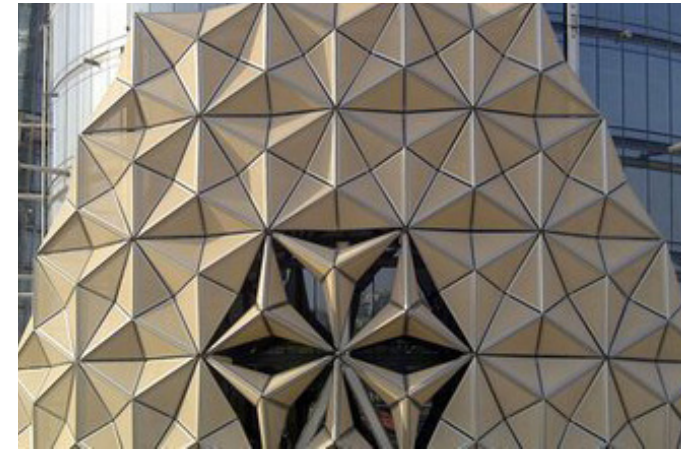


Figure 1.1.18 The Al Bahar Towers Facades utilizes motorized folding louvers to control the amount of sunlight that can pass through.



Figure 1.1.19 The Umbrellas in Medina opens and closes to open up the space as well as offer environmental protection depending on the weather and time of day.



Figure 1.1.20 The Roof of the Rogers Center in Toronto opens and closes to provide outdoor or indoor experiences depending on the exterior conditions.

building, but can now be carried out as part of a larger ecosystem of connected objects, environments, and buildings that autonomously interact with each other.”^[15]

These spaces, the spaces previously only associated with science fiction, are becoming increasingly possible as newer and cheaper technology becomes available. Imagine spaces where furniture moves to accommodate the number of occupants, where the lighting changes depending on identity, and the walls flex to accommodate circulation. Imagine elevators that can *sense* the flow of people arriving and adjust accordingly, spaces that know the identity of its occupants and tailor its functionality for them, and infrastructure that can perform certain tasks depending on the time of day and where its occupants are located.

While some of these technologies are still in their infancy, more and more people are beginning to experiment with these new forms of spaces. Every year, art festivals such as Nuit Blanche Toronto^[16] allow artists and designers to prototype new forms of dynamic spaces (Fig. 1.1.21 - 23) and conferences such as ACADIA (Association for Computer Aided Design in Architecture)^[17] are showcase new forms of spaces and technology integration in utility and planning. (Fig. 1.1.24 - 26) In the book *Alive: Advancements in adaptive architecture*, Manuel Kretzer and Ludger Hovestadt assemble a collection of essays that challenge questions concerning “temporality and decay, or concepts dealing with performance, feedback, and progression” categorized into the following chapters from the Alive 2013 symposium:

“Bioinspiration highlights a sensitive observation of biological processes and their transfer into novel design methodologies for the creation of innovative architectural explorations. [...] Materiability addresses the potential to control and design matter at a nano—or micro—scale and construct materials that are dynamic, active, and responsive to environmental conditions. [...] Interaction elaborates on concepts concerning interaction and adaptation that exceed pure control and automation mechanisms but attempt to change, learn, and evolve dynamically.”^[18]

This collection shows the variety of spaces that these technologies can influence and the vast amount of strategies that can be utilized to create them. Annual collaborations such as these act as platforms and environments that allow architects to push technology to new bounds, creating prototype spaces and tools that forecast what the future of architecture may hold.

15 Fox, *Interactive Architecture: Adaptive World*, 11.

16 “Nuit Blanche,” City of Toronto, accessed October 10, 2019, <https://www.toronto.ca/explore-enjoy/festivals-events/nuitblanche/>.

17 “About ACADIA,” ACADIA, accessed October 16, 2019, <http://acadia.org/>.

18 Manuel Kretzer and Ludger Hovestadt, *ALIVE: Advancements in Adaptive Architecture* (Basel: Birkhäuser, 2014), 21-22.



Figure 1.1.21 Starscape



Figure 1.1.22 Ocean



Figure 1.1.23 Cushion



Figure 1.1.24 Acadia conference in Michigan



Figure 1.1.25 Acadia lecture in Michigan

Figure 1.1.24 - 1.1.26

ACADIA stands for the Association for Computer Aided Design In Architecture. Every year, digital design researchers and professionals hold a conference to facilitate critical investigation into the role of computation in architecture.

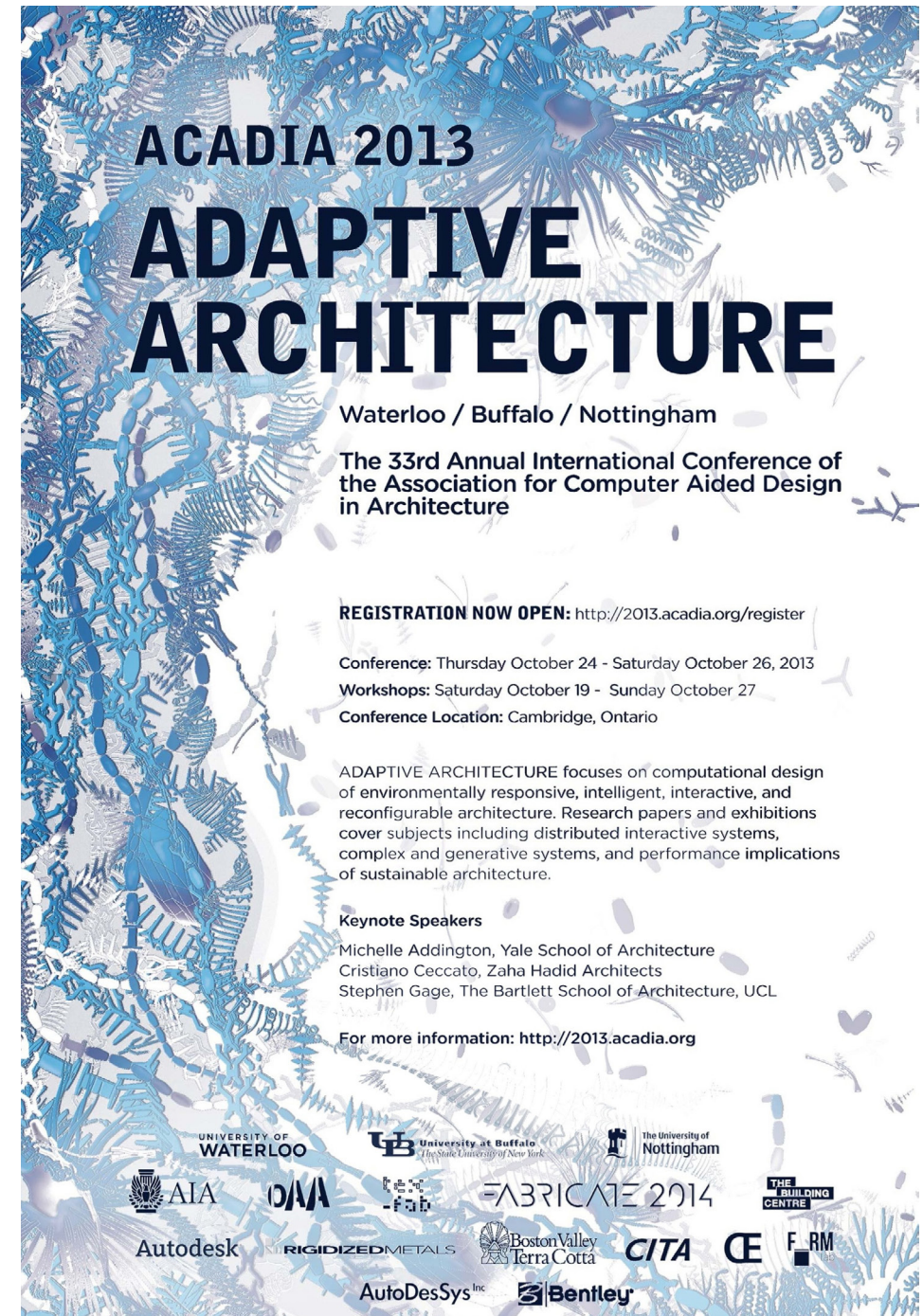


Figure 1.1.26 Acadia 2013 poster

Chapter 1.2 | Inadequacy of Current Visualization Methods

With these new forms of dynamic architecture comes increased complexity in both operation and design. As such, current visualization workflows must be updated to accommodate this development. Fortunately, technological progression benefits not only the construction of these new dynamic spaces but also the tools for designing and visualizing them. Richard Sennet states in *The Craftsman*: “We need to visualize what is difficult in order to address it. This is probably the greatest challenge facing any good craftsman: to see in the mind’s eye where the difficulties lie.”^[1] From this, one can infer that visualization can facilitate access to complexity, which in turn will allow the development of new technologies that can develop even better visualization tools. This concept, where one development feeding into another as a cyclical progression is nothing new, as visualization methods have progressed as such throughout history.

The first of these progressive leaps in regards to architectural visualization was perhaps the development of linear perspective in the 1400s by Italian architect Filippo Brunelleschi.^[2] This method was later documented within the treatise *Della Pittura (On Painting)* by Leon Battista Alberti^[3] that established the preservation and accessibility of this knowledge to later generations. John R. Spencer noted within his translation of Alberti’s *De pictura* that “By substituting the pyramid for a cone Alberti made the one-point perspective system possible, for in pyramidal vision the size of the object seen varies as the height of the observer’s eye and the distance to the object. Although he was physiologically incorrect, Alberti made it possible to represent objects on a plane surface with greater apparent exactitude.”^[4] (Fig. 1.2.1 - 2) Before this development, most art and visualization depictions consisted of mostly two-dimensional images with little attempt to portray depth or three-dimensionality, and where such attempts within medieval paintings, were exceedingly incorrect. (Fig. 1.2.3 - 4) With “this greater apparent exactitude” however, visualization evolved to better portray the dimensionality of the world and, in essence, the architecture within the world.

1 Richard Sennett, *The Craftsman* (London: Penguin, 2009), 230.

2 “Early Applications of Linear Perspective.” Khan Academy, accessed July 26, 2019, <https://www.khanacademy.org/humanities/renaissance-reformation/early-renaissance1/beginners-renaissance-florence/a/early-applications-of-linear-perspective>.

3 Khan Academy, “Early Applications of Linear Perspective.”

4 Leon Battista Alberti, *On Painting*, trans. with an Introduction and Notes by John R. Spencer (New Haven, 1966), 103.

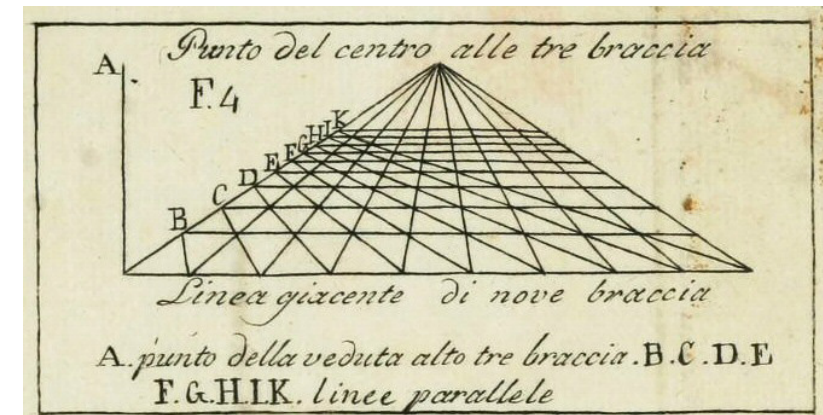


Figure 1.2.1 Vanishing point, depicted in *Della Pittura* by Alberti

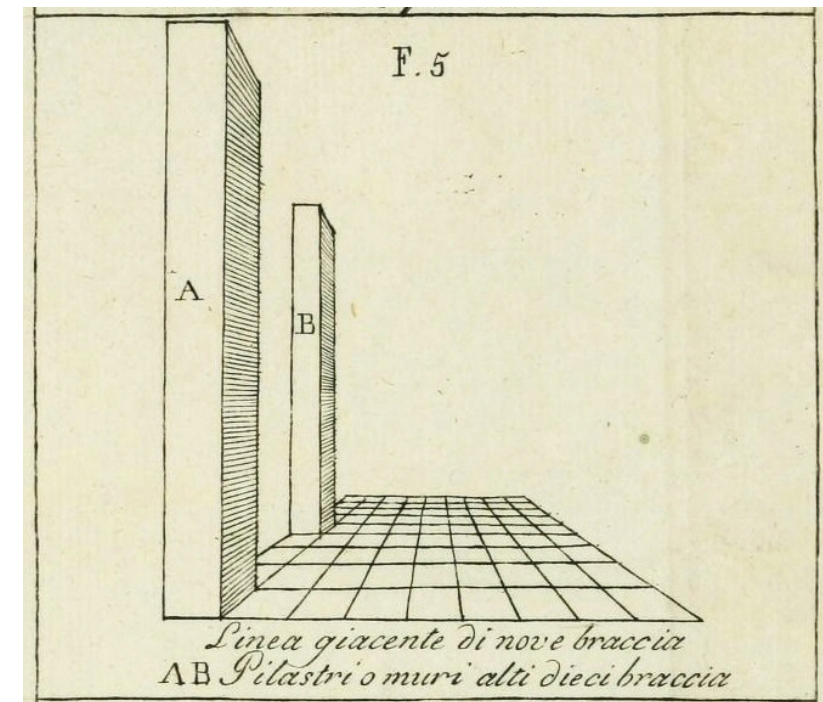


Figure 1.2.2 Perspective pillars on grid, depicted in *Della Pittura* by Alberti.



Figure 1.2.3 *The calling of the Apostles Peter and Andrew by Duccio, 1308-11*

Figure 1.2.3 - 1.2.4

These figures compare an medieval painting from before the development of linear perspective to an Renaissance painting from the time after. While they are produced by different artists, it is interesting to note the increased realism within the portrayed proportions.

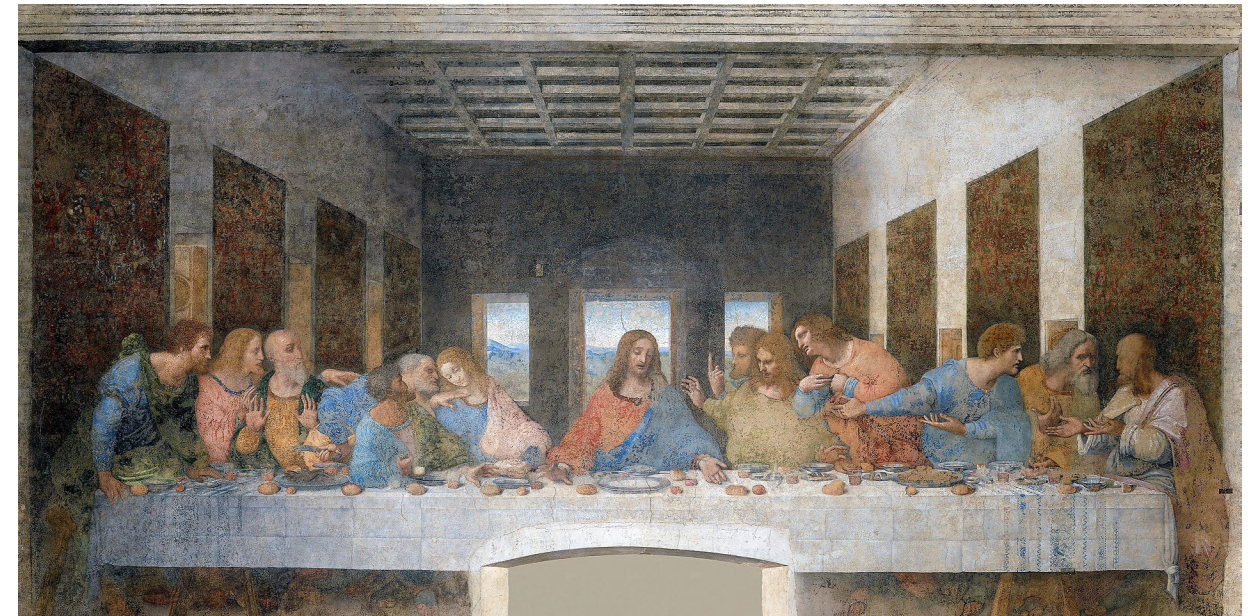


Figure 1.2.4 *The Last Supper by Leonardo Da Vinci, 1495-96*

This capability of portrayal was enhanced even further with the invention of photography in the 1800s.^[5] In a drawing, every line is deliberate, but a photograph captures the location with context, whether accidental or deliberate. This aspect of photography gained it its credibility as a tool for documentation, as it was a way to confirm and validate, but at the cost of visual flexibility.^[6] It wasn't until French architect Henri Labrouste and his deliberate tracing of photographs that allowed the removal of unwanted features and the highlighting of important details to regain that visual flexibility. (Fig. 1.2.5 - 6) Neil Levine describes Labrouste's tracings in *The Template of Photography in Nineteenth-century Architectural Representation*:

“Labrouste’s tracing of the photograph involved more than removing unwanted features. His redrawing highlighted important aspects of the building that were somewhat indistinct in the photograph. The lack of clarity of detail in parts of the photograph is ironic given the emphasis on the medium’s ‘precision’ and ‘exactitude’ in the photographic discourse. [...] Labrouste built on photography’s putative strengths to give the image an even greater degree of precision, exactitude, and mechanical definition than the photograph itself provided. [...] Finally, the removal of all trace of human occupation transformed the photographic scene into an abstracted, airless, uncanny representation of reality combining in almost equal measure the rational character of the building’s design with its pronounced structural expression.”^[7]

Within this passage, Levine notes the irony of Labrouste’s tracings: how by removing features—thus reducing the clarity of the photograph—he was able to enhance the clarity of the building design by highlighting the “important aspects of the building.” This of course benefited greatly in architectural visualizations as it allowed designers to visualize what is important in the design while keeping the building within context—essentially becoming an early form of architectural rendering.

Although each of these advancements facilitated progression, none of them have influenced the modern world as rapidly and profoundly as digital computation. Since its conception in the mid-1900s, many industries conformed by moving towards digital mediums, changing not only architectural visualization, but the rest of the world as well. Christoph Schindler gives a brief summary of these developments in his dissertation *Information-Tool-Technology: Contemporary digital fabrication as part of a continuous development of process technology as illustrated with the example of timber construction*:

Figure 1.2.5 - 1.2.6

These figures show how Labrouste removed all traces of human activity in preparation of engraving the photo. In doing so, he reinforced the clarity of the design.



Figure 1.2.5 Photograph of Bibliothèque Sainte-Geneviève by Bisson Frères



Figure 1.2.6 Perspective view of Bibliothèque Sainte-Geneviève, traced from Bisson Frères' photograph by Henri Labrouste, engraving by Jacques-Joseph Huguenet

5 "Invention of Photography," The British Library, accessed July 28, 2019, <https://www.bl.uk/learning/timeline/item106980.html>.

6 Neil Levine, "The Template of Photography in Nineteenth-Century Architectural Representation," *Journal of the Society of Architectural Historians* 71, no. 3 (January 2012): 308, <https://doi.org/10.1525/jsah.2012.71.3.306>.

7 Levine, "The Template of Photography in Nineteenth-Century Architectural Representation," 308.

“William Shockley developed the first efficient transistor in 1947 at the US Bell Laboratories; in 1958 Jack Kilby started to cast integral circuits into a germanium ‘microchip’; and in 1970 IBM produced the first silicon ‘microprocessor chip’. From this point onwards the agenda was set to produce computers small and low priced enough to be built into machines to automate complex formalized processes economically.”^[8]

By observing the delta in time between these developments, it can be seen how rapidly technology has progressed in the past hundred years relative to human history. Even before the observation of Moore’s Law, within 30 years, computation production went from the first efficient transistor to an agenda of producing smaller and cheaper computers. Comparing this to the few hundred years between the development of linear perspective and photography, it can be presumed just how fast visualization progression will not only continue but accelerate from this point onwards.

Schindler views this technological progress in the context of timber construction as waves of development “divided into three essential production technologies in the history of mankind: hand-tool-technology, machine-tool-technology, [and] information-tool-technology.”^[9] He states, “to this extent the three ‘waves’ of technology are not to be understood as competing, incompatible principles, but rather as the gradual substitution of formalized physical and later also formalized intellectual operations by machines. Man is not replaced, but revalued. His function shifts from processor to process designer.”^[10] This view highlights how the progression of these new technologies can gradually allow the outsourcing of tedious work away from low-efficiency humans to high-efficiency computers, thus not only improving efficiency in mundane tasks but also allowing humans to focus on more purposeful work—both of which can drive progression even faster than it is now.

The potential ramifications of these new technologies generated excitement, invoking avant-garde movements such as neo-futurism, where designers began thinking new ways of programming spaces, facilitating new architectural ideas that transcended norms. The London-based architecture group Archigram was one such example, where they published a series of magazines throughout the 1960s, featuring futuristic concept designs on what they imagined computation could bring to architecture. (Fig. 1.2.7 - 11) Within *Beyond Archigram*, Hadas Steiner notes an excerpt from *Design Quarterly* in an IDEA conference pamphlet in 1966 that “the strength of Archigram’s appeal stems from many things ... But chiefly it offers an image-starved world a new vision of the city of the future, a city of components on racks, components in stacks, components plugged into networks and grids, a city of components

8 Christoph Schindler, “Information-Tool-Technology: Contemporary digital fabrication as part of a continuous development of process technology as illustrated with the example of timber construction,” (PhD diss., 2007), 12, accessed June 26, 2019, http://www.caad.arch.ethz.ch/wiki/uploads/Organisation/2007_Schindler_Information-tool-technology.pdf.

9 Schindler, “Information-Tool-Technology: Contemporary digital fabrication as part of a continuous development of process technology as illustrated with the example of timber construction,” 2.

10 Schindler, “Information-Tool-Technology: Contemporary digital fabrication as part of a continuous development of process technology as illustrated with the example of timber construction,” 17.



Figure 1.2.7 Archigram Information Tear-off Sheet

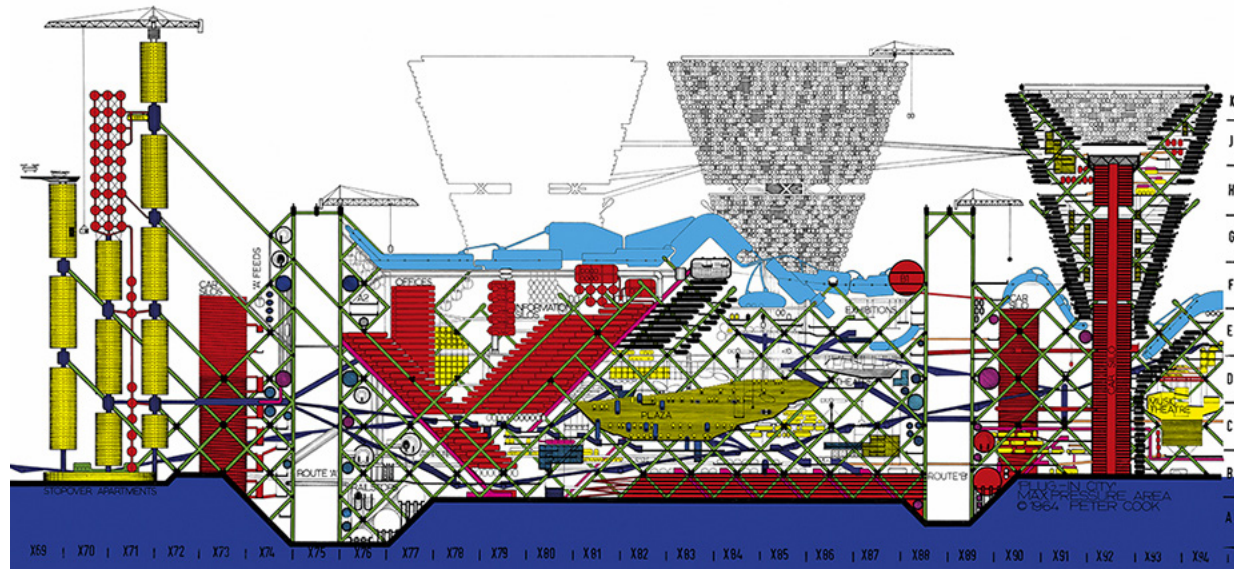


Figure 1.2.8 Plugin City concept by Peter Cook (Archigram), 1964



Figure 1.2.10 Instant City concept by Peter Cook (Archigram), 1969

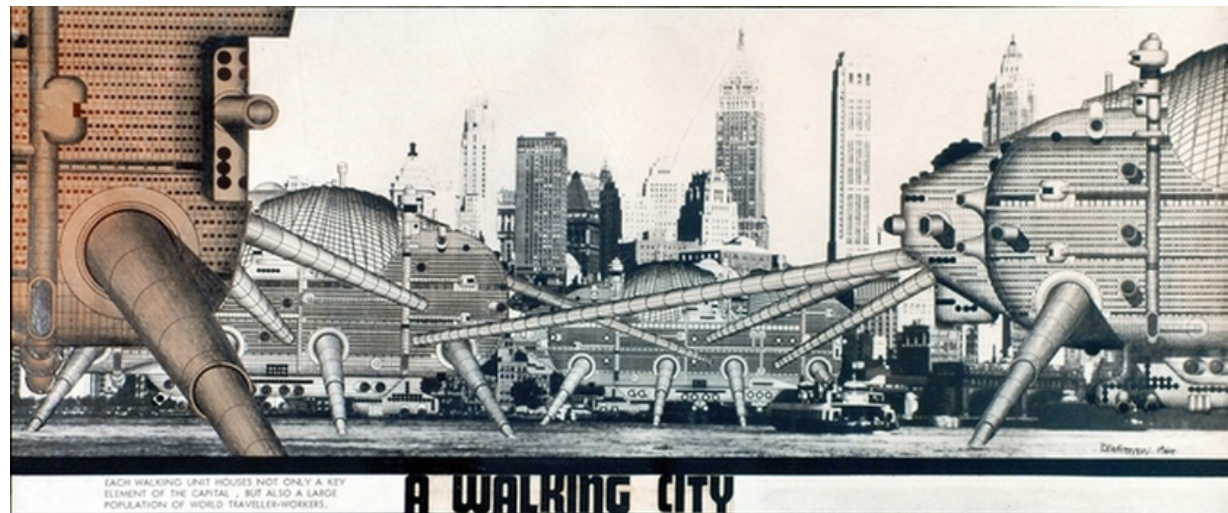


Figure 1.2.9 Walking City Concept by Ron Herron (Archigram), 1964

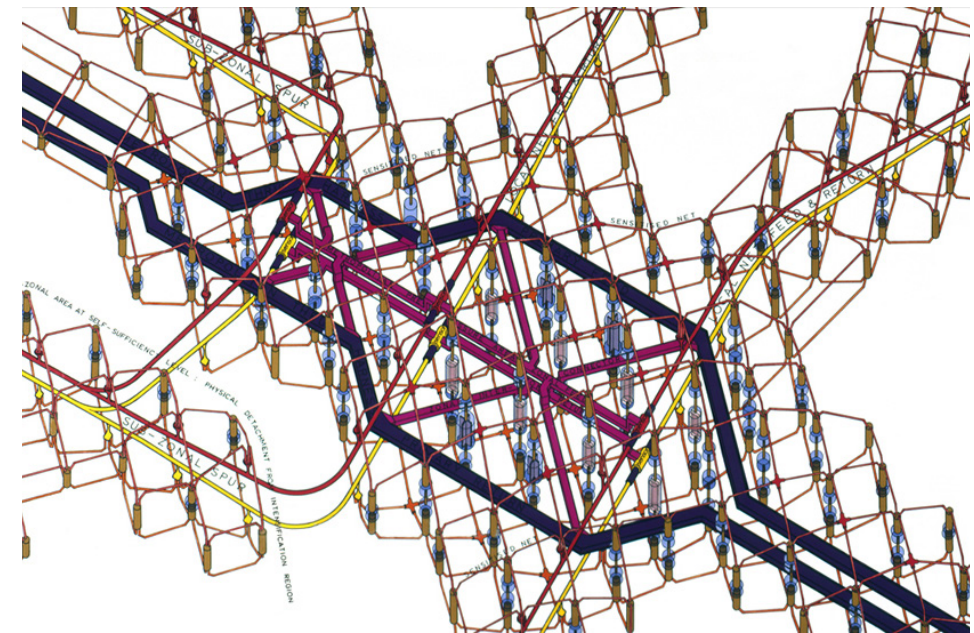


Figure 1.2.11 Computer City concept by Dennis Crompton (Archigram), 1964

being swung into place by cranes.”^[11] This excerpt highlights the reason for Archigram’s success, which stems from—amongst many cultural drivers such as new technologies and the world wars—satisfying the world’s aspirations for the future; how this new mindset of embracing change allows the mere idea of new technologies to drive new forms of creation, further reinforcing the observed influence of new technologies on the world.

Nevertheless, as technology continues to progress at an exponential rate, more of these seemingly quixotic designs become increasingly possible. The advent of Computer Aided Design (CAD) software facilitated more efficient workflows that increased the productivity, quality, and communication of ideas, allowing the production of better tools and materials. Michelle Addington and Daniel Schodek notes in *Smart Materials and New Technologies: For the Architecture and Design Professions* that “through advancements in CAD/CAM (Computer Aided Design/Computer Aided Manufacturing) technologies, engineering materials such as aluminum and titanium can now be efficiently and easily employed as building skins, allowing an unprecedented range of building facades and forms.”^[12] Sennet’s view is in line with this, stating that:

“Thanks to the revolution in micro computing, [...] modern machinery is not static; Though feedback loops machines can learn from their experiences. [...] Computer-assisted design has become nearly universal in architectural offices because it is swift and precise. [...] The modern material world could not exist without the marvels of CAD. It enables instant modeling of products from screws to automobiles, specifies precisely their engineering, and commands their actual production.”^[13]

This once again has enormous potential in not only building design but also building construction, which in itself also benefits building design. As such, architects now have faster ways to design and visualize, as well as a plethora of new materials to choose from. This multidisciplinary progression further illustrates the aforementioned claim of cyclical progression at the beginning of this chapter. CAD software evolved from 2D to 3D. (Fig. 1.2.12 - 13) The addition of this spatial dimension allowed architects to simulate buildings in virtual space, where one can program an environment with complete creative freedom with essentially no physical limitations or cost associations. These virtual spaces have the potential to not only simulate the building form but also its materials and lighting. With this, comes the emergence of modern photorealistic renderings. (Fig. 1.2.14)

Even with these new technologies, however, the majority of commercially produced present-day architectural visualizations are still static images. While modern visualizations have become increasingly photorealistic and more efficient to produce, their static nature still limits the amount of information they can communicate. The problem with this is that the real world is rarely

11 Hadas A. Steiner, *Beyond Archigram: The Structure of Circulation* (New York: Routledge, 2009), 202.

12 D. Michelle Addington and Daniel L. Schodek, *Smart Materials and New Technologies: For the Architecture and Design Professions* (London: Routledge, 2016), 3.

13 Sennett, *The Craftsman*, 38-39.



Figure 1.2.12 2D floorplans, created within Autodesk AutoCAD.

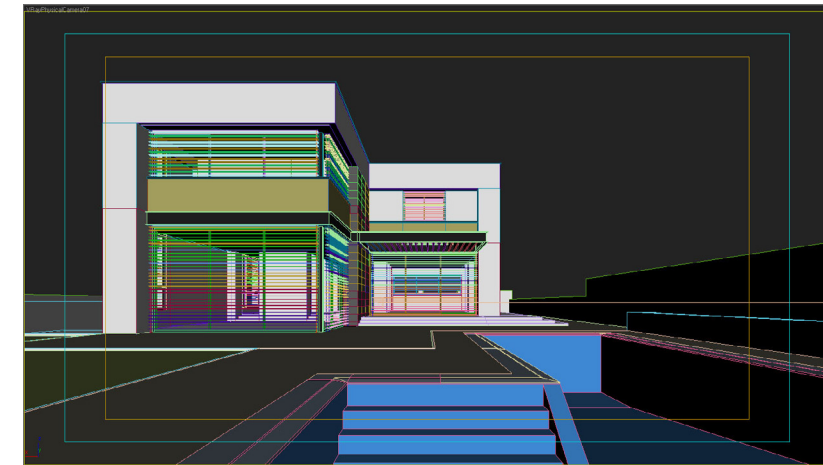


Figure 1.2.13 A building model rendered within 3D space on a viewport in Autodesk 3DS Max



Figure 1.2.14 The same building model rendered out with Vray

Figure 1.2.12 - 1.2.14

These figures show how new software has facilitated progression in visualization from 2d virtual spaces to 3d virtual spaces to photo-realistic rendered visualizations.

static. Rudolf Arnheim states in *The Dynamics of Architectural Form*, “A building [...] is an experience of the senses of sight and sound of touch and heat and cold and muscular behavior, as well as of the resultant thoughts and strivings.”^[14] This means that buildings invoke senses beyond just human sight, and as such, require more than a static frame to portray its full effect. People are inherently dynamic, thus, even in static spaces, once they become occupied, they also become dynamic. Singular images only show a snapshot of the design frozen in time, which makes it challenging to capture the dynamic impact of occupants. While this is acceptable for portraying unoccupied static spaces, it falls short at portraying anything more. This issue then becomes compounded with the introduction of dynamic architecture, as now there are two dynamic systems interacting with each other, bringing additional complexities that render current methods further inadequate at spatial representation.

Although motion can be *suggested* within a single frame through means such as motion blur, (Fig. 1.2.15 - 17) the more accurate way to represent such motion is to simply add more images as the scene changes. (Fig. 1.2.18 - 19) This concept, again, is nothing new, as there are entire industries focused on videography and cinematography. The issue, however, is the way this is utilized within architectural visualization. Since the late 1900s, there has been a plethora of films with visualizations that are borderline photorealistic, yet there have only been a select few architectural visualizations that could be said to rival this quality. By comparing a rendered architecture video against a film in the early 2000s, it is evident that the architectural rendering is severely lacking in terms of spatial and environmental representation. (Fig. 1.2.20 - 21)

Figure 1.2.15 - 1.2.17
These photographs convey the movement of people by utilizing motion blur within a single image.

14 Rudolf Arnheim, *The Dynamics of Architectural Form* (Berkeley: University of California Press, 2009), 4.

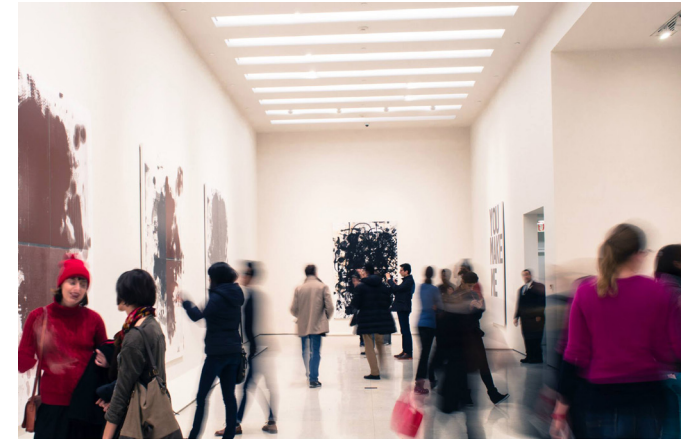


Figure 1.2.15 People moving through a gallery space within the Solomon R. Guggenheim Museum in NYC



Figure 1.2.16 Macy's Thanksgiving Day parade, NYC



Figure 1.2.17 People moving through a gallery space within the MOMA in NYC

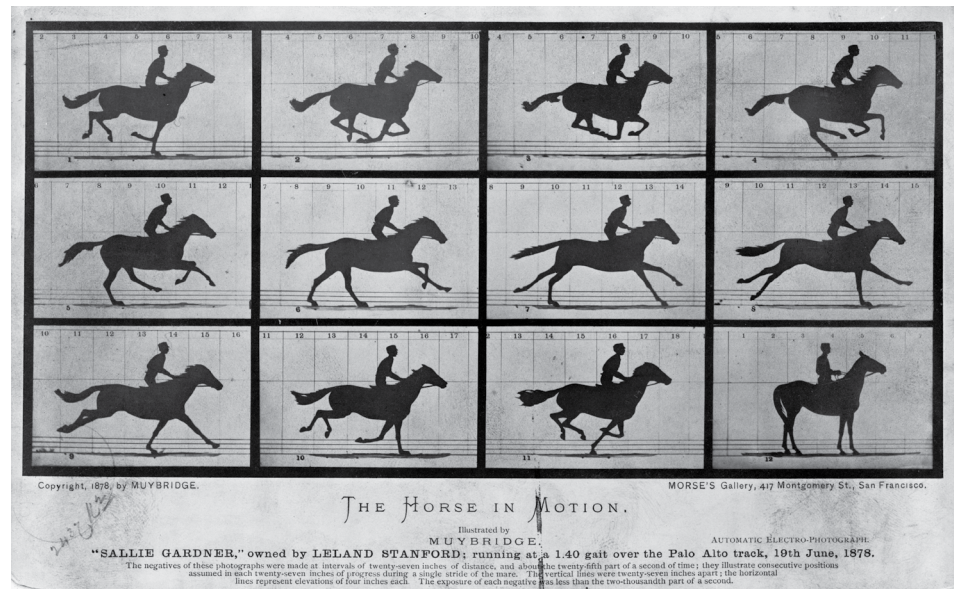


Figure 1.2.18 *The Horse in Motion* cabinet cards by Eadweard Muybridge, 1878



Figure 1.2.19 Animation made from Eadweard Muybridge's cards



Figure 1.2.20 *Lord of the Rings Return of the King*, 2003



Figure 1.2.21 *Royal Ontario Museum architectural walk-through*, 2003

Figure 1.2.20 - 1.2.21

These 2 figures shows the difference in quality between a film from 2003 and an architectural walk through video from the same year. It can be seen that the quality is vastly different between the 2 in not only the rendering aspect, but also the crowd dynamics.

Since then, architectural visualization has somewhat caught up in terms of rendering capabilities, the most notable of which being Alex Roman's short film *the third and the seventh*.^[15] (Fig. 1.2.22) Although Roman's film exhibits excellent visuals that are comparable to the film industry, it can also be argued that this film is more of a passion project, thus it does not have the same constraints and limitations compared to an average commercial project. Comparing this then, to various architectural commercial animations, the relative degradation of quality appears once again. (Fig. 1.2.23 - 24) Further comparing these visualizations to a rasterized viewport screen capture—which is what is generally used as visualization tools during the design phase—the degradation of quality becomes even more apparent. (Fig. 1.2.25)

While this might not seem like a fair comparison due to the differing priorities and budgets within the respective industries, clients—who for the most part does not understand these technology and industry specific limitations—are so accustomed to seeing the higher quality images that they somewhat expect architectural images to be of the same quality. As such, these comparisons can provide some insight to identify a few of the problems present in current visualization methods. It can then be speculated that this is the difference between what is used as a final product, as a pitch, and as a design tool. Analyzing these examples further, one can see that a commonality between them is the lack of occupancy dynamics in some form or another. This causes the potential utility of these extra frames to be wasted as they are only utilized to show the spatial qualities of the space instead of the ambiance and dynamic interactions within the space.

¹⁵ Alex Roman, "The Third & The Seventh," uploaded November 24, 2009, Vimeo, 12:29, accessed July 26, 2019, <https://vimeo.com/7809605>.



► **Figure 1.2.22** *The 3rd and the Seventh* by Alex Roman, 2009
This Short Film is comparable to the film industry, however, the occupancy dynamics portrayed is fairly simple and this does not have the same time constraints as most commercial projects.

Figure 1.2.22 - 1.2.24

These 3 figures shows some of the shortcomings in current architectural visualization renderings. While their quality have gotten better significantly over the years, they are still playing catchup to the film industry from 15 years ago.



► **Figure 1.2.23** *Architecture Walk-through by Framemakers Creative SB, 2015*
While the visualization here looks decently photorealistic, the lack of people makes the space feel empty and desolate.



► **Figure 1.2.24** *Architecture Walk-through by Momo Graphics, 2016*
This walk-through does include people to convey a more believable space, however, it is easy to notice how these people are manually placed in, instead of actually utilizing the environment. This not only produces weaker visuals, but also takes quite a bit of time to do.

The main culprit of this large discrepancy in rendering quality between the film and architecture industries is likely a combination of budget and time constraints. While videography consisted of simply pointing a video camera at the scene, animation requires creating and rendering the scene from scratch. This requires vastly more time expenditure compared to single images, as one would need to render a minimum of twenty-four images to provide a single second of video. (**Fig. 1.2.26**) To put this into perspective, with current architectural rendering methods, if a single frame takes 10 minutes to render, then a 1-minute video would take 10 days, not counting the post processing that goes with it. Because of this time expense, it is often unrealistic to utilize this medium within architectural design, where deadlines are consistently present. In the few projects that do utilize architectural videos for client pitches, architects often do not have the time or budget to allocate the resources required to make these visualizations at the same visual quality as films, let alone animating crowd dynamics on top of this. Film studios are tasked with delivering the resulting video; therefore, it makes sense that they will allocate the majority of their budget to perfecting the final video file, and thus have the capacity and flexibility to absorb this large time expenditure in animation, or avoid animation entirely by utilizing extras and practical effects.

In contrast, architecture firms are tasked with developing a design, where their visualization mediums are merely methods used to communicate said design; therefore, it should make sense that architectural visualizations are of lower priority than the actual design. What usually happens during the design phase is that the perspective visualizations are often ignored until the design needs to be communicated externally to another person or client. This is done once again not because perspectives are unnecessary during the design phase, but because of the additional time required to produce them compared to orthographic drawings. While the design phase is arguably the most important phase—since it will influence all the other phases after it, thus having the largest impact on the resulting building—most architects are forced to spend less time on it due to budget constraints. By the time adequate visuals are required to portray the space, they are often rushed due to time limitations. On top of this, architecture projects can operate at many scales ranging from exterior site planning to interior designs, and anything in-between. This further forces architects to prioritize their renderings on the larger scales first to convey the overall design intent, while allocating less time to the interior visualizations even though the interiors might have a higher impact on occupancy.

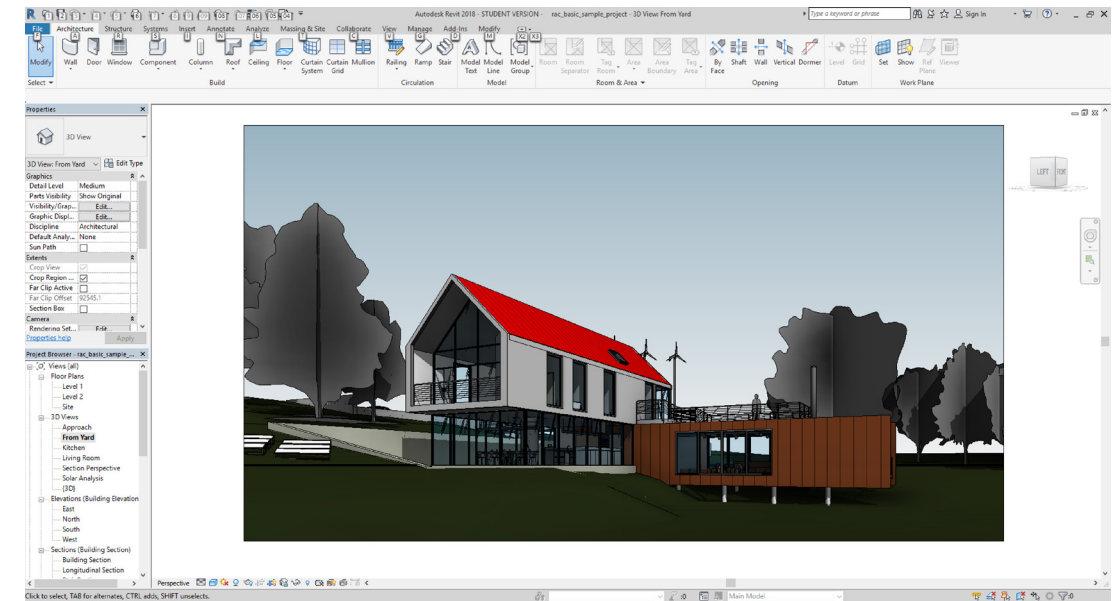


Figure 1.2.25 Revit Perspective Viewport

During the design phase, most architects would simply utilize the shaded view within their drafting software to get a sense of the current building. While this is quick and effective in getting feedback for the design, it is still lacking in dynamics, materiality, and atmosphere.

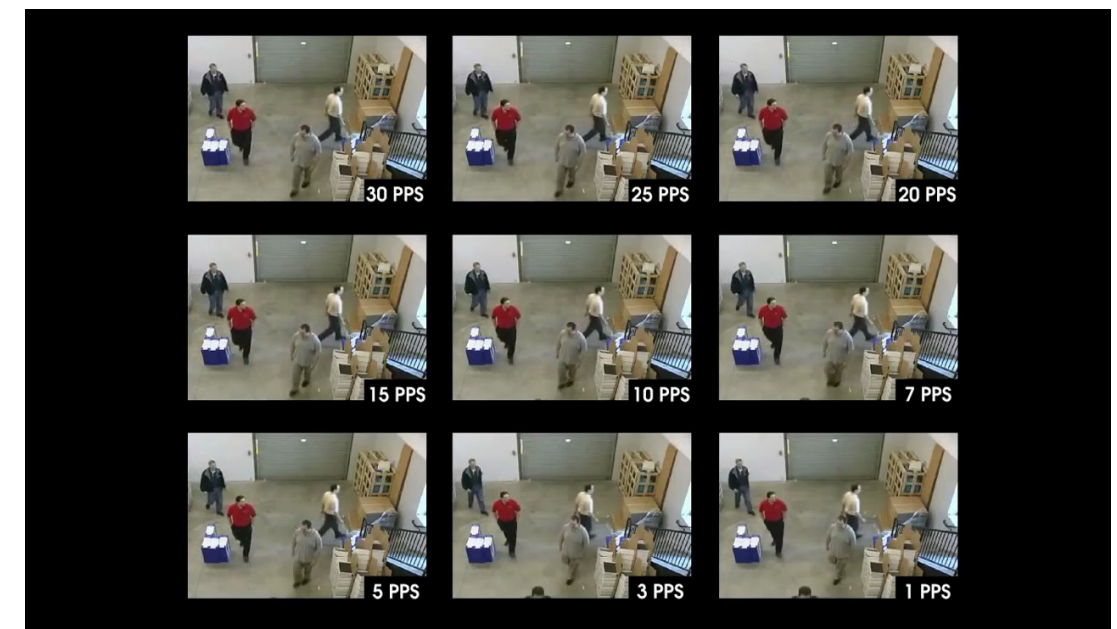


Figure 1.2.26 Security footage showing various frame rates

This is made worse by the fact that the bulk of the architect's fees actually come from the production of construction documents, where most of the detailed design is to be conveyed through orthographic drawings. (Fig. 1.2.27 - 28) Comparing this to perspective drawings—which not only take longer to produce but are also not required within a construction documentation package—it makes sense that the visualization budget within architecture firms is much smaller than that of film studios.

Because of these factors, architects are usually so stripped of time that they must prioritize on visualizing the architecture rather than how the architecture will be used, having no choice but to *throw* people into the final rendering as an afterthought, or even leaving them out completely, resulting in a barren, lifeless space. What this means is not only are architects rushing to introduce them at the end of the project, thus compromising the quality of their pitch, but they are also not visualizing them as they design, thus compromising the design's potential.

While this lack of occupancy visualization is not the end of the project, it does not change the fact that people will occupy these exterior and interior spaces in the physical world. As such, ignoring the ability to portray crowd dynamics at these varying scales would be ignoring a large aspect of both interior crowd interactions as well as exterior crowd flows from the surrounding context. These concerns become even more substantial with dynamic spaces due to the increasing interactions between people and architecture. It is disheartening that architects design buildings to be occupied by people yet don't have the time to consider them within their visualization tools. If architects can barely afford the time to even produce static perspectives, then it is by no means a surprise that many firms are choosing to not utilize dynamic perspective videos within their design workflows and client pitches.

Architects are hired to provide good designs to clients, meaning clients hire architects because they trust them to provide high quality design. However, now as the film and gaming industries expand, client expectations for visualizations may also become higher due to their increased exposure to everyday media. Good renderings have become ordinary and commonplace, as such, to keep up with these expectations, architecture must also adopt better visualization methods.

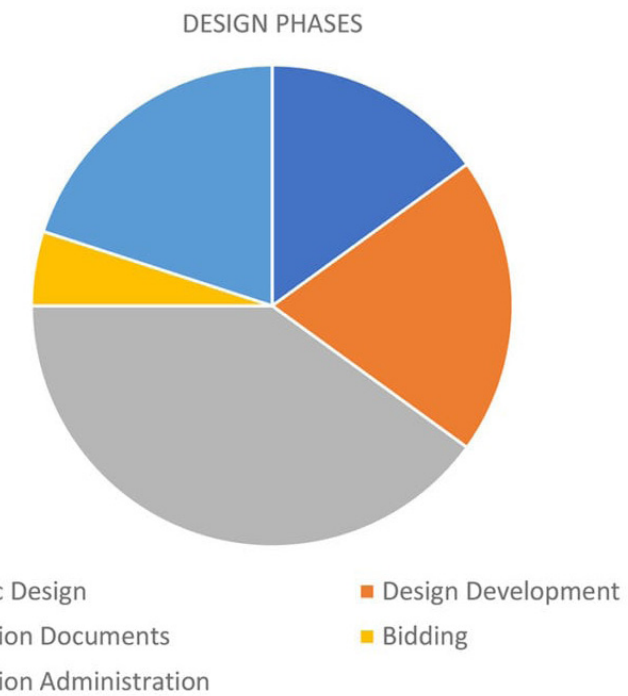


Figure 1.2.27 Typical breakdown of architectural fees

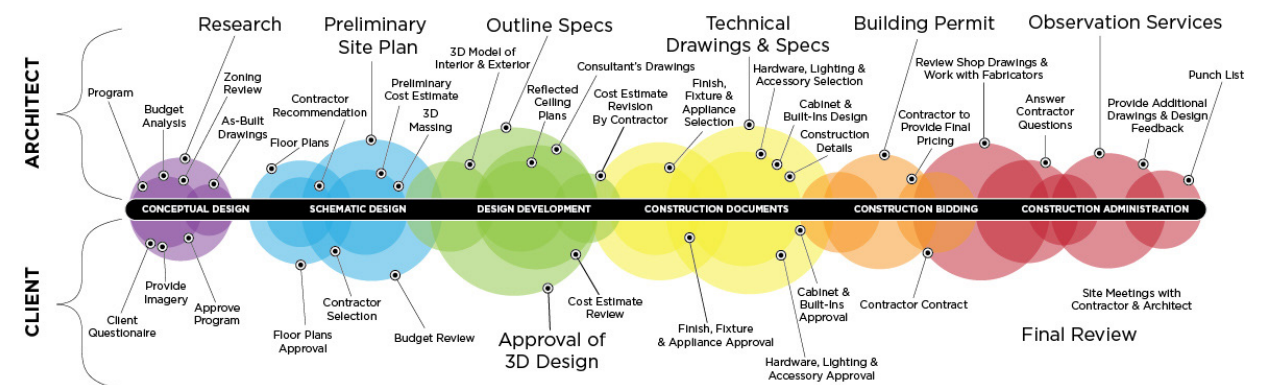


Figure 1.2.28 Typical time-line of architectural design phases

Chapter 1.3 | Advent and Progression of the Gaming Engine

The progression of technology has also facilitated a plethora of new creative industries, with game design being one of them. Although architecture has been around since the beginning of civilization, digital game design is a relatively new field that came about with the onset of the digital revolution. Historically, these were very separate fields, with architecture focusing on building design and physical drafting and game designers initially focusing on developing interactive two-dimensional scenario representations. But as technological advancements progressed, both fields have found themselves increasingly dependent on digital spatial environments. Architects moved from pencil and paper to three-dimensional CAD environments while virtual games shifted from two-dimensional representations such as “Pong, Space Invaders, PacMan, [and] Donkey Kong” to three-dimensional representations such as “Wolfenstein 3D and Catacomb Abyss.”^[1] (Fig. 1.3.1 - 4) Now as software and hardware continues to improve, both industries are approaching the territory of photorealistic visualizations. (Fig. 1.3.5 - 6) However, while this is the case, “this rapid development in computer game technology is almost unnoticed by the users of professional CAD-, GIS-, and illustration software.”^[2] Architects, who produce designs for real-world applications, and game developers, who produce immersive digital experiences for people, both benefit from quality simulations in this modern age, and yet, architecture is lagging behind, as investigated in the past chapter (1.2). However, the potential to catch up is there. With game developers essentially tasked with *simulating spaces* and architects tasked with *designing spaces*; along with gaming graphics becoming increasingly realistic, and architecture depending more on computation; the fields are beginning to overlap, and as such, the integration and unification of skill sets, workflows, and tools within these respective industries is becoming increasingly beneficiary.

The most notable of these tools with regards to architectural visualization is undoubtedly the *game engine*, which can be described as a “collection of modules of simulation code that do not directly specify the game’s behavior (game logic) or [the] game’s environment (level data).”^[3] As such, one can think of these engines as a form of Integrated Development Environment (IDE) that is specialized in game creation, or more generally, “an assemblage of reusable software functionalities.”^[4]

“sensory and responsive technologies expose new and surprising ways to make connections across disparate fields”^[1]

“Video games do not constitute finished text presented to an audience, but a system or a world with which players interact. The creation of video games can be described as the building of a universe of possibilities relying more on systems and cybernetic principles than on aesthetic rules attached to the production of an object. Video game creation can be described as a process of metacreation where a certain number of possibilities are crafted and played upon by game developers and players.”^[2]

1 Michael Fox, *Interactive Architecture: Adaptive World* (New York: Princeton Architectural Press, 2016), 7.

2 Damien Charrieras and Nevena Ivanova, “Emergence in Video Game Production: Video Game Engines as Technical Individuals,” *Social Science Information* 55, no. 3 (September 2016): 338, <https://doi.org/10.1177/0539018416642056>.

1 Andrian Herwig and Philip Paar, “Game Engines: Tools for Landscape Visualization and Planning?,” (November 2014): 5, accessed October 16, 2019, https://www.researchgate.net/publication/268212905_Game_Engines_Tools_for_Landscape_Visualization_and_Planning.

2 Herwig and Paar, “Game Engines: Tools for Landscape Visualization and Planning?,” 1.

3 Michael Lewis and Jeffrey Jacobson, “Game Engines in Scientific Research,” *Communications of The ACM* 45, no. 1 (January 2002): 28, accessed October 16, 2019, <https://www.cse.unr.edu/~sushil/class/gas/papers/GameAlp27-lewis.pdf>.

4 Damien Charrieras and Nevena Ivanova, “Emergence in Video Game Production: Video Game Engines as Technical Individuals,” *Social Science Information* 55, no. 3 (September 2016): 341, <https://doi.org/10.1177/0539018416642056>.

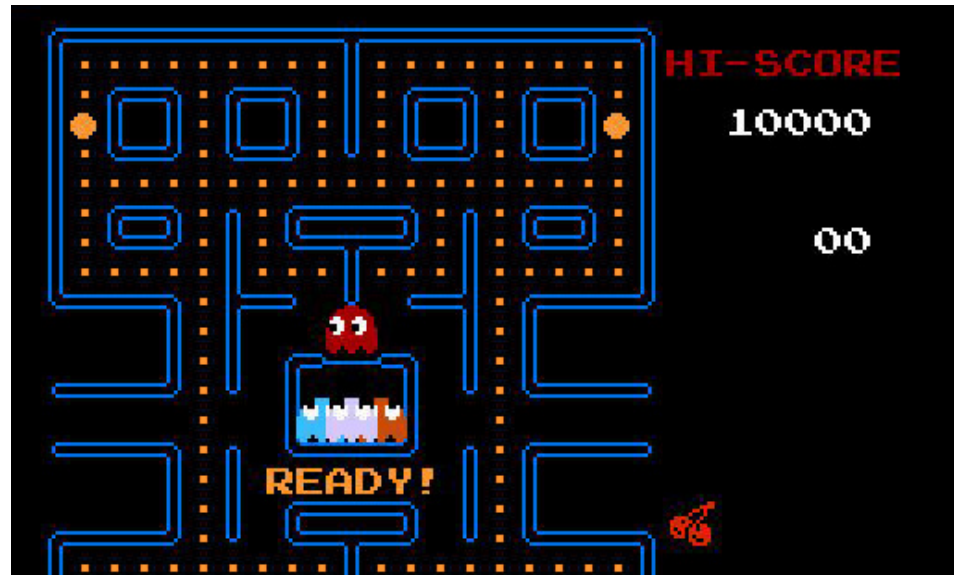


Figure 1.3.1 Pac-Man, Namco, 1980



Figure 1.3.2 Catacomb Abyss, Softdisk, 1992

Figure 1.3.1 - 1.3.2

These 2 figures show how early games transitioned from 2D representations to 3D, better portraying the spatial dimensions of real life.

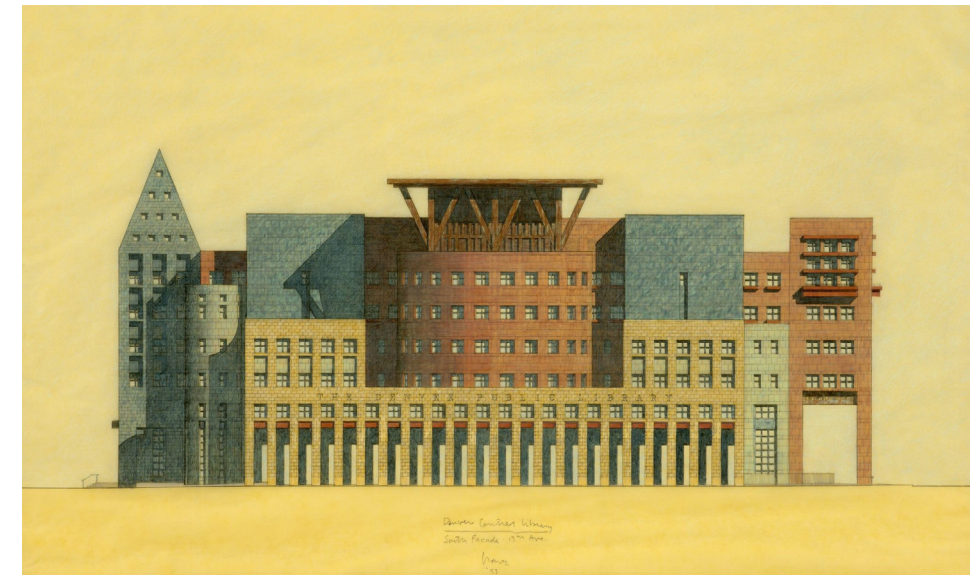


Figure 1.3.3 Hand Drafted South Elevation of Denver Library by Michael Graves, 1994

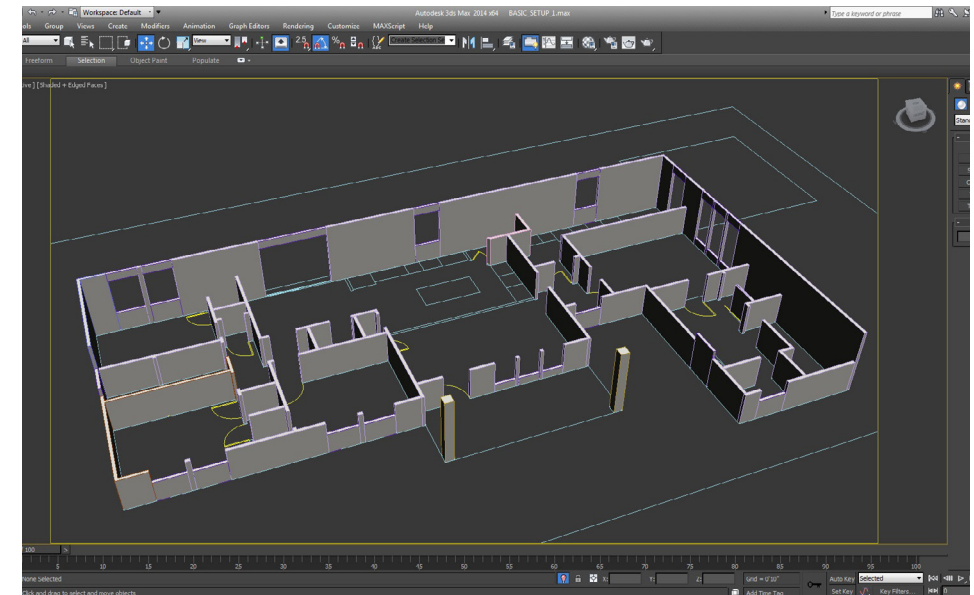


Figure 1.3.4 Creating 3D building walls from a 2D Building plan in virtual space

Figure 1.3.3 - 1.3.4

These 2 figures show how Architectural visualizations transitioned from hand drafting to 3D virtual modeling.



Figure 1.3.5 *The Division*, Massive Entertainment, 2016



Figure 1.3.6 *Hudson Yards Rendering* by KPF, 2019

Figure 1.3.5 - 1.3.6

These 2 figures show how both the game design and architecture industries are moving towards realistic renderings.

The origin of these game engines can be described as a product of necessity—to handle the increased complexity of modern games—as well as cost—to provide increased efficiency and cost savings compared to building a game from scratch.^[5] It achieves this by prioritizing “concepts such as reusability (a given GE is not tied to one game but can be used for different games), modularity (how an object can be accessed and modified in the GE) and extensibility (the possibility of adding extra functionalities to a given GE).”^[6] As such, the computational infrastructure of the game engine itself is comprised of many sub-systems that are brought together within one application software, each responsible for various functionalities such as Audio, Input, Physics, Rendering, Artificial Intelligence, Core, Scripting, and Networking.^[7] (Fig. 1.3.7) This allows game engines to utilize various file types in the form of assets within the same software, ranging from “texture bitmaps, 3D mesh data, animations, audio clips, collision and physics data, game world layouts, and [more].”^[8] Because of this diverse file type utilization, as well as their modularity and reusability, they can be used in multiple disciplines, “[operating] at the junction of creative and engineering practices.”^[9] Michael Lewis and Jeffrey Jacobson outline this nicely in their article “Game Engines in Scientific Research”:

“The cost of developing ever more realistic simulations has grown so huge that even game developers can no longer rely on recouping their entire investment from a single game. This has led to the emergence of game engines—modular simulation code—written for a specific game but general enough to be used for a family of similar games. This separability of function from content is what now allows game code to be repurposed for scientific research.”^[10]

While this “separability of function” allows game engines to be utilized for scientific research—and much more—this aspect is especially potent within architectural design. This is evident when considering a video game production pipeline:

“Video game production is a complex process involving different technical and artistic expertise as well as a diverse range of technologies. Several tools can be used at the stage of prototyping (Manker, 2012). 2D software like Photoshop is used by 2D artists to create textures (from photographs sometimes), 3D software is used to produce 3D models to be put into the game environment at a later stage. 3D animation software (3DS Max, Maya, Blender) involves keyframing

5 Charrieras and Ivanova, “Emergence in Video Game Production: Video Game Engines as Technical Individuals,” 340-341.

6 Charrieras and Ivanova, “Emergence in Video Game Production: Video Game Engines as Technical Individuals,” 344.

7 Björn Nilson and Martin Söderberg, “Game Engine Architecture,” (May 26, 2007): 3-6, accessed October 16, 2019, <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.459.9537&rep=rep1&type=pdf>.

8 Jason Gregory, *Game Engine Architecture* (Boca Raton; London; New York: CRC Press, 2019), 481.

9 Charrieras and Ivanova, “Emergence in Video Game Production: Video Game Engines as Technical Individuals,” 339.

10 Lewis and Jacobson, “Game Engines in Scientific Research,” 28.

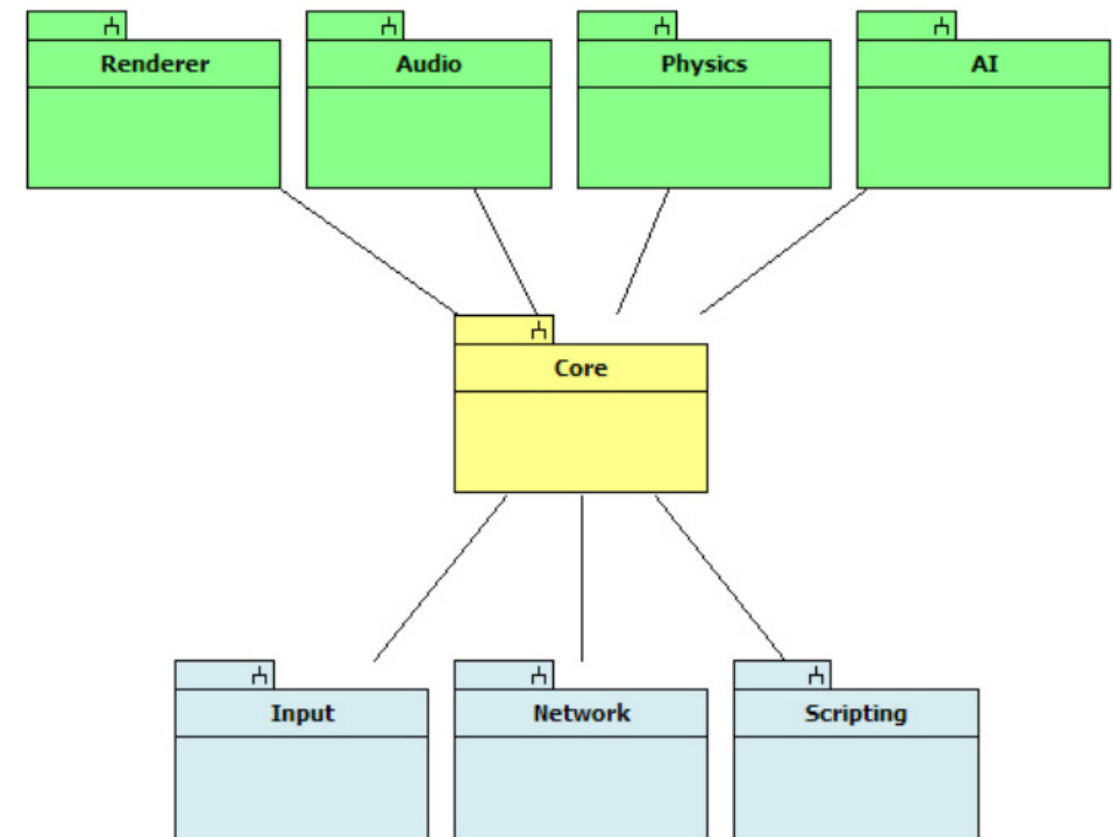


Figure 1.3.7 “An abstract model of how an engine might be put together”

(an animation technique based on smooth transition of movements) or motion capture data. Surfacing tools like Mudbox (Autodesk) enable character artists to sculpt very fine details into the 3D models.”^[11]

From this, the amount of software that is utilized within the game design industry becomes apparent. What is more noteworthy, however, is how much of this software is already utilized within the architectural visualization industry. It is clear that transferable skills are present; therefore, the utilization of game engines in architecture becomes even more plausible. Software such as Photoshop, 3DS Max, and Maya are already used extensively in current architectural visualization pipelines as modeling and rendering tools, thus implementing the game engine would then allow it to “govern the relations between these [modeled] objects to build the game space,”^[12] allowing the creation of increasingly dynamic simulations and visualizations.

Of course, this becomes even more compelling in the consideration of occupancy dynamics, which has been a recurring theme from the visualization studies of the previous chapter (1.2). This factor is particularly relevant within the game design industry since most games rely on the player interacting with NPCs (non-player characters) in one form or another, which, in most cases, are simulated people or forms of people. Although early games such as Grand Theft Auto 3 (GTA 3) lacked adequate crowd dynamics, much like current architectural visualizations, (Fig. 1.3.8) newer games such as The Witcher 3 are beginning to alleviate this by utilizing better artificial intelligence (AI) tools and computational hardware to add more depth and complexity to simulated beings. (Fig. 1.3.9) While the quality of these crowds depends on the type of game, the budget of the project, and the studio that makes them, it still offers a valid confirmation as to what it is possible to create with game engines.

Within the book *Game Engine Architecture*, Jason Gregory describes games as “what computer scientists would call *soft real-time interactive agent-based computer simulations*.”^[13] He then notes on the game engine’s ability to approximate and simplify reality, specifying the various aspects of this description:

“In most video games, some subset of the real world—or an imaginary world—is *modeled* mathematically so that it can be manipulated by a computer. The model is an approximation to and a simplification of reality (even if it’s an *imaginary* reality), because it is clearly impractical to include every detail down to the level of atoms or quarks. Hence, the mathematical model is a *simulation* of the real or imagined game world. [...] An *agent-based* simulation is one in which a number of distinct entities know as ‘agents’ interact. This fits the description of most three-dimensional computer games very well, [...] Given the agent-based nature of most games, it should come as no surprise that most games nowadays are implemented in an object-oriented, or least

11 Charrieras and Ivanova, “Emergence in Video Game Production: Video Game Engines as Technical Individuals,” 340.

12 Charrieras and Ivanova, “Emergence in Video Game Production: Video Game Engines as Technical Individuals,” 340.

13 Gregory, *Game Engine Architecture*, 9.



Figure 1.3.8 *Grand Theft Auto 3*, DMA Design, 2001



Figure 1.3.9 *The Witcher 3*, CD Projekt, 2015

loosely object-based, programming language. [...] All interactive video games are *temporal simulations*, meaning that the virtual game world model is *dynamic* [...] most video games present their stories and respond to player input in real time, making them *interactive real time simulations*. [...] A “soft” real-time system is one in which missed deadlines are not catastrophic. Hence, all video games are *soft real-time systems*.^[14]

It is evident from these notes how well this description of video games falls in line with the required components for simulating occupancy dynamics within a virtual space, as well as simulating the space itself as a dynamic architectural visualization. This *approximation and simplification of reality* allows the game engine to simulate architectural spaces from the physical world. The *object-oriented agent-based simulation model* then allows the creation and visualization of crowd dynamics within this simulated architectural space. The *dynamic interactive real-time aspect* of this makes it practical and efficient for simulating and visualizing the interactions between the crowd dynamics and the static or dynamic architectural elements of these emerging dynamic spaces. The “soft” description of this, then means that the creation of this simulation model does not need to be too *strict* in order to function, which increases the realistic potential for an architecture student (without a background in software development) to utilize and create such a simulation and succeed. From these reasons, it is then possible to list the following benefits of utilizing game engines within architectural visualization workflows:

Higher abstraction tools for virtual simulations

Perhaps the most valuable aspect of game engines for visualizing dynamic spaces is their ability to utilize scripting languages and tools alongside various file types within the same software environment.^[15] This allows the designer to establish interactions between entities which allows the creation of various simulation systems within this software—essentially becomes a virtual playground for simulating the physical world. With this, it is possible to not only script autonomy to simulate human crowds and dynamic architectural elements but also allow the relatively easy integration of such autonomy with existing architectural visualization models and frameworks. This allows the simulation of architectural spaces as they are used in the real world with little regard to how complex they may become.

Real-time rendering

Beyond these tools, game engines also offer vastly more efficient rendering methods compared to traditional CPU based ray-traced methods from software such as V-Ray and Mental-Ray.^[16] While these older methods can produce extraordinary results, they can take hours or even days to render a single frame, which can be a time-consuming endeavor within the design process. (Fig. 1.3.10)

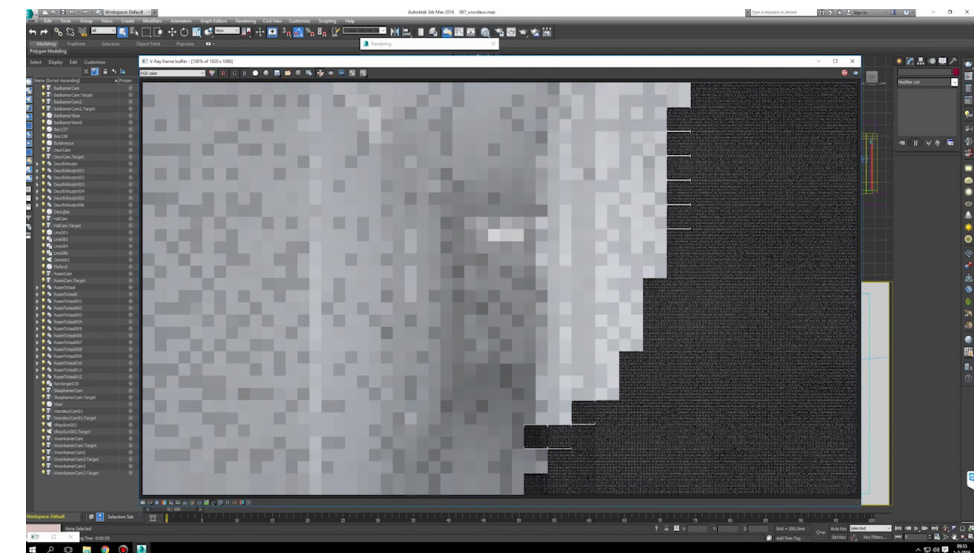
¹⁴ Gregory, *Game Engine Architecture*, 9–10.

¹⁵ Gregory, *Game Engine Architecture*, 481.

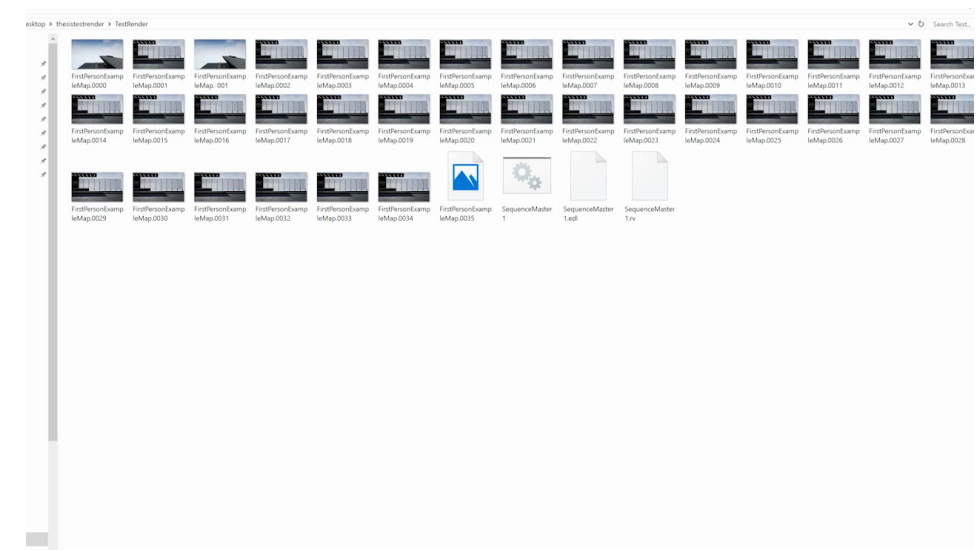
¹⁶ Brian Caulfield, “What’s the Difference Between Ray Tracing, Rasterization?,” The Official NVIDIA Blog, April 11, 2019, accessed October 16, 2019, <https://blogs.nvidia.com/blog/2018/03/19/whats-difference-between-ray-tracing-rasterization/>.

Games on the other hand must run in real-time due to their reliance on interactivity, which (as already mentioned in Chapter 1.2) to the human eye is at least 24 frames per second to convey the “illusion of motion,”^[17] and even more so to not feel delayed when the visualization is also required to respond to human input. This is many times faster than what can be achieved with traditional rendering methods, and as such, the rendering engines that are built into these game engines must prioritize speed to meet this demand. (Fig. 1.3.11)

¹⁷ Gregory, *Game Engine Architecture*, 10.



▶ Figure 1.3.10 Rendering a frame from Vray



▶ Figure 1.3.11 Rendering Frames from Unreal Engine 4

This rendering speed is largely achieved by utilizing rasterization—which is a rendering method that *approximates* lighting by converting the vertices of the virtual mesh into pixels^[18]—instead of ray tracing—which is a rendering method that *calculates* lighting where the paths of simulated “light rays” bouncing throughout the environment is traced back to the source of the camera.^[19] (Fig. 1.3.12 - 13) These virtual games also take advantage of the parallelism of GPUs along with various techniques—such as precomputing lighting and texture baking details onto low poly models—to maximize computational efficiency.^[20] (Fig. 1.3.14 - 15) These steps greatly reduce unnecessary calculations, which allows game engines to render in real-time. The downside to this workaround however is the reduced detail within the scene such that it only approximates of the scene. As such, the resulting renderings still lack a level of realism compared to traditional ray tracing methods that calculate the scene “correctly.”

However, with advancements in both hardware and software, along with pipelines that are utilizing *Physically Based Rendering* (PBR)^[21] and *linear space lighting*,^[22] *Photogrammetry*,^[23] and even *GPU-based real time ray tracing*,^[24] it is possible to obtain results approaching that of traditional ray tracing. As technology continues to progress, the differences between these two rendering methods are lessening, leaving only the benefits of these game engines without the shortcomings. As such, this aspect of gaming engines will become increasingly useful as their rendering quality begins to catch up with traditional rendering engines—even more so for architectural visualization as they do not need to be as strict on performance requirements as games do, and thus will have more flexibility in pushing the boundaries of utilizing the game engine as a simulation tool.

18 Caulfield, “What’s the Difference Between Ray Tracing, Rasterization?”

19 Arthur Appel, “Some Techniques for Shading Machine Renderings of Solids,” *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference on - AFIPS 68 (Spring)*, 1968, <https://doi.org/10.1145/1468075.1468082>.

20 “Optimizing Graphics Performance,” Unity, accessed October 17, 2019, <https://docs.unity3d.com/Manual/OptimizingGraphicsPerformance.html>.

21 “Physically Based Materials,” Unreal Engine Documentation, accessed October 17, 2019, <https://docs.unrealengine.com/en-US/Engine/Rendering/Materials/PhysicallyBased/index.html>.

22 “The PBR Guide - Part 1,” Substance Academy, accessed October 17, 2019, <https://academy.substance3d.com/courses/the-pbr-guide-part-1>.

23 Sébastien Lachambre, Sébastien Lagarde, and Cyril Jover, *Photogrammetry Workflow*, 2017, accessed October 17, 2019, https://unity3d.com/files/solutions/photogrammetry/Unity-Photogrammetry-Workflow_2017-07_v2.pdf.

24 David Cardinal, “How Nvidia’s RTX Real-Time Ray Tracing Works,” ExtremeTech, August 21, 2018, accessed October 17, 2019, <https://www.extremetech.com/extreme/266600-nvidias-rtx-promises-real-time-ray-tracing>.

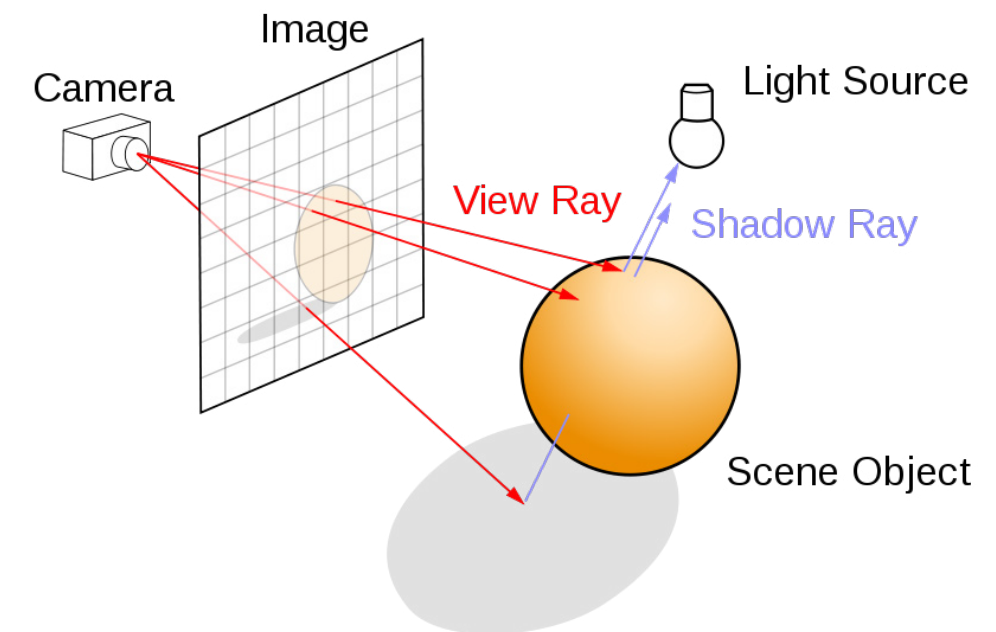


Figure 1.3.12 Ray tracing

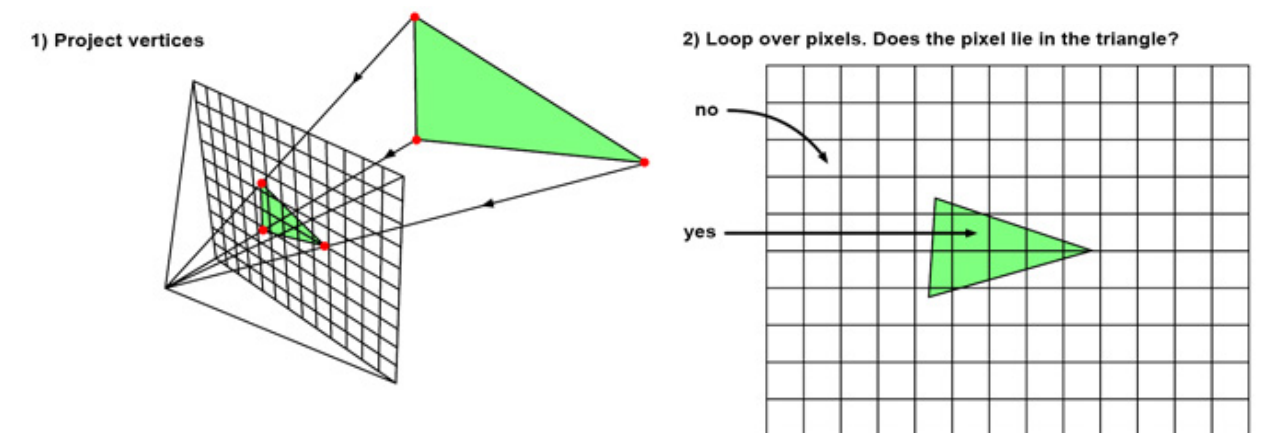
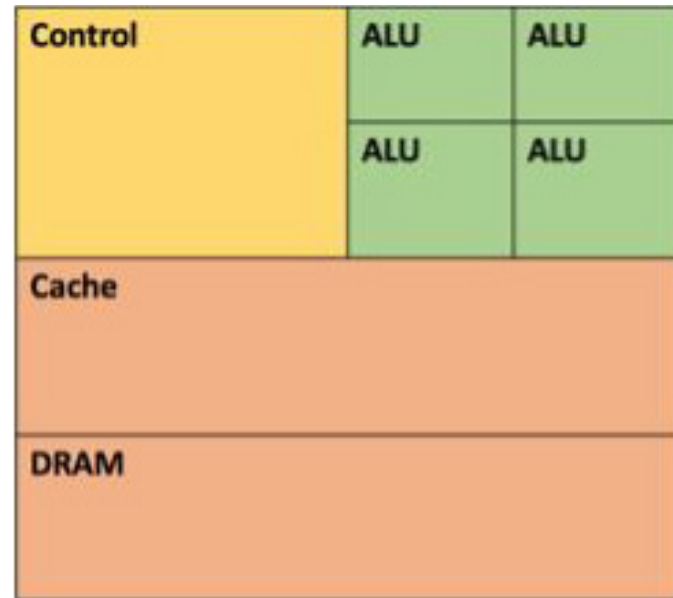
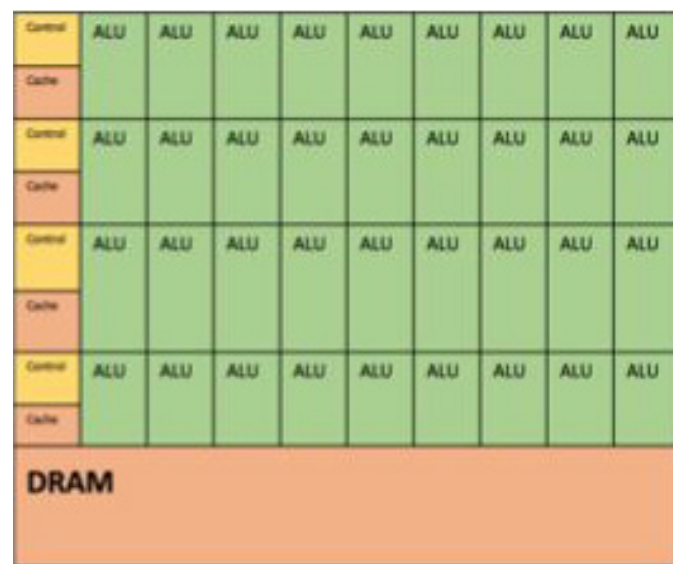


Figure 1.3.13 Rasterization



CPU



GPU

Figure 1.3.14 CPU vs GPU Processing

GPUs have many times more Arithmetic Logic Units (ALU) compared to CPUs, which allows it to better process “simple operations in parallel.”

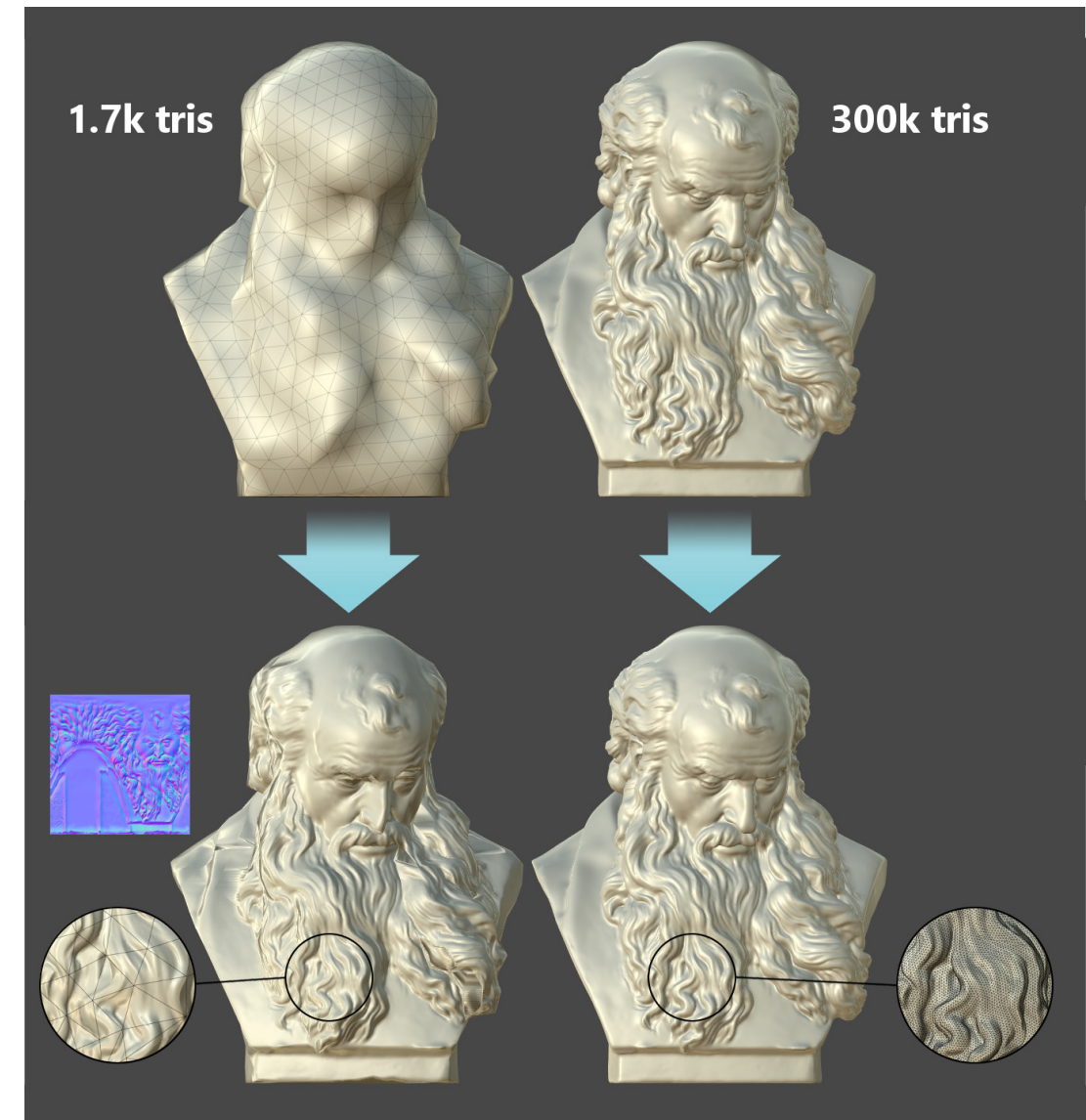


Figure 1.3.15 Texture Baking utilizes normal maps to preserve detail without the additional polygons

Interactivity

While the time savings from the increased rendering efficiency would be valuable, perhaps what will be more beneficial is the possibility for interactivity within architectural visualization. This opens up a wide range of mediums to communicate design intentions within current architectural pipelines, ranging from simple mouse and keyboard inputs with a monitor screen output to fully tracked virtual reality (VR) and augmented reality (AR) headsets and various forms of body tracking.^[25] (Fig. 1.3.16 - 17)

Beyond the static image, and even beyond the animated film, one will have the ability to experience these digital environments by directly interacting with them. This turns traditional static architectural renderings into immersive interactive real-world walkthroughs that allow clients and designers to better understand the experiential and spatial qualities of the design, adding further utility and credibility to the visualizations produced for design and presentation.

With all these benefits, the game engine can be thought of as another progression leap in architectural visualization. With so much potential in not only current game engines but also the direction of which these technologies are heading, it becomes evident how the game design industry may benefit the architecture industry. This is already noticed by some architecture firms such as HOK, who are already implementing this technology within their workflows, albeit still skimming the surface of what game engines are capable of.^[26] (Fig. 1.3.18) By investigating this overlap of tool utilization, it becomes possible to not only close the gap between architectural visualizations and films, but also to facilitate new modes of visualization and interaction—thus satisfying the increasing expectations of people as a result of the increasing general exposure to both film and game media. With the direction that technology and coincidentally both the architecture and game development fields are headed, moving towards the Game Engine for visualization seems to be the appropriate direction.

25 Pierre Pita, "List of Full Body VR Tracking Solutions," Virtual Reality Times, February 21, 2017, accessed October 17, 2019, <https://virtualrealitytimes.com/2017/02/21/list-of-full-body-vr-tracking-solutions/>.

26 Ken Pimentel, "HOK on Architectural Visualization: Aggregate, Iterate, Communicate," Unreal Engine, March 13, 2019, accessed October 17, 2019, <https://www.unrealengine.com/en-US/spotlights/hok-architectural-visualization-aggregate-iterate-communicate>.



Figure 1.3.16 A traditional desktop setup with a monitor and various input devices such as mouse and keyboard, game controller, and joysticks

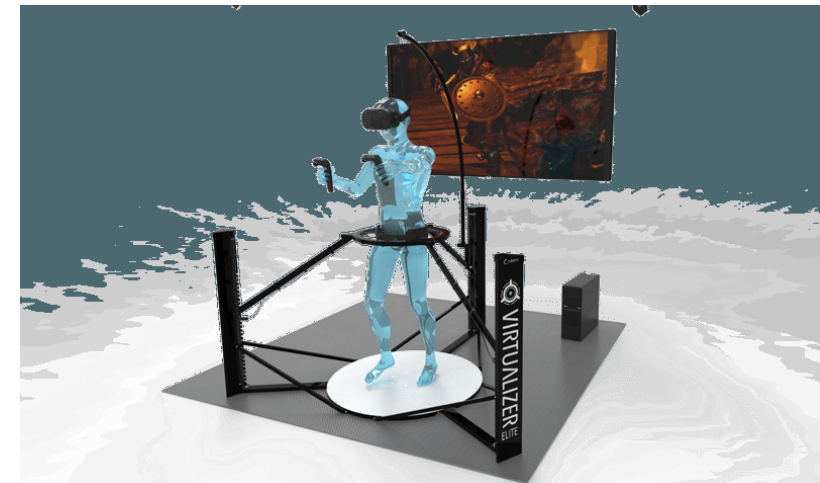


Figure 1.3.17 Possible VR setup with various trackers for interactive inputs



Figure 1.3.18 Architectural Rendering by HOK

Chapter 1.4 | Proposed Framework

To move forward, architecture needs to look beyond the static frame, but to do that, the issue of time-constrained visualizations needs to be addressed. Rather than wait for computation to progress to a point that renderings within current native architectural pipelines can be calculated instantly, it is possible to reduce the amount of time required by moving towards a more efficient visualization pipeline.

As mentioned in Chapter 1.2, current architectural visualization workflows have a large disconnect between design and production visualizations, the cause of which can be due to the lethargy of current visualization tools, which forces compromises throughout the stages of architectural design. These problems can be alleviated somewhat by utilizing a more efficient visualization workflow. The tools of pitch and design were originally separated due to limitations within current software, but now with faster computational power, as well as the advent of game engines, there is the potential to integrate these tools within the same workflow. In doing so, it becomes possible to incorporate dynamic visualizations at the earlier stages of design, which can benefit not only the resulting design but also the resulting pitch to the clients.

To be realistic with the scope of this thesis, however, it is necessary to focus on a singular beneficial aspect to improve upon within architectural visualization. Comparing static and dynamic architectural spaces, the one thing that remains consistent is the existence of occupants, and the thing that remains problematic when visualizing these spaces is the inadequate portrayal of these occupants. No matter the function of architecture, it is usually intended to be designed to be inhabited. As stated in Chapter 1.1, occupants are at the root of dynamics within the space, even before introducing any dynamic architectural elements. People are inherently dynamic, and as such, any space that is occupied, be it originally static or dynamic, will naturally become dynamic, for people themselves are dynamic elements.

People are a constant in static and dynamic architecture, as well as exterior and interior spaces, therefore, being able to visualize them in these various contexts and scales can be practical in both design and communication between the architect, client, and user groups. As seen in Chapter 1.2, however, adding people to renderings usually seems to be a rushed endeavor which results in a lackluster portrayal of how these occupants interact with the space. Architects design buildings for occupants to inhabit, and yet they rarely visualize them during the design phase due to time constraints and the lack of integration within current workflows. While there are commercial crowd simulation tools available for architectural visualization—in standalone forms such as Massive^[1] and Massmotion^[2], plug-ins such as Miarmy^[3] and Golaem^[4], or

1 "What Is Massive?," Massive Software, accessed October 17, 2019, <http://www.massivesoftware.com/applications.html>.

2 "Crowd Simulation Software: MassMotion," Oasys, accessed October 17, 2019, <https://www.oasys-software.com/products/pedestrian-simulation/massmotion/>.

3 "Miarmy," Basefount Company, accessed October 17, 2019, <http://www.basefount.com/miarmy.html>.

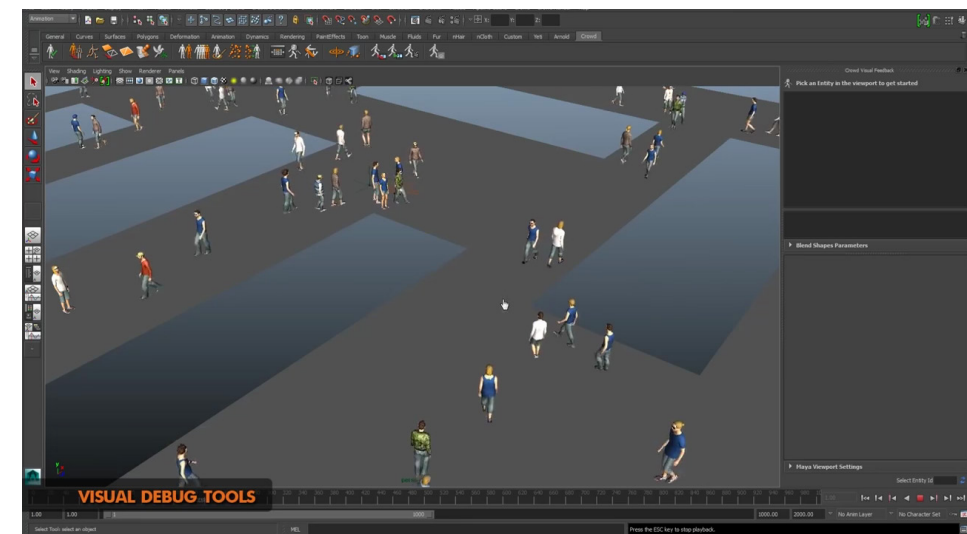
4 "Digital Extras at Your Fingertips," Golaem, accessed October 17, 2019, <http://golaem.com/>.

Figure 1.4.1 - 1.4.3

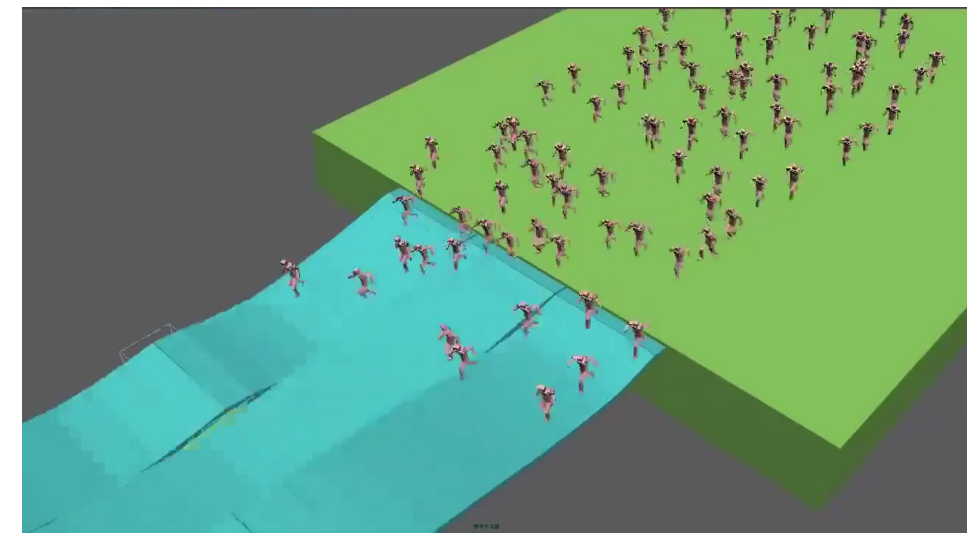
These are 3 examples of current crowd simulation tools on the market that are tailored towards film production. While these are impressive tools for generating complexity within crowd simulations, they prioritize more on controlling the crowds through the space instead of visualizing what the crowds might do within such a space. On top of this, most of them require a substantial setup as well as rendering time to obtain results close to what is shown in the films, which makes it harder to integrate within current architectural visualization pipelines. These extra steps and time requirements can incentivize architects to skip visualizing crowds entirely in tight deadlines.



▶ **Figure 1.4.1** *Massive*



▶ **Figure 1.4.2** *Golaem*

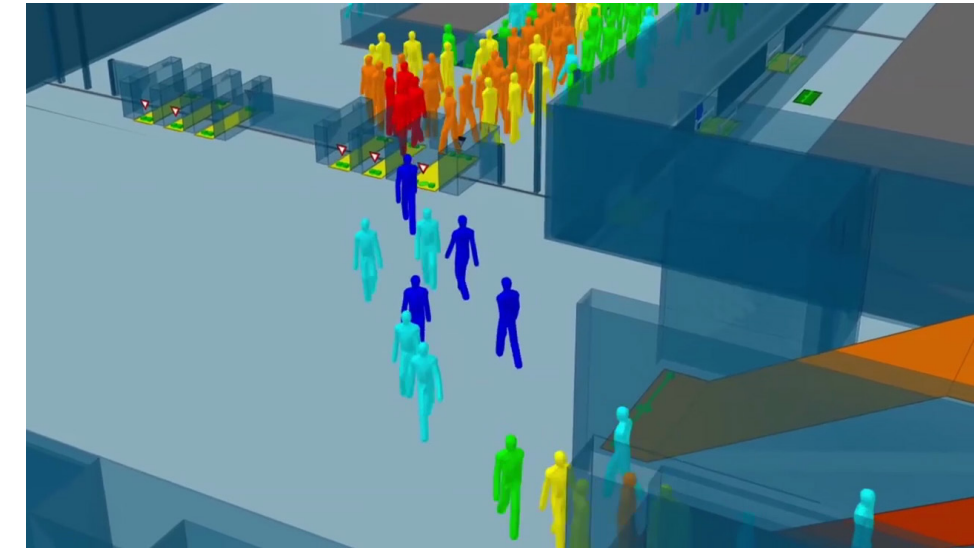


▶ **Figure 1.4.3** *Miarmy*

built into 3D applications such as 3ds Max^[5]—most of them do not consider the existence of new forms of dynamic spaces alongside the integration of a robust visualization pipeline. (Fig. 1.4.1 - 5) Because of these limitations, they are harder to integrate within contemporary architectural workflows and are unsuited for portraying the high complexity of emerging intelligent spaces, especially during the design phase when changes to the building design are constantly made.

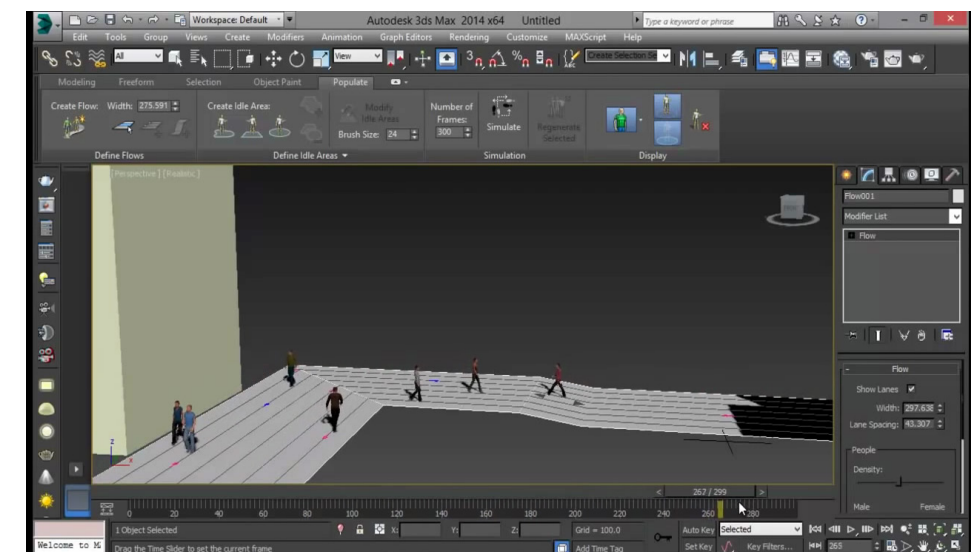
For these reasons, this thesis will propose the creation of a means—to both analyze and visualize dynamic spaces—by first establishing a methodology for simulating human crowds, and then integrating it within a game engine. In doing so, it will be shown to be possible to build a foundational framework that can be built upon to be tailored to specific needs, from which one can both extrapolate information on occupancy as well as visualizations from the same workflow. Of course, this is venturing into unknown territories, especially for an architecture student with no coding background. Therefore, personally, this thesis is an exploration of technology, a learning process, and a challenge to oneself. Ideally, this thesis is the creation of a tool and a framework with the intent to provide architects with a tool that they can both use during the design process and also evolve.

As architecture is evolving, so too must architectural visualization. Technology is changing, and this thesis is looking at creating a framework to facilitate that change for architectural design workflows. The benefits in doing so set up the means to improve efficiency, flexibility, and design, while also providing the skill and knowledge to understand, manipulate, and create new tools to tailor to any type of project, most particularly during the design process, at a point where change can be most easily incorporated. By approaching knowledge in an interdisciplinary manner, one can broaden the scope of what is possible within the field of architecture.



► **Figure 1.4.4** *Oasys mass motion*

This is a stand alone crowd simulation tool where architects must export their model into. While there is nothing wrong with this, it is an extra step, which can incentivize architects to skip visualizing crowds entirely in tight deadlines.



► **Figure 1.4.5** *A crowd Visualization Tool in Autodesk 3ds Max*

Here, the user defines a path where people are generated to follow. This is a simple approach for visualizing people for a pitch, but it does not show how the space will be used, rather how the architect thinks the space will be used.

⁵ "Example: Using Populate," Autodesk Support & Learning, accessed October 17, 2019, <https://knowledge.autodesk.com/support/3ds-max/learn-explore/caas/CloudHelp/cloudhelp/2017/ENU/3DSMax/files/GUID-BEA89C57-3A7B-4AB5-AAF7-02494AA01CFA-htm.html>.

Part 2 | Technical Research

Analog to Digital

Analog to Digital

To begin this exploration, initial research is required. Therefore, as a starting point to creating a crowd simulation tool, it is important to first look at the basis of a computer simulation, not just what it is, but how it works at the fundamental level. This section then will mainly focus on developing the methodologies required to create a model for human crowd simulation. Chapter 2.1 will investigate the concepts to consider when creating a simulation model by looking into the different components of a simulation and the concept of emergence and how it relates to complex systems and human crowds. Chapter 2.2 will then investigate how these concepts can be used to create a simulation model for simulating crowds by looking into the concept of autonomous agents. Chapter 2.3 will utilize this model to translate human behaviors to machine logic by establishing a set of human systems that can drive the agents within the simulation model. Chapter 2.4 will investigate a framework for simulating a variety of architectural objects by understanding how dynamic elements may work and what the resulting spaces may entail. Chapter 2.5 will then take these established systems and create a prototype to validate the validity of this model based on the framework established within Chapter 2.2.

Chapter 2.1 | Simulations Ideology

A simulation—as defined by Jerry Banks et al. in their book *Discrete-Event System Simulation*—is an “imitation of the operation of a real-world process or system over time.”^[1] Since the advent of digital computation and computer simulations, their utility have become much greater, and are now utilized within a wide range of applications, including various types of manufacturing, construction engineering, military, logistics, transportation, distribution, business processes, and human systems.^[2] Because of this array of uses, it is important to first determine the type of system that the simulation is required to imitate. The system—as defined by Banks et al.—is “a group of objects that are joined together in some regular interaction or interdependence toward the accomplishment of some purpose.”^[3] He states, “In order to understand and analyze a system, a number of terms need to be defined. An entity is an object of interest in the system. An attribute is a property of an entity. An activity represents a time period of specified length. If a bank is being studied, customers might be one of the entities, the balance in their checking accounts might be an attribute, and making deposits might be an activity.”^[4]

Learning from this, it is then possible to investigate various other systems. Within a pool of water, the entities would be the water particles, with their location, velocity, and mass being some of the possible attributes, and colliding with each other being one of the possible activities. **(Fig. 2.1.1)** The same can be observed in a traffic system, where the cars would be the entities, with their location, size, color, and car typology being possible attributes, and starting or stopping being possible activities. **(Fig. 2.1.2)** Taking this investigation then, to a crowd of people, it can be abstracted that the individual people would be the entities, with their location, gender, height, weight, etc. being possible attributes, and their various interactions with one another being activities. **(Fig. 2.1.3)**

Of course, these are just generic assumptions, since “a complete list cannot be developed unless the purpose of the study is known.”^[5] However, by investigating these assumed systems, a pattern can be noticed. From the water example, each particle interacts with surrounding particles through collision, which are affected by the particle attributes such as velocity and mass. If one were to toss a rock into the water, the rock would offset local particles at the point of impact, which will interact with particles around it, producing ripple waves within the system. From the traffic example, it can be observed that when one car slows down the subsequent car slows down as well, producing an offset until there is a wave of phantom traffic within the road. From the crowd example, it can then be observed that the people, much like the water particles, collide into one another, each instigating interaction locally as they ripple throughout the space, once again producing a wave-like pattern.

1 Jerry Banks et al., *Discrete-Event System Simulation* (Upper Saddle River, NJ: Prentice Hall, 2001), 3.
 2 Banks et al., *Discrete-Event System Simulation*, 3.
 3 Banks et al., *Discrete-Event System Simulation*, 10.
 4 Banks et al., *Discrete-Event System Simulation*, 10.
 5 Banks et al., *Discrete-Event System Simulation*, 10.



▶ **Figure 2.1.1** Various wave patterns seen on-top of the ocean surface



▶ **Figure 2.1.2** Video showing phantom traffic jam



▶ **Figure 2.1.3** This crowded concert shows how the interaction between each individual human produces various wave patterns throughout the entire crowd.

To better understand this phenomenon, it is important to understand *emergence*. This term is generally used to characterize the behavior of systems in which its components interact in various ways by following local rules, producing nonlinear behaviors often resulting in greater complexity than the sum of its parts. Johnson explains this in his book *Emergence: The Connected Lives of Ants, Brains, Cities*:

“What features do all these systems share? In the simplest terms, they solve problems by drawing on masses of relatively stupid elements, rather than a single intelligent “executive branch.” They are bottom-up systems, not top down. They get their smarts from below. In a more technical language, they are complex adaptive systems that display emergent behavior. In these systems, agents residing on one scale start producing behavior that lies one scale above them: ants create colonies; urbanites create neighborhoods’ simple pattern-recognition software learns how to recommend new books. The movement from low-level rules to higher level sophistication is what we call emergence.”^[6]

He then conceptualizes a billiard table with motorized billiard balls programmed to alter their movement based on interactions, stating that “such a system would define the most elemental form of complex behavior: a system with multiple agents dynamically interacting in multiple ways, following local rules and oblivious to any higher-level instructions. But it wouldn’t truly be considered emergent until those local interactions resulted in some kind of discernible macrobehaviour.”^[7]

What can be extracted from this text is that this “movement from low level rules to high level sophistication” is what is known as emergence and can be observed in various complex system models, as it is a fundamental property of the universe. (Fig. 2.1.4 - 6) It can then be observed from these water/traffic/crowd examples that due to emergence, a wave-like pattern emerges from the local interactions of the particles. These interactions, while simple, propagate throughout the system using the particles as a medium, producing this wave-like pattern that can only be observed from a scale larger than the particles but can only be understood by looking at the particles themselves.

While these systems are complex in nature, and can be hard to comprehend, they generally exhibit common traits that can be investigated to help break down their complexity. By observing these simulations as complex systems, it can become easier to understand the basic components that make up the system. These basic components can then be modified to create a simulation model, meaning that by following this methodology, it is possible to produce relatively complex simulations with relatively simple components.

Figure 2.1.4 - 2.1.6
Emergence is a fundamental property of the universe. Subatomic particles such as protons, neutrons, and electrons combine to form atoms that are responsible for all the various emergent properties of matter that we come in contact with on a daily bases. These figures show some examples of emergent behaviors within nature from further interactions of these emergent properties.

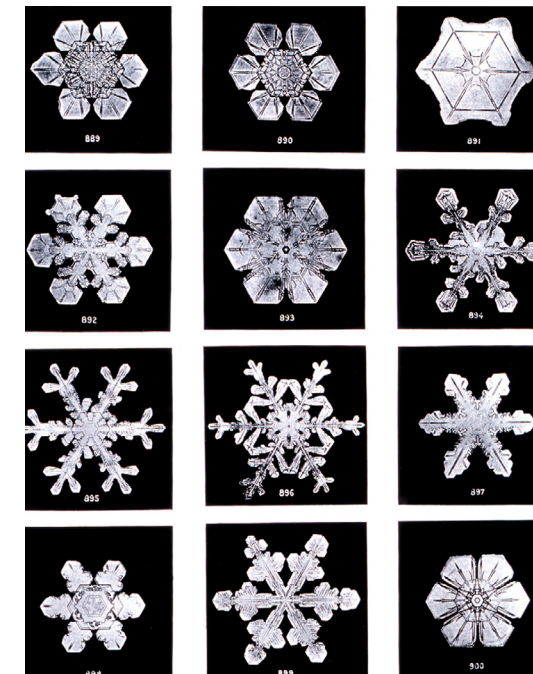


Figure 2.1.4 Snowflakes



Figure 2.1.5 Termite mound



▶ Figure 2.1.6 Starling murmurations

6 Steven Johnson, *Emergence: The Connected Lives of Ants, Brains, Cities, and Software* (New York: Scribner, 2004), 18.
7 Steven Johnson, *Emergence: The Connected Lives of Ants, Brains, Cities, and Software*, 19.

This concept is especially important when simulating humans due to their inherent complexity. The human brain is a complex object and to this day is still not fully understood, therefore, simulating human behaviors *perfectly* is beyond the current capabilities of collective human knowledge. As stated by Delaney and Vaccari in *Dynamic Models and Discrete Event Simulation*: “one approach with coping with such difficulties is to relax the requirements on the model so that approximation can be employed even though they yield fewer and/or less accurate results”^[8] This means that simulations can vary in levels of complexity, ranging from simple real time calculations for approximating visualizations to high accuracy scientific models. Banks et al. also acknowledges this, stating:

“The art of modeling is enhanced by an ability to abstract the essential features of a problem, to select and modify basic assumptions that characterize the system, and then to enrich and elaborate the model until a useful approximation results. Thus, it is best to start with a simple model and build towards greater complexity. However, the model complexity need not exceed that required to accomplish the purposes for which the model is intended. Violation of this principle will only add to model-building expenses. It is not necessary to have a one-to-one mapping between the model and the real system. Only the essence of the real system is needed.”^[9]

It should be noted how this aligns with Jason Gregory’s description of video games back in Chapter 1.3, which is a good indication of the validity of this approach when the time comes to integrate this model within a Game Engine. With this in mind, it can then be determined that a crowd simulation for architectural visualization purposes—that can also be evolved by architects—would benefit more from simplicity and speed, rather than perfect accuracy from the beginning. This is already an improvement over not simulating at all, and since the behavior of people is not absolute and exact, an approximation is likely sufficient to gauge whether or not a space is working properly in a crowd situation. It is not required to figure out all the nuances of the system, but instead focus on determining simple agents that can utilize emergent behaviors from a bottom-up approach. These agents would then act upon simple embedded rules to move around the space, and if need be, higher accuracy can be generated by adding additional simple rules to these agents. One does not need to simulate behaviors perfectly, but rather account for enough variance that the system as a whole is realistic enough to provide information and believability to inform the design process. In doing so, it makes the creation process much simpler; instead of creating artificial intelligence, this simulation can be based on simple rules to give the illusion of distributed intelligence. **(Fig. 2.1.7 - 8)**

8 William Delaney and Erminia Vaccari, *Dynamic Models and Discrete Event Simulation* (New York: M. Dekker, 1989), 323.
 9 Banks et al., *Discrete-Event System Simulation*, 15.

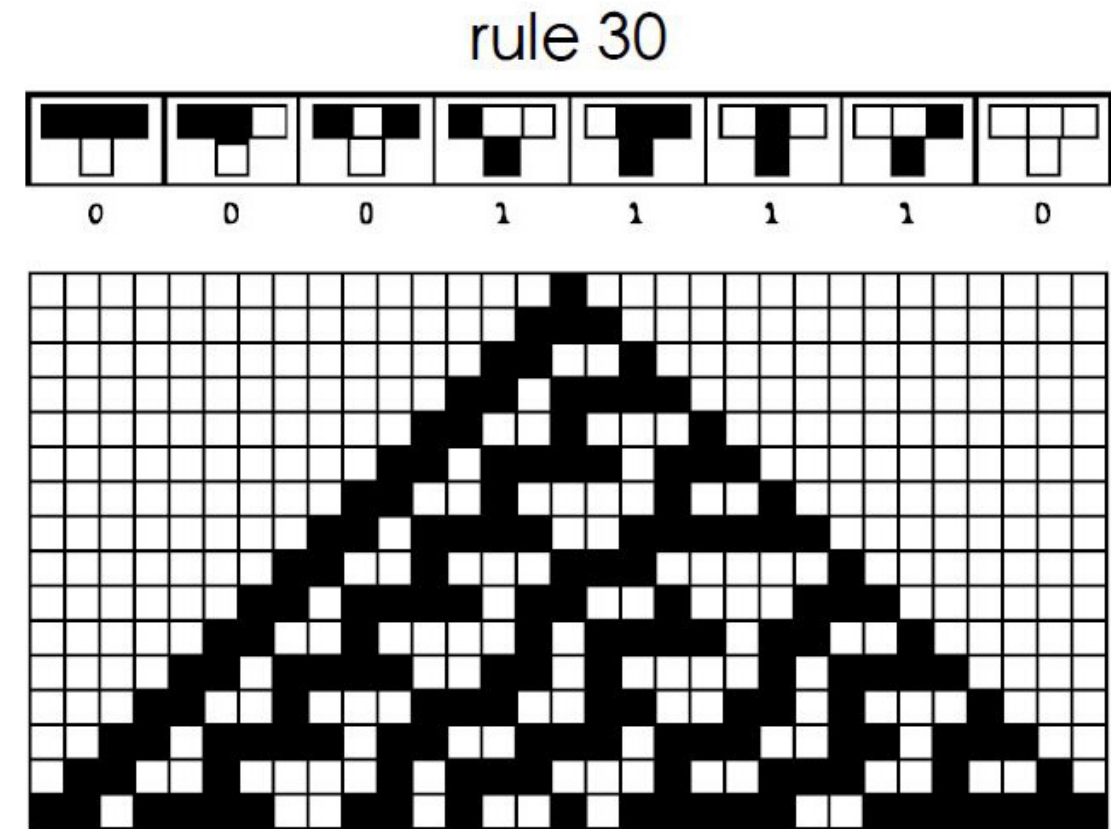


Figure 2.1.7 Rule 30 as introduced by Stephen Wolfram, 1983
 What is compelling about the phenomenon of emergence in the context of crowd simulations is how one can utilize relatively simple logic to generate complex behaviors.

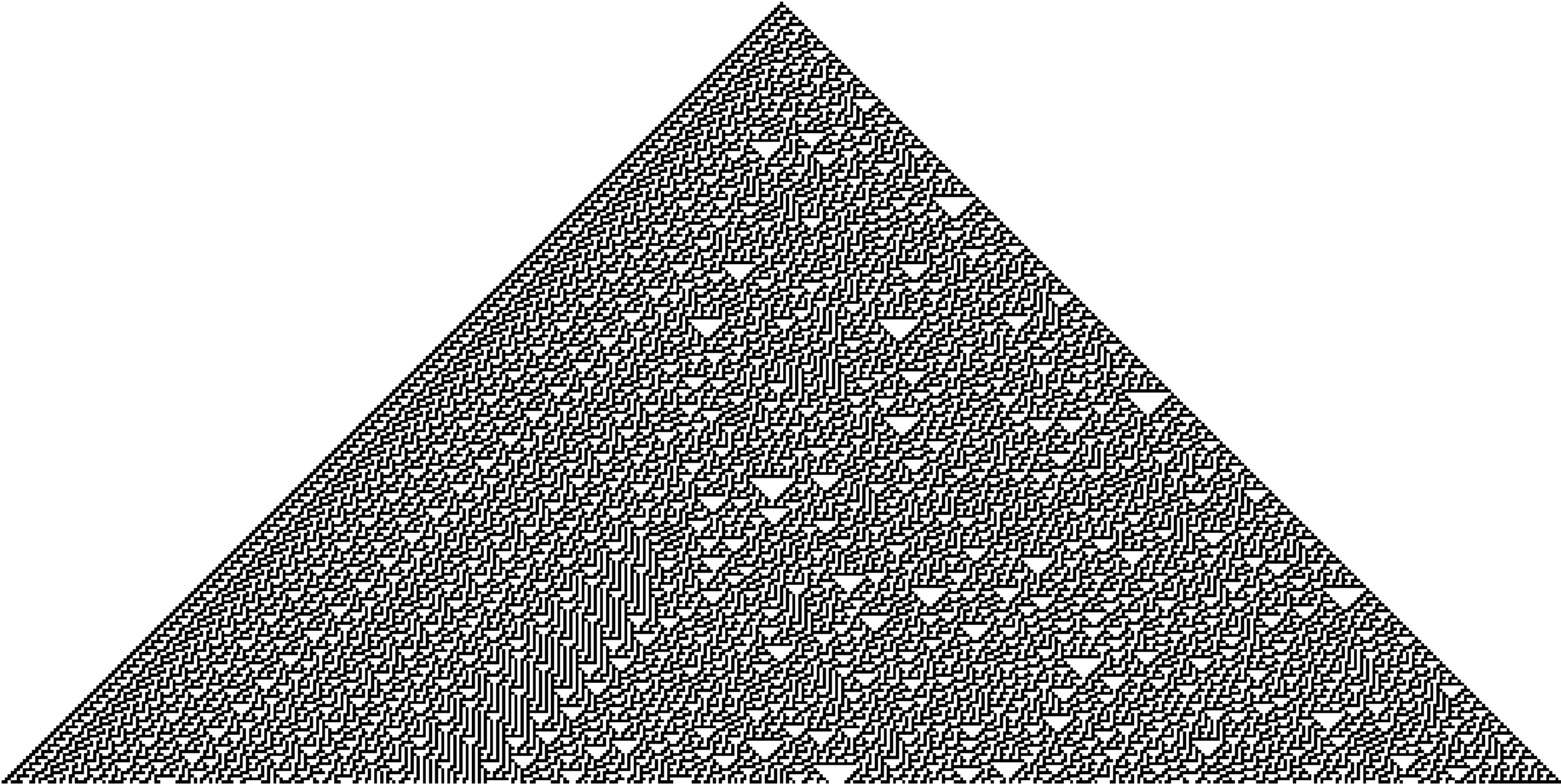


Figure 2.1.8 250 iterations of Rule 30

Chapter 2.2 | Establishing Model Methodology

Now that the foundational principles of complex system simulation have been investigated, and the plan of utilizing a bottom-up simulation methodology has been established, it is possible then to dive deeper into the creation of this simulation. Within *Discrete-Event System Simulation*, Jerry Banks et al. determined a set of steps to facilitate a thorough simulation study to aid in creating a simulation model.^[1] (Fig. 2.2.1) He breaks this down into four phases:

“The first phase, consisting of steps 1 (Problem Formulation) and 2 (Setting of Objective and Overall Design), is a period of discovery or orientation. [...] The second phase is related to model building and data collection and includes steps 3 (Model Conceptualization), 4 (Data Collection), 5 (Model Translation), 6 (Verification), and 7 (Validation). [...] The third phase concerns running the model. It involves steps 8 (Experimental Design), 9 (Production Runs and Analysis), and 10 (Additional Runs). [...] The fourth phase, implementation, involves steps 11 (Documentation Reporting) and 12 (Implementation).”^[2]

These phases describe a very logical approach, as such they line up nicely with the chapter structure of this thesis. The first phase of discovery and orientation is covered in *Part 1: Introduction and Theory*, where the proposed framework was established; the second phase of model building and data collection is covered within this current part (*Part 2: Technical Research*), as well as *Part 3: Tool Creation* for when the simulation model is updated for the game engine environment; The third phase concerning running the model is covered within *Part 4: Simulation Applications*, where the simulation will be utilized within a variety of applications to review their effectiveness in architectural crowd simulation; The fourth phase of implementation is finalizing the resultant simulation as a tool within visualization applications. Since the initial scope of this thesis was meant to provide a foundational framework for this tool, the full completion of this phase may be beyond the scope of this thesis and would be suited to further development at a later date.

As it can be seen, this relatively simple framework provides a good starting point and guideline in creating a custom model for crowd simulation. Therefore, by applying this further, it is possible to provide a more thorough breakdown of what each step may entail in relation to this thesis.

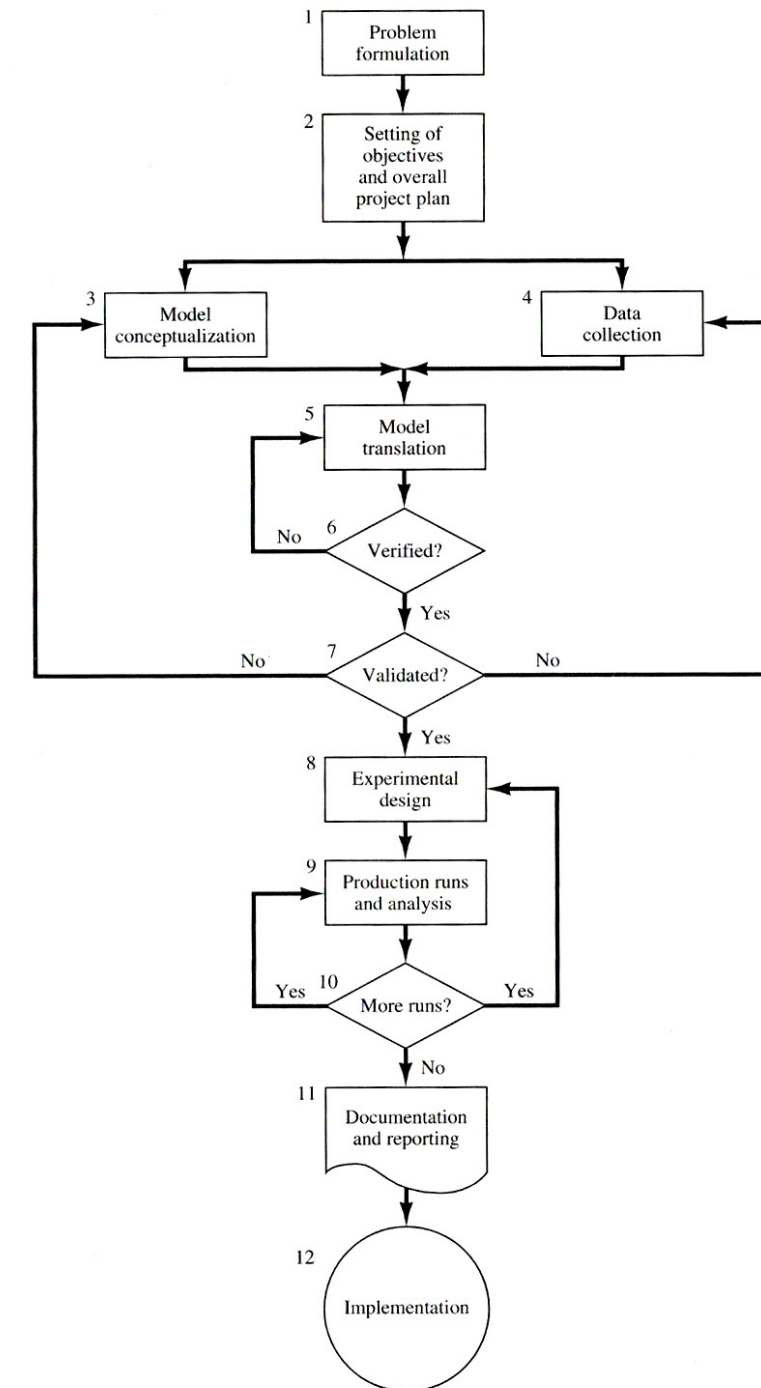


Figure 2.2.1 Simulation Study Diagram

1 Banks et al., *Discrete-Event System Simulation*, 15.
 2 Banks et al., *Discrete-Event System Simulation*, 19-20.

Problem formulation: It is important to begin this process by first stating the problem. This was already established in *Part 1: Introduction and Theory* by investigating the emergence of dynamic architecture in *Chapter 1.1*, the inadequacy of current visualization methods in *Chapter 1.2*, and the advent of game engines in *Chapter 1.3*.

Setting of objectives and overall project plan: This step aims to indicate the questions that can be answered with the simulation. In the context of this thesis, the objective would then be finding a way to simulate and visualize human crowd movements within a dynamic architectural space.

Model conceptualization: This step is the most technical, as it deals with establishing the various components within the simulation model, as well as defining their interactions within the system. This is what this current chapter (2.2) will investigate and what the next chapter (2.3) will establish. This model will then be updated as the thesis moves on from the processing prototype to the game engine environment.

Data Collection: This step focuses on the collection of input data for the model, and as such, constant interplay will be present between this step and the last. In the context of this thesis, Chapter 2.3 will be investigating various properties of human behaviors and how that can be utilized and represented within the system model. The initial data from this step, as well as the established methodology from the initial model conceptualization, will be invaluable when re-establishing the model within the game engine environment.

Model Translation: This step deals with the technical translation of this simulation model into machine logic. The Java-based program processing will be utilized first to prototype this methodology as a way to validate the concept of emergent complex systems within the context of crowd simulations. Once this processing prototype is validated, the model will then be translated into a game engine environment to utilize its higher abstraction tools.

Verification: This step mainly focuses on debugging to ensure the software operates according to the model. Fixing certain software bugs can take minutes to days, therefore, this step arguably provides the greatest unknown in terms of time expenditure due to the inherent complexity and uncertainty within the debugging process.

Validation: The aim of this step is to determine whether the simulation model is an accurate representation of the real system. As such, this step may be repeated until the resulting model accuracy is judged acceptable. In the context of this thesis, the processing prototype will first need to be validated to confirm that the concept of emergent behaviors from autonomous agents is enough to convey the movements of human crowds. Then, once the model is re-updated for the game engine environment, this step will need to be revisited to confirm this new model within the context of human crowds.

Experimental design: This step determines the various alternative systems that can be simulated within this model. This thesis will be investigating this in Part 4, where it will be utilizing the developed simulation tool within a variety of architectural scenarios to determine the usability of this tool within alternative spatial applications.

Production runs and analysis: This step utilizes the simulation to estimate measures of performances within these alternative systems. This step can be utilized to analyze the architectural scenarios from the experimental design step, which can then be used to measure their effectiveness in visualizing spatial typologies.

More runs: Additional runs may or may not be required depending on the analysis and updates of the previous runs.

Documentation and reporting: The documentation of the methodologies and concepts obtained from creating this model will allow potential upgrades to the system model in order to provide increased accuracy and performance metrics in the future.

Implementation: At this point, it is possible to establish a rough workflow within architectural visualization that can benefit with the utilization of this crowd simulation tool.

As shown by the various steps of this framework, the next stage in creating this crowd simulation—as well as the focus of this and the next few chapters—is developing a model for this simulation system. A model as defined by Banks et al. is “an abstract representation of a system, usually containing structural, logical, or mathematical relationships which describe a system in terms of state, entities and their attributes, sets, processes, events, activities, and delays.”^[3] He remarks, “Just as the components of a system were entities, attributes, and activities, Models are represented similarly. However, the model contains only those components that are relevant to the study.”^[4] Going by this, it is then important to first simplify architectural spaces into its fundamental relevant components. The purpose of this model, then becomes to define the interactions between classes of entities within the system, which can be defined as the active human agents and the passive (or active in the case of dynamic spaces) architectural objects.

³ Banks et al., *Discrete-Event System Simulation*, 64.

⁴ Banks et al., *Discrete-Event System Simulation*, 13.

One way to simplify the interactions between these entities may be to compare it with a simpler system. By looking back to Chapter 2.1, one can re-investigate the similarities between the crowd and water simulations. From this, it can then be observed that the crowd simulation, when dense enough, can arguably be described as a fluid. Fluid flows can be described in either the Lagrangian description—where each fluid particle is calculated as discrete particles—or the Eulerian description—where the fluid properties are only calculated at the boundary of set volumes.^[5] (Fig. 2.2.2 - 3) While there are pros and cons to both methods, the Lagrangian methods will provide a more accurate model for non-continuum mass as well a more fluid simulation that would be more beneficial for crowd simulation. The downside to this is that extra computation is required to calculate each particle as more people are added into the system. However, by designing this model as a framework that can be scaled up, this limitation becomes less of an issue, especially with computation becoming faster with technological progression. However, humans are undeniably not as elementary as water particles. They do not simply flow, but have desires, motives, and goals. While it can be seen from the concert example that individuality starts to break down in larger crowds—and thus began to behave like the flow of water as the forces exerted on the individual people move in a direction that is determined by the crowd—it is still important to distinguish people in smaller groupings. Therefore, these rules at the bottom level will not only need to accommodate movement but also human behaviors. To create such a system, one must determine the logic behind one’s actions and look at ways to translate them into machine logic, essentially breaking them down to their fundamental elements.

5 John M. Cimbala, “Descriptions of Fluid Flows,” Penn State Engineering, accessed August 3, 2019, https://www.mne.psu.edu/cimbala/Learning/Fluid/Introductory/descriptions_of_fluid_flows.htm.

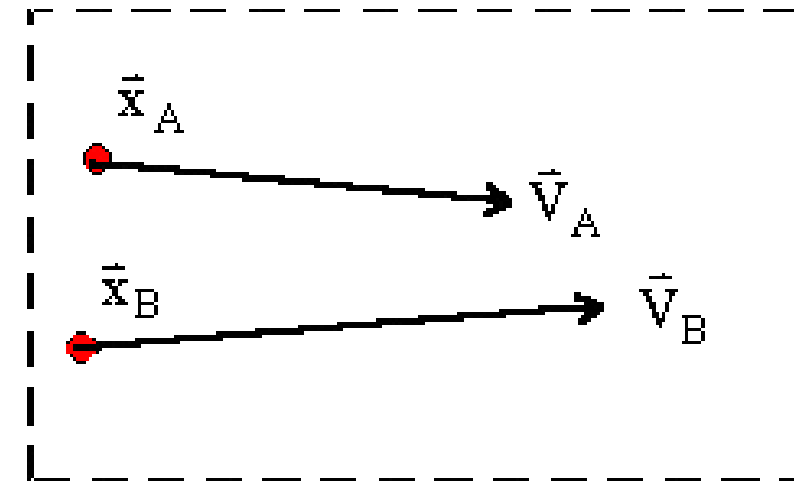


Figure 2.2.2 The Lagrangian description calculates the position and velocity of the individual particles within the fluid

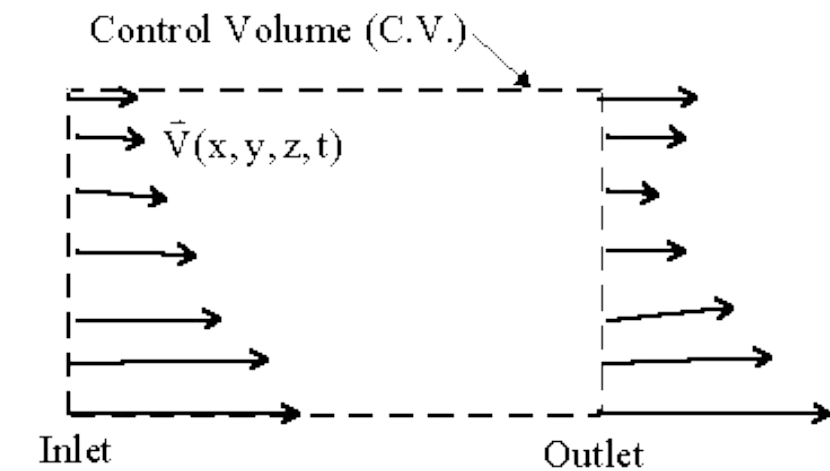


Figure 2.2.3 The Eulerian description calculates the output velocities from the input velocities, in which the space inside the control volume is assumed to be completely filled as a continuous mass

There are, of course, many ways to do this. A quick investigation reveals a plethora of existing simulation methodologies, ranging from Eulerian-based models such as Cellular Automata,^{[6][7]} to Lagrangian-based particle models such as Craig Reynold’s Boids,^[8] Helbing’s Social Forces,^[9] Van den Berg’s Reciprocal Velocity Obstacles,^[10] Adaptive Roadmaps,^[11] Centroidal Particles,^[12] and HiDAC.^[13] **(Fig. 2.2.5 - 10)** While all these different methods can be overwhelming for someone without an extensive coding background, one commonality among these models is the utilization of autonomous agents. **(Fig. 2.2.4)** Taking a cue from the Lagrangian methodology, it is then imperative to treat crowd simulations as an example of a multi-agent dynamic system, where the overall system can be broken down into *agents* and *operations*. Much like how the water simulation can be broken down into water particles, and the mathematical models used to describe how individual particles affect each other, a crowd simulation can be broken down into the people (agents), and the human interactions between them (operations). To create this system, one must understand the principles of intelligent autonomous agents, which are entities that can act in their environment without external influences from a leader or global plan. These entities have three key concepts that should be kept in mind when establishing them:

“An autonomous agent has a *limited* ability to perceive the environment.”^[14]

“An autonomous agent processes the information from its environment and calculates an action.”^[15]

“An autonomous agent has no leader.”^[16]

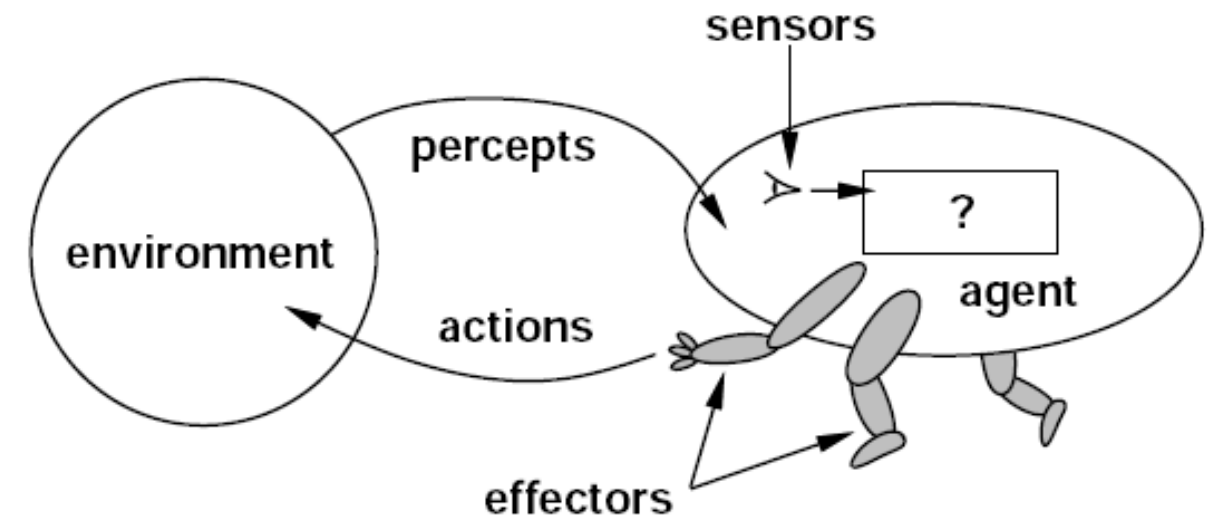


Figure 2.2.4 Much like how humans interact with spaces, Autonomous Agents act within the simulation through its perception of the environment

6 Jana Dadova, “Cellular Automata Approach for Crowd Simulation” (Master’s thesis, Comenius University, Bratislava, 2012), 1-58, accessed August 3, 2019, http://www.sccg.sk/~dadova/phd/rigorozka_dadova_final.pdf.

7 Hubert Klüpfel, “A Cellular automaton model for crowd movement and egress simulation,” (July 2003): 1-136, accessed December 26, 2019, https://www.researchgate.net/publication/29800160_A_Cellular_automaton_model_for_crowd_movement_and_egress_simulation.

8 Craig W. Reynolds, “Steering Behaviors For Autonomous Characters,” Reynolds Engineering & Design, accessed October 17, 2019, <http://www.red3d.com/cwr/steer/gdc99/>.

9 Dirk Helbing and Péter Molnár, “Social Force Model for Pedestrian Dynamics,” *Physical Review E* 51, no. 5 (1995): 4282-286, doi:10.1103/PhysRevE.51.4282.

10 Jur Van Den Berg, Ming Lin, and Dinesh Manocha, “Reciprocal Velocity Obstacles for Real-Time Multi-Agent Navigation,” *2008 IEEE International Conference on Robotics and Automation*, (May 2008): 1928-1935, <https://doi.org/10.1109/robot.2008.4543489>.

11 Avneesh Sud et al., “Real-time Navigation of Independent Agents Using Adaptive Roadmaps,” *ACM SIGGRAPH 2008*, (2008): doi:10.1145/1401132.1401207.

12 Omar Hesham and Gabriel Wainer, “Centroidal Particles for Interactive Crowd Simulation,” *2016 Summer Computer Simulation Conference (SCSC 2016)*, (2016): <https://doi.org/10.22360/summersim.2016.scs.012>.

13 Nuria Pelechano, Jan M. Allbeck, & Norman I. Badler, “Controlling Individual Agents in High-Density Crowd Simulation,” *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, (2007): 99-108, <http://repository.upenn.edu/hms/210>.

14 Daniel Shiffman, “Chapter 6. Autonomous Agents,” in *The Nature of Code* (United States: D. Shiffman, 2012), accessed October 17, 2019, <https://natureofcode.com/book/chapter-6-autonomous-agents/>.

15 Shiffman, “Chapter 6. Autonomous Agents.”

16 Shiffman, “Chapter 6. Autonomous Agents.”

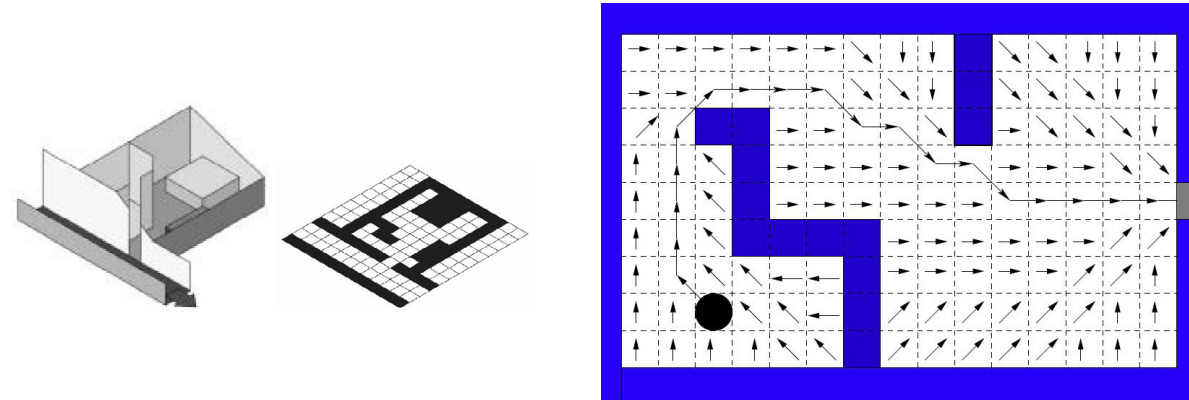


Figure 2.2.5 *2-Dimensional Cellular Automata*

The Cellular Automata approach divides a space into a grid of cells. Rule 30 (introduced at the end of chapter 2.1) is an example of a 1-Dimensional Cellular Automata that utilizes simple rules to specify the next color of a cell based on its color and its neighbors. This concept can then be used to calculate position amongst the grid as a way to establish pathfinding.

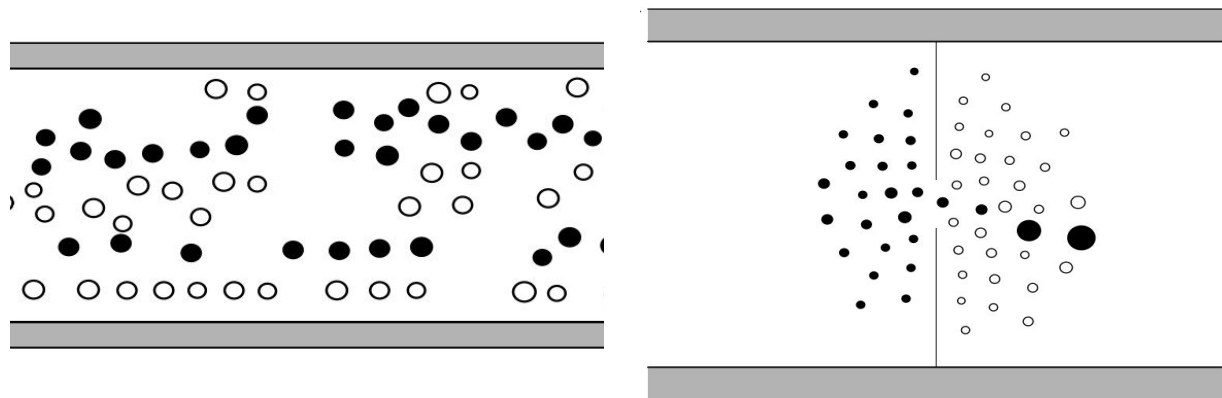


Figure 2.2.6 *Social Forces*

The Social Forces approach utilizes “social forces” that are “a measure for the internal motivations of the individuals to perform certain actions.” The diagrams above shows how this can produce the formation of lanes as well as follow behaviors for agents with the same desired walking directions through narrow doors.

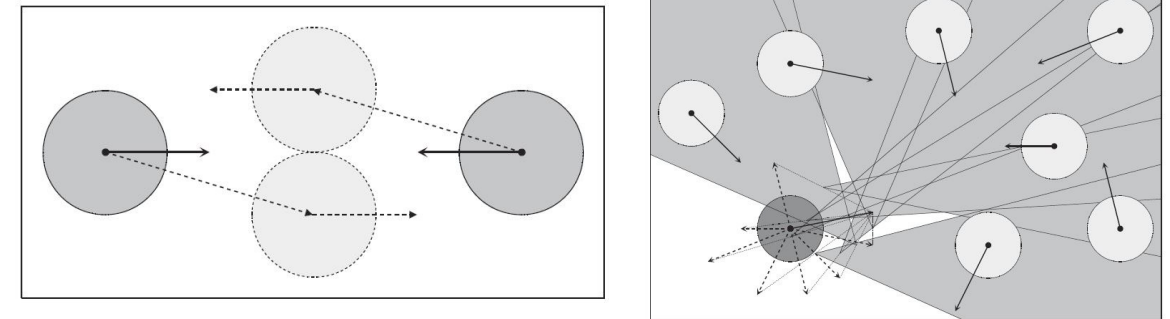


Figure 2.2.7 *Reciprocal Velocity Obstacles*

The RVO approach accounts for the reactive behavior of other agents by assuming that all agents make similar collision-avoidance reasonings within navigation. The diagrams above shows how the paths of two agents have opposite preferred velocities and how each agent calculates a combined velocity from the union of velocities from other agents.

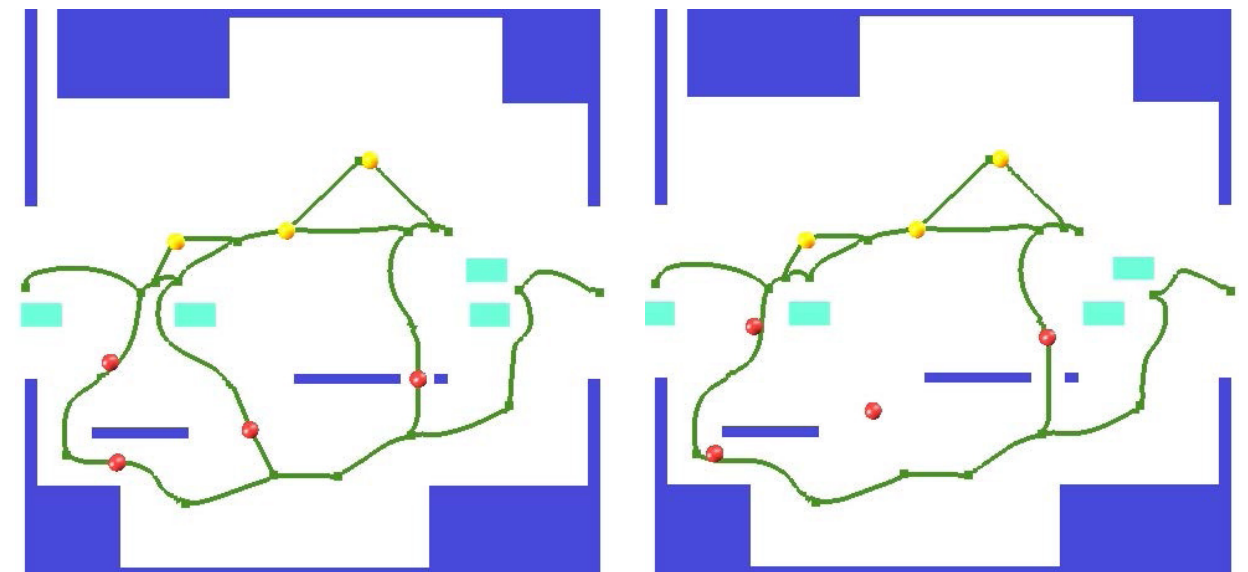


Figure 2.2.8 *Adaptive Roadmaps*

The Adaptive Roadmaps approach utilizes Adaptive Elastic ROadmaps (AERO) to perform global path planning for each agent. The diagrams above shows how dynamic obstacles such as cars can affect paths leading to the resulting goal.

Distinguishing these concepts allows the creation of custom categories that are specific to crowds, which will ensure a simpler foundation in the translation from physical behaviors to virtual code. From here, one can begin to establish the systems that allow these autonomous agents to operate as humans within the simulation. As such, this simulation model will be focusing on determining the interactions between autonomous agents and their environment.

Ehsan Baharlou states in his thesis *Generative Agent-Based Architectural Design Computation: Behavioral Strategies for Integrating Material, Fabrication and Construction Characteristics in Design Processes*:

“Complex systems, which exhibit holistic behaviors, are difficult to comprehend. Complex behaviors, which arise from interactions among parts, need a model to gain insight into processes that exhibit emergent phenomena. [...] Therefore, a model should be simple enough to represent its main purpose. The purpose of self-organizing a system is adaptation with dynamic complexity arising from emergent phenomena. Avoiding unnecessary details allows models to maintain a reasonable level of complexity (Miller and Page 2007, pp. 36-37).^[17] This method of modeling allows further analyses and investigations; otherwise, another level of simplification would be required to explain the growing complexity.”^[18]

This communicates that there is merit in avoiding unnecessary details in order to simplify the model. Complex systems are inherently “difficult to comprehend,” as such it becomes more effective to simulate this complexity by utilizing the emergent phenomena that is present within self-organized adaptive dynamic models. “Once [this] model is deemed valid, it is extendable to different variables and parameters,”^[19] which allows the process of refinement and improvement upon future iterations of this simulation. From this, it becomes clear that, at this stage, it is beneficial to not get carried away with higher complexity models, and instead investigate a more basic approach, not only to simplify the process but also to build a deeper understanding of how these systems work at a fundamental level. As such, this thesis will investigate and build upon the algorithmic steering behaviors developed by computer scientist Craig Reynolds in the late 1980s, specifically for its relatively simple and comprehensive nature.^[20] While this may not produce results at the level of higher complexity models, such as HiDAC that utilizes both “psychological and geometrical rules [alongside] social and physical forces,”^[21] it will build a solid foundation that can be expanded later to encompass additional considerations.

17 John H. Miller and Scott E. Page, *Complex Adaptive Systems: An Introduction to Computational Models of Social Life* (Princeton, NJ: Princeton University Press, 2007), 36-37.
 18 Ehsan Baharlou, *Generative Agent-Based Architectural Design Computation: Behavioral Strategies for Integrating Material, Fabrication and Construction Characteristics in Design Processes* (Stuttgart: Institute for Computational Design and Construction, 2017), 46.
 19 Baharlou, *Generative Agent-Based Architectural Design Computation: Behavioral Strategies for Integrating Material, Fabrication and Construction Characteristics in Design Processes*, 74.
 20 Reynolds, “Steering Behaviors For Autonomous Characters.”
 21 Pelechano, Allbeck, & Badler, “Controlling Individual Agents in High-Density Crowd Simulation.”

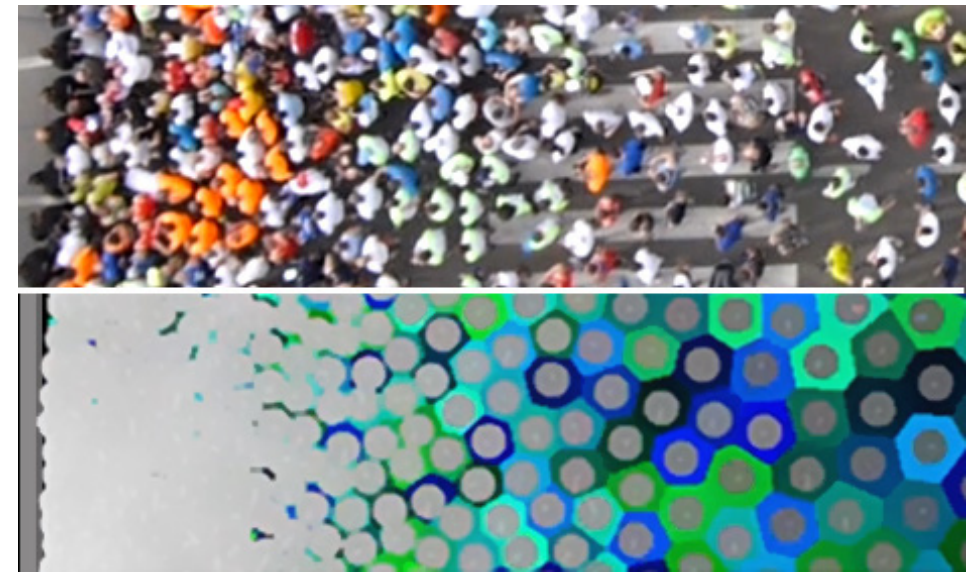


Figure 2.2.9 Centroidal Particles
 The Centroidal Particles approach utilizes personal spaces to calculate compressional forces between the agents to calculate agent interactions.

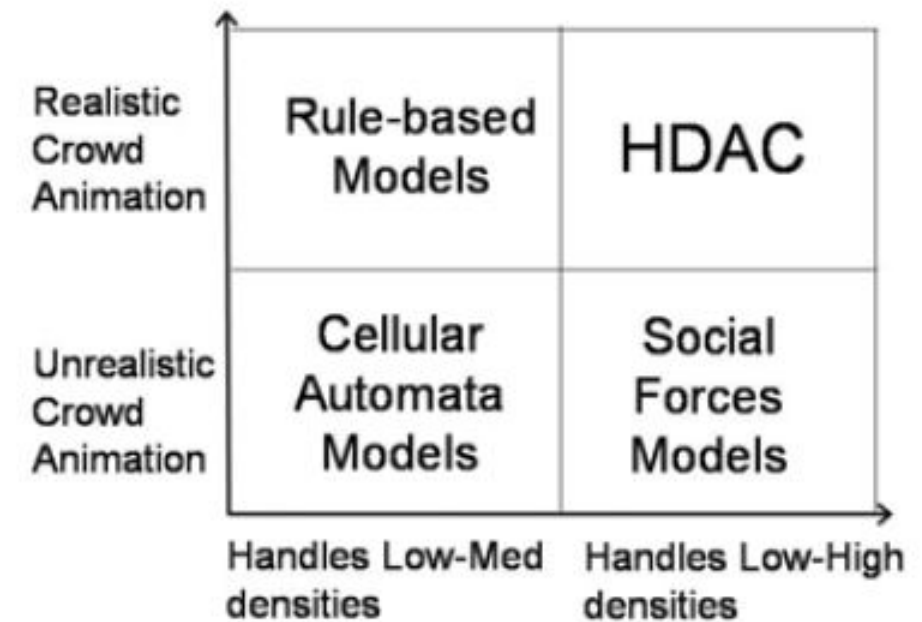


Figure 2.2.10 HiDAC
 The HiDAC approach utilizes “a combination of psychological and geometrical rules with a social and physical forces model” to create a variety of emergent behaviours such as line formation and pushing.

► **Figure 2.2.11** *Craig Reynold's flocking Boid flocking model describes these complex patterns with 3 simple steps along with 3 simple steering behaviors*



Chapter 2.3 | Abstracting the Human Systems

As stated in Chapter 2.2, occupied architectural space can be broken down into its components of active human agents and passive (or active in the case of dynamic spaces) architectural objects. This chapter will focus on establishing the simulation model for the active human agents by building upon Craig Reynolds’ *boids* model. Within Reynolds’ paper “Steering Behaviors For Autonomous Characters,” he refers to autonomous agents as “idealized vehicles” or “boids,” and described their motion as three layers: *action selection*, *steering*, and *locomotion*.^[1] (Fig. 2.3.1)

Action selection is the process of selecting the action depending on the calculated goal of the vehicle.

Steering is the calculated forces that is applied to the vehicle after deciding the action.

Locomotion is the method of how the vehicles move to their goals.

Naturally, human agent behaviors are a bit more complex than vehicles. The three layers established by Reynolds only described motion; therefore, it is vital to build upon this to describe the other attributes of human agents. While these three layers present an idea of how the agents can move around the environment, the question still remains as to how the agents obtain information and decide on their tasks. Considering these functional requirements, it is then possible to simplify human behaviors into three main systems: *the sensory system*, *the decision logic*, and *the pathfinding*.

The sensory system allows the agents to see their surroundings.

The decision logic allows the agents to choose a goal based on what they see.

The pathfinding allows the agents to find a way to the goal that they chose.

These systems function in a hierarchal sense in which one system feeds into the next. The agents would utilize the sensory system to obtain information from the environment. It would then use this information within the decision logic to decide which entity from the environment interests them. It would then utilize the pathfinding system to navigate to the entity of interest. By acknowledging this, one can begin building the systems by looking from the bottom level up. (Fig. 2.3.2 - 3)

¹ Reynolds, “Steering Behaviors For Autonomous Characters.”

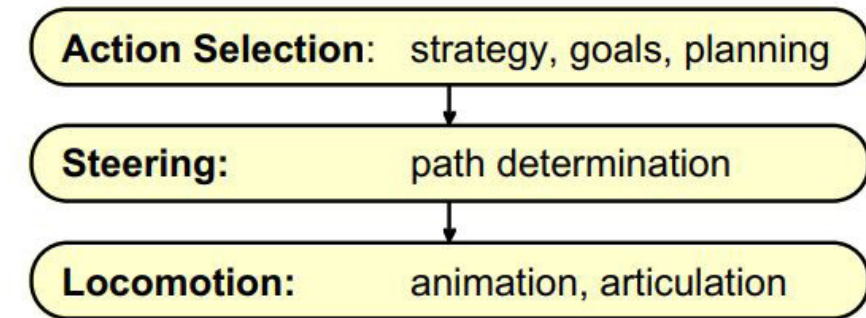


Figure 2.3.1 Three hierarchical layers of motion behaviors

Craig Reynold’s Flocking model describes these complex patterns with 3 simple steps along with 3 simple steering behaviours.

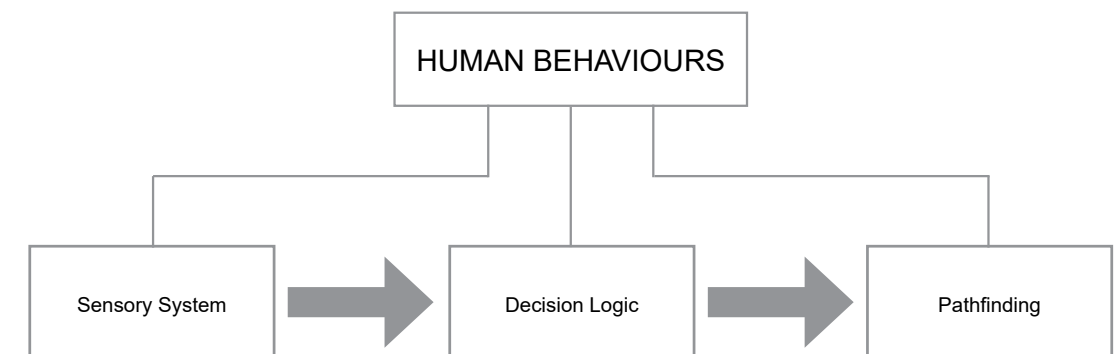


Figure 2.3.2 Three hierarchical layers of human behaviors

Human behaviors can be simplified into 3 main systems: the Sensory System, the Decision Logic, and the Pathfinding. Each system requires the consideration of their human counterparts to set up.

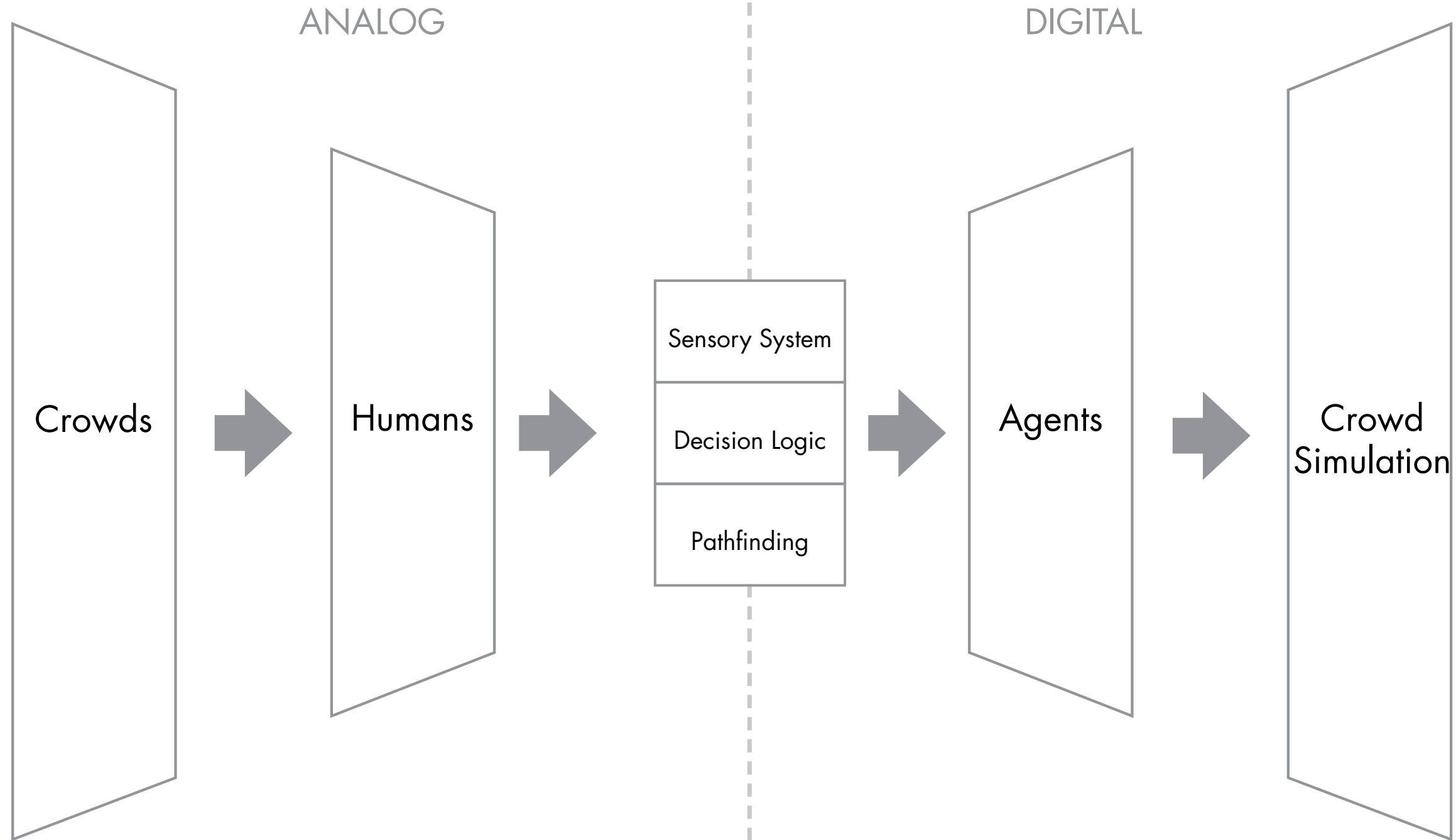


Figure 2.3.3 *Analog to Digital*

By breaking down the complex analog system of human crowds into its fundamental elements and rules, they become simple enough to be translated and approximated by machine logic. Then by reintroducing these agents back into the virtual space, it becomes possible to rebuild the crowd system from the bottom up.

Pathfinding

The Pathfinding system determines how these agents move within the simulation. If they decide to go to a door, how do they walk over? If there are other people in the way, how do they avoid them? If there are walls, how do they not bump into them? While this system is the most technical and dynamic of the three—as it deals with real-time calculations due to environmental inputs—it is also the most well-established, as it can be described relatively intuitively by Craig Reynolds’ description of the three layers of motion: *action selection, steering, and locomotion*. These steps, much like the human systems defined earlier in this chapter, are hierarchal and feed into the next.

Locomotion

The bottom level contains Locomotion, which is the method by which the vehicles move to their goals. There are many ways to calculate this in machine logic, but relating back to the physical world, its kinematics can be broken down into *position, velocity, and acceleration*.^[2] In programming, these can be expressed by vectors, which can be defined as entities that have both magnitude and direction.^[3] These vectors can be thought of as the difference between two points, and can be used to describe the location and movement of autonomous agents within the virtual space.^[4] (Fig. 2.3.4 - 6) This virtual space, much like the physical world, can be understood in the three spatial dimensions of x, y, and z. As such, these vectors can be broken down in the same way to describe the agent’s relation within virtual space. By familiarizing oneself with these vectors, and their relations within these spatial dimensions, one can begin to calculate and manipulate these vectors to establish a variety of possible functions.^[5] (Fig. 2.3.7 - 11)

From this model, it is then possible to describe *position* with a vector that determines a point relative to the origin, whereas *velocity* can be described with a vector that determines a point relative to *position* (the rate of change of *position*), and *acceleration* can be described with a vector that determines a point relative to *velocity* (the rate of change of *velocity*). To utilize these vectors in this way, one can add them into one another to calculate the resulting position of the agent. In simpler terms, the software will take the following steps every frame to achieve motion:

1. Calculate Acceleration
2. Add Acceleration to Velocity
3. Add Velocity to Position
4. Draw object at Position.

2 “Displacement, Velocity, Acceleration,” NASA, accessed August 4, 2019, <https://www.grc.nasa.gov/www/k-12/airplane/disvelac.html>.
 3 “Scalars and Vectors,” NASA, accessed August 4, 2019, <https://www.grc.nasa.gov/www/k-12/airplane/vectors.html>.
 4 Daniel Shiffman, “Chapter 1. Vectors,” in *The Nature of Code* (United States: D. Shiffman, 2012), accessed October 17, 2019, <https://natureofcode.com/book/chapter-1-vectors/>.
 5 Shiffman, “Chapter 1. Vectors.”

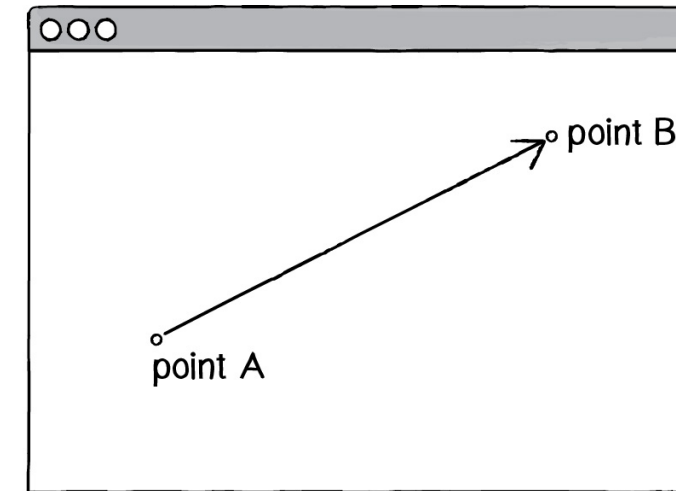


Figure 2.3.4 Vectors can be thought of as the difference between 2 points.

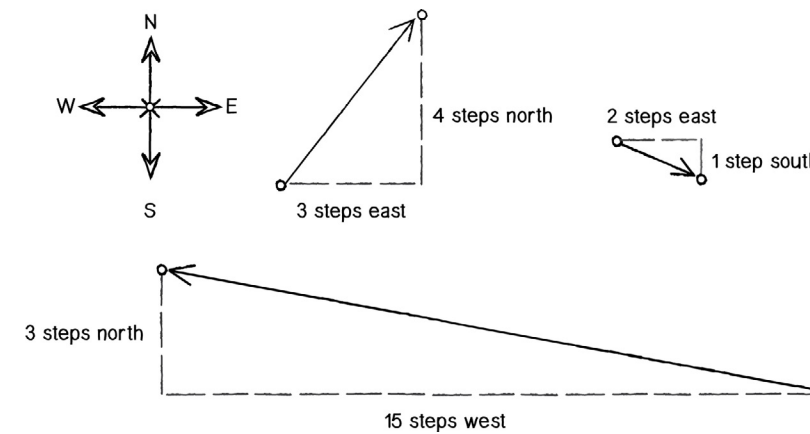


Figure 2.3.5 Vectors can be described by 2 scalar variables.

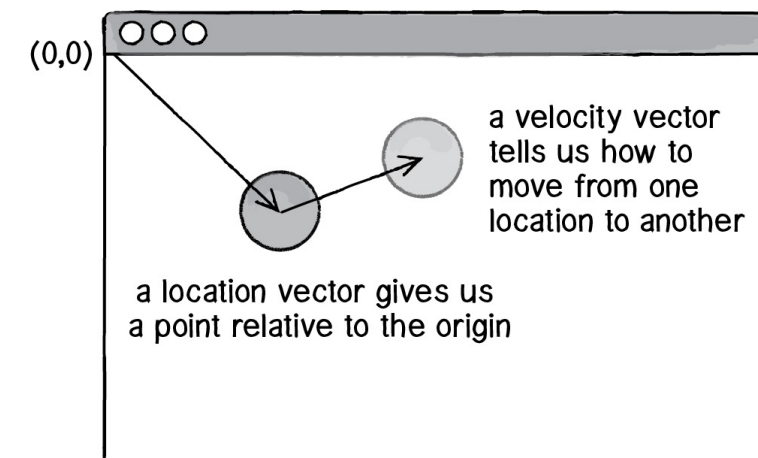


Figure 2.3.6 Velocity vector updates position
 This figure shows how adding a velocity vector to position vector calculates a new position vector for the agent to move to.

- add() — add vectors
- sub() — subtract vectors
- mult() — scale the vector with multiplication
- div() — scale the vector with division
- mag() — calculate the magnitude of a vector
- setMag() - set the magnitude of a vector
- normalize() — normalize the vector to a unit length of 1
- limit() — limit the magnitude of a vector
- heading() — the 2D heading of a vector expressed as an angle
- rotate() — rotate a 2D vector by an angle
- lerp() — linear interpolate to another vector
- dist() — the Euclidean distance between two vectors (considered as points)
- angleBetween() — find the angle between two vectors
- dot() — the dot product of two vectors
- cross() — the cross product of two vectors (only relevant in three dimensions)
- random2D() - make a random 2D vector
- random3D() - make a random 3D vector

Figure 2.3.7 A list of Vector operations that can be used within Processing.

These operations can be defined by simple words within processing, but It is important to understand how these operations work to understand how to utilize them within the simulation.

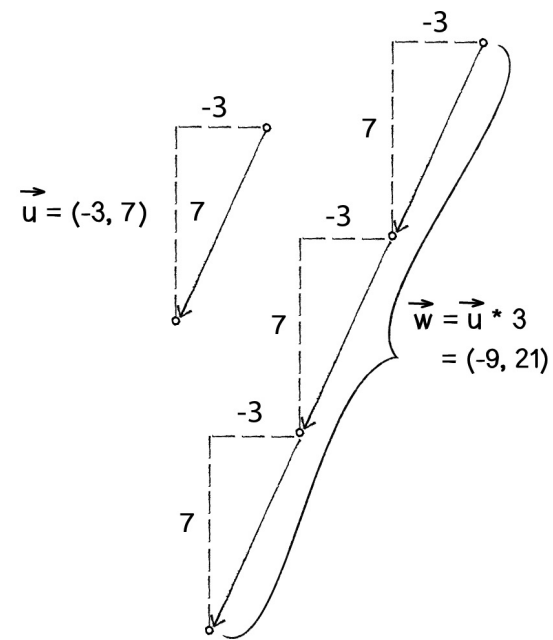


Figure 2.3.8 Vector Multiplication

It can be seen from this figure that multiplying a vector keeps its direction, but increases its length. As such, vector multiplication can be used as a way to scale the magnitude of vectors.

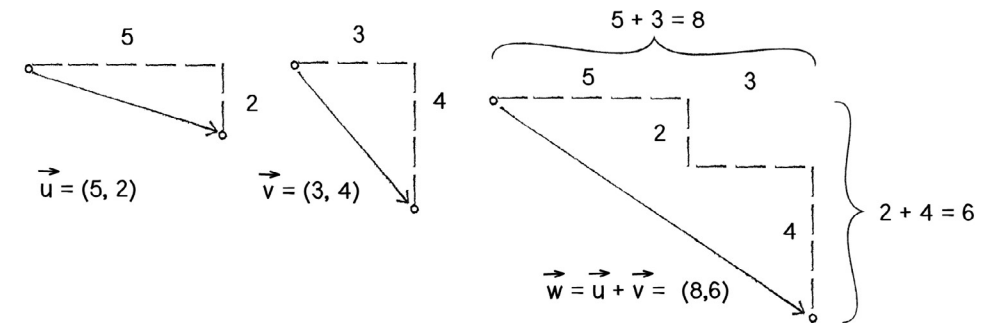


Figure 2.3.9 Vector Addition

adding two vectors together results in a new vector location that is the result of going along each vector individually. As such this can be utilized to update position by adding additional vector forces to it.

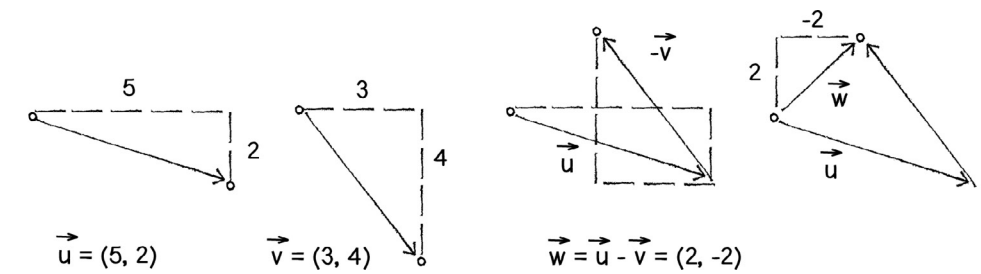


Figure 2.3.10 Vector Subtraction

Vector subtraction can be used to calculate the distance between two points, which in turn have many uses within a crowd simulation, such as determining how far away an object is to the agent or calculating a vector that points from the agent to its goal.

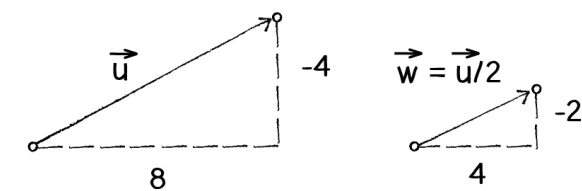


Figure 2.3.11 Vector Division

Dividing a vector by a scalar value is similar to multiplying it, except its length is decreased instead of increased. Dividing a vector by itself will normalize a vector and result in a magnitude of 1. This normalized vector can then be scaled by multiplying the vector by a scalar variable in order to control how big a vector can become.

When the software runs, it is essentially re-rendering an object at the point of its location every time the frame is refreshed. This means a velocity of 0 would entail no change in location from the previous frame, and an acceleration of 0 would mean constant velocity. **(Fig. 2.3.12)** This *trickledown* effect produces non-linearity within vehicle movements, producing greater complexity within the system. While this non-linearity can provide life-like movements within the simulation, there is still a crucial aspect that needs to be considered for a crowd simulation. In the physical world, there are limits to forces- a ball does not instantly fall to the ground, vehicles do not instantly go from 0 to 100km/h, and humans do not instantly get from one location to the next. Therefore, an additional scalar variable is needed to represent that maximum speed.

Although humans can move at a large range of speeds, they generally have a preferred speed that varies depending on personal factors such as value of time,^[6] energetics,^[7] biomechanics,^[8] visual flow,^[9] and exercise,^[10] which in turn can be influenced by environmental or social factors such as temperature, “population size, economic conditions, and cultural values.”^[11] This preferred speed can then be broken down into two main modes of locomotion: walking and running.^[12] Walking is the slower of the two forms, with speeds ranging from 0.3 to 2.0 m/s, whereas running is the faster state with speeds ranging from 2.0 to 5.0 m/s.^[13] Of these two modes, walking is the generally more preferred mode due to its lower energy usage,^[14] with running generally reserved for urgency and exercise. Considering this, it is then possible to utilize these general statistics as a starting point for setting the maximum speed to simulate a believable human crowd. While these various factors can all be utilized to calculate this scalar value, it would increase the complexity of the model substantially. As such, simply predefining an approximate number at this stage should suffice for conveying human motion. Translating this into machine logic is then a simple case of normalizing the velocity vector variable and multiplying it by the maximum speed scalar variable.

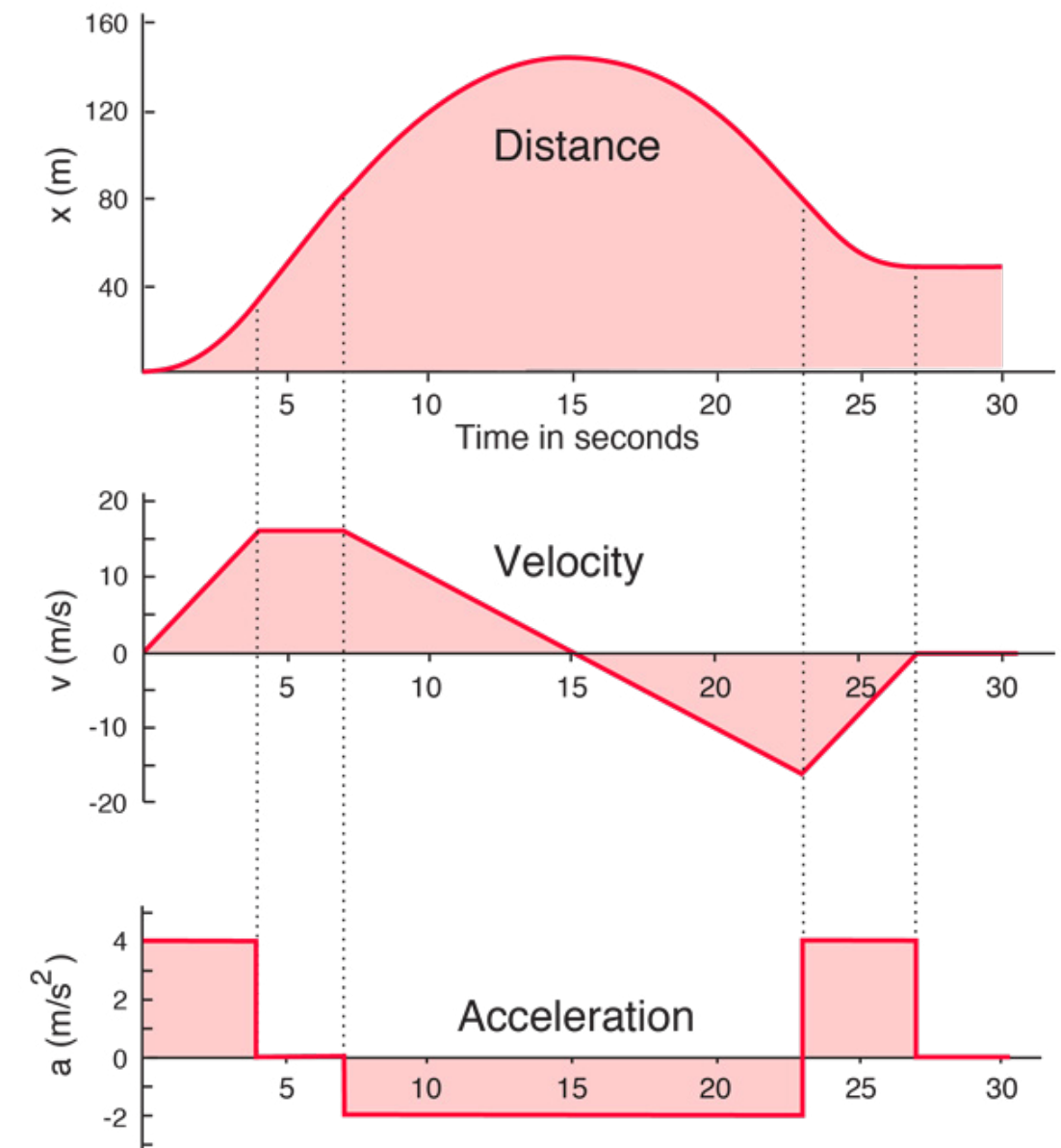


Figure 2.3.12 The relationship between distance, velocity, and acceleration

6 Mark Wardman, “Public Transport Values of Time,” *Institute of Transport Studies, University of Leeds, Working Paper 564* (2001): 1-56, accessed October 17, 2019, http://eprints.whiterose.ac.uk/2062/1/ITS37_WP564_uploadable.pdf.

7 R. Mcneill Alexander, “Energetics and Optimization of Human Walking and Running: The 2000 Raymond Pearl Memorial Lecture,” *American Journal of Human Biology* 14, no. 5 (2002): 641-48, doi:10.1002/ajhb.10067.

8 J. Maxwell Donelan, Rodger Kram, and Arthur D. Kuo, “Mechanical Work for Step-to-Step Transitions Is a Major Determinant of the Metabolic Cost of Human Walking,” *The Journal of Experimental Biology* 205 (August 2002): 3717-3727.

9 Betty J. Mohler et al., “Visual Flow Influences Gait Transition Speed and Preferred Walking Speed,” *Experimental Brain Research* 181, no. 2 (2007): 221-228, <https://doi.org/10.1007/s00221-007-0917-0>.

10 Catrine Tudor-Locke and David R Bassett, “How Many Steps/Day Are Enough?,” *Sports Medicine* 34, no. 1 (2004): 1-8, <https://doi.org/10.2165/00007256-200434010-00001>.

11 Robert V. Levine and Ara Norenzayan, “The Pace of Life in 31 Countries,” *Journal of Cross-Cultural Psychology* 30, no. 2 (1999): 201, doi:10.1177/0022022199030002003.

12 Mohler et al., “Visual Flow Influences Gait Transition Speed and Preferred Walking Speed,” 221-222.

13 A. E. Minetti, “The three modes of terrestrial locomotion,” In *Biomechanics and Biology of Movement*, ed. Benno Maurus Nigg, Brian R. MacIntosh, and Joachim Mester (Human Kinetics, 2000), 69-72.

14 Alexander, “Energetics and Optimization of Human Walking and Running: The 2000 Raymond Pearl Memorial Lecture,” 641.

Steering

The next level up is *steering*, which is the calculated force that is applied to the vehicle to tell it how it should move. The key word here is *force*, which in traditional Newtonian motion can be defined as a vector that causes an object with mass to accelerate.^[15] Looking back at the steps in *locomotion*, it can be seen that step one requires a vector that represents *acceleration* to determine to *location* of the agent, therefore in this stage, it is imperative to find a way to calculate that *acceleration*.

Shiffman relates this concept back to the physical world by investigating Newton's second law of motion—which states that *Force = mass x acceleration* ()—and then solving for acceleration, producing the formula: *acceleration = Force / Mass*. From this, Shiffman remarks, “Now, in the world of Processing, what is mass anyway? Aren't we dealing with pixels? To start in a simpler place, let's say that in our pretend pixel world, all of our objects have a mass equal to 1. $F / 1 = F$. And so: $A = F$.”^[16] This makes sense in the context of a crowd simulation, as the calculated force is a translation of behavior, and not a relation to the object's mass—such as gravity. What this means is that to calculate the acceleration vector in locomotion, one must first calculate a steering force that can be added to the acceleration vector.

This steering force can be an accumulation of many forces within the system and can represent many things, ranging from wind to friction to gravity, etc. depending on the simulation. While these forces can seem complex, Craig Reynolds developed a simple formula to calculate the steering force within these models:^[17] (Fig. 2.3.13 - 14)

$$\text{Steering Force} = \text{Desired Velocity} - \text{Current Velocity}$$

This formula then requires two variables: the *desired velocity* and the *current velocity*. The *current velocity* can already be derived from the last frame of locomotion; therefore, the critical part of this calculation is determining the *desired velocity*. This is a vector that points from the agent's current position to the target position, therefore it can be calculated by the formula:^[18] (Fig. 2.3.15)

$$\text{Desired velocity} = \text{Target Position} - \text{Agent Position}$$

Vectors as stated earlier are entities that have both magnitude and direction. With these two equations in place, the *direction* of the steering force can be calculated. One thing to keep in mind, however, is the *magnitude* of such forces. Much like how humans do not instantly get from one location to the next, they also do not instantly turn from one direction to another. As such, this stage will require

15 “Newton's Second Law,” NASA, accessed August 4, 2019, <https://www.grc.nasa.gov/www/k-12/airplane/newton2.html>.
 16 Daniel Shiffman, “Chapter 2. Forces,” in *The Nature of Code* (United States: D. Shiffman, 2012), accessed October 17, 2019, <https://natureofcode.com/book/chapter-2-forces/>.
 17 Shiffman, “Chapter 6. Autonomous Agents.”
 18 Shiffman, “Chapter 6. Autonomous Agents.”

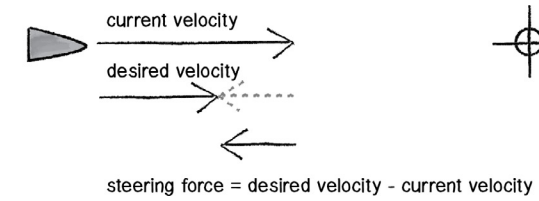


Figure 2.3.13 The calculated opposing Steering force, when added to current velocity, will bring it closer to desired velocity

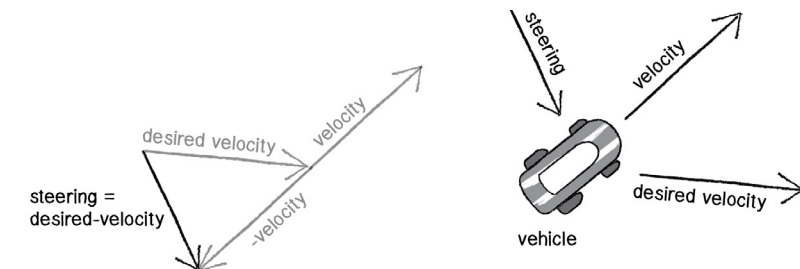


Figure 2.3.14 The steering force is pushing down on the vehicle to steer it towards desired velocity

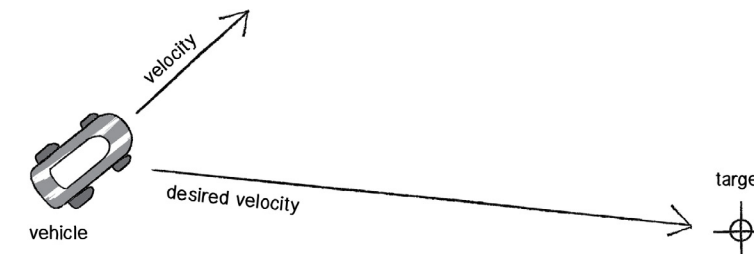


Figure 2.3.15 Desired velocity can be calculated by obtaining the Vector distance between the vehicle position and agent position

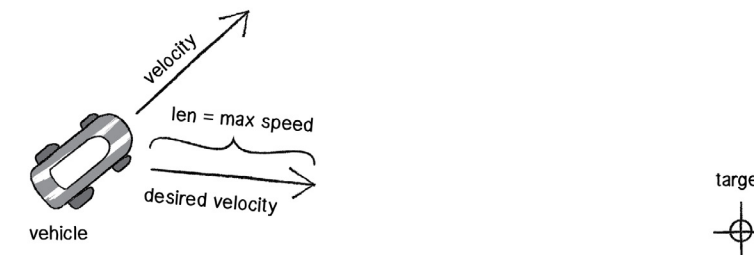


Figure 2.3.16 We must then limit this distance vector to obtain our desired velocity so our vehical, or human, can't move too fast

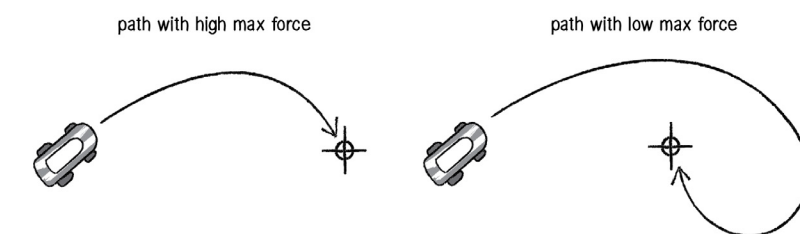


Figure 2.3.17 How Max force can affect radius

a scalar variable that defines maximum force to limit the forces that steer these agents. (Fig. 2.3.16 - 17) This, along with the maximum speed defined from the last stage, creates the limits within the system that helps define the agents as humans within a crowd.

Setting this maximum force is not as straightforward as the maximum speed; in reality, humans in normal locomotion are steered by intent rather than forces. What this means is that this way of fabricating locomotion by calculating the steering force is simply a way of simulating the intent of the agents among the simulation; a way of translating human intent into numbers. These forces may exceed human limits if it represented things such as hurricane winds, or falling due to gravity, but if this model is utilizing forces as a representation for human steering, then it is logical to limit them as such. In doing so, acquiring this maximum force-defining number becomes a matter of trial and error to find a variable that portrays realistic movements of human steering.

Once these limits are set, one can begin manipulating the magnitudes of these vectors to create both attraction and repulsion forces of varying strength. In doing so, it becomes possible to create algorithms for simple actions such as seek and flee, to more complex actions such as collision avoidance, object avoidance, path following, arrival behaviors, as well as the three rules of flocking, all of which can be calculated into a steering force and accumulated into the current acceleration.^[19] (Fig. 2.3.18 - 20)

Action Selection

The last level of defining *pathfinding* is the *action selection*, where the agents decide what actions to take depending on their desires. These actions are ultimately what calculates the final steering force vector every frame; therefore, this layer requires the most consideration for defining the simulation model of the system. Until this point, other than the human limits, the basic calculations from locomotion and steering are to a degree universal and can be applied to many types of simulation models. (Fig. 2.3.21) Therefore, at this stage, it becomes important to determine the elements that define this system as a *crowd simulation*.

While many of the actions mentioned in the last section (steering) may pertain to crowds, it is the manner in which to apply them that matters. The particles in a water simulation might only ‘desire’ that they do not occupy the same space as another particle, therefore, they would utilize functions for *collision avoidance*; the planets within a solar system simulation might desire attraction to the other planets while keeping their current momentum, therefore, they would utilize functions such as *seek* to calculate their trajectory; the vehicles in a traffic simulation might desire to follow a path as well as not collide with other vehicles, in which case they might utilize a combination of functions for *path following* and *object avoidance*. Compared to these examples, however, the people within a crowd simulation carry another layer of complexity. People generally have goals that they want

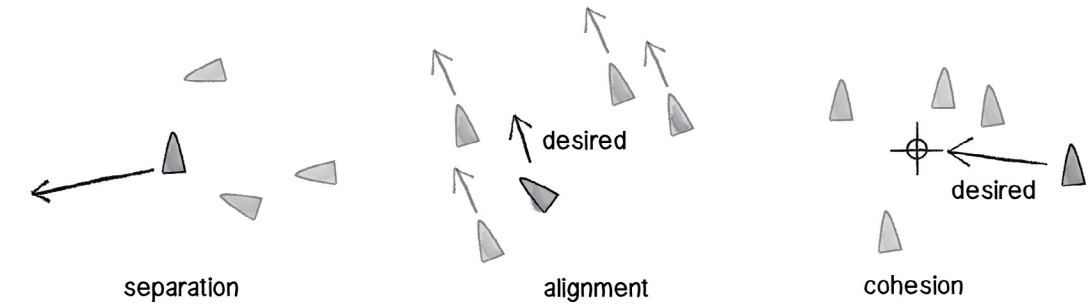


Figure 2.3.18 The 3 rules of flocking is defined by Reynolds as Separation, Alignment, and Cohesion

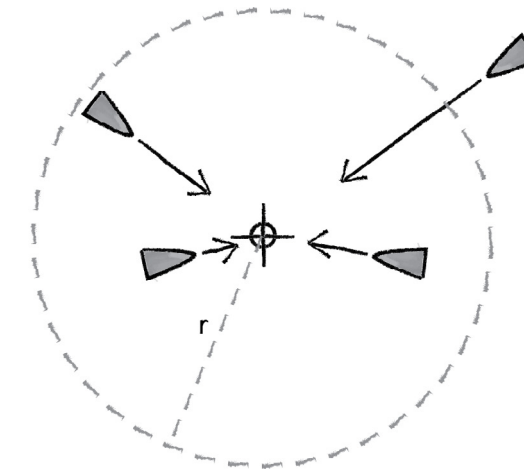


Figure 2.3.19 Arriving behavior once they get to a certain distance from target location

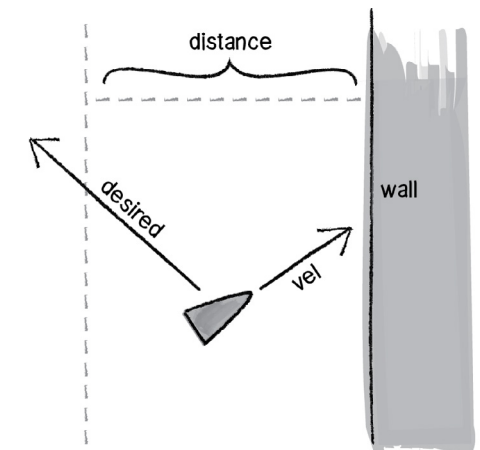


Figure 2.3.20 Avoiding walking into walls

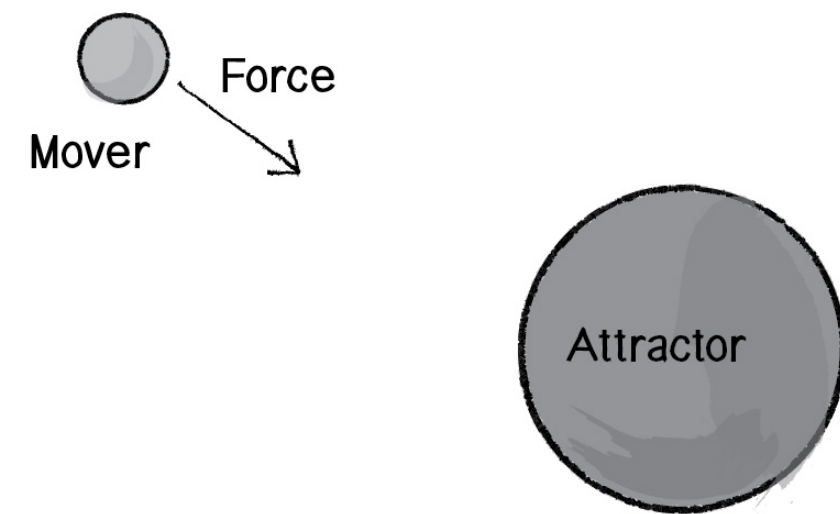


Figure 2.3.21 These same forces are useful in other types of simulations as well, such as planetary motion

19 Shiffman, "Chapter 6. Autonomous Agents."

to achieve on top of their instinctive desires, thus one would need to utilize a plethora of functions to calculate a final velocity vector that determines their movement.

To define action selection within a crowd simulation, one must first identify the individual goals and desires of the people that make up the crowd. These goals and desires will change based on the location and type of space that the person is occupying, but at this stage of prototyping, it is possible to simply define some generic desires as a starting point to approximate human behavior. From here, it is then possible to add in other desires as the conditions change and require it. These can be separated into *conscious* goals and *subconscious* desires, which can in turn be calculated separately and added to the individual acceleration vectors of the agents. At the *conscious* level, they might have goals such as reaching a destination, doing a task, meeting someone, or any combination of these, but at the *subconscious* level, these people may desire to do their task without walking into objects such as walls, furniture, or other people.

While action selection is mainly responsible for this unconscious level of obstacle interaction, both levels of desire can be influenced by factors such as stress, energy, and comfort. An example of such an influence would be how people might consciously choose to sit down if their energy level is low, but they might also subconsciously take a longer route there to avoid higher traffic areas. The correlation between these attributes and conscious goals are quite clear but becomes less obvious when dealing with subconscious desires. In order to better understand the influence of these attributes, one can investigate proxemics, which is the study of human spatial requirements, and its effects on behavior and social interactions. Edward T. Hall coined this term in 1963 in his book *The Hidden Dimensions*, in which he explores social and personal spaces and man's perception of it. Here, he divides spaces into four distinct regions, defining the interpersonal distances of man.^[20] (Fig. 2.3.22) Examining these four zones, it can be inferred that the inner zones have a higher role in the subconscious desires of object and people avoidance, whereas the outer zones have a higher role in the conscious desires of tasks and goals. The space within these two inner zones is called personal space, which can be described as a space around a person they associate as theirs. Entering such spaces often indicates familiarity, therefore, it is logical to assume that most individuals prefer to be able to control these spaces, and that in an unfamiliar crowded public realm, preserving personal space becomes an important desire.

This notion of personal space is a useful consideration not only in how it influences human behavior but also how easily it can be described by scalar variables as distances around the person. With this deduction, one can utilize these defined personal distances as a radius in which the agents can interact within the simulation. Translating this into machine logic, it can be coded that when another agent or object goes within the agent's personal space, a steering force will be calculated in the opposite direction to steer them away, or towards the other

20 Edward T. Hall, *The Hidden Dimension* (Garden City, NY: Doubleday, 1966), 107-122.

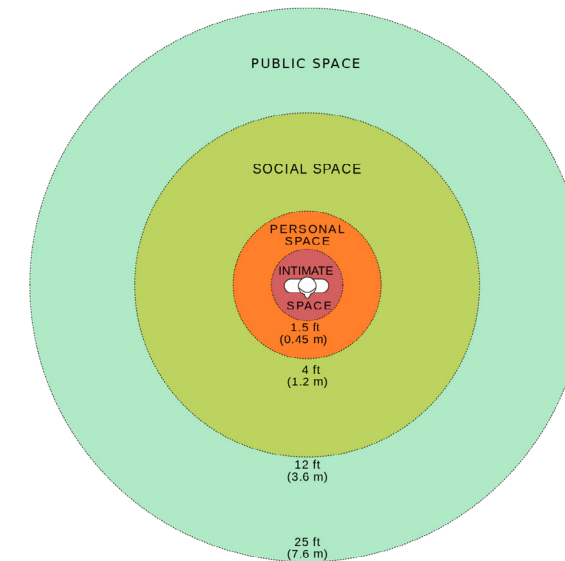


Figure 2.3.22 Edward T Hall's Interpersonal Distances of man

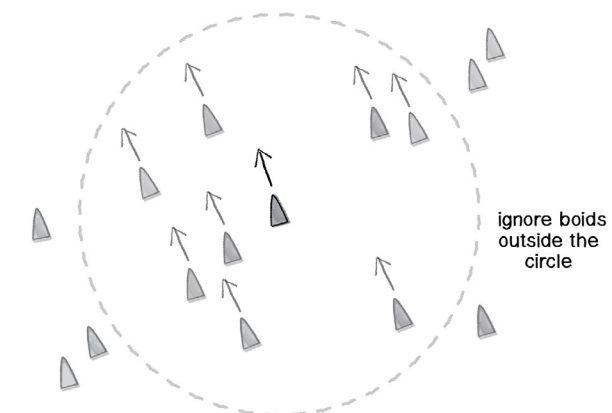


Figure 2.3.23 We can use these defined personal spaces to determine the area around the agent in which they will be affected

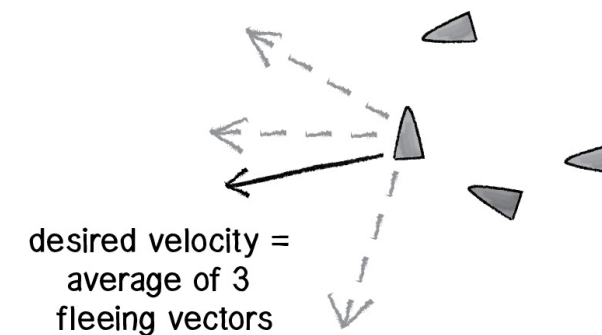


Figure 2.3.24 Utilizing Fleeing behaviour to avoid other agents that may have entered the Agent's personal space

agent to interact with them, or anything in between depending on the objective. Simply put: if distance is less than r , apply force. (Fig. 2.3.23 - 24)

This basic distance-based calculation, while simple, is enough to set a rule where agents have a higher urge to avoid something the closer they are to it, emulating a behavior in the physical world where people gradually move to avoid something they see in the distance, but move much more quickly if they're about to walk into it. This can also cover physical behaviors, such as congestion when the space around the agent becomes limited. (Fig. 2.3.25) In the future however, one can consider additional levels of calculation; instead of a direct distance-to-force correlation, this distance data—which itself can be influenced by geography and culture—can instead be used to drive attributes such as comfort and stress, which in turn, can be used to manipulate the acceleration force that drives the agents. This additional level of inference can introduce even more complex behaviors, such as leaving if it becomes too uncomfortable, or subconsciously choosing the lowest stress path towards their goal as they are moving. For this prototype, however, simple distance-based calculations should be enough to portray simple crowd movements at a macro scale.

With these three steps of Action selection, Steering, and Locomotion, one can create a basic pathfinding system. This gives the agents a means of navigation, allowing them to avoid various elements within the simulation. (Fig 2.3.26)

Crowd Density v Crowd Flow Rate

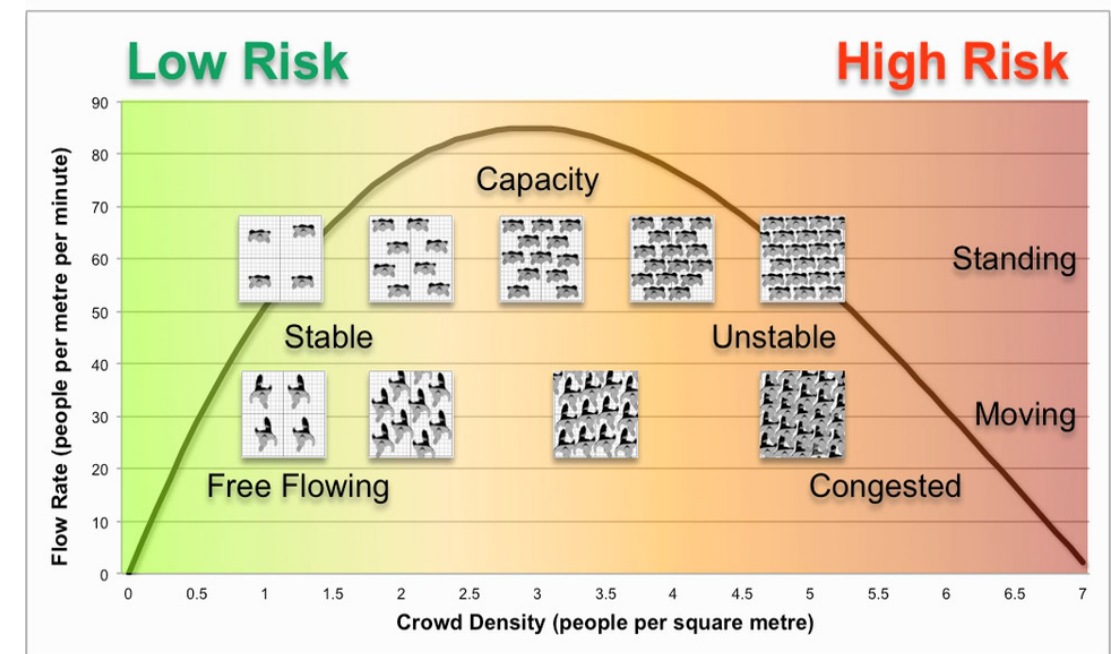


Figure 2.3.25 Crowd density vs crowd flow rate graph

This crowd density vs crowd flow rate graph shows one example of how one might establish a correlation for the personal spaces that drives the pathfinding of these agents.

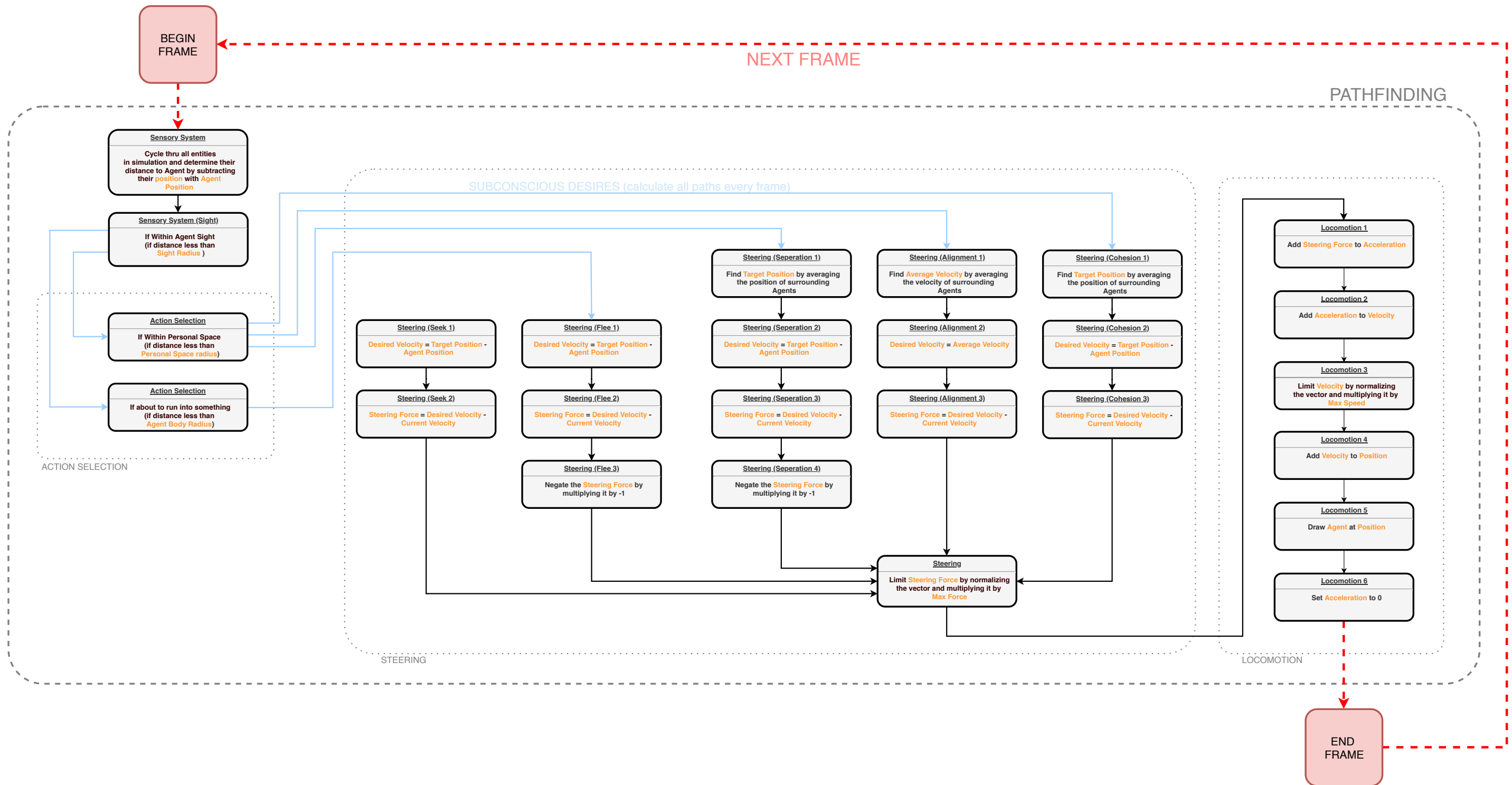


Figure 2.3.26 Pathfinding flowchart

Decision Logic

After establishing the foundation that is the *pathfinding system*, the next step is to build upon it by establishing the *decision logic system*. This system determines how the agents choose their actions within the simulation and is responsible for providing the main steering force of directing where the agents need to go. While the previous section investigated how action selection within locomotion dealt with the subconscious *in-the-moment* calculations of deciding which way to turn to avoid an upcoming obstacle, this current section will investigate how the decision logic deals with the conscious overall goal of the agent within the space. In doing so, the agents can acquire a sense of objective instead of mindlessly walking around the space.

The first step is to establish the method by which the agents can decide, which at the simplest form, can be defined by a series of logic gates within a decision matrix, composing of *yes* and *no* outcomes. These gates are created by utilizing if/else statements within the programming language and can be controlled using probability by generating a number between 0 and 100. This makes it possible to define actions based on percentages, which allows for the injection of non-uniformity with simple inputs. While this method may seem crude in simulating individual human decisions, it is important to remember the complexity that comes with human decisions and acknowledge “there are certain phenomena and events in any environment where we have to consider them as random because we simply have no better way of characterizing them.”^[21] The validity of this approach is further supported within *Design and Use of Computer Simulation Models*, where they state that “most situations in the real world have stochastic (randomly varying) properties because of real (or assumed) ignorance of details. Sometimes these properties must be modeled explicitly, but it is often sufficient to model situations as if they were deterministic by using expected values of the variables.”^[22]

Furthermore, the observation that individually can break down in larger crowds means that populations can often be broken down into percentages as well. The Canadian Census is a good example of this concept, in which the population is broken down by categories and numbers. (Fig. 2.3.27) It is shown here that in 2016, Canada had an employment rate of around 61%,^[23] with 74% of Canadians driving to work and 12% using public transit.^[24] With this, a simple city-sized traffic simulation might be created where each agent would have a 61% chance of going to work, with a 74% chance of using a car and 12% of taking transit. Even with each individual only utilizing one form of transportation, the overall simulation will behave like a collective city. This is in line with Sokolowski and C. Banks’ points, where they stated, “Random

21 John A. Sokolowski and Catherine M. Banks, *Principles of Modeling and Simulation: A Multidisciplinary Approach* (Hoboken, NJ: John Wiley, 2009), 36.
 22 James R. Emshoff and Roger L. Sisson, *Design and Use of Computer Simulation Models* (New York: MacMillan Etc., 1976), 13.
 23 “Labour Force Characteristics, Monthly, Seasonally Adjusted and Trend-Cycle, Last 5 Months,” Statistics Canada, accessed October 17, 2019, <https://www150.statcan.gc.ca/t1/tb1/en/cv.action?pid=1410028701#timeframe>.
 24 “Journey to Work, 2016 Census of Population,” Statistics Canada, November 29, 2017, accessed October 17, 2019, <https://www150.statcan.gc.ca/n1/pub/11-627-m/11-627-m2017038-eng.htm>.

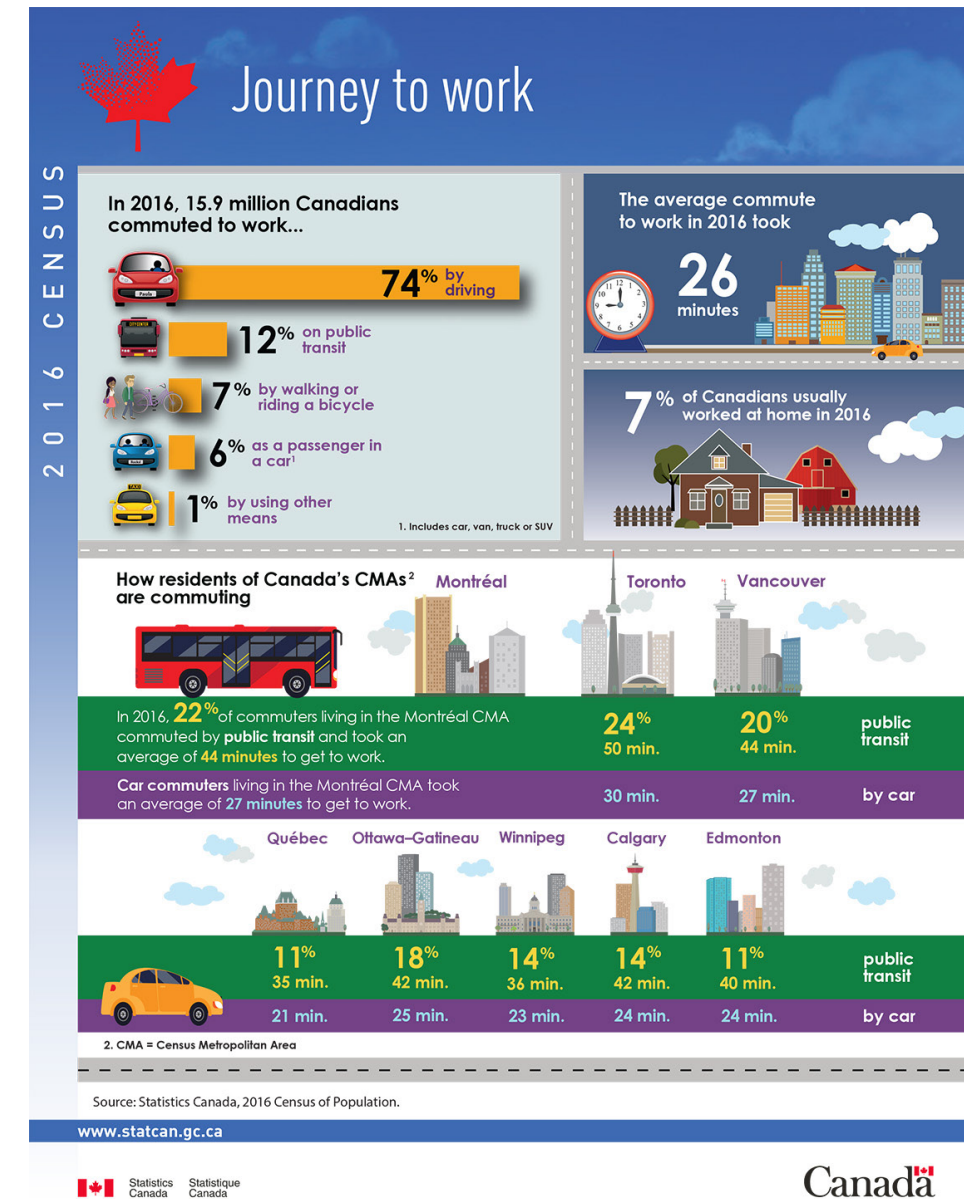


Figure 2.3.27 Canadian census info-graphic breaking down the population into percentages

event refers to occurring without a recognizable pattern. Random events can be represented by statistical distributions that allow one to simulate these seemingly random occurrences.”^[25]

This is, of course, only one example of utilizing population percentages within a crowd simulation. Different locations and cultures will need different studies, and as such, it is impossible to accommodate all scenarios. This method of using probability within decision making, however, can be utilized at varying degrees of scale and accuracy, meaning the actual numbers of probability does not need to be 100% accurate to create a believable decision within a crowd. Without doing extensive population studies, logical assumptions can still be made to supplement these variables. This means that at this stage of prototyping, it is possible to use simple logical assumptions based on previously-studied crowd behaviors as a starting point. It then becomes possible to generate probability variables and tweak it as required based on the element within the system. For instance, within a gallery, it can be assumed that larger displays will attract more attention than smaller displays; within a shopping center, it can be assumed that larger or more general stores will attract more attention than smaller or more niche stores; within a train stations, it can be assumed that trains heading to downtown will attract more people than trains going to the suburbs as a function of the time of day. While the resulting crowd might not be as accurate as ones made with specific population studies, it will still have enough nonlinearity to establish emergence within the system, allowing it to pass as a believable visualization for architectural design.

These probabilities can be further manipulated by variables such as object distances, attraction, and crowding, as well as agent energy, comfort, goals, and hunger. For example, “[c]ognitive processing theories predict that people who move quickly are less likely to find time for social responsibilities, particularly when those responsibilities involve strangers.”^[26] As such, one of these variables could be correlated to their walking speed, where a faster speed would reduce the chance of the agent interacting with others. While these additional variables are not mandatory at larger scales to create realistic crowds, the consideration of each additional variable would provide added complexity and realism at the smaller scale to each individual agent within the system. It is, however, not feasible to implement all these considerations at this stage of initial prototyping, thus it is important to be selective and focus on implementing the more notable ones—such as *object distance* and *crowding*. (Fig. 2.3.25) The notability of these two particular variables comes from the fact that they relate to agent densities. By prioritizing this on the microscale, it is possible to improve agent movements within higher traffic spaces on the macro scale. This in turn smooths movement flows within the simulation.

Taking all this into account, the decision logic becomes like a probability-based *choose your own adventure game* for the agents. If you are in a room with a door and a chair, do you go to the chair? Or do you go to the door? If you choose the door, what do you do in the next room? (Fig. 2.3.28 - 29) Each person might choose something different, and while it may be impossible to know what each person will choose, through the choices of many agents, the overall system will reflect a scenario where a certain percentage of the population may choose the chair while another

25 Sokolowski and Banks, *Principles of Modeling and Simulation: A Multidisciplinary Approach*, 49.
 26 Levine and Norenzayan, “The Pace of Life in 31 Countries,” 200.

percentage may choose the door. In an average population there will be a certain percentage of people who would choose A, another percentage would choose B, and yet another that would choose C. Depending on what these choices represent, some of them might have a higher probability of being chosen compared to others. None of these choices are right or wrong, but instead are utilized as a way of creating distributions within the crowd in order to inform the decision-making process. This is the potential of distributed agents; by simplifying these choices into percentages on the individual scale, the illusion of choice is created for the individual and the illusion of complexity is created within the collective.

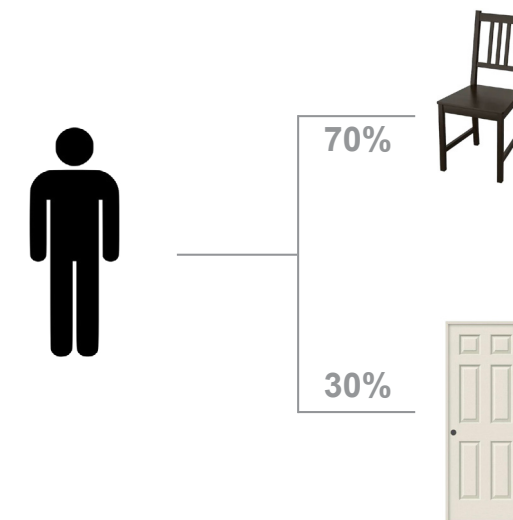


Figure 2.3.28 If you are in a room with a door and a chair, do you go to the chair? Or do you go to the door?

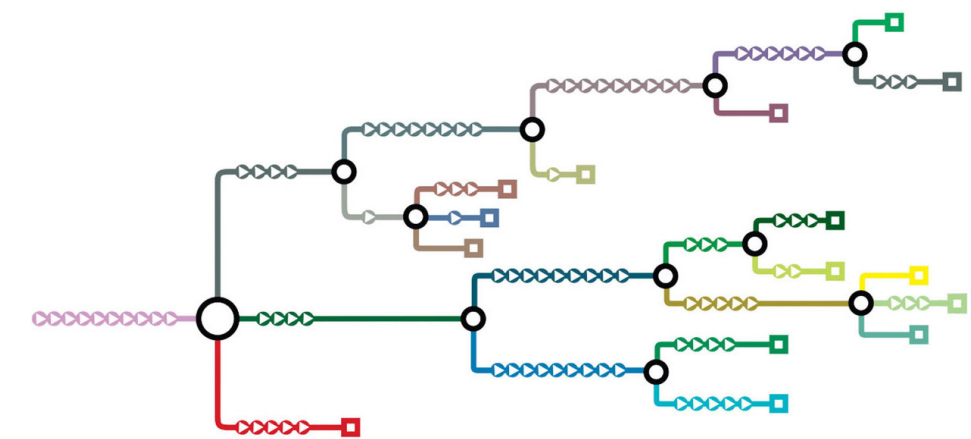


Figure 2.3.29 A decision Tree based on percentages

Sensory System

The last system to establish is the *sensory system*, which allows the agents to collect input data from the environment, similar to how humans can perceive their surroundings. This system functions at the highest level of the human systems and is responsible for generating a list of available elements to choose from within the space before deciding which task to perform within the decision logic system.

The five traditional senses of the human body consist of sight, hearing, taste, smell, and touch.^[27] Of these five senses, however, the most utilized sense in typical crowd navigation is undoubtedly sight. This is evident when one considers that while most people would have little trouble walking around with headphones on, they would find it much more challenging with their eyes closed. For this reason, this prototype will mainly utilize sight for obtaining information for the agents within their virtual environment. While there are benefits in adding other primary motivators (such as sound-based navigation for visually-impaired occupants), it is beneficial to keep the systems simple at this proof-of-concept prototyping stage. Much like other aspects, additional senses can be introduced later depending on the project and the medium being utilized for visualizing said project.

Recalling the introduction of autonomous agents in *Chapter 2.2*, one of the main concepts is that they have a *limited* ability to perceive the environment. Therefore, it is time once again to define some human limits for the system. A quick investigation reveals that “the visual field of a normal human eye measures (from point of fixation) 100 degrees temporally, 60 nasally, 75 superiorly and 60 inferiorly. Binocular (using both eyes) visual fields are approximately 200 degrees wide and 135 degrees tall, with a region of binocular overlap that is 120 degrees wide.”^[28] (Fig. 2.3.30 - 31) These angles can be translated to machine logic by utilizing the dot product, which allows the acquisition of the angle between any two vectors. In doing so, an *ifelse condition* can be set, where only objects within a certain angle of the agent’s line of sight will be added to the list of available elements. (Fig. 2.3.32)

Once a list of objects that the agent can see is acquired, the agent will then need to differentiate what those objects are. When an agent sees a door, they might be attracted to it, whereas when they see a wall, they might try to avoid it. This is of course highly situational to the intended purpose of the space; objects within a gallery will attract people as it can be assumed that people came to see them, whereas advertisements at a train station might have less attraction since they are not the main purpose of the space. Similarly, an emergency door might not attract people until there is an emergency, whereas a highly ornate wall might attract people even though they cannot go through it. Nevertheless, due to these varying requirements, defining categories such as walls, thresholds, people, and architectural elements becomes necessary. In

27 A. Chapanis, “Review of the Human Senses,” *Psychological Bulletin* 51, no. 1 (01, 1954): 100-101, doi:http://dx.doi.org.proxy.lib.uwaterloo.ca/10.1037/h0050962.
 28 Gislin Dagnelie, *Visual Prosthetics: Physiology, Bioengineering and Rehabilitation* (New York: Springer, 2011), 398.

doing so, the agents will be able to differentiate between the type of object they are seeing and calculate an action accordingly. The agent will undoubtedly see many elements at once, therefore, the ability to distinguish between these objects will allow it to better choose which element they want to interact with.

The next step is to then identify and categorize the various elements within the simulation system. Once this is done, individual IDs can be assigned to each class of entities, where other descriptive variables can be assigned to further customize these elements depending on their functionality within the space. These objects will require customization for each design instance as a function of the purpose of the space, which will be discussed further in the next chapter (2.4) of this thesis.

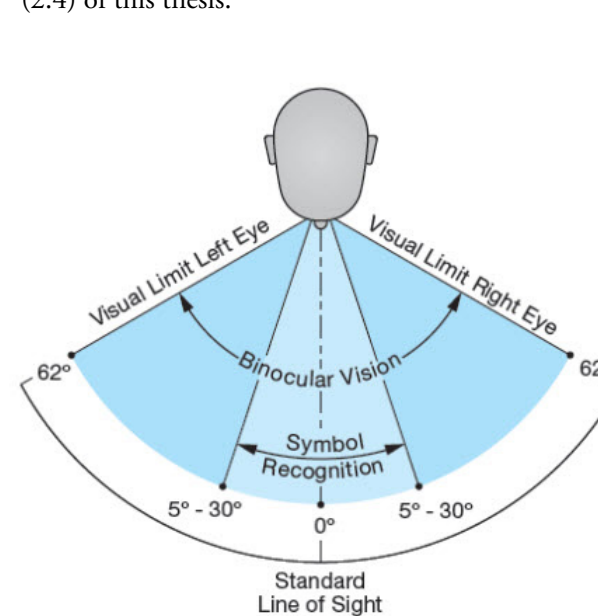


Figure 2.3.30 Human Visual Limit- Top View

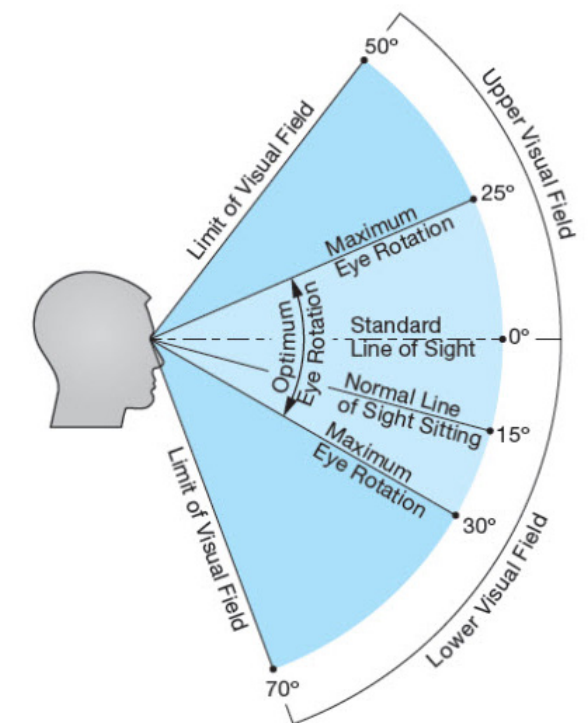


Figure 2.3.31 Human Visual Limit- Side View

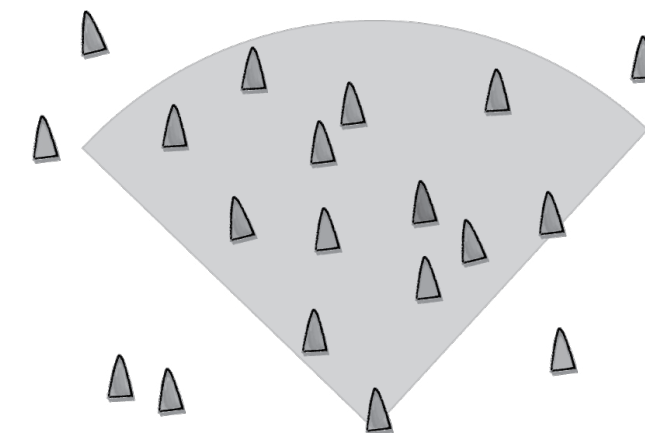


Figure 2.3.32 Sensory limit within simulation

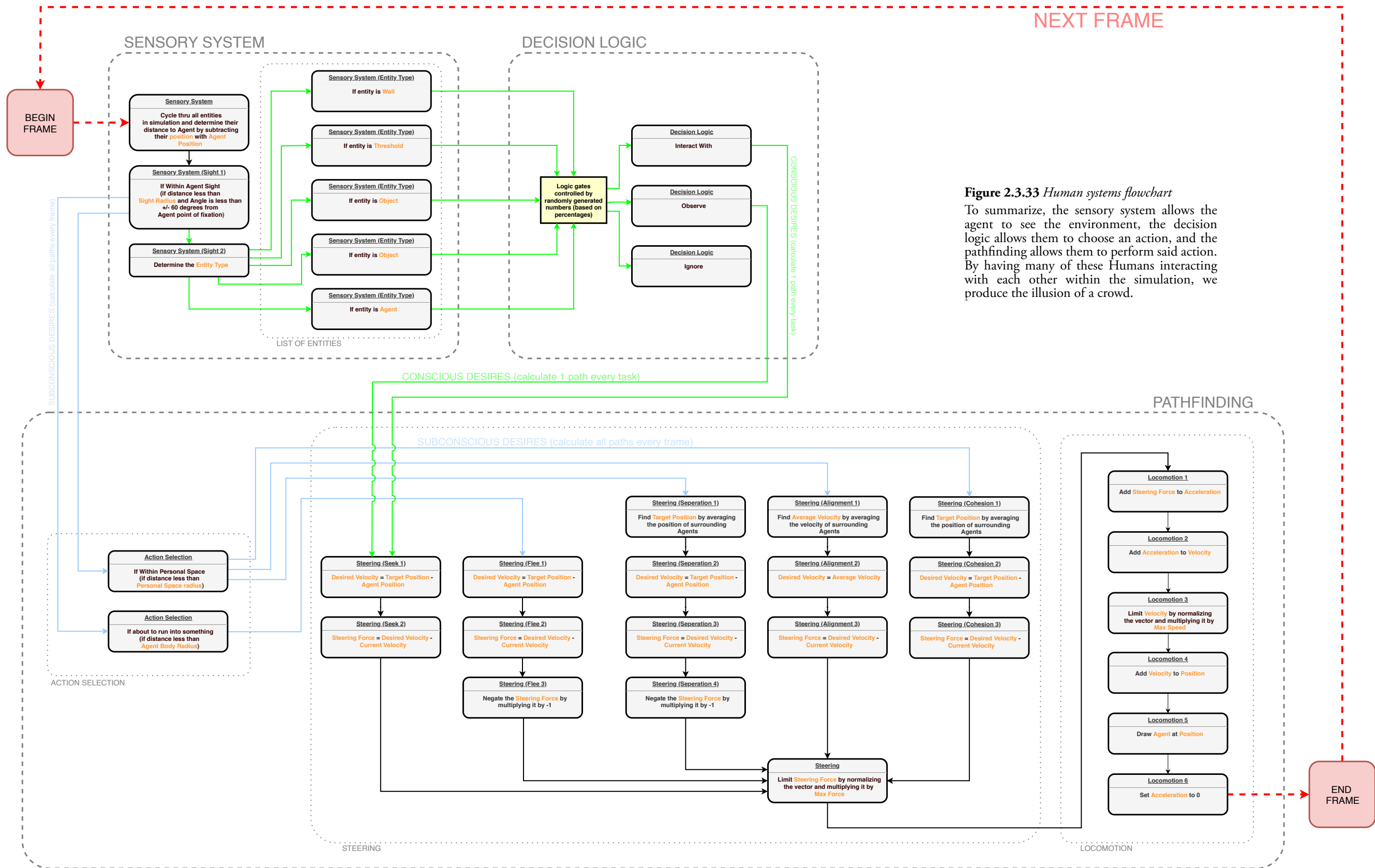


Figure 2.3.33 Human systems flowchart
 To summarize, the sensory system allows the agent to see the environment, the decision logic allows them to choose an action, and the pathfinding allows them to perform said action. By having many of these Humans interacting with each other within the simulation, we produce the illusion of a crowd.

Chapter 2.4 | Spatial Functions

With the emergence of interactive and dynamic architecture, the occupants are no longer the only possible dynamic elements within the space. The human systems model that was established in the previous chapter (2.3) may have been adequate for visualizing conventional static spaces, but would not suffice in fully visualizing dynamic spaces, where elements of the space can also change position or level of attraction. Since these interactive spaces are dynamic, one must account for this and find a way to translate these dynamics into a logic that the machine can understand. As such, much like for human behaviors in the last chapter (2.3), the goal of this chapter (2.4) is to unpack the methodologies for simulating these dynamic spaces and how they can function within the virtual space.

As mentioned in *Chapter 1.1*, dynamic spaces can utilize both simple passive elements such as water and sand features, as well as more complex computational elements such as elevators, motorized louvers, and sensors. While these spaces do not necessarily require computation and mechatronics to be dynamic, the integration of such technologies greatly increases the possibilities of spatial utilities within the space. These systems can incorporate many elements with sensors and motors to create features that can respond to natural forces, occupants, as well as operate on repetitive patterns. These features can come in the form of projections on walls to motorized doors to raising floors and ceilings, or anything else that designers can dream of, all of which has the potential to communicate with one another, and at a multitude of scales ranging from a single system imbedded within a room to networks of subsystems that can span entire cities.

The typology of these elements can be very diverse, and is becoming even more so with the addition of data-driven computational systems within infrastructure. The problem with this intrinsic diversity is that it causes complications in simulating these objects, as the variety of functions makes it challenging to develop a singular system that works for every scenario. Since each object can potentially have a different function, the software that controls them in the physical world would need to be customized and tailored to each specific instance, meaning that this logic would then need to be recreated within the simulation model depending on the functionality and utility of the object. As such, one cannot use the same code for every element, but instead must create a generic model and tailor to the functionality of the object, the typology of the space, and the specific scenario of the simulation. While it is true that this might make it more challenging to find a universal system that works for every scenario, one can overcome this by simplifying its processes into parts. Much like the process of establishing the human systems in the last chapter (2.3), one can create a framework where functionalities can be added depending on the typology and functionality of the space and its elements.

A basic understanding of these systems is required to create this framework—specifically on how they work and what these resulting spaces may entail. In Mike Crang and Stephen Graham’s article, “Sentient Cities Ambient intelligence and the politics of urban space,” they talk about three different typologies in which an urban environment can become automated through ubiquitous computing systems.^[1]

Augmenting space relies on the fact that the existing environment has already been saturated with information. Computing systems can utilize this physical information using sensors and tracking to overlay new digital media on top of the existing structures. This allows users to both see the physical world as well as the dynamic graphical information of the virtual world. This produces a reactive environment where emphasis is placed on the user’s activity.

Enacting space relies on the fact that computation inhabits everything around us, ranging from the things we carry on our bodies, to the cars on the streets, to the infrastructures of our cities. Unlike *augmenting space*, which emphasizes the user’s activity, this approach further utilizes intermediary processes, which reallocates agency back to the environment. This allows the computer system to suggest through the interaction of space or the display of data.

Transducing space relies on the digitalization and identification of people, where the layering and cross-referencing of identities allow the system to form a technological consciousness through the automation of data without cognitive inputs. This type of space can recognize its occupants and allow for autonomous tasks but comes at the cost of user awareness and user agency.

It is evident that these approaches all utilize data in some form. Environments have always been saturated with information in the forms of signage and the existence of occupants and objects, but it has only been recently that technology has begun to utilize this information by converting it into digital data with various sensors, cameras, and machine vision. This translation of analog-to-digital allows us to blur the boundary between the physical and the virtual, which not only provides greater flexibility in data utilization but also allows us to redefine spatial functionalities by creating homogeneous spaces of technological integration. By manipulating this data in different ways, it is possible to influence the distribution of agency within the space, which can in turn affect spatial operation in unforeseen ways.

¹ Mike Crang and Stephen Graham, “Sentient Cities Ambient Intelligence and the Politics of Urban Space,” *Information Communication and Society* 10, no. 6 (2007): 792–794, <https://doi.org/10.1080/13691180701750991>.

This redistribution of agency presents the concept of architectural consciousness, which can result from emergence introducing unpredictability within the architectural space. It has been shown that it does not take much to infer a sense of consciousness in humans, as the human brain has been wired to see patterns and relate to them. (Fig. 2.4.1) As such, when a space becomes unpredictable, humans instinctively try to create their own patterns from what they infer from the environment. This phenomenon can be seen in the installation “Fearful Symmetry” by Ruairi Glynn, which consists of a moving light that interacts with the public using sensors.^[2] (Fig. 2.4.3) Through intermediary processes between the human interactions and the movement of light, the illusion of personality is given to the object, blurring the line between the conscious humans and the unconscious objects. In Fox’s words, “as we embrace a world in which the lines between the physical and the digital are increasingly blurred, we see a maturing vision for architecture that actively participates in our lives.”^[3]

If something as simple as a light can infer consciousness to the occupants, imagine the potential ramifications when these dynamic objects are connected to a network and gain the capacity to communicate with each other. Within such a space, the boundary between the occupants and the architecture break down as agency is exchanged through interaction. Not only do humans have an identity, but the objects have an identity as well. Since everything is connected, the system has the capacity to know not only where the user is but also where the object is. Because of this, the system can catalog datasets of *user identity* as well as *object identity*. It is then possible to consider the implications of a *collective identity*, where a database of memory can track the history of every object and user within the system. (Fig. 2.4.2)

As current infrastructures become increasingly intelligent, new considerations for identity as well as agency becomes increasingly relevant. This city of distributed intelligence then becomes the container of the identities of both human and objects alike. While not all systems will be this extreme, it is important to accommodate for this within the simulation model to maximize the variation that can be supported by the framework. A sensory based system will function differently than an identity-based system, as such it becomes important to consider these aspects when creating the overall system, as they have much impact on the space’s overall function.

This idea of a collective identity also presents an interesting proposition in replicating these results within this simulation. Having a database of history for not only the objects, but the users as well, means that the physical world begins to operate much like the virtual world, which allows a more direct approach to simulating these spaces. By utilizing these identities to track, categorize, and organize the various entities within the space, a system of interactions can be created within the simulation, which further blurs the boundary of human occupants and dynamic objects. As these dynamic architectural elements become increasingly similar to the human crowds that occupy these spaces,

2 Ruairi Glynn, “Fearful Symmetry,” accessed October 18, 2019, <http://www.ruairiglynn.co.uk/portfolio/fearfulsymmetry/>.

3 Fox, *Interactive Architecture: Adaptive World*, 9.

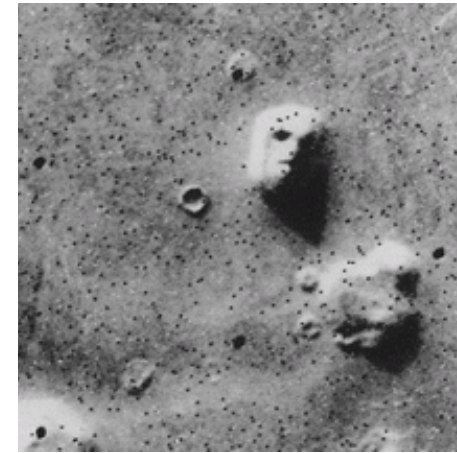


Figure 2.4.1 The face on mars

Although simply a rock formation, bears resemblance to a human face due to our tendencies of seeing patterns within nature. This is known as Pareidolia, which is also what causes us to see shapes in the clouds.

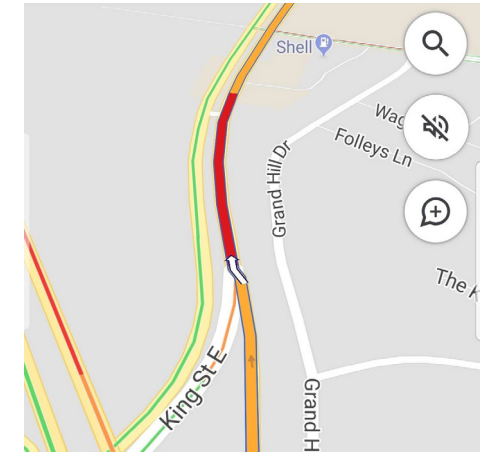


Figure 2.4.2 Google Maps

Google maps is one example of how collective identities within a system can influence physical spaces. It utilizes datasets of maps and users to generate real time traffic navigation overlaid on top of updated maps. While entirely digital, it has the capacity to influence the physical flow of traffic through its distribution of digital information to the general population. This changes people’s behaviors, which in turn allows the virtual platform of information to indirectly influence various physical platforms of the city.

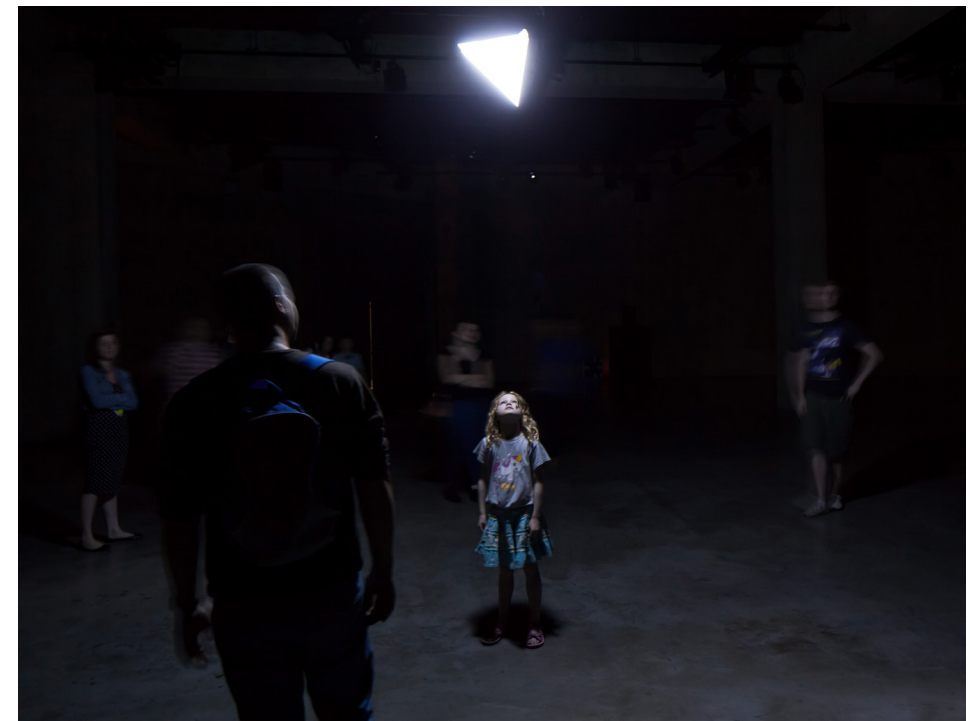


Figure 2.4.3 Fearful Symmetry by Ruairi Glynn

one can revisit the methodologies from establishing the human systems in Chapter 2.3 and adopt a similar mindset for simulating these objects. This approach allows the adoption of a framework for establishing the entities within the system regardless of what typology they may be, whether they are the human agents or the architectural objects.

Revisiting the human systems in Chapter 2.3, it can be seen that they are based on the concept of autonomous agents, which are defined by their limited ability to perceive the environment, their ability to process the information from its environment to calculate an action, and their lack of a leader. While these three rules all applied to autonomous human crowds, the third rule of lacking a leader does not necessarily need to apply to these dynamic objects, as they can be programmed to follow a leader if need be. This leaves us with the first two rules, which are the limited ability to perceive the environment and the ability to process the information to calculate an action. This can be broken down further into the basic stages of an input, processing, and an action, which allows the deduction of how these systems can be manipulated by relating it back to the three human systems that were established within Chapter 2.3.

The input, which is functionally similar to the *sensory system* of the human, determines how the object can perceive the environment and what kind of trigger necessitates an interaction from the object. This data can be obtained from various sources such as the occupants, the environment, or self-generated from algorithms, and can be stored as variables for use in the processing stage.

The processing, which is functionally similar to the *decision logic* of the human, determines how the object might utilize the data if it has to decide on an action. Within this stage, data is utilized to manipulate variables depending on the function of the object. This manipulation can be very flexible, ranging from nonexistent to space altering; where the input variable is unchanged and directly used within the output, to an algorithm where the color of a light can correlate to the number of occupants within the space.

The action, which is functionally similar to the *pathfinding* of the human, determines how the object might respond due to environmental and human contact. These are functions that utilize the processed variables to update the attributes of the object.

Simplifying this method into its basic stages helps to overcome the potential complexities that come with the varied typology of these objects. However, while this might be enough for simulating smaller spaces with a limited number of objects, for larger, more complex spaces, one will also need to account for the whole picture and investigate the organization and movement of data throughout the system. In Rob Kitchin and Martin Dodge's book,

Code/Space, they talk about software that is embedded in everyday life at four levels of activity, terming *coded objects*, *coded infrastructure*, *coded processes*, and *coded assemblage*.^[4]

Coded objects rely on software to function, which can include credit cards to flash drives to phones. This is the most personal level of activity since most objects on this level belong to the user. As such, they offer a primary source of identification as the user travels throughout the environment. Examples of this can be observed in credit cards and how they contain identification to bank accounts, or how phones can contain various forms of personal information ranging from e-mail accounts to GPS locations to microphone recordings. This is also the current main form of human computer interaction through electronic devices such as personal laptops or phones. This level of activity is important as it allows the most precise form of data collection in a distributed system, where the identification of the object is essentially the identification of the human.

Coded infrastructures are networks that can link coded objects together. They are an infrastructure that can be monitored, regulated, or interactive. Unlike coded objects, these elements are mostly built within the environment, and as such need to be integrated directly into the design of the space. As these systems are largely physical elements that need to be incorporated into spaces, elements within this layer contain the main challenge of transitioning an existing space into an intelligent interactive space.

Coded processes are the transition of data across coded infrastructure. It acts as a technological unconsciousness that drives the hardware within everyday space. This invisible layer of data is only revealed through the inference of mechanical elements or the graphical visualizations of a user interface. It is because of this technological unconsciousness that allows these complex systems to function as well as to connect to each other.

Coded assemblages are where the convergence of multiple sources of coded infrastructure are present to create a nested system that is in parallel. These systems include automation of local spaces such as hospitals, warehouses, transportation, etc. where they essentially allow the organization of local spaces within the distributed intelligence as a whole. From this, they can form almost-closed systems with minimal input and output to minimize impact from external noise.

⁴ Rob Kitchin and Martin Dodge, *Code/Space: Software and Everyday Life*, Software Studies (MIT Press, 2011), 5.

By understanding the various levels of activity that are present within these dynamic spaces, one can begin to realize the different ways these objects can function not only individually but also together within a space of distributed intelligence. With this, all the required concepts of this framework are now established. As such, it is now possible to logically deduce the best way of simulating these spaces.

A motorized window louver for example, can operate by *time of day*, where a rotation value is set by a specific time variable; by *environmental temperature*, where the louver communicates with other coded infrastructures such as temperature sensors to acquire a temperature variable; or by *occupancy number*, where the amount of people within the space can be determined either by coded infrastructure such as proximity sensors, or coded objects such as cellphones that the occupants are carrying. Within all three of these scenarios, the louver operates at the coded infrastructure level but has the option of utilizing different forms of coded processes as well as different forms of input to achieve a similar result. The dynamics of these louvers can arise as a pre-defined action or as a function of human and environmental interaction, where the output action can be as simple as defining a rotational degree variable, to additional deployment percentages, opacity, and any other attributes depending on the typology of the louver.

This example is just one possibility, but in reality the typology of these objects can be limitless, ranging from lights, to projections, to mechatronics, to furniture, all of which may react to sound, temperature, ambient lighting, occupancy numbers and identities, or be simply pre-programmed from a pattern or noise. As already discussed, this diversity poses a challenge for developing a singular method for simulating these objects, but fortunately, unlike human crowds where one must translate from analog behaviors to digital functions, these dynamic objects already operate on a code-based hierarchy.

This digital to digital translation makes this a much simpler process compared to establishing the human systems from the past chapter (2.3). There is no longer a need to interpret analog behaviors to create a new system, instead, one can replicate directly the digital logic of the software that controls the dynamic objects in the physical world. Fox remarks on this by stating, “The sensors and robotic components are now both affordable and simple enough for the design community to access; and all of the parts can be easily connected to each other. Designing interactive architecture in particular is not inventing so much as understanding what technology exists and extrapolating from it to suit an architectural vision.”^[5] What this means is that as long as one has a rough understanding of how these objects operate in the physical world, as well as considered the different urban typologies that may arise, as well as how these objects might be connected within a larger system through software, then one can simulate these objects by establishing its input, its processing, and its action. In essence, simulating these objects is less about developing a specific algorithmic model, but rather developing a methodology to understand and simplify these objects into their fundamental qualities.

5 Fox, *Interactive Architecture: Adaptive World*, 12.

Therefore, when creating such an object, the steps can be as follows:

1. Determine its functionality, whether it is a louver, a light, or a piece of furniture.
2. Assign attributes to the object that defines what it can do. A light might have an RGB variable as well as a brightness variable. A louver might have a rotational degree variable, to additional deployment percentages, opacity, and any other attributes depending on the typology of the louver. A reactive mechatronic sculpture might have a position variable along with lighting RGB and brightness, as well as a sensor that tracks the number of nearby people.
3. Consider its *input*, whether it is a human intervention such as a touch input from touchscreen, or proximity sensor, or sound; An environmental intervention such as temperature, daylighting, or air pressure; Or predefined algorithms such as a written message, or a generated pattern.
4. Consider its *processing* in relation to its attributes, whether it is a direct translation such as simply displaying the temperature on a medium, or if it utilizes intermediary processes such as generating a color and a location based on the number of occupants within a set space, or creating various profiles based on user identities within the building.
5. Consider its *action/output*, whether it is simply an object that can be moved, or if it lights up from a source, or if it projects onto a wall, or if it moves within a space, or if it deforms from interaction, or if it does all of these and more.

This same methodology can also be utilized to simulate some of the simpler elements mentioned in Chapter 1.1 such as a water fountain. Undisturbed, the fountain will have a pre-defined function to determine the state of its water texture within the environment. Upon a touch input by the occupants, however, it will utilize a function to generate ripples, in which it will output to the location of the touch input.

Chapter 2.5 | Prototyping the System Model

Up until this point, the systems that were developed in Chapters 2.3 and 2.4 are still theoretical concepts of how behavioral patterns might be translated into machine logic and how that might interact with coded infrastructure. Now that all the major components of this methodology have been considered and developed, we can begin the technical exercise of coding these concepts within a simulation system to prototype and assess the validity of this approach.

At this stage of prototyping, we can utilize Integrated Development Environments (IDEs), which are software applications that facilitate the development of software.^[1] These programs generally include various tools built on top of a text-based source code editor to help maximize programmer productivity.^[2] While there are many forms of IDEs with various degrees of technicality and flexibility, the Java-based IDE *processing* has been chosen for this prototype, due to its familiarity (since I have already used this software in previous elective courses) and its visual user interface (which better facilitates my learning process of computer programming). Because of these aspects, I can both assess the validity of these human systems to see if they work in practice as well as in principle while I familiarize myself with the fundamentals of computer programming, which will be an invaluable skill moving forward with the creation of this framework tool and in life.

Within the earlier chapters (2.3 & 2.4) we broke down the simulation into the autonomous human agents and static/dynamic architectural elements, where they can be further classified as different objects within the system. In computer science, this concept is known as object-oriented programming (OOP), where the code is organized based on defining individual objects to interact with each other within the system. As already suggested by the autonomous agents defined in *Chapter 2.3: Abstracting the Human Systems*, and the methodology specified in *Chapter 2.4: Spatial Functions*, these objects themselves can contain attributes in the form of variables that can be modified by functions in the form of procedures.^[3] Following this approach then allows the objects to effectively represent real life entities, which in turn allows us to simulate a variety of entities ranging from fluid particles to autonomous human agents to both static and dynamic architectural elements, making this a highly appropriate method for prototyping such a simulation.

While this prototype can be somewhat crude, it shows that the concept of these human systems is able to produce results somewhat resembling humans moving through the space, which demonstrates that there is merit in this methodology of simulating complex human crowds by these simple systems. (Fig. 2.5.1 - 2)

1 "What Is an Integrated Development Environment (IDE)?," Veracode, May 9, 2019, accessed October 18, 2019, <https://www.veracode.com/security/integrated-development-environment>.
 2 "What Is an IDE?," Codecademy, accessed October 18, 2019, <https://www.codecademy.com/articles/what-is-an-ide>.
 3 John Lewis and William Loftus, *Java Software Solutions: Foundations of Program Design* (Boston: Addison-Wesley, 2012), 44-51.



▶ **Figure 2.5.1** Prototype 2D Simulation created in Processing, based on the Nuit Blanche Installation Cushion



▶ **Figure 2.5.2** Cushion invites people to walk through a narrow corridor. The light filled balloons change color as people interact with them.

Part 3 | Tool Creation

Constructing the Simulation Tool

Constructing the Simulation Tool

This section will focus on translating the methodologies developed throughout *Part 2: Technical Research* and integrating them within a game engine. Chapter 3.1 will investigate various types of game engines and the reasoning behind choosing a specific one. Chapter 3.2 will look into the fundamentals of the chosen game engine to establish what is required to create a crowd simulation tool within this new software environment. Chapter 3.3 will be focusing on creating the human agents within this simulation by re-establishing the human systems model that was defined in Chapter 2.3, as well as defining additional considerations and requirements inherent in moving to this new Game Engine software environment. Chapter 3.4 will be establishing different ways of creating various architectural objects that the human agents can interact with, as well as defining some generic base objects within the tool that can be utilized for many project scenarios. Chapter 3.5 will investigate ways to create and import context from both within and outside of the game engine. Chapter 3.6 will define a possible workflow for setting up and utilizing this simulation tool for practical architectural visualization use.

Chapter 3.1 | Utilizing the Gaming Engine

Now that we have established the methodology of these human systems and verified the validity of them within a processing-based simulation prototype, we can begin to integrate this within a game development environment to create a usable framework for architectural visualization. At this stage, it is possible to add extra features to the system to enhance it from a two-dimensional dot-based representation of spatial movement to a system that resembles a crowded space. There are of course many ways to do this in processing, but the problem with this approach lies in the relatively low levels of abstraction available compared to other applications such as the game engines mentioned in Chapter 1.3. While processing is great for learning the fundamentals of programming, the game engine is a much more effective tool to continue developing this framework.

As expected, there is a large variety of game engines, examples of which includes the Quake family of engines, Unreal Engine, Half-Life Source Engine, DICE's Frostbite, Rockstar Advanced Game Engine (RAGE), CRYENGINE, Sony's PhyreEngine, Microsoft's XNA Game Studio, and Unity.^[1] While all these engines have their own strengths and weaknesses, as well as their own production pipelines, I have chosen to utilize Epic Game's Unreal Engine 4 (UE4) for its extensive documentation, photorealistic rendering engine, and visual scripting system, as these seem most in sync with the expectations of an architectural study.

Extensive Documentation: This is an important consideration when learning a new software. UE4 offers extensive documentation on their website^[2] as well as a tutorial series and live training on platforms such as YouTube and Vimeo.^[3] On top of this, there are also many forms of third-party tutorials online from various sources.^[4]

Photorealistic Rendering Engine: This feature is important for architectural visualization to better situate these visualizations within their physical context. UE4 offers many features to aid in the production of photorealistic visualizations, including: Physically-based Materials, pre-calculated bounce light via Lightmass, Stationary lights using IES profiles (photometric lights), and post processing, reflections.^[5]

Visual Scripting System: This is perhaps the most significant reason for choosing UE4 in this tool creation. This feature, which is aptly named blueprints within UE4, is a *Node based graph editor* with an interface that is very similar to other software such as Grasshopper and Dynamo, both of which are utilized within the architectural

1 Gregory, *Game Engine Architecture*, 31-36.
 2 "Unreal Engine 4 Documentation," Unreal Engine Documentation, accessed October 18, 2019, <https://docs.unrealengine.com/en-US/index.html>.
 3 "Unreal Engine," YouTube, accessed October 18, 2019, <https://www.youtube.com/channel/UCBobmJyZsJ6LI7UbthI4iwQ>.
 4 "Unreal Engine Tutorial in Videos on Vimeo," Vimeo, accessed October 18, 2019, <https://vimeo.com/search/page:2?q=unreal+engine+tutorial>.
 5 "Realistic Rendering," Unreal Engine Documentation, accessed October 18, 2019, <https://docs.unrealengine.com/en-US/Resources/Showcases/RealisticRendering/index.html>.

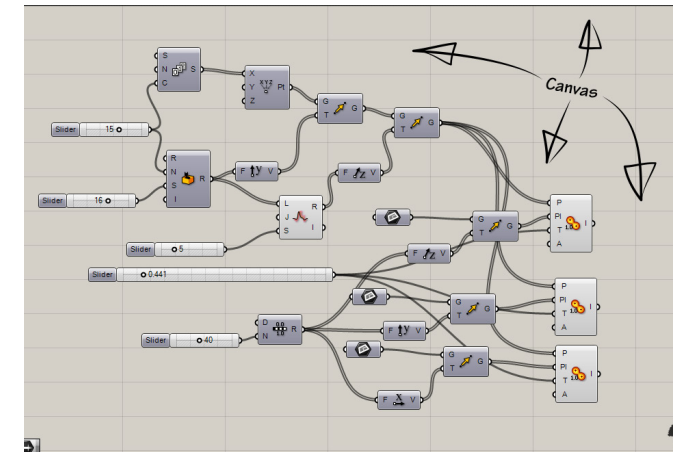


Figure 3.1.1 Parametric node system within Grasshopper

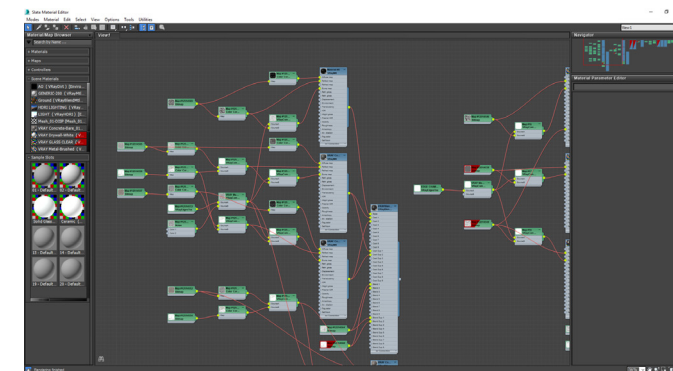


Figure 3.1.2 Material node system within 3ds Max

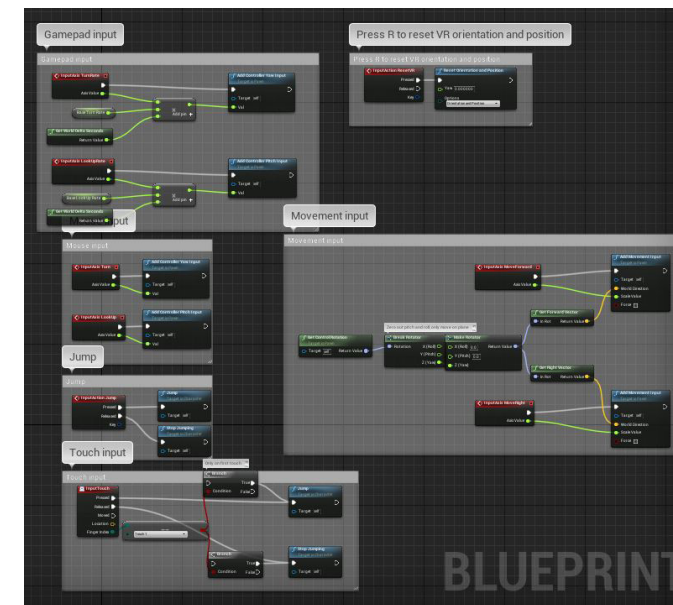


Figure 3.1.3 Scripting node system within Unreal Engine 4

industry for parametric modeling.^[6] (**Fig. 3.1.1 - 3**) This familiarity provides a type of visual scripting that is intuitive for people coming from predominantly visual fields such as architecture, which in turn provides a smooth translation of skillsets and toolsets already within architectural design and visualization.

While this tool may seem vastly different when compared to processing, one should remember that the syntax or the code is not what is important, but rather the methodology. Fortunately, we have already established this methodology through *Part 2: Technical Research* of this thesis; as such, this part will investigate the utilization of this established methodology to recreate the *human agents* and the *spatial entities* of this system—within this new software environment that is Unreal Engine 4.

⁶ "Blueprint Editor Reference," Unreal Engine Documentation, accessed October 18, 2019, <https://docs.unrealengine.com/en-US/Engine/Blueprints/Editor/index.html>.

Chapter 3.2 | Asset Creation

The organization of UE4 follows an object-based approach, which is similar to the OOP methodologies that were utilized within the processing prototype in Chapter 2.5. Because of this, we already have a rough idea of what we need to create within this new environment—we just need to familiarize ourselves with the local syntax.

Compared to processing, Unreal Engine has a more intuitive file-based system for handling entities within its software environment. As such, its base building blocks are appropriately referred to as *objects*.^[1] These objects contain much of the lower-level code required to provide “under the hood” functionalities, and when sterilized to a file, they are referred to as an asset, which is a piece of content within Unreal Engine 4.^[2] (Fig. 3.2.1) Of these assets, the ones that allow additional functions to be scripted are referred to as blueprint classes. These blueprint classes can then be broken down into the following generic types:^[3]

Actor: “[A]n *object* that can be placed or spawned in the world.”^[4]

Pawn: “[A]n *Actor* that can be “possessed” and receive input from a Controller.”^[5]

Character: “[A] *Pawn* that includes the ability to walk, run, jump, and more.”^[6]

Player Controller: “[A]n *Actor* responsible for controlling a Pawn used by the player.”^[7]

Game Mode: “[D]efines the game being played, its rules, scoring, and other faces of the game type.”^[8]

The fact that assets are *objects that are serialized to a file* means that they can be utilized within many different UE4 projects. Therefore, if we want to create a crowd simulation tool that can be used within many types of architectural projects, we must create an asset package. As such, this chapter will investigate the creation of these assets based on our defined methodologies from *Part 2: Tool Creation*. Looking back at our system model, we then need to establish two main asset types within UE4: the *human agents* and the *architectural objects*.

1 “Unreal Engine 4 Terminology,” Unreal Engine Documentation, accessed October 18, 2019, <https://docs.unrealengine.com/en-US/GettingStarted/Terminology/index.html>.

2 “Assets and Packages,” Unreal Engine Documentation, accessed October 18, 2019, <https://docs.unrealengine.com/en-US/Engine/Basics/AssetsAndPackages/index.html>.

3 “Blueprint Class,” Unreal Engine Documentation, accessed October 21, 2019, <https://docs.unrealengine.com/en-US/Engine/Blueprints/UserGuide/Types/ClassBlueprint/index.html>.

4 Unreal Engine Documentation, “Blueprint Class.”

5 Unreal Engine Documentation, “Blueprint Class.”

6 Unreal Engine Documentation, “Blueprint Class.”

7 Unreal Engine Documentation, “Blueprint Class.”

8 Unreal Engine Documentation, “Blueprint Class.”

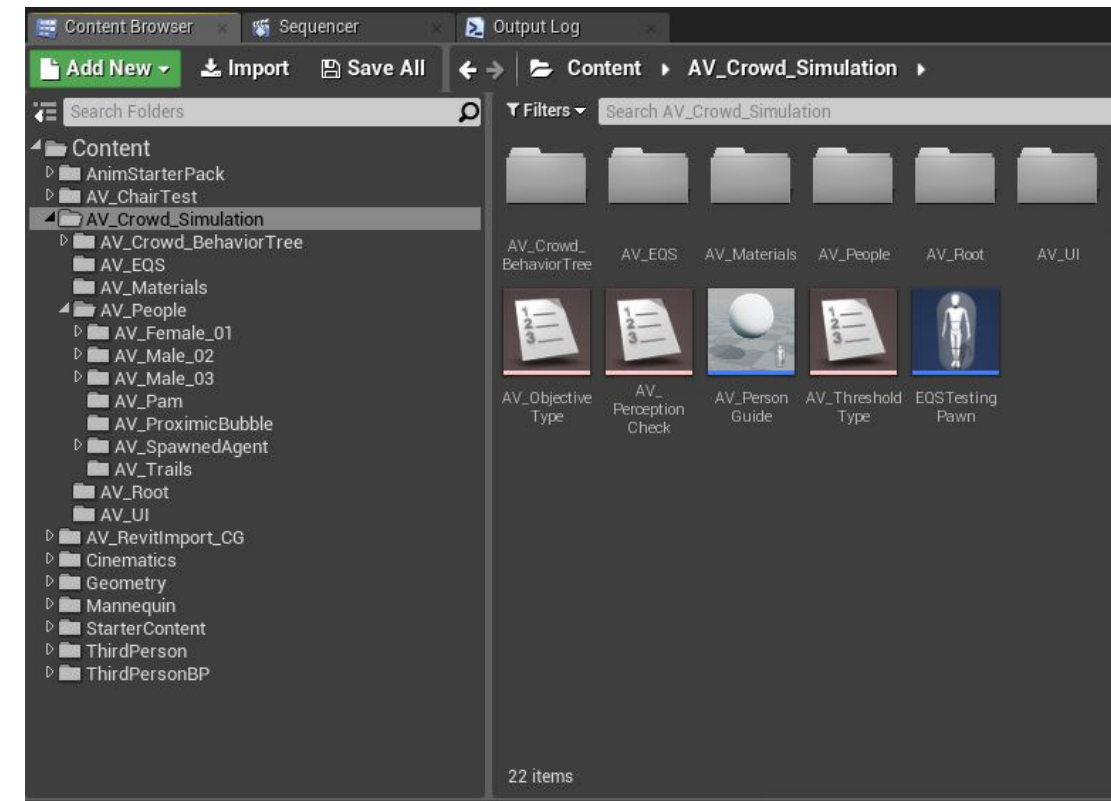


Figure 3.2.1 Game assets within project browser

Chapter 3.3 | Human Agents

The first step to creating the human agents within this software environment is to set up the data structure of the agent. As such, we need to create the following assets within the content browser: *AI Controller*, *Blackboard*, *Behavior Tree*, and *Character Class*.^[1]

AI controller: This is the blueprint controller that controls the agent’s mental actions, which acts as the container for all decision branches the agent will utilize. From this, the AI controller will select a resulting state according to the agent’s choice and set it within the blackboard.

Blackboard: This is a container to store all the different states the agent can be in.

Behavior tree: This is a tool that controls all the physical actions of the agents. It will command a series of pre-established actions depending on the state of the agent set within the blackboard.

Character class: This blueprint class acts as a container for the physical attributes of the agents and is the physical entity that moves within the game environment. As such, this class would generally include assets such as the 3D model of the agent as well the textures that may be associated with it.

With this, we can establish a rough idea of how the agents will function within the software. As the agents explore the space, they will receive environmental input. They will then use this input to decide what action to take within the AI controller. This choice will then be used to set the agent state within the blackboard which will then be utilized by the behavior tree to determine a set of tasks the agent will perform. Once the agent finishes performing this task, it can then do additional tasks or return to a default state, in which the cycle will start over. Within the simulation, each agent will have their own independent copy of these assets, as such, they will function and interact independently from each other—much like how real humans do.

From here, the next step would be to re-establish the human systems from Chapter 2.3, which are the sensory system, the decision logic, and the pathfinding. We already established the components of this system, however, we can revisit our human systems from the top down instead of bottom up. This allows us to approach this problem in a sequential logical order to create our agents. There are, of course, various tools within UE4 to help us recreate these systems; therefore, this chapter will investigate the processes of doing so.

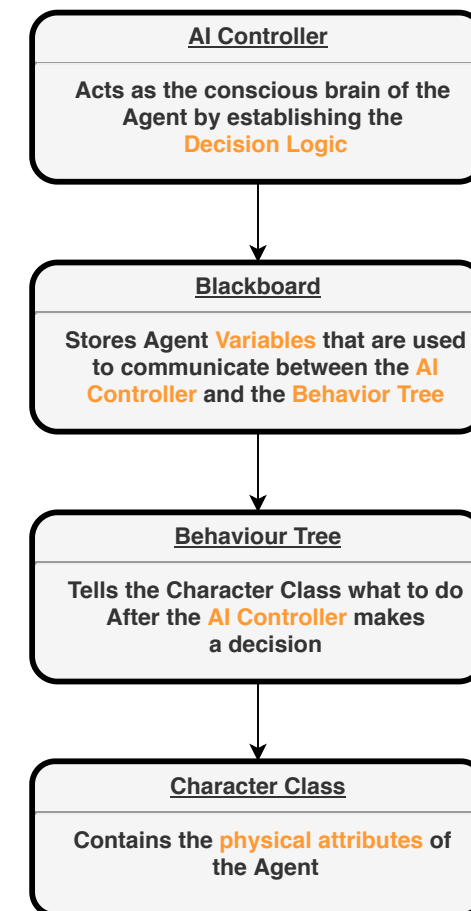


Figure 3.3.1 How each asset will be utilized within this software environment

¹ “Behavior Tree Quick Start Guide,” Unreal Engine Documentation, accessed October 18, 2019, <https://docs.unrealengine.com/en-US/Engine/ArtificialIntelligence/BehaviorTrees/BehaviorTreeQuickStart/index.html>.

Sensory System

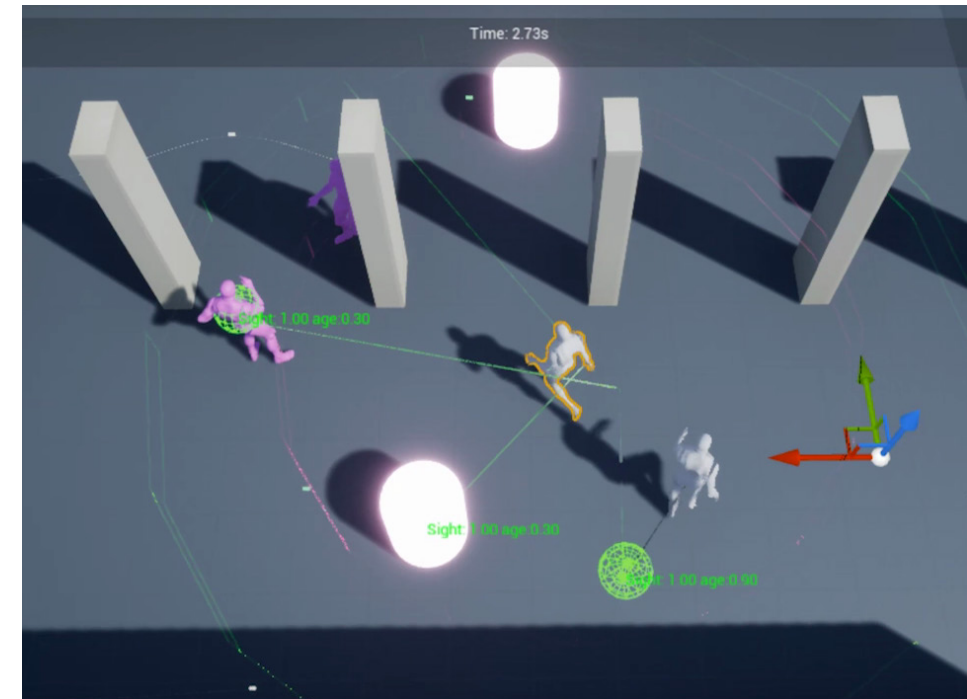
The first and top level of these systems is the sensory system, which allows our agents to be able to see the environment. Within processing, we had to manually calculate this by utilizing the dot product and then limiting the angles with *iffelse* statements, but within UE4, this becomes much easier and more intuitive to establish. The reason for this is largely due to the tools provided within UE4, such as *AI perception* and the *Environment Query System (EQS)*.

AI perception is a component that can be added to the AI controller, which can then be used to define various senses that the agents can utilize. (Fig. 3.3.2) These senses can include traditional ones such as sight, hearing, and touch, to nontraditional game-related ones such as damage and team.^[2] As established in Part 3, sight is by far our most dominant sense so at this stage we will only be implementing sight for the agents. The agents can then use this to determine the location and type of anything they see, as well as the time in which they last saw it.

Environment Query System (EQS) is a feature that allows the agents to collect data from the game environment. It does this by performing a series of tests to determine the best location option depending on the set parameters.^[3] The agents can use this system to determine the best location around other entities based on a multitude of other factors such as distance and sight. (Fig. 3.3.3)

Both of these tools would then be able to return a variable within their respective systems, which can then be used within the next stages of these human systems.

Figure 3.3.2 - 3.3.3
AI perception and EQS offer different ways for the agents to acquire data from their surrounding environment.



▶ **Figure 3.3.2** AI perception

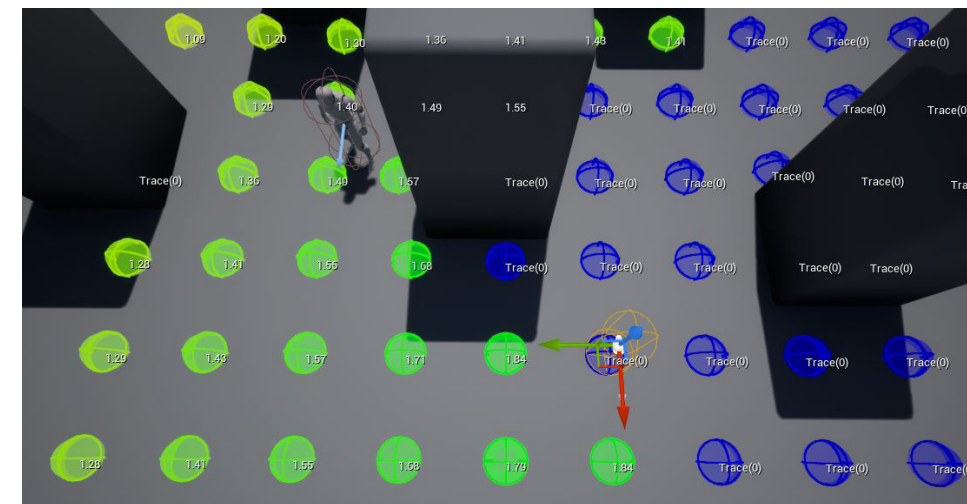


Figure 3.3.3 EQS trace test

2 "AI Perception," Unreal Engine Documentation, accessed October 18, 2019, <https://docs.unrealengine.com/en-US/Engine/ArtificialIntelligence/AI Perception/index.html>.
 3 "Environment Query System Quick Start," Unreal Engine Documentation, accessed October 18, 2019, <https://docs.unrealengine.com/en-US/Engine/ArtificialIntelligence/EQS/EQSQuickStart/index.html>.

Decision Logic

The next level of these systems is the decision logic, which allows our agents to choose an action when they see an object within the environment. This can be accomplished within the AI controller class, which contains an EventGraph that allows a form of visual scripting with various elements such as:^[4]

“**Events** are nodes that are called from gameplay code to begin execution of an individual network within the EventGraph.”^[5]

Functions are node graphs within blueprints that can be executed to perform a specific function.^[6] These can be utilized throughout the EventGraph to change variables and divert the flow of execution.

“**Variables** are properties that hold a value or reference an Object or Actor in the world.”^[7] These can be utilized to store or reference various types of data within the blueprint.

Components are sub-objects that can be attached to actors. These can be used to give additional functionalities to the actor.^[8]

There are certainly many more types of elements within these event graphs, but for the most part, these are the main elements that make up most of this decision logic system. These elements can then be connected to form a decision network, which allows the agents to take a step-by-step approach to establishing their choices. (Fig. 3.3.5) While this may look very complicated, it is a rather straightforward process:

The first step begins with the *Event Node*. These elements begin the execution of nodes down an individual network of functions, and can be triggered in many ways, including but not limited to every frame, set time interval, when a key is pressed, and in the context of AI perception, whenever the agent sees something.^[9]

Once the agent sees something, the *AI perception Event Node* begins a line of execution down the network where we can utilize a number of Boolean-driven branch nodes to divert the execution line. (Fig. 3.3.6) The Booleans that drive these branch nodes can be defined in various ways, but for this simulation, we can utilize the concept of percentages we defined from Chapter 2.3. With this, we can generate a random number between 0 and 1, where 1 is 100% yes, 0 is 0% yes or 100% no, and 0.56 is 56% yes or 44% no, etc. (Fig. 3.3.7) With this we can perform a series of checks throughout the logic network to

4 “EventGraph,” Unreal Engine Documentation, accessed October 18, 2019, <https://docs.unrealengine.com/en-US/Engine/Blueprints/UserGuide/EventGraph/index.html>.
 5 “Events,” Unreal Engine Documentation, accessed October 18, 2019, <https://docs.unrealengine.com/en-US/Engine/Blueprints/UserGuide/Events/index.html>.
 6 “Functions,” Unreal Engine Documentation, accessed October 18, 2019, <https://docs.unrealengine.com/en-US/Engine/Blueprints/UserGuide/Functions/index.html>.
 7 “Blueprint Variables,” Unreal Engine Documentation, accessed October 18, 2019, <https://docs.unrealengine.com/en-US/Engine/Blueprints/UserGuide/Variables/index.html>.
 8 “Components,” Unreal Engine Documentation, accessed October 18, 2019, <https://docs.unrealengine.com/en-US/Programming/UnrealArchitecture/Actors/Components/index.html>.
 9 Unreal Engine Documentation, “Events.”

determine the final action of the agent. Is the agent seeing an object? If yes, is the agent currently doing something? If no, is the agent interested in this object? If yes, does it want to observe and admire it or interact with it? If interact, then set *AgentState* to *Interact*. (Fig. 3.3.8) However, if the agent is currently busy or is not interested in the object, then the execution line will move down and ask if the agent if it is seeing another agent. If yes, is the agent current doing something? And so on. It would then continue these checks for every type of entity the agent comes across. (Fig. 3.3.9)

This is naturally an oversimplification, as there are other steps required due to the various nuances of UE4, but these are the essential steps required to create these behaviors to give the illusion of choice to the agents. With this, different functions within the blueprint will be triggered depending on the path of this execution, therefore allowing us to define the agent actions based on these paths.

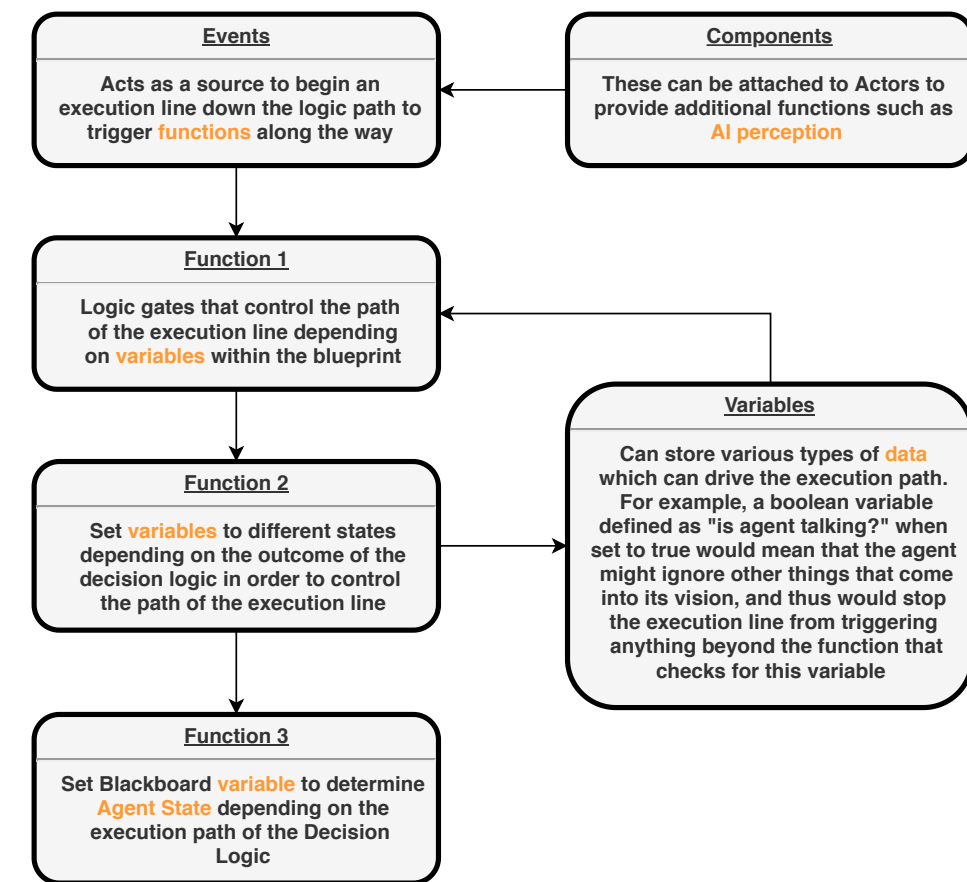


Figure 3.3.4 Simplified visual scripting process within UE4

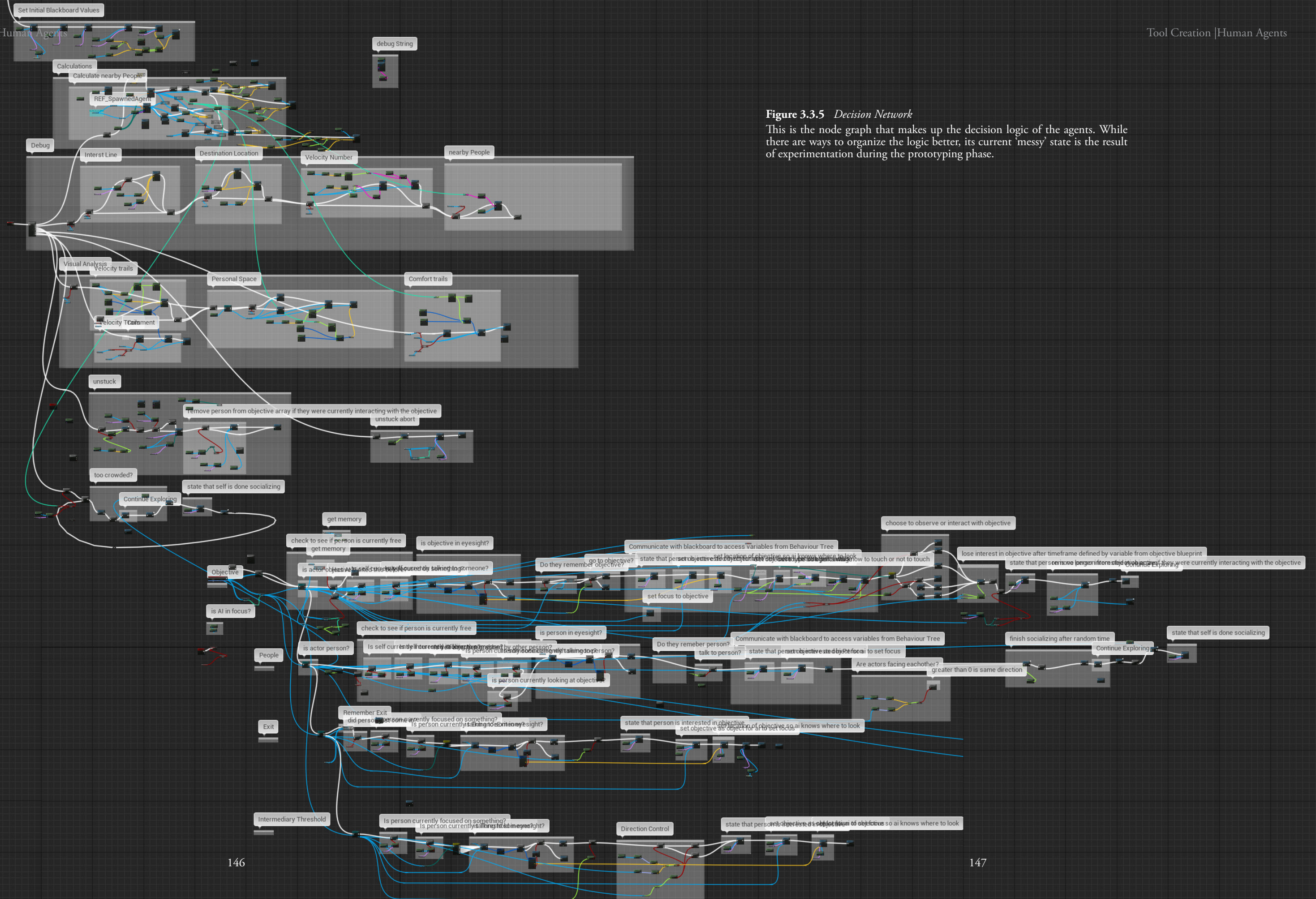


Figure 3.3.5 *Decision Network*
 This is the node graph that makes up the decision logic of the agents. While there are ways to organize the logic better, its current 'messy' state is the result of experimentation during the prototyping phase.

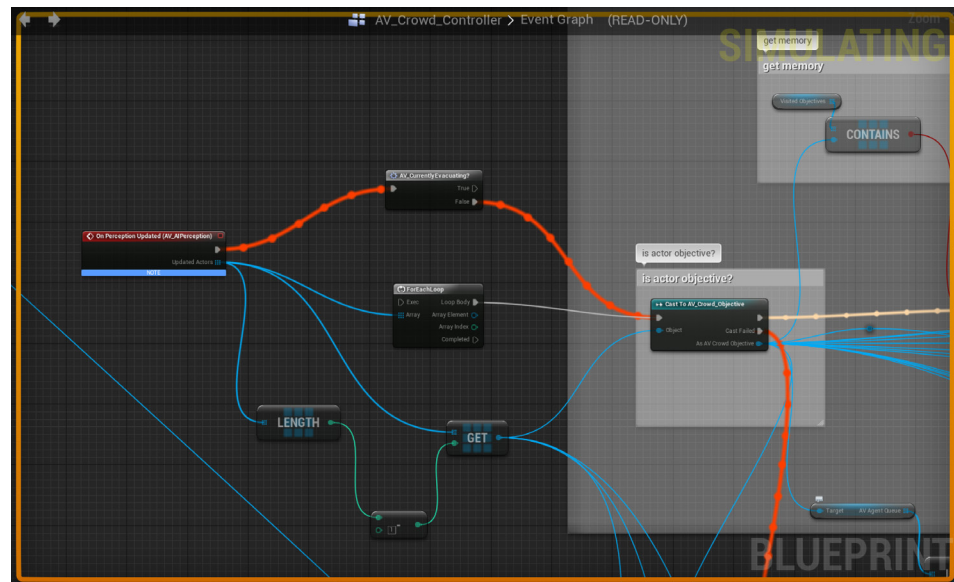


Figure 3.3.6 Event Node begins the execution line

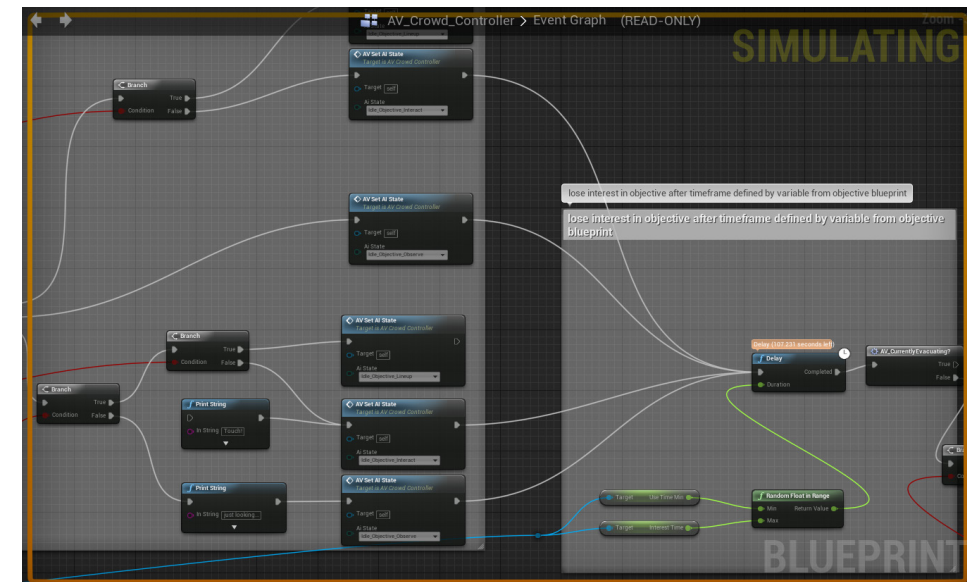


Figure 3.3.8 Once the execution line reaches a decision, a node is triggered to set the AgentState and then apply a timer to establish how long the agent might be in that state

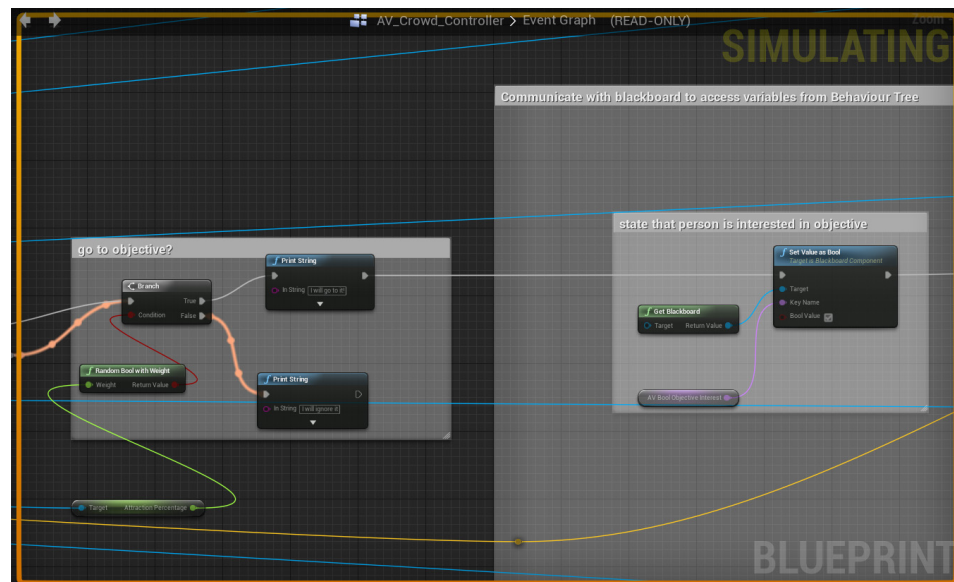


Figure 3.3.7 Boolean percentages controls which path the line takes

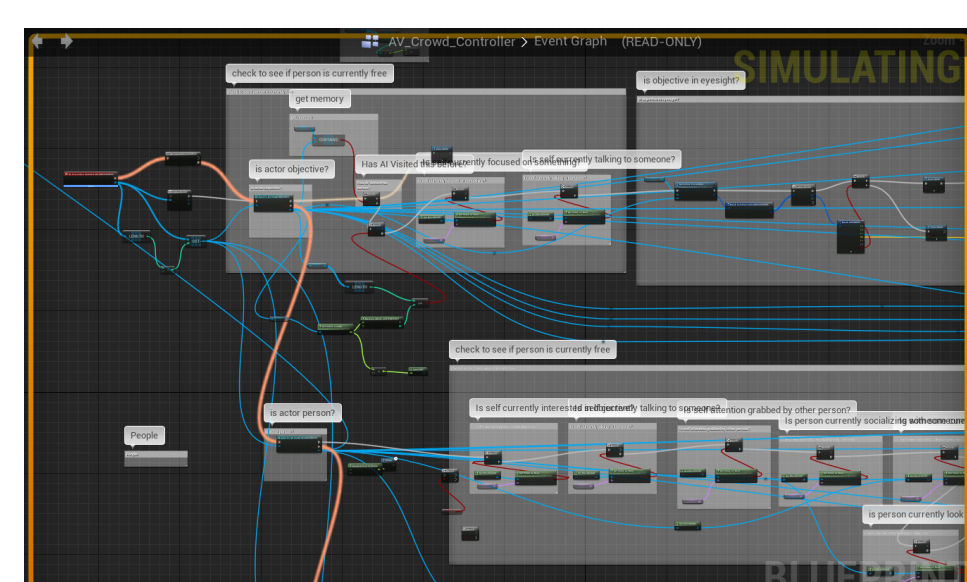
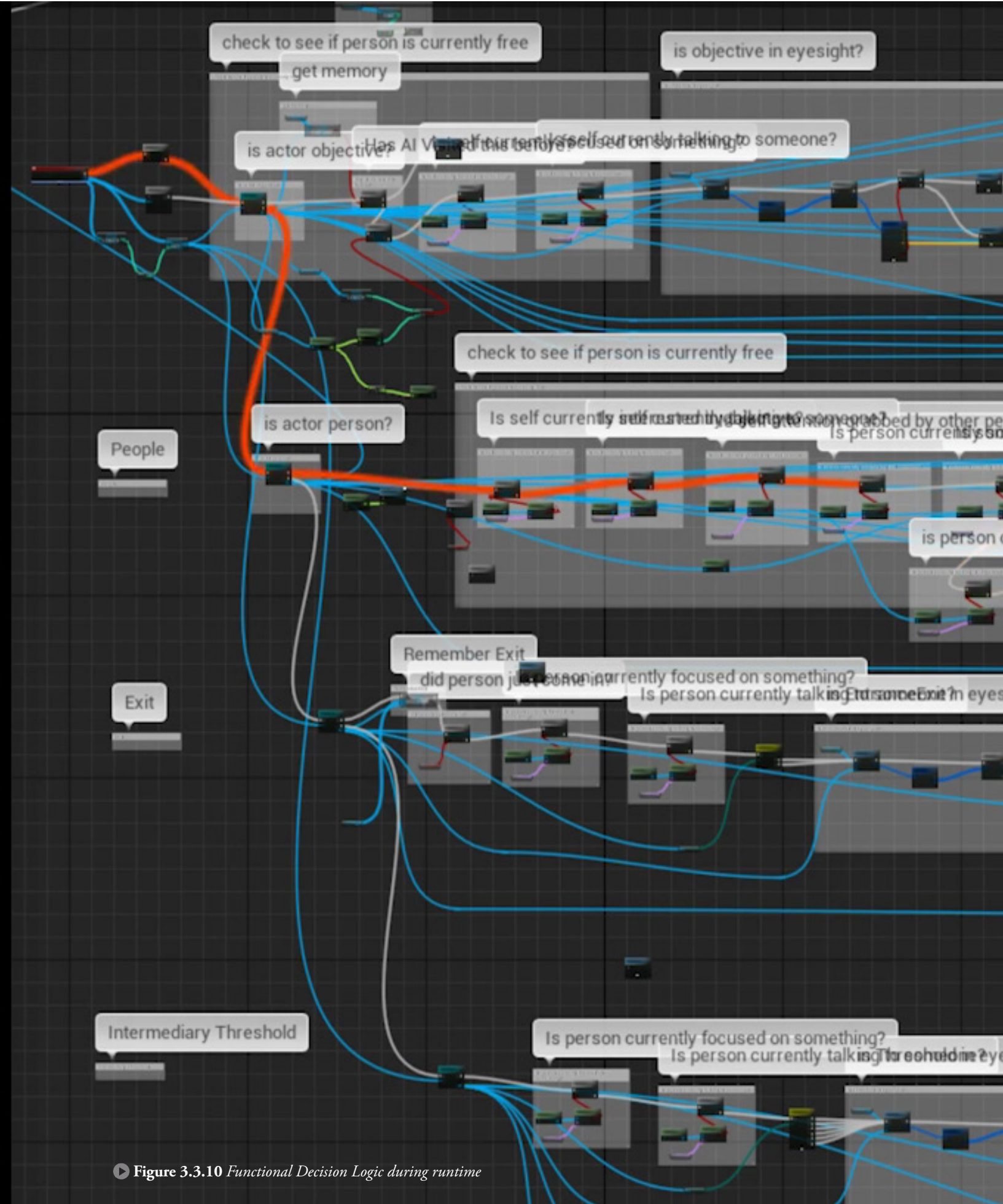
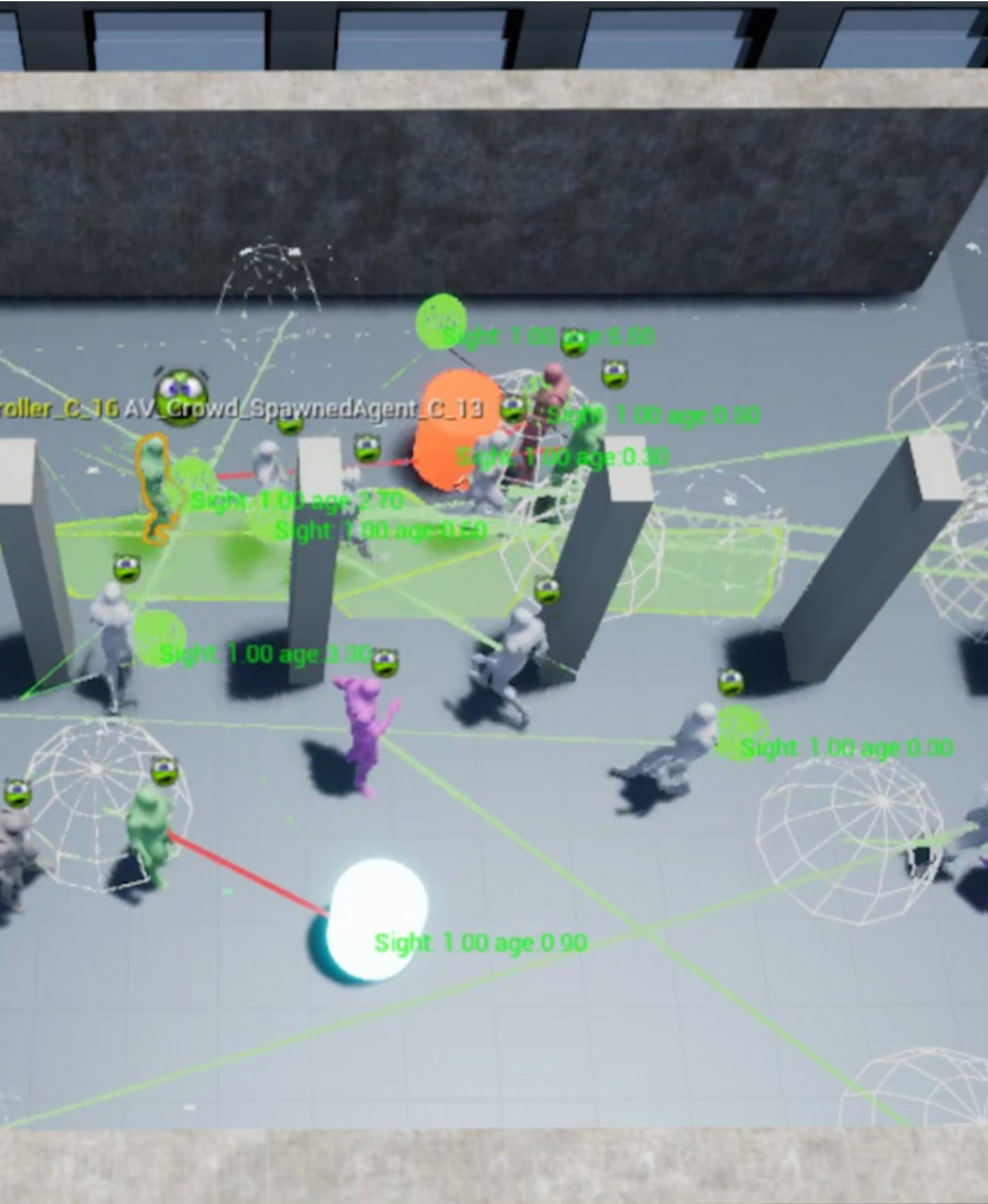


Figure 3.3.9 The execution line moves down and repeats all the checks for every entity type if it does not reach the end of the logic



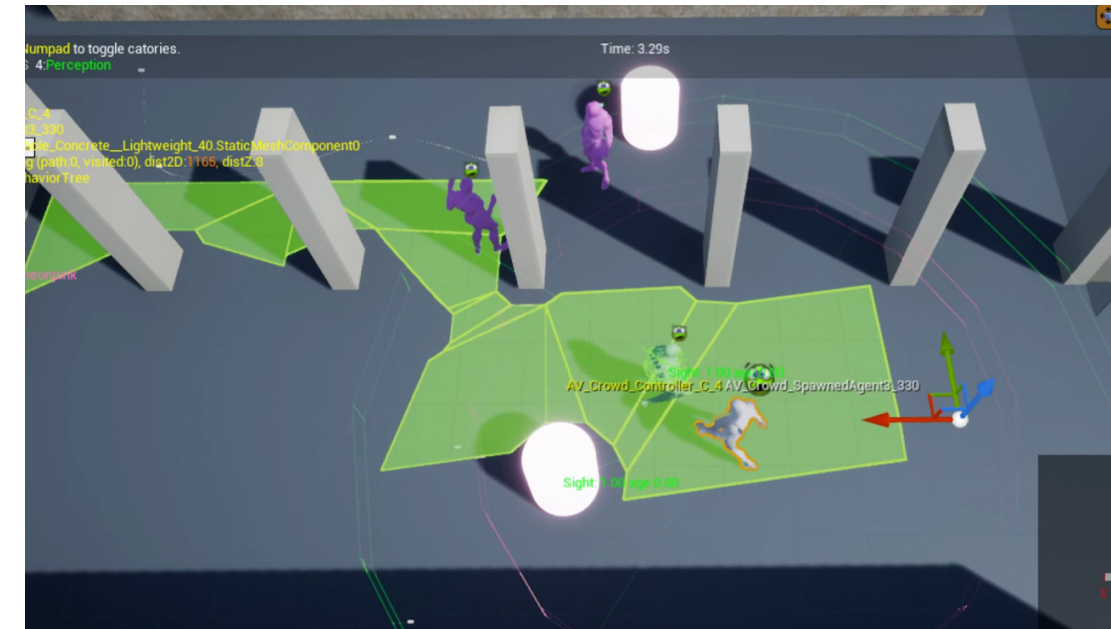
▶ Figure 3.3.10 Functional Decision Logic during runtime

Pathfinding

The last human system to reestablish is the pathfinding system. Recalling from *Chapter 2.3: Abstracting the Human Systems*, the purpose of this system is to provide a way for the agents to move to their goals through the consideration of their subconscious desires, which we achieved by modifying Craig Reynolds’ description of the three layers of motion: *action selection, steering, and locomotion*. From this, we investigated Edward T. Hall’s *proxemics* to establish personal space for the agents, where action selection defined their subconscious desires of preserving this space, steering provided the forces that drove this preservation, and locomotion translated this force into their current position within the virtual environment.

Translating this methodology into UE4 is once again, fairly straightforward, considering that the software has a built-in navigation system by means of a *NavMesh volume* within the simulation environment.^[10] (Fig. 3.3.11) As it can be seen, compared to simple text-based IDEs such as processing, where we had to essentially create everything from the ground up, UE4 offers a vastly more efficient workflow. Rather than having to work down the levels of code to calculate the vector position of these agents while simultaneously implementing various vector calculations to account for collision and avoidance, we can now just utilize UE4’s built-in navigation for the agents to move to their desired conscious goals.

In order to implement subconscious goals such as personal space, however, we must implement our modified layers of motion on top of this built in NavMesh. This can be done by utilizing a Tick Event (which is triggered every frame) to trigger an offset position node within the AI controller. In doing so, we can utilize a variety of vector calculation nodes to mimic the steering forces from Reynolds’ description of motion. (Fig. 3.3.12) This will not only make the crowd flow smoother but also more human-like.



▶ Figure 3.3.11 Navmesh at work

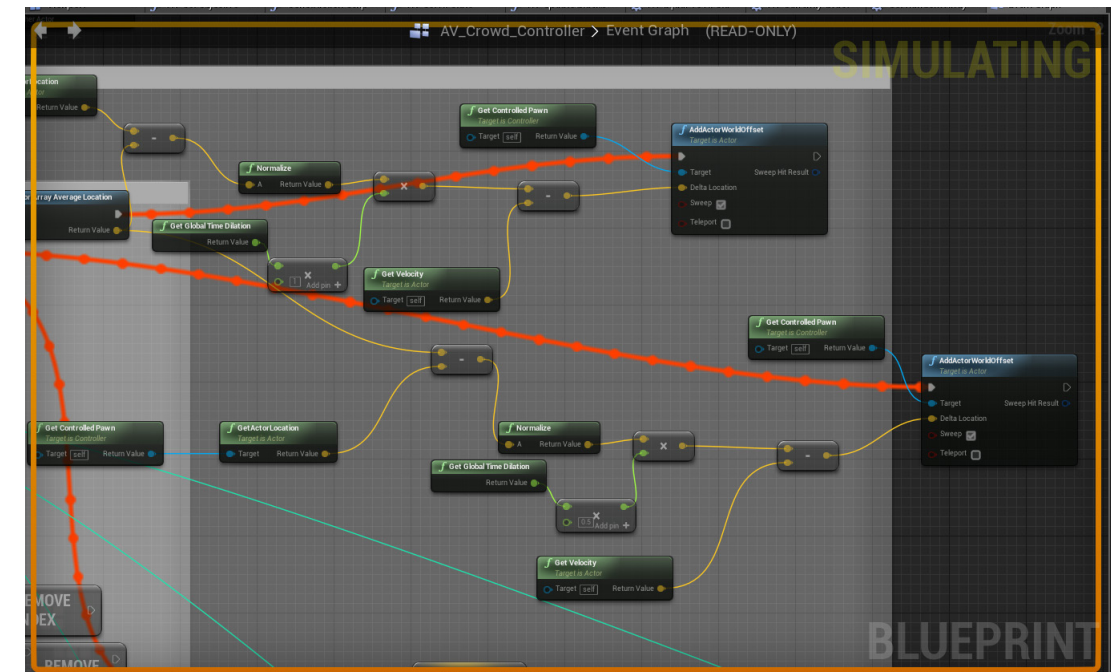


Figure 3.3.12 Flocking Behaviors recreated with Vector and World-Offset nodes within UE4

10 “Navmesh Content Examples,” Unreal Engine Documentation, accessed October 18, 2019, <https://docs.unrealengine.com/en-US/Resources/ContentExamples/NavMesh/index.html>.

Agent-State Actions

Now that we have established the three human systems within our simulation model, our agents should be able to move around freely within the space. However, since these agents are no longer simple dot representations—but actual human agents walking in a virtual 3D environment—we require additional considerations to portray them within the simulation.

The first of these considerations are the actions the agents need to perform once they choose a state from the decision logic. Within our processing prototype, they are simply represented by dots that try to avoid one another to get to a target. While this was fine when the level of detail was represented by dots, it seems rather unrealistic now that these agents are more representative of the human form. As such, to retain this credibility, we need to define a set of actions for the agents to perform depending on the task that they choose to act upon. In doing so, this crowd simulation becomes much more comparable to reality.

Fortunately, UE4 provides a useful tool called the *Behavior Tree* specifically for scripting artificial intelligence. This tool—much like blueprints—provides a visual method of integrating functionality by utilizing a series of nodes. The difference from blueprints, however, is that this Behavior Tree Graph executes logic by priority from left to right and top to bottom.^[11] (Fig. 3.3.13 - 14) This allows the Behavior Tree to work alongside the *blackboard* and *AI controller* to provide an intuitive method of establishing these actions based on the choices the agents have made. The decision logic tells the blackboard what state the agent is in, and the behavior tree will read that state to determine which branch of tasks it should perform. From this, we can define a series of states that the agent may be in.

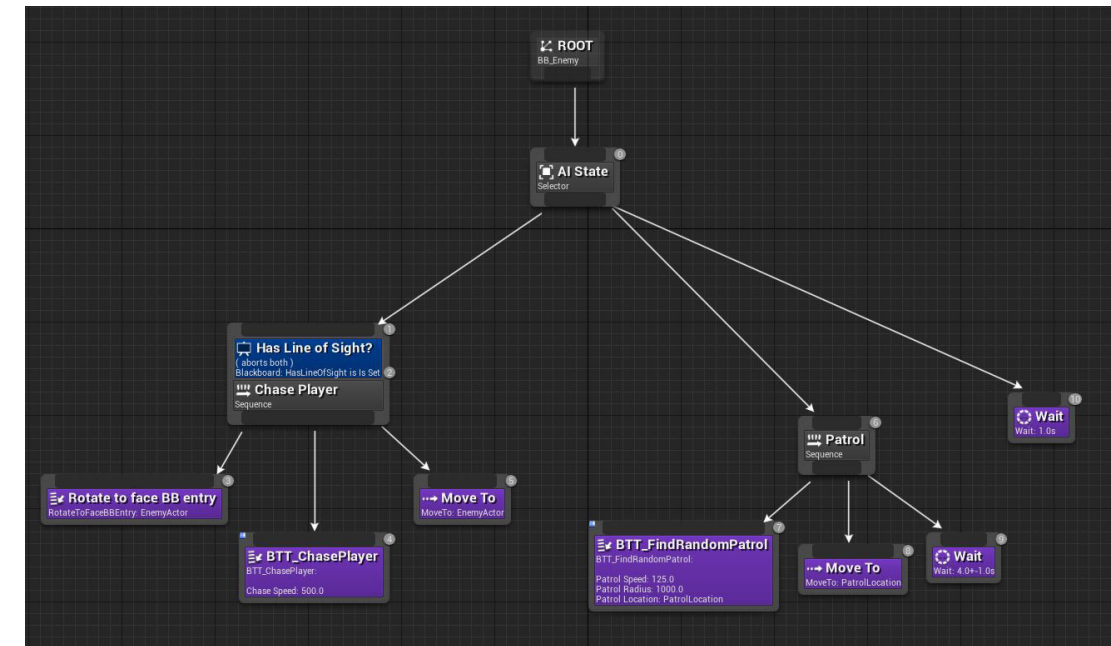


Figure 3.3.13 The Behavior Tree allows a visual way to define AI tasks within UE4

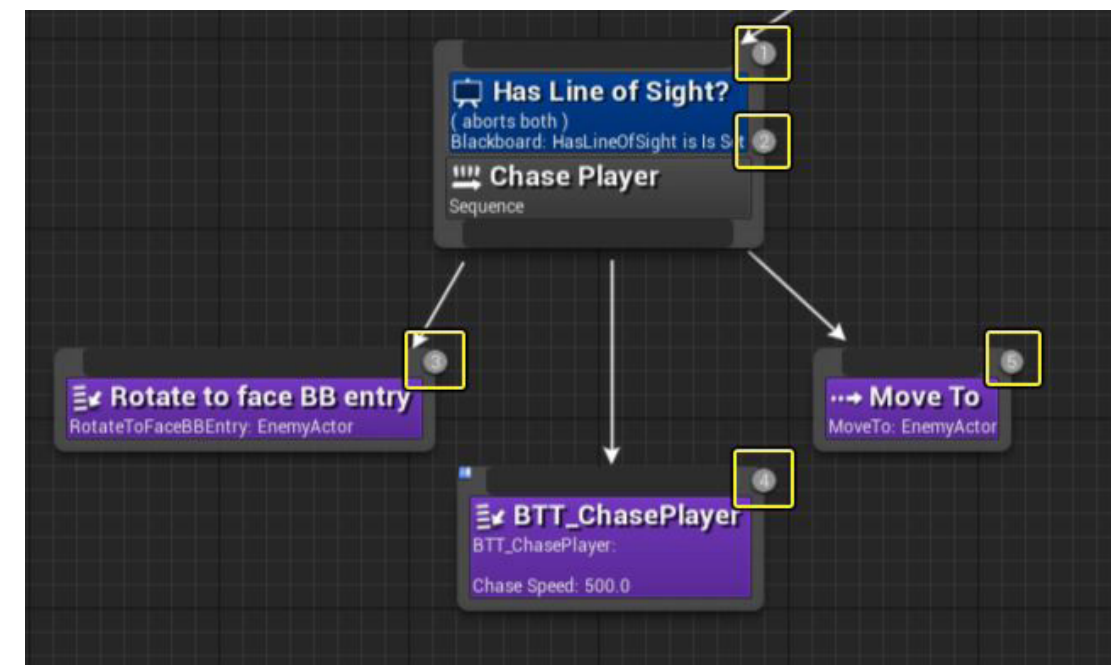


Figure 3.3.14 Behavior Tree Execution Order from top to down and left to right

11 "Behavior Tree Overview," Unreal Engine Documentation, accessed October 18, 2019, <https://docs.unrealengine.com/en-US/Engine/ArtificialIntelligence/BehaviorTrees/BehaviorTreesOverview/index.html>.

Default State

Due to the responsive nature of these agents, one of the important questions we must first ask is: What do the agents do when the surrounding space is empty? In other words, we must first define a base state in which the agents can operate with limited knowledge and no environmental information to work from.

In order to define this state, we must make some assumptions. Imagine if people were left in an empty room; What would they do? Would they do nothing, or would they explore? Chances are, most people would explore to see if there is anything around them rather than just doing nothing. Ideally, we would be able to perform a survey from a population to deduce an average base state, but since this simulation only needs to be an approximation, this generic assumption of *exploration* should be adequate to visualize crowd flow.

To simulate this task of *exploration*, we should ask ourselves what it means to explore the space within an empty room. The lack of features means that there is nothing to attract our attention, therefore the only variance within such a space would be the distance of the agent to the boundary of this space. In that context, it makes logical sense that the furthest point offers the most mystery; therefore, it should also offer the highest incentive to investigate.

Translating this to game engine logic then, we can configure the EQS tool to query the environment around the agent every second and provide a location that is the farthest point to the agent. (Fig. 3.3.15 - 16) In doing so, it provides the illusion of exploration. (Fig. 3.3.17)

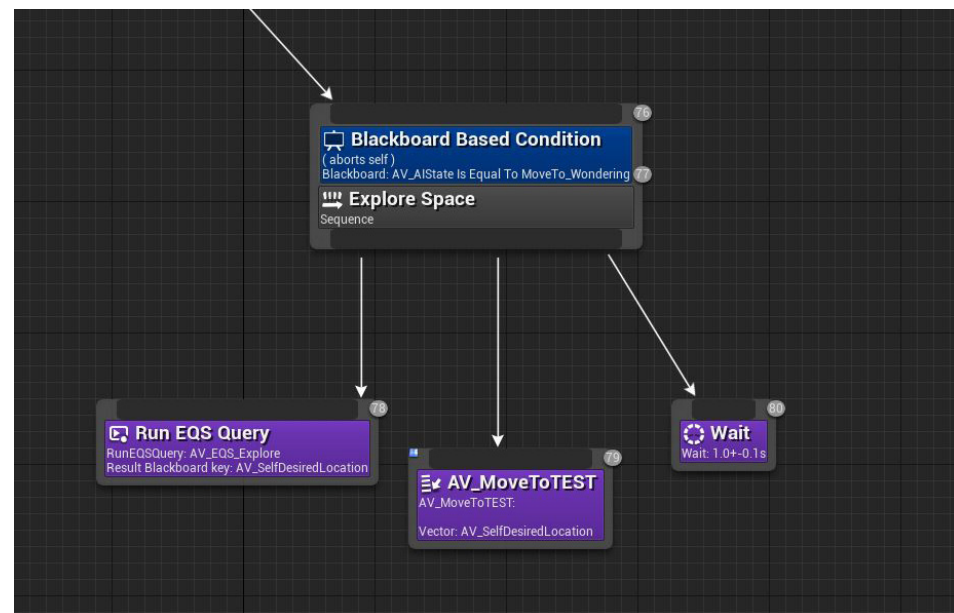


Figure 3.3.15 Explore Tasks defined within the Behaviour Tree

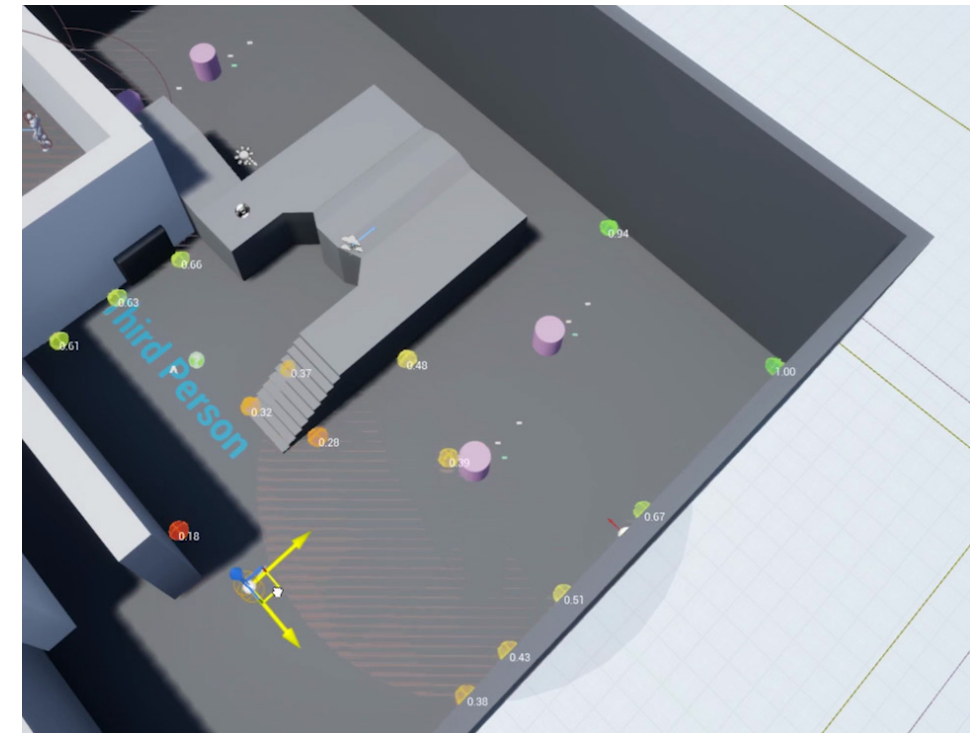


Figure 3.3.16 EQS allows us to check the surrounding environment to calculate locations based on how far away they are from the agent



Figure 3.3.17 Agent Exploring the space by utilizing this data

Object Looking State

The object looking state defines what happens when an agent chooses to admire an object without touching it. Again, a population survey based on the location and demographic of the architectural project would give the most accurate simulation result within a specific context, however, in the ethos of this simulation tool—where there is no specific context and the tool is designed to be applicable within different typologies—a generic set of actions would provide the most flexibility. To establish these generic actions, we can consider a passage from *The Dynamics of Architectural Form*, where Rudolf Arnheim states:

“These ‘proxemics’ normals influence also the choice of preferred distances between objects, e.g., the placement of furniture, and they are likely to affect the way people determine and evaluate the distances between buildings. [...] We feel impelled to juggle the distances between objects until they look just right because we experience these distances as influencing forces of attraction and repulsion. [...] In order for an object to be perceived appropriately, its field of forces must be respected by the viewer, who must stand at the proper distance from it. I would even venture to suggest that it is not only the bulk or height of the object that determines the range of the surrounding field of forces, but also the plainness or richness of its appearance. A very plain façade can be viewed from nearby without offense, whereas one rich in volumes and articulation has more expansive power and thereby asks the viewer to step farther back so that he may assume his proper position, prescribed by the reach of the building’s visual dynamics.”^[12]

From this, Arnheim examines the proxemics described by Edward T. Hall (which we have also investigated to establish the personal spaces of our agents), and reasons it within the context of object distances as well as human distances. While Arnheim is specifically referring to the distances between the placement of furniture in this case, by changing the reference point of view, it can be argued that these distances can also be utilized to describe the preferred viewing distances of agents when admiring an object. These viewing distances can then be influenced by factors such as the size of the object as well as “the plainness or richness of its appearance.”

We can then recreate this description by establishing a series of movements within the behavior tree. (Fig. 3.3.18) By utilizing the EQS, we can first determine a random location around the object. (Fig. 3.3.19) From this, we can then tell the agent to move there, turn towards the object, and admire it within a randomized timeframe. After this timeframe, the agent would then pick a different location around the object and repeat the process. From this, we can establish the following steps:

1. Find location around the object with EQS.
2. Move to location.
3. Turn towards the object.
4. Loop observing animation for randomized time interval.
5. Repeat action from step 1, or set AgentState to something else (such as ObjectInteract or AgentDefault).

This series of actions would give the illusion of the agents trying to “juggle the distances between objects until they look just right.” (Fig. 3.3.20) Throughout this process, the agent may also decide to interact with the object or become bored with the object. While this is by no means a perfect representation, it does provide a relatively simple and generic approach for portraying this action at this stage of tool creation.

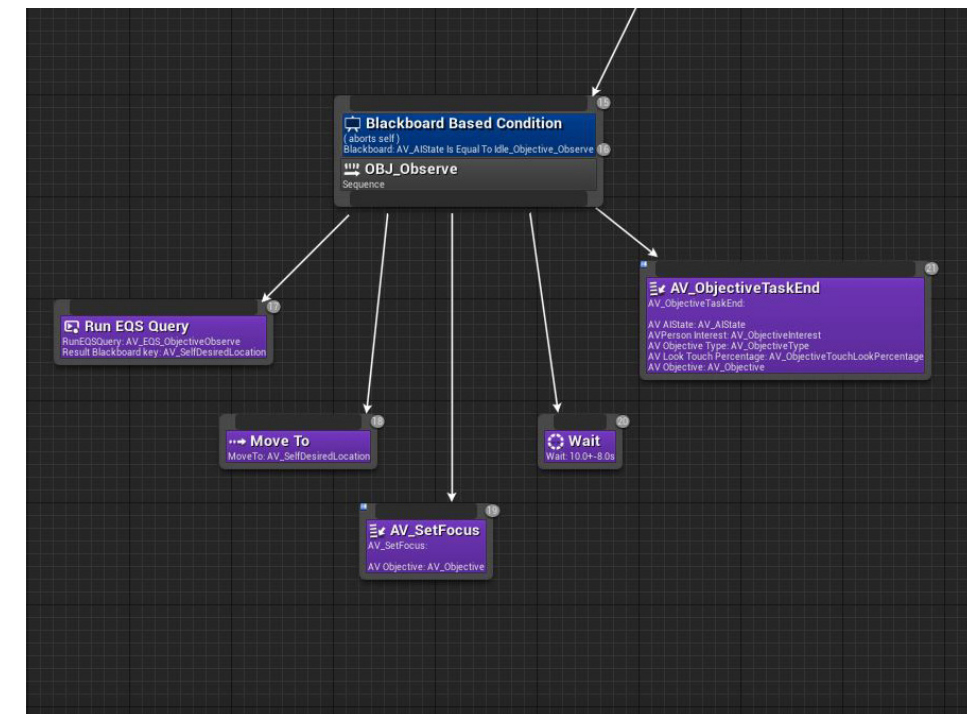
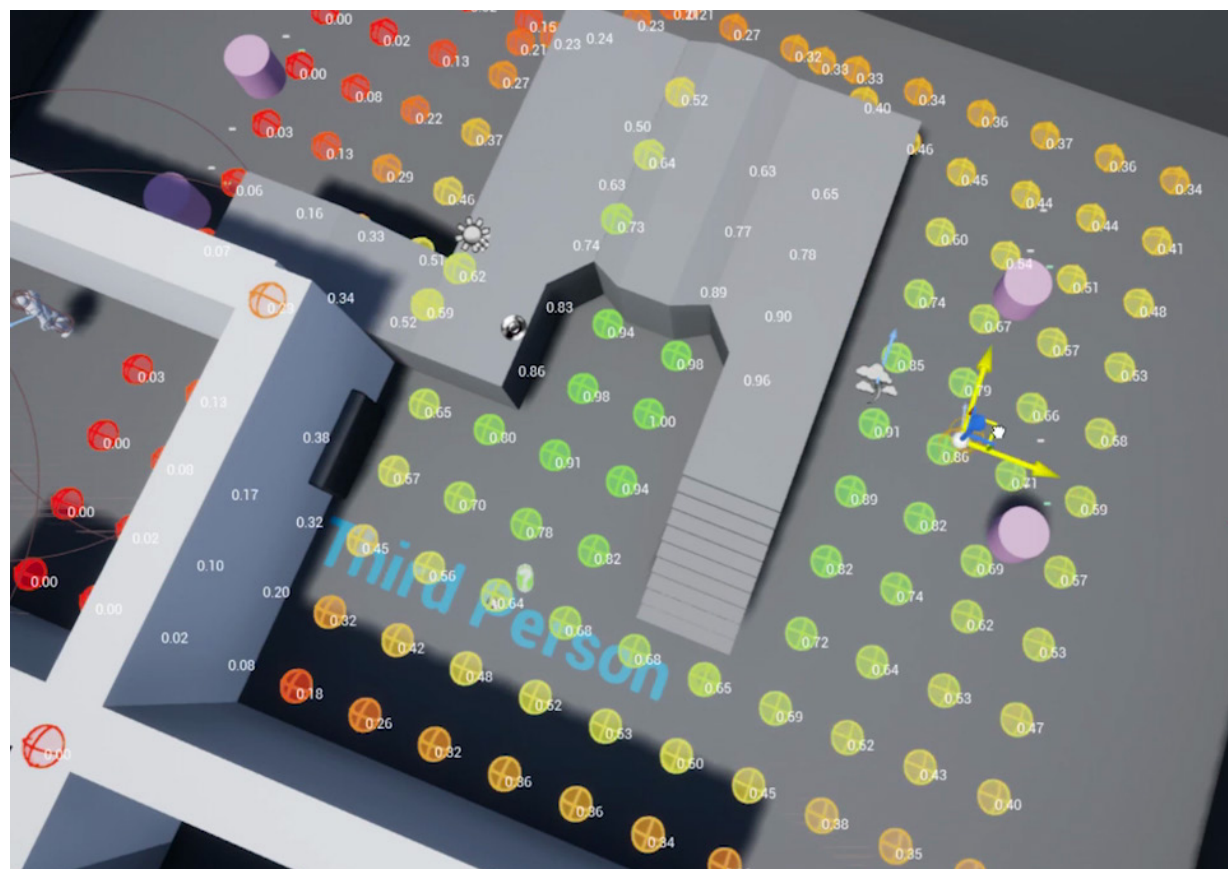
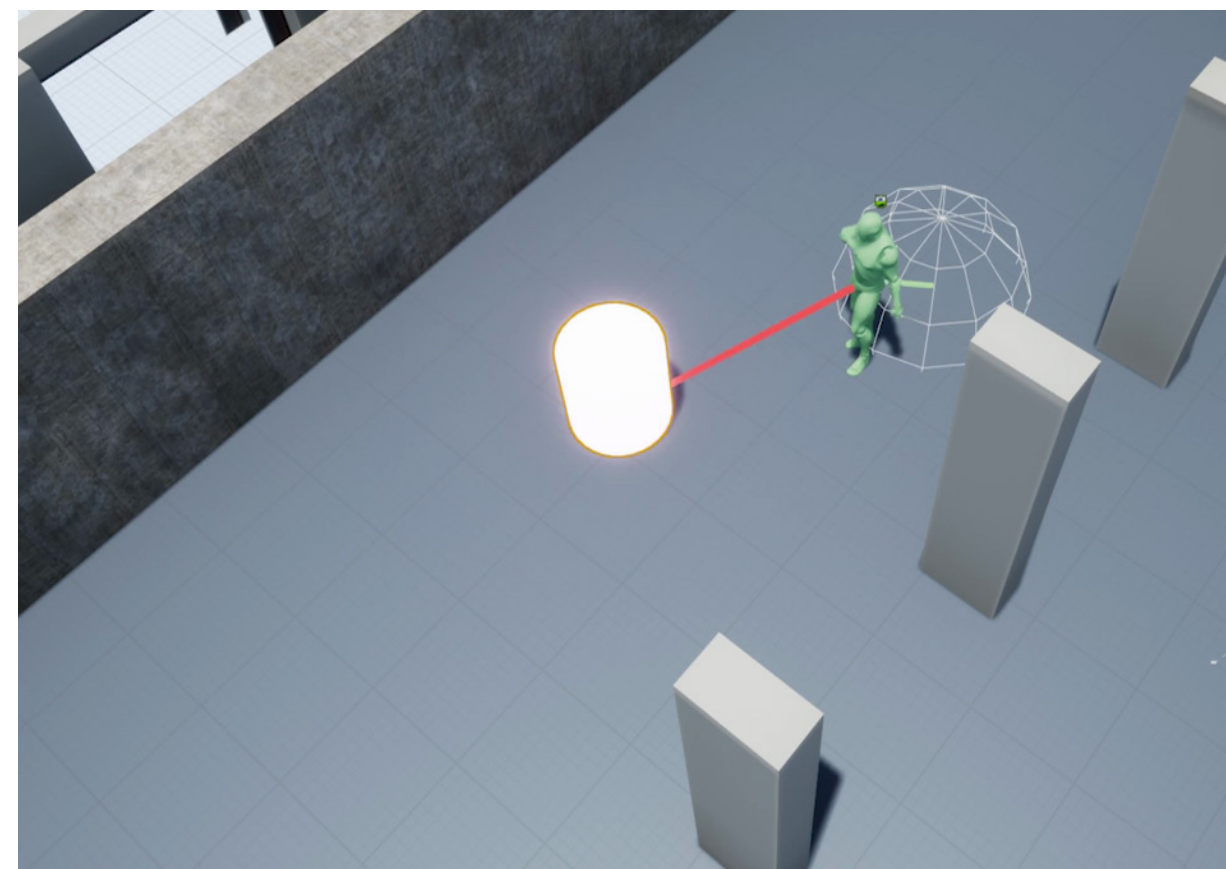


Figure 3.3.18 Behavior Tree Object-Looking state tasks

¹² Arnheim, *The Dynamics of Architectural Form*, 20, 28.



► **Figure 3.3.19** EQS also allows us to calculate a random location that is biased to how far away they are from an object



► **Figure 3.3.20** Agents 'juggling' the distances to the objects until they look right

Object Interact State

This state defines what the agent does when they choose to interact with an object. This process undoubtedly depends on the typology of the object; if the object is a book, the agent might go and read it; if the object is a touch screen, the agent might go and touch it; if the object is a chair, the agent might go and sit on it.

In all these cases, we would need to break down these tasks into their individual logical actions. With the chair example, the agent would first need to calculate the location of the chair. It would then need to walk to that location and align its body with the chair. Only when all these steps are taken would the agent be able to sit on the chair. Once the agent sits down, it may then have a randomized range of time before the agent decides to get up again. Translating this then into machine logic, we would have the following steps:

1. Get location of the front of the object.
2. Walk to location.
3. Align agent body to the chair.
4. Play sitting down animation.
5. Play sit animation and loop it for a randomized time interval range.
6. Play standing up animation.
7. Set *AgentState* to *Default* State.

Naturally, we would require a version of this state for every typology of an object within the simulation. Interacting with a bookshelf for example would require the agent to calculate a location in front of the bookcase, turn to it, and choose a book. The agent then might quickly skim the book before choosing another book and continue to do so until the agent finds the right book or becomes uninterested.

8. Get random location in front of the object (depending on the size of the bookcase; this can vary a lot)
9. Walk to location.
10. Turn towards the bookshelf.
11. Play retrieve book animation.
12. Play reading animation and loop it for a randomized time interval.
13. Play animation for keeping the book OR putting the book back on bookshelf (depending on what the agent chooses).
14. Set *AgentState* to *Default* (or another task) OR repeat step one to find another book from the bookshelf.

Ordering food at a counter on the other hand, would require the agent to first check if there is a lineup. If there is, the agent would need to find the location of the end of the line and move there. It would then have to calculate a new position each time the line updates until it reaches the front of the line. Once it is the agent's turn, it would then need to define the location of the counter, move to it, and start playing an animation for portraying the action of ordering food.

1. Check if there is a lineup for the counter.
2. Get location depending on if there is a lineup.
 - 2a. If there is a lineup, get the location for end of line
 - 2b. If there is no lineup, get the location of the counter.
3. Walk to location.
4. Play Animation depending on if there is a lineup or not.
 - 4a. Play wait animation and check once per second if the line is moving forward. If moving forward, update location for the agent to walk to. If the agent is at the start of line, and the counter is free, go to step 2b.
 - 4b. If there is no lineup, and the agent is at the counter, play food ordering animation and loop it for a randomized time interval.
5. After the agent finishes ordering, set *AgentState* to another task (such as going to food pickup or finding a table).

As it can be seen, all these scenarios follow a similar logic, in which the agent requires a *location*, a *direction*, and an *animation*. The use of these three attributes within the behavior tree provides us with an intuitive method of portraying these agents performing various tasks. With this, we can set up a series of nodes that can accommodate any type of object interactivity within the simulation environment. (**Fig. 3.3.21**) It is of course unrealistic to account for every scenario within the scope of this thesis, therefore the best course of action at this stage of tool creation is to establish a generic base object with definable parameters that works with a wide range of use cases and can be easily modified depending on future use cases and object typologies.

In order to do this, we need to first consider some basic parameters that can describe both the most common forms of object interactivity, as well as the most influential forms on crowd movement. With this in mind, we can logically break down these parameters into the ability to touch, the ability to sit, and the need for queuing (lining up). The ability to touch would be the default form of interaction, as this can be modified to be any other form of interaction (such as typing, taking, painting, pushing, etc.) by changing its animation and looping timeframe. The ability to sit is perhaps the most common form of interaction amongst varying spatial typologies. Since rest is such an important human need, the presence of chairs can usually be found no matter what type of space is being designed. The need for queuing comes into play when there is overdemand on a specific object. This can be seen in scenarios such as checking tickets at entrances, ordering food at counters, talking with bank tellers, etc. While this won't be relevant in every space typology, it is common enough in public spaces to warrant its consideration. Translating these into the behavior tree, we can utilize the following steps: (Fig. 3.3.22 - 24)

1. Check to see if the object can be sat upon, or if there is a lineup required.
2. Define location based on object type.
 - 2a. If the object cannot be sat upon, and does not have a lineup, Utilize EQS to define a location around the object.
 - 2b. If the object functions as a seat, define the location in front of the seat.
 - 2c. If there is a lineup, define the location at the end of the lineup.
3. Walk to location.
4. Set rotation direction of the agent based on object type.
 - 4a. If the object cannot be sat upon, and does not have a lineup, turn towards the object.
 - 4b. If the object functions as a seat, align the agent body to the seat.
 - 4c. If there is a lineup, turn towards the start of line.
5. Perform animation based on object type.
 - 5a. If the object cannot be sat upon, and does not have a lineup, play touch animation
 - 5b. If the object functions as a seat, play sitting animation
 - 5c. If there is a lineup, play waiting animation. The agent might also need to continuously update its position as the line moves forward.
6. Wait for randomized time interval based on object type.

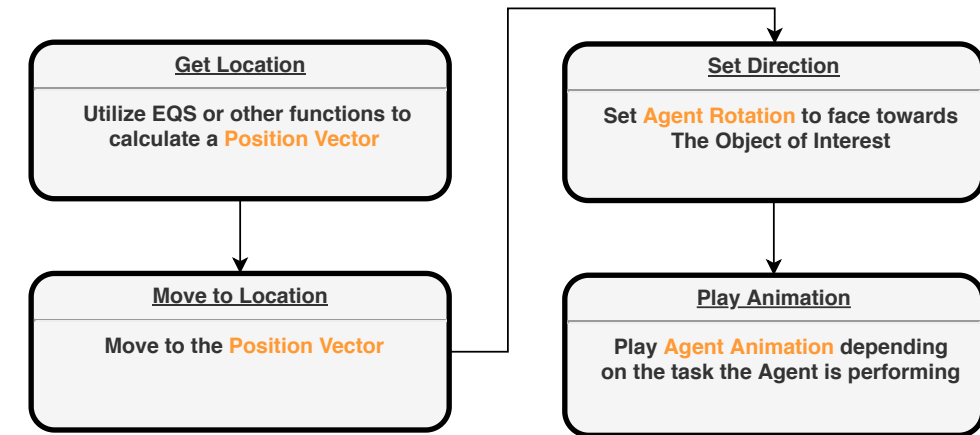


Figure 3.3.21 Simplified steps for establishing object Interaction tasks

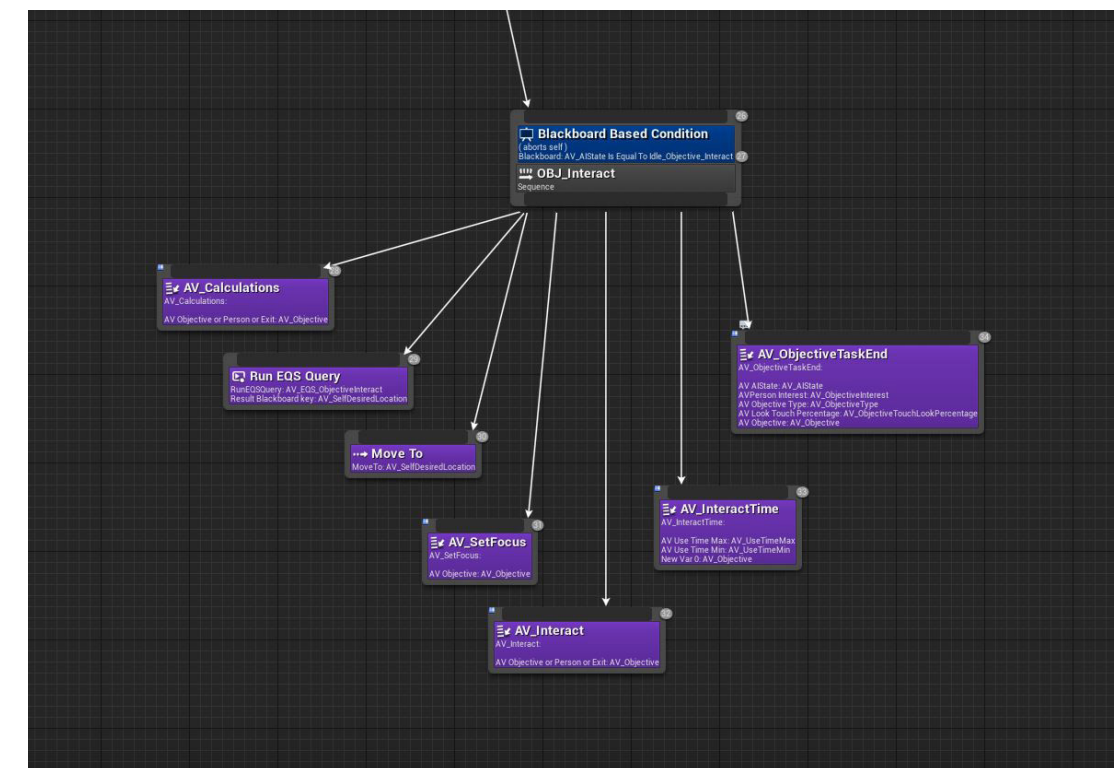
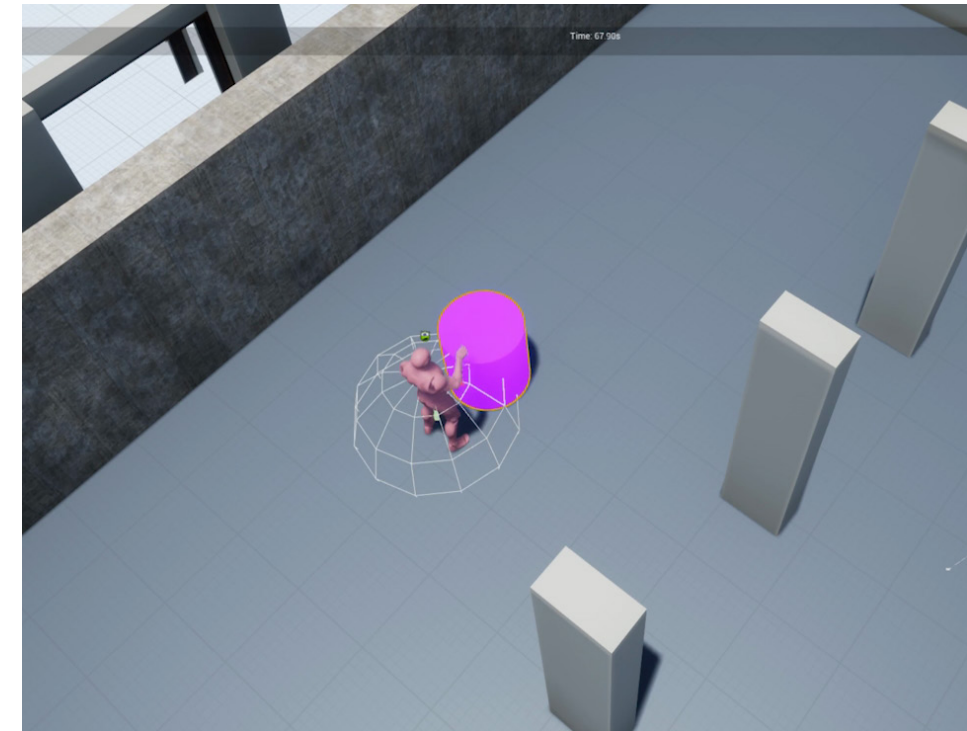


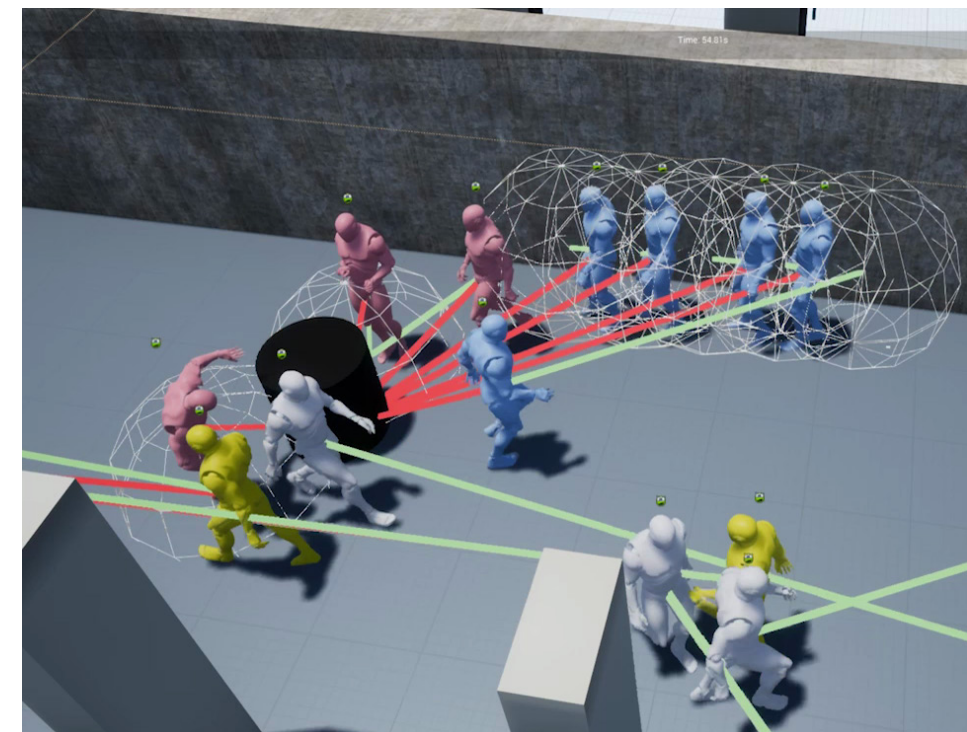
Figure 3.3.22 Object-Interact State tasks within behavior tree

- 6a. Touch based objects can vary in length based on the object typology (e.g. a simple button will take less time to operate compared to navigating a touchscreen to fill out a waiver)
- 6b. A seat can vary in length depending on how tired the agent is or if the agent is waiting for someone or watching something.
- 6c. A lineup can vary in length depending on the space. A line at the DMV will probably move slower than a line at a grocery store.
7. Repeat task or change to new AgentState.
 - 7a. The agent might touch it in 1 spot, then move to another spot to touch it again, or it could be done with the task and do something else, in which case we can set AgentState to something else.
 - 7b. When the agent is finished resting, it may get up from the chair by playing an animation, and then go do another task.
 - 7c. When the agent reaches the front of the line, it can then go back to step 2a, where it will finally be able to interact with the object.

While these three branches won't cover every scenario within the behavior tree, we can always modify or add to them in the future if the need arises. Since these steps depend largely on the typology of the object however, we will also need to create such an object—which we will establish in Chapter 3.4—and update this state depending on the requirements of the object.



▶ **Figure 3.3.23** Agent interacting with an object



▶ **Figure 3.3.24** Agents lining up before interacting with the object

Agent Interact State

The agent interact state defines what happens when an agent decides to interact with another agent. The methodology for this is similar to what we just established within object interact state, except we now must consider these steps in the context of human interactions.

When an agent decides to interact with another agent, this interaction could either be with someone the agent knows (an *acquaintance*) or have never met (a *stranger*). As such, we can logically deduce these interactions into these two basic types. When the agent wants to interact with a stranger, they might first need to walk up to them before engaging in conversation. However, when the agent wants to interact with an acquaintance, they might both recognize each other and meet halfway. To simply this into its fundamental actions, we can proceed with the following steps: (Fig. 3.3.25 - 26)

1. Check if the other agent is facing towards or away the agent (by using the dot product).
2. Define location depending on the direction of the other agent.
 - 2a. If facing away, move to the location of the other agent.
 - 2b. If facing towards, define random location between the two agents.
3. Get attention of the other agent by setting the *AgentState* of the other agent to *AgentInteract* (with an optional tap animation).
 - 3a. If facing away, get their attention after walking to them.
 - 3b. If facing towards, get their attention before walking to the random location between the two agents.
4. Loop talking animation for a random time interval.
5. After finished talking, play goodbye animation.
6. Set both agents' *AgentState* to *DefaultState*.

While this method might imply that the people facing away are presumed to be strangers, and people facing towards are acquaintances, the fact that the location generated in between the two agents are random in step 2b means that there is a gradient of possible meeting locations between the agent. This gradient then can portray both acquaintances meeting in the middle, or the agent going up to a stranger to talk, or the agent signaling a stranger or acquaintance to come to them, or anything in between.

Of course, if other senses, such as sound, were added, we would then need to introduce another option, where the first agent might shout at the other agent to get their attention, in which the other agent would then turn around and they can meet at a random location between them. This being said, to keep the simulation simple at this stage, we will mainly be utilizing sight.

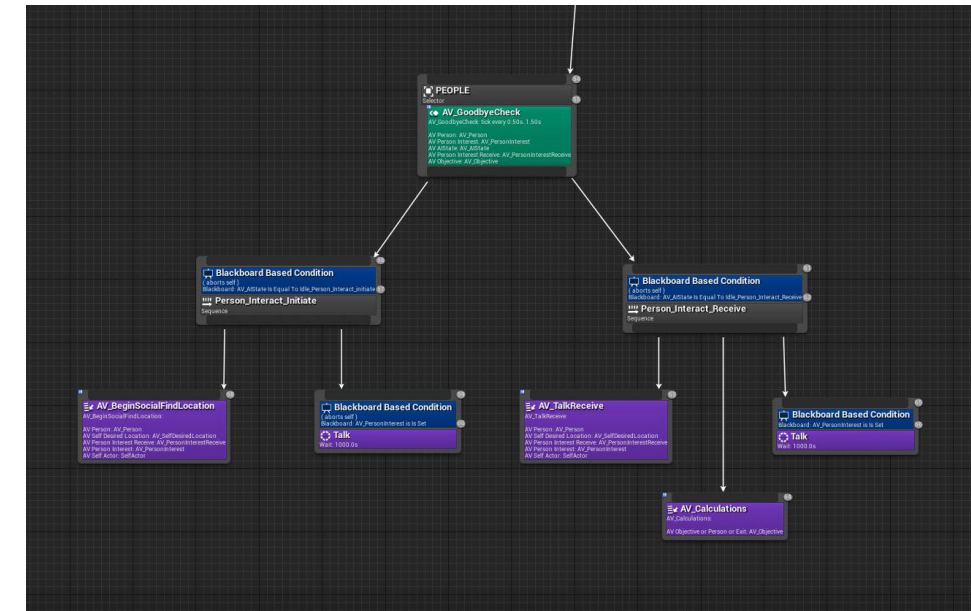


Figure 3.3.25 Agent Interact state tasks within Behavior Tree

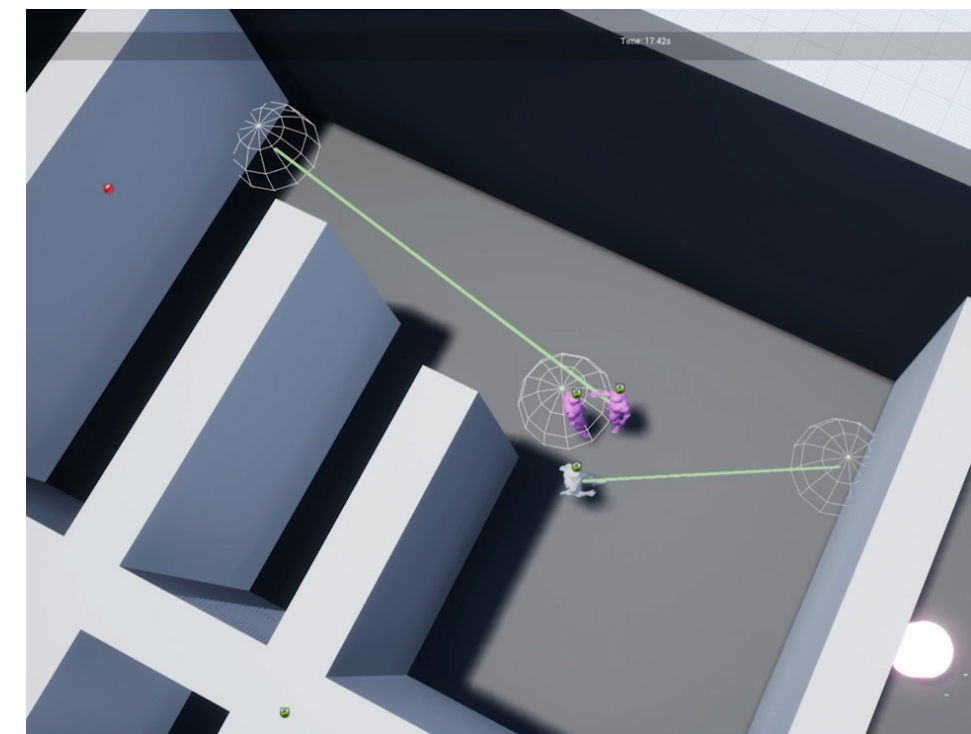


Figure 3.3.26 Agents interacting with each other

Threshold State

While *explore* provides a good generic way for agents to explore the space, we should remember that there are other forms of attractors other than distance. To accommodate for this, we can define another class of objects that represents thresholds. These objects would act as doors or openings that offer attraction to passing agents as they see it, which allows slightly more control in diversifying the crowd movements within the simulation space.

The series of actions required for this state can be thought of as a simplified version of the object interact state, as the agents are essentially *interacting* with the door or threshold. As such, we only require a slight modification in animation and position when compared to the object interact state. To create a series of actions for this then, we can define the following steps: (Fig. 3.3.27 - 28)

1. Check to see if the threshold is open or closed (if there is a door built into the opening)
2. Calculate location depending on if there is a door or not.
 - 2a. If there is a door that is closed, define a random location on the same side of the threshold.
 - 2b. If there is no door or if the door is open, define a random location on the other side of the threshold.
3. Go to location.
4. Play animation depending on if there is a door or not.
 - 4a. If there is a door, turn towards the door and play open door animation. Then go back to step 2b.
 - 4b. If there is no door or if the door is open, proceed to step 5.
5. Change *AgentState* to *DefaultExplore*.

With these steps, the agent will walk through the threshold on to the other side and continue to explore the space beyond. If there is a door in the way, the agent will then proceed to open the door first before walking through to the other side.

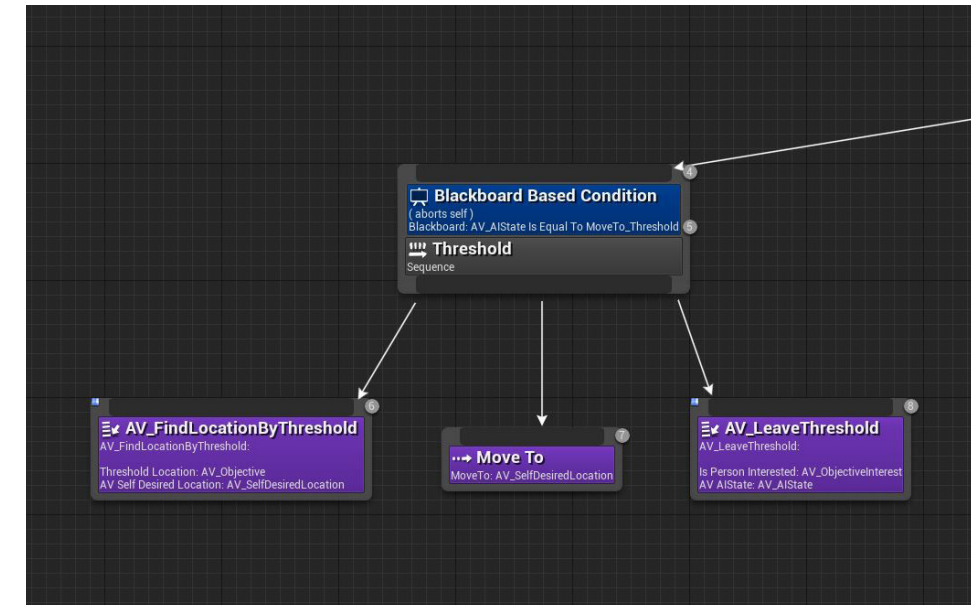


Figure 3.3.27 Threshold State tasks within Behavior Tree

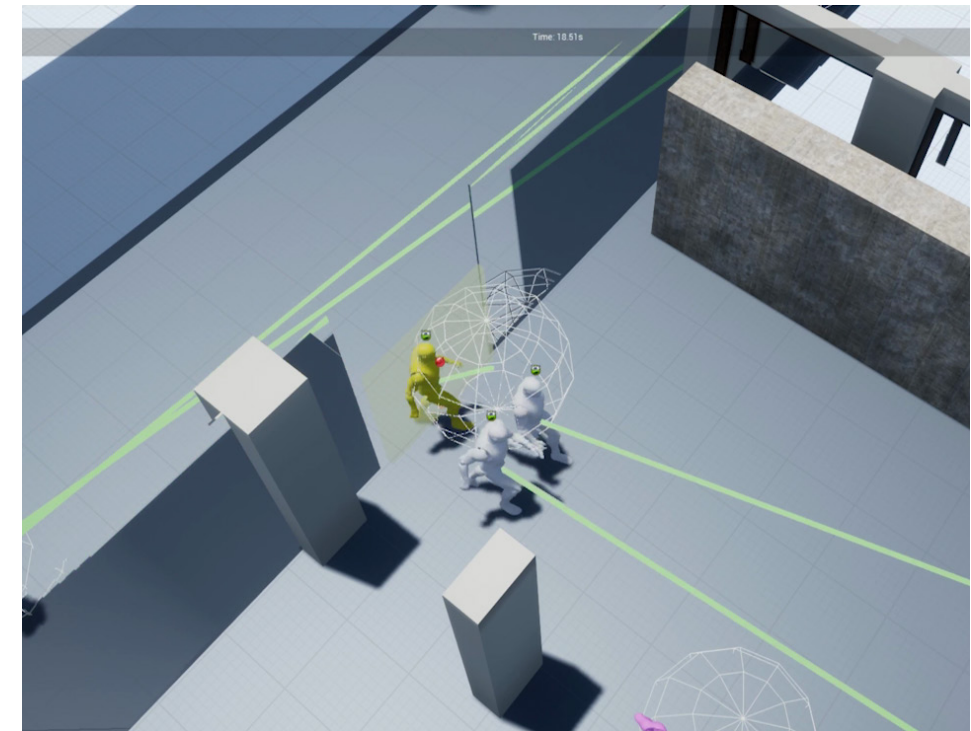


Figure 3.3.28 Agent moving through threshold

Enter/Exit State

Since most spaces are not closed systems, we will need a way for the agents to enter and leave. As such, entrances and exits need to be defined within the simulation to establish the crowd flow within the space. Because these entrances and exits are essentially a type of threshold however, we can use the same methodology defined in *ThresholdState*, with the addition of *spawning the agents* in the case of *entering*, and *de-spawning the agents* in the case of *exiting*. (Fig. 3.3.29 - 30) From this logic, we can establish the following steps:

Enter State

1. Spawn agent along the entrance threshold.
2. Check if threshold has a door or not
3. Calculate location depending on if there is a door or not.
 - 3a. If there is a door, play door opening animation, then to go step 2b.
 - 3b. If there is no door, set agent rotation to face perpendicularly away from the entrance threshold.
4. Set AgentState to *DefaultExplore*.

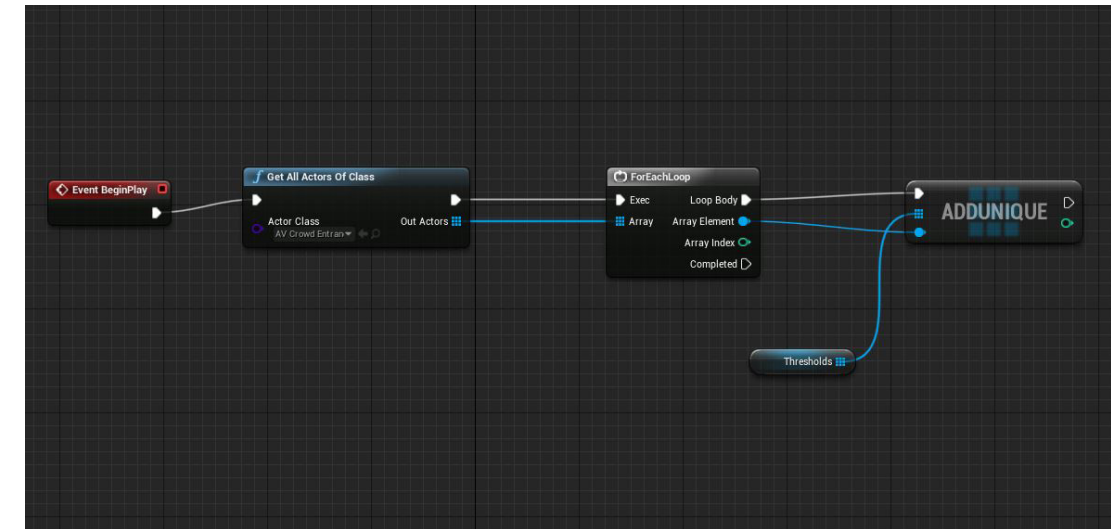


Figure 3.3.29 To spawn Agents, we must first make a list of all the Entrance Threshold objects at the start of the simulation

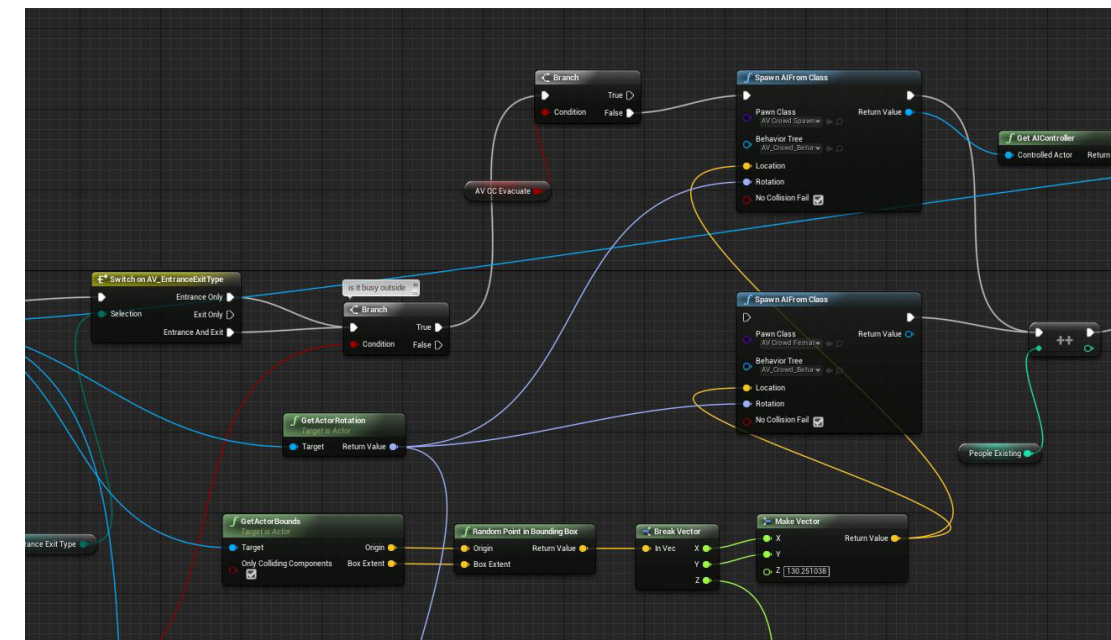


Figure 3.3.30 Spawning Agents at Entrance Thresholds

Exit State

5. Check to see if the threshold is open or closed (if there is a door built into the opening)
6. Calculate location depending on if there is a door or not.
 - 6a. If there is a door that is closed, define a random location on the same side of the threshold.
 - 6b. If there is no door or if the door is open, define a random location on the other side of the threshold.
7. Go to location.
8. Play animation depending on if there is a door or not.
 - 8a. If there is a door, turn towards the door and play open door animation. Then go back to step 2b.
 - 8b. If there is no door, proceed to step 5.
9. De-spawn agent.

With these two states, the agents can now enter and exit the simulation. (Fig. 3.3.31 - 32) In the case of evacuation however, the agent must also remember the location of the last threshold they've encountered, which can be recorded within their AI controller as an object array. In such a case, the agent would simply need to pull up the array to obtain the location, and then execute the steps from ExitState.

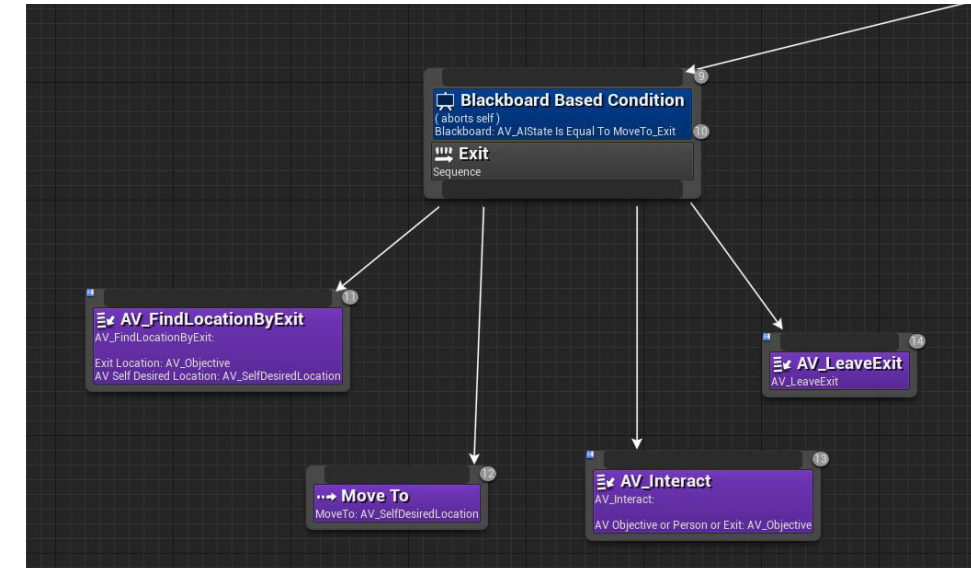


Figure 3.3.31 Enter State tasks within the Behavior Tree

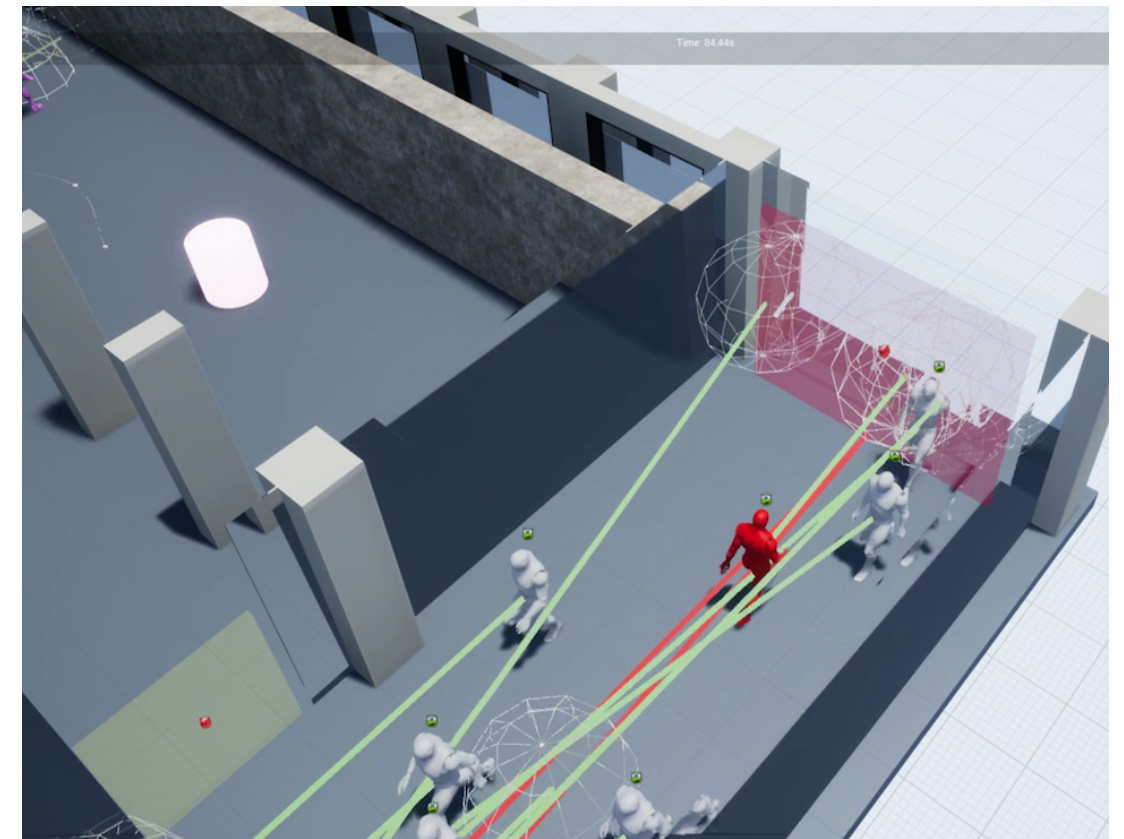


Figure 3.3.32 Agents entering and exiting the building

Unstuck State

Since all these states depend on a series of logic to function, there are instances where the agents might become stuck within the environment. For example, an agent within the *ExploreState* might become stuck in a corner due to the EQS continuously calculating the furthest point in front of the agent. (Fig. 3.3.33) Another example might be an EQS query establishing a location that is unavailable to the agent, in which case the agent might freeze on the spot as it does not know how to get there through the pathfinding system. There are certainly many more instances when the agents might become stuck, so rather than trying to debug every instance, we can simply utilize a state for this. To establish this unstuck state, we can perform a check every few seconds based on the agent's location and their desired location. If these locations are unchanged for too long when the agent is not interested in an object or agent, we can tell the agent to recalculate a location near them that is valid and reset their state to *DefaultExplore* with this new location. (Fig. 3.3.34)

Because of the higher abstraction of this tool, we do not need to worry about the technical aspects of teaching the agents how to walk for each task—since this is already taken care of with our new pathfinding system—but rather just focus on the sequence of individual actions that they need to do within their respective tasks. This allows for a quickly implementable and relatively intuitive way to add agent actions to establish them for future use as well. Now that we have established the main states, we can put them together within a functioning behavior tree. (Fig. 3.3.39)

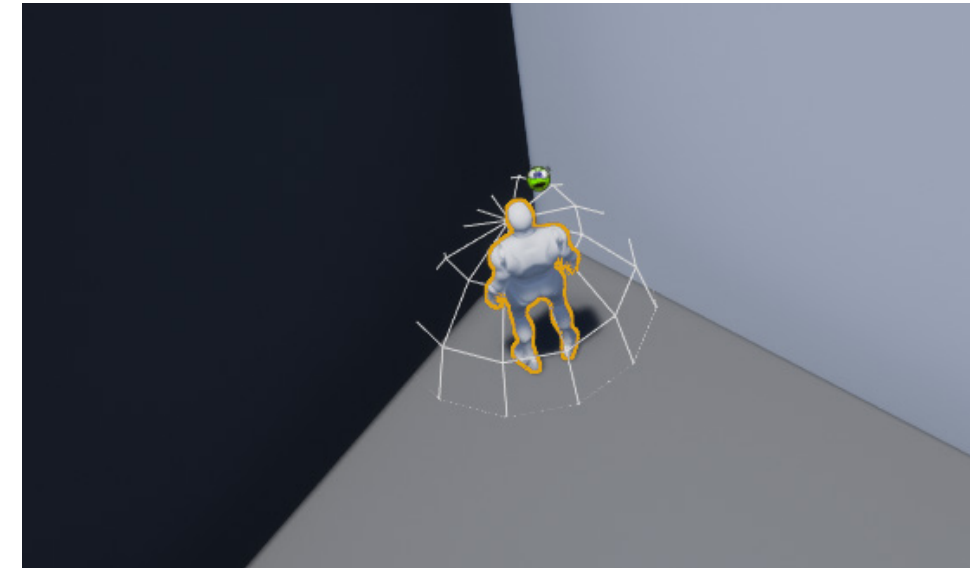


Figure 3.3.33 Agents sometimes becomes stuck in the corner due to the EQS continuously perceiving the corner in front of the agent to be the furthest point within the environment

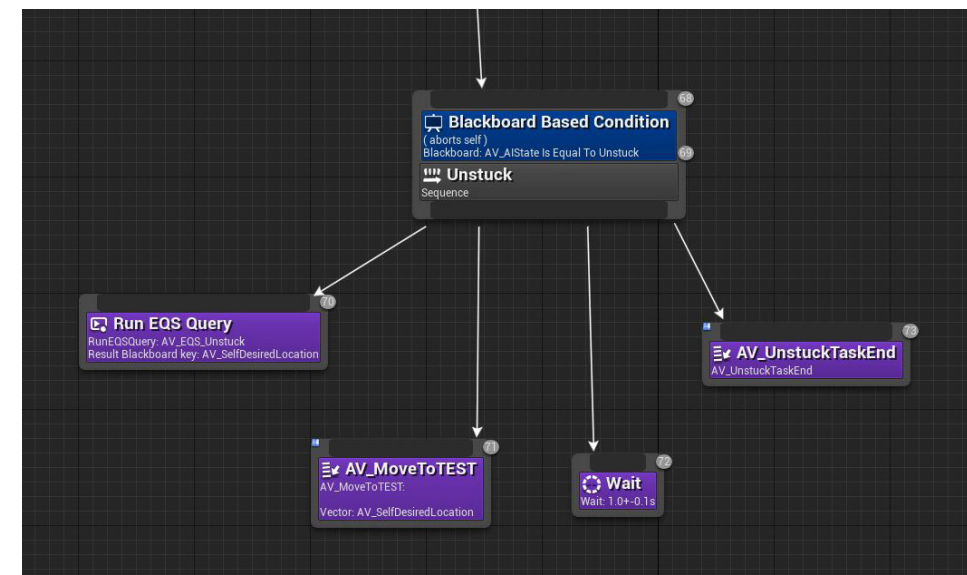
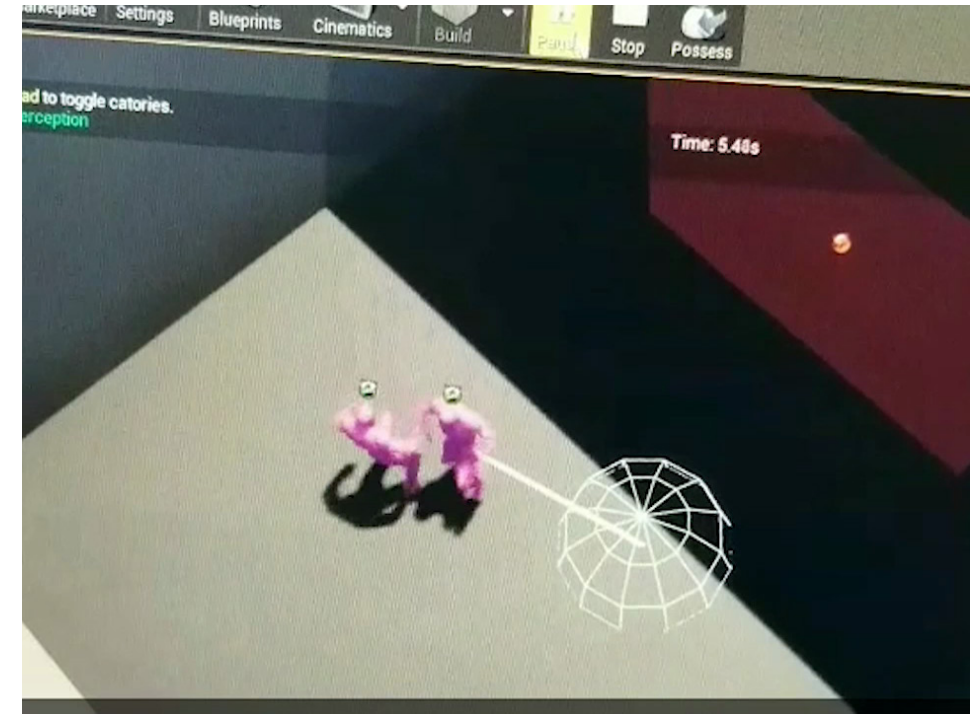


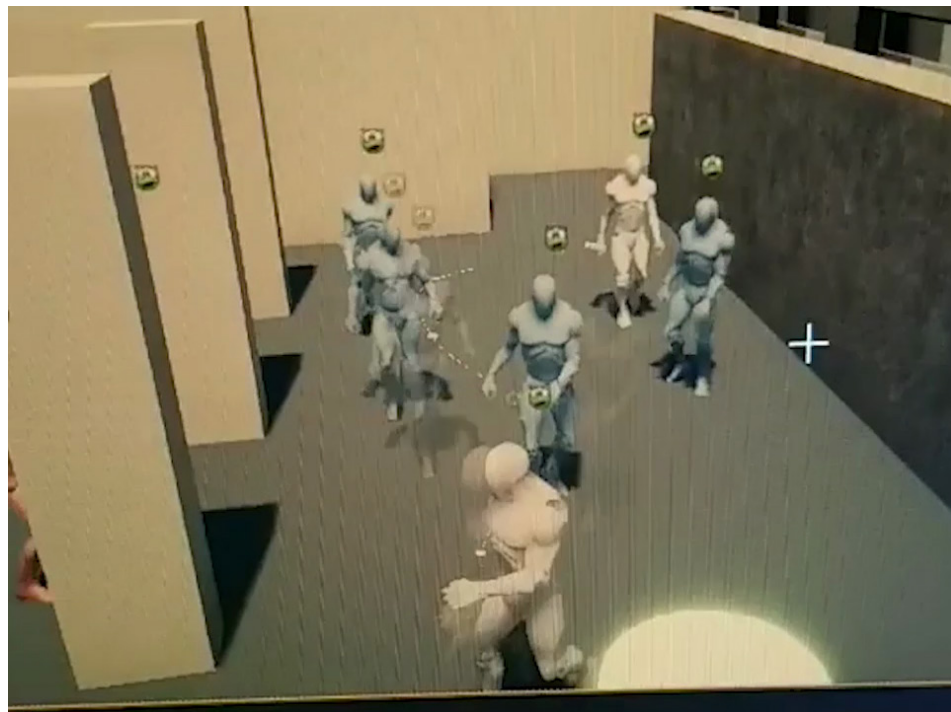
Figure 3.3.34 Unstuck State within Behavior Tree tells the agent to query a new location behind them if they don't move for longer than a specified time interval



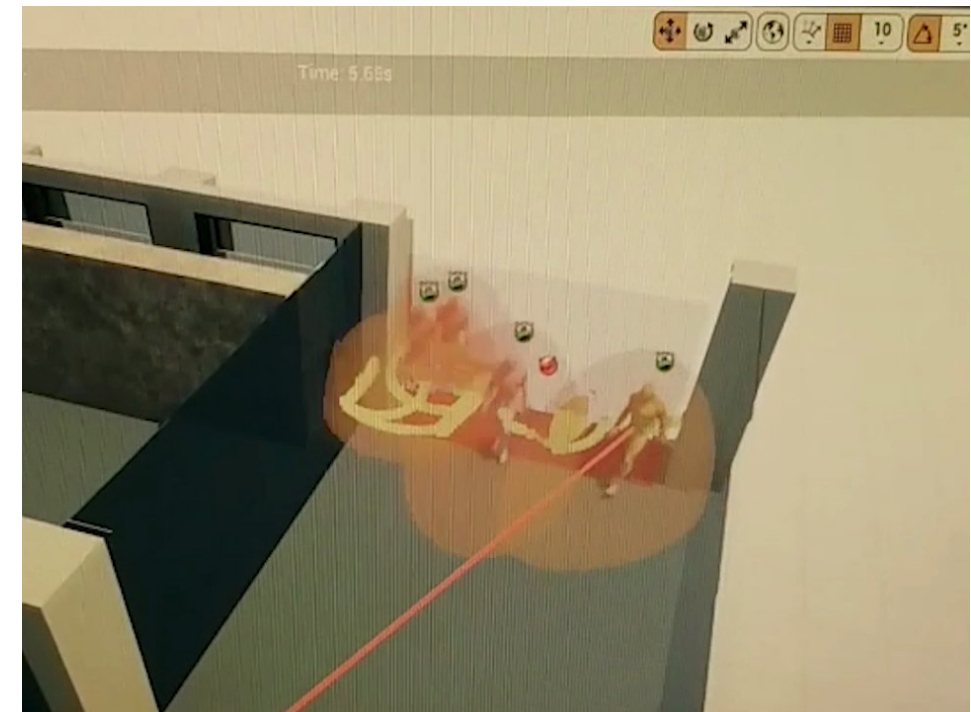
▶ **Figure 3.3.35** Bug with obtaining a position vector for establishing agent global offset



▶ **Figure 3.3.37** Bug with animation looping for the Agent Interact State



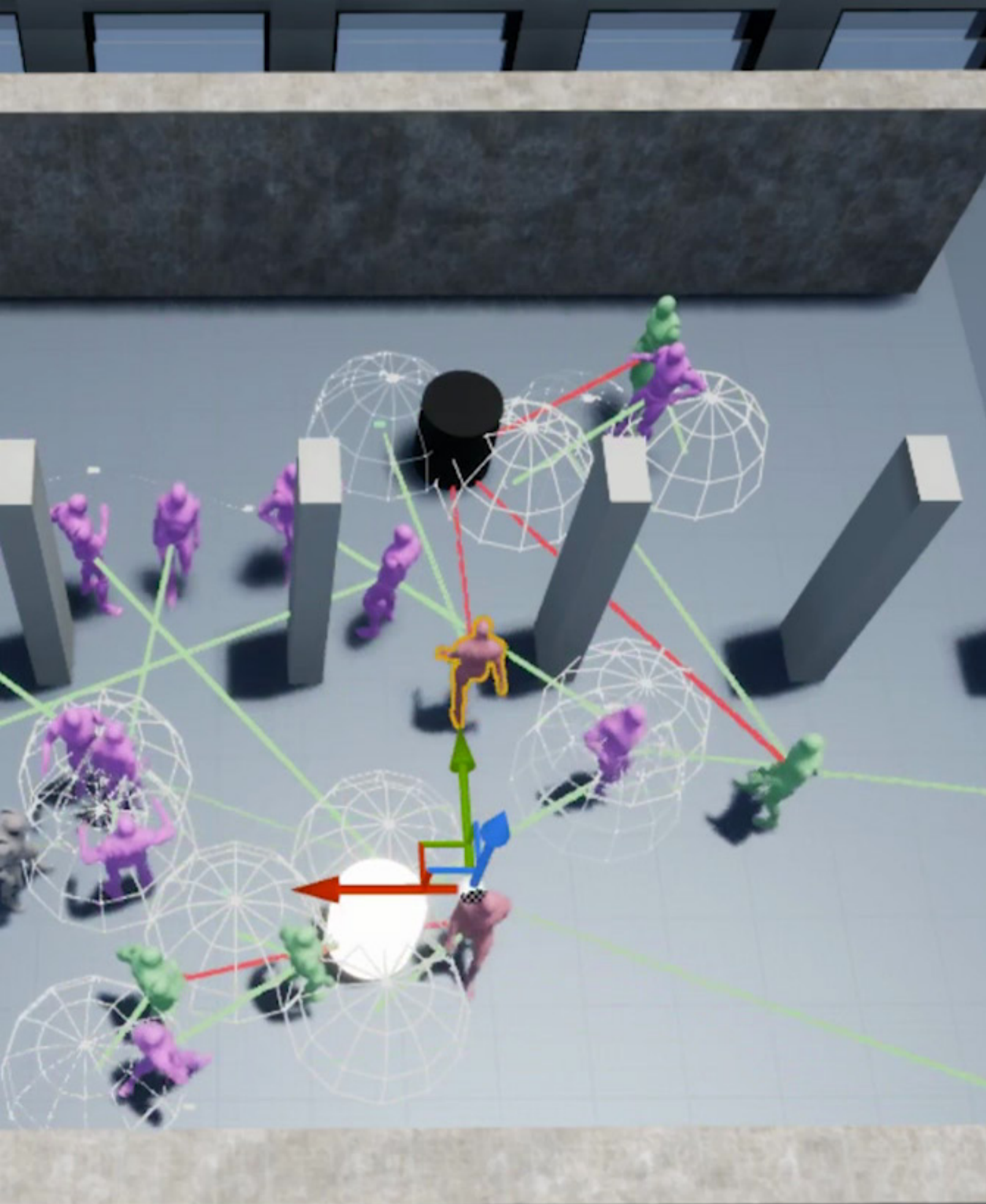
▶ **Figure 3.3.36** Bug with obtaining a position for the agents to form a line



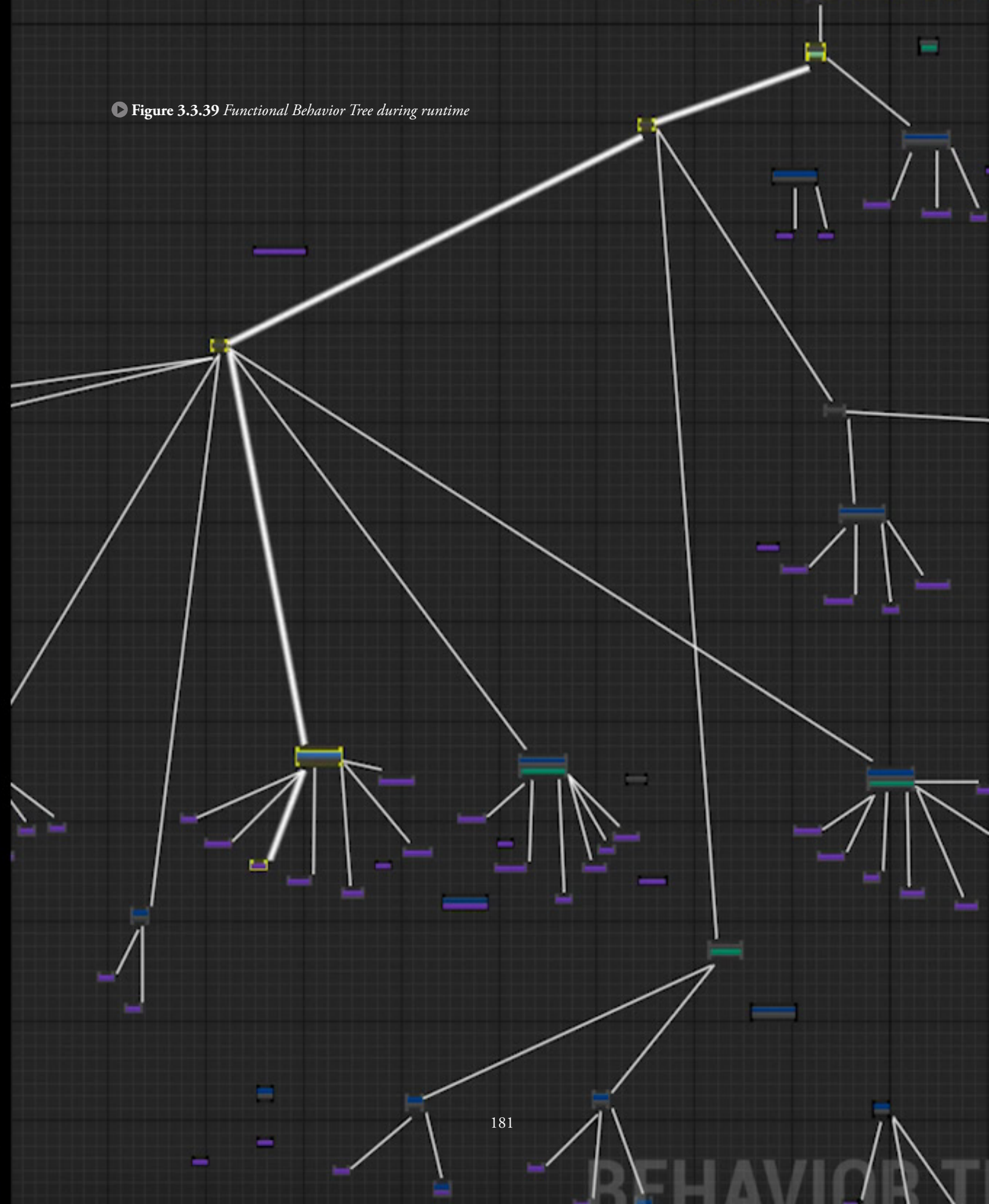
▶ **Figure 3.3.38** Another global offset bug when establishing a position vector from the agent's personal space

Figure 3.3.35 - 3.3.38

These figures showcase some of the various bugs that are encountered during the prototyping phase. This further demonstrates how prototyping can be an unpredictable time-sink since most of these bugs appear at random and thus becomes hard to calculate the time-frame it would take to fix them.



▶ Figure 3.3.39 Functional Behavior Tree during runtime



Character Model and Animation

Beyond the additional consideration of *AgentStates*, we must also consider the physical form of these agents, or rather a virtual description of their physical form. To do so, additional assets will be required to define how these agents look as they are moving throughout the environment. This can be broken down into skeletal assets and animation assets within UE4.

Skeleton Assets

Skeleton assets within UE4 are comprised of the skeletal mesh and the skeleton. This can be described as a set of “meshes bound to a hierarchical skeleton of bones which can be animated for the purpose of deforming the mesh.”^[13] Put simply, the surface of the skeletal mesh asset is responsible for the visual representation of the agent, whereas the skeleton asset is responsible for translating animation data to this surface mesh. (Fig. 3.3.40)

These assets can be obtained in many ways. UE4 comes with a ready-to-use solution in the form of a generic humanoid robot model (Fig. 3.3.43); however, they can also be created within external applications such as 3ds Max and Maya or be purchased and downloaded through various asset stores such as Unreal Marketplace^[14] and CGTrader^[15].

Of these external applications, one tool that may be worth investigating further is Autodesk’s web-based character generator.^[16] (Fig. 3.3.41) This tool allows us to generate a variety of generic character models based on their height, gender, facial and body features, and clothing. With this, we can quickly export a variety of model assets and create a library of people to spawn within the simulation space. While these models are not particularly detailed, they are perfectly suited to blend into a simulated human crowd where the agents are meant to represent normal everyday people. From here, we can then import these models as an FBX file and implement them within the character asset.^[17]

One thing to keep in mind with these models, however, is the hardware limitations present within a multiagent simulation system, where higher model detail will increase rendering time and slow down the performance of the overall visualization. For this reason, I have decided to stay with UE4’s default robot model to prioritize performance, with the intention of utilizing more detailed model assets in the future. Once we have access to faster hardware and a more optimized simulation code, it may become possible to utilize these more detailed model assets in not only final rendering outputs, but in the design phase as well.

13 “Skeletal Meshes,” Unreal Engine Documentation, accessed October 18, 2019, <https://docs.unrealengine.com/en-US/Engine/Content/Types/SkeletalMeshes/index.html>.
 14 “Characters,” Unreal Engine Marketplace, accessed October 18, 2019, <https://www.unrealengine.com/marketplace/en-US/content-cat/assets/characters>.
 15 “UASSET 3D models - download UnrealEngine (UASSET) file format 3D assets,” CGTrader, accessed October 18, 2019, <https://www.cgtrader.com/3d-models/ext/uasset>.
 16 “Autodesk Character Generator,” Autodesk, accessed October 18, 2019, <https://charactergenerator.autodesk.com/>.
 17 “Setting Up a Character,” Unreal Engine Documentation, accessed October 18, 2019, <https://docs.unrealengine.com/en-US/Engine/Animation/CharacterSetupOverview/index.html>.

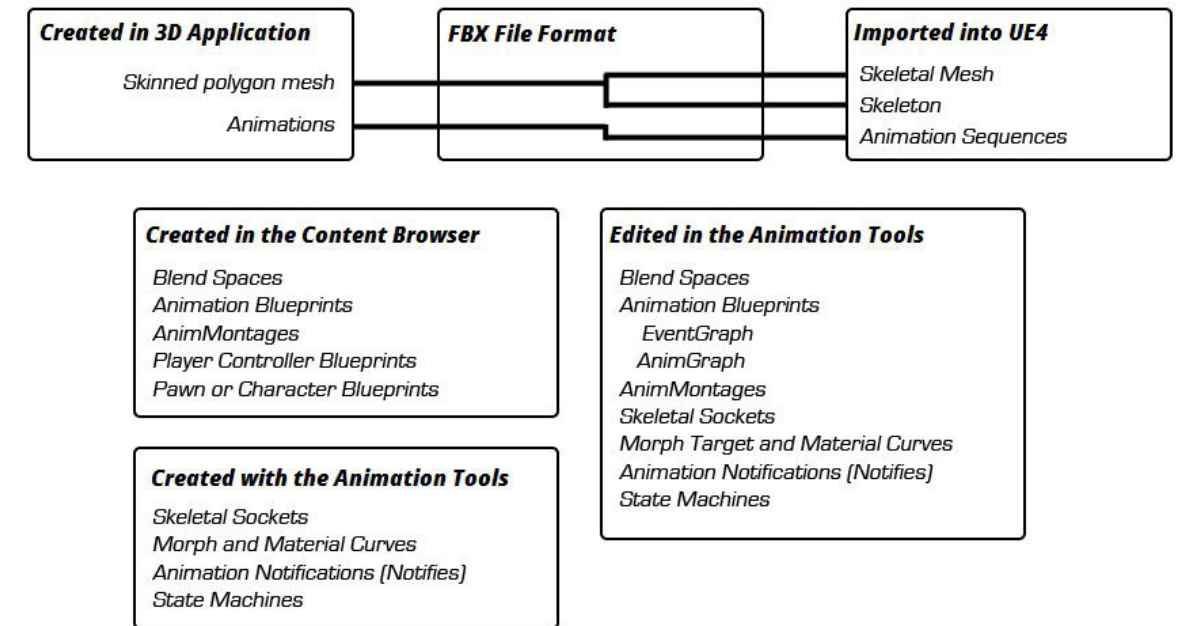


Figure 3.3.40 UE4 animation system breakdown

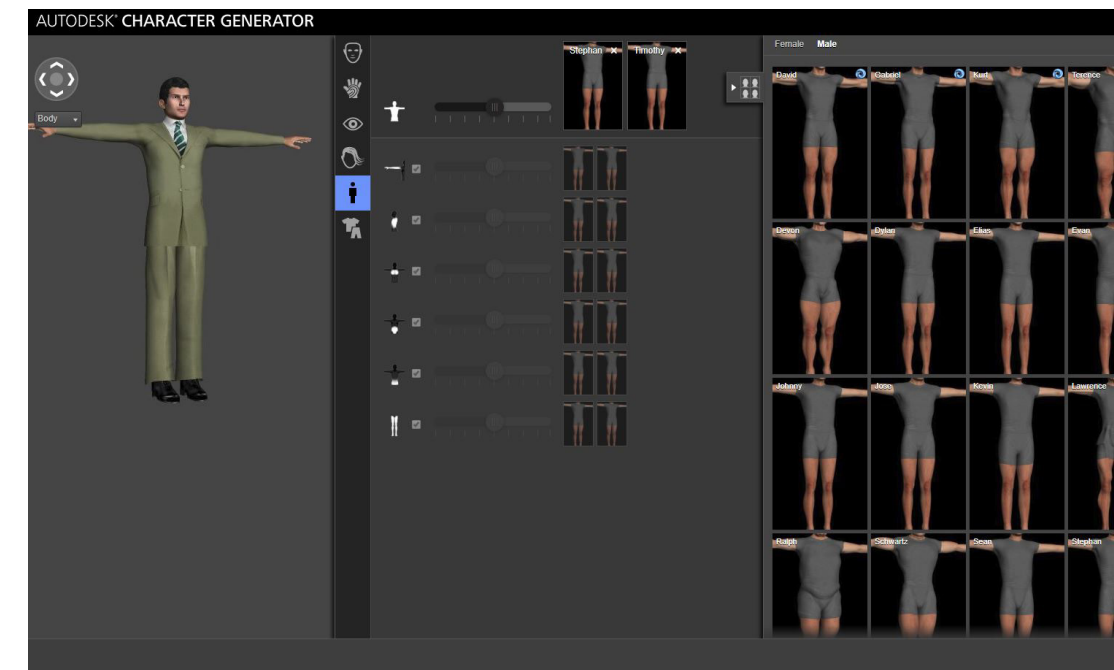


Figure 3.3.41 Autodesk Character Generator

Animation Assets

Beyond the skeletal mesh, animations are also required to fully portray these agents. Within the simulation visualization, these animations are what allows these human agents to visually differ from dynamic architectural elements shaped like humanoids. As such, without this step, the agent models will be lifeless objects that float about within the simulation. (Fig. 3.3.42)

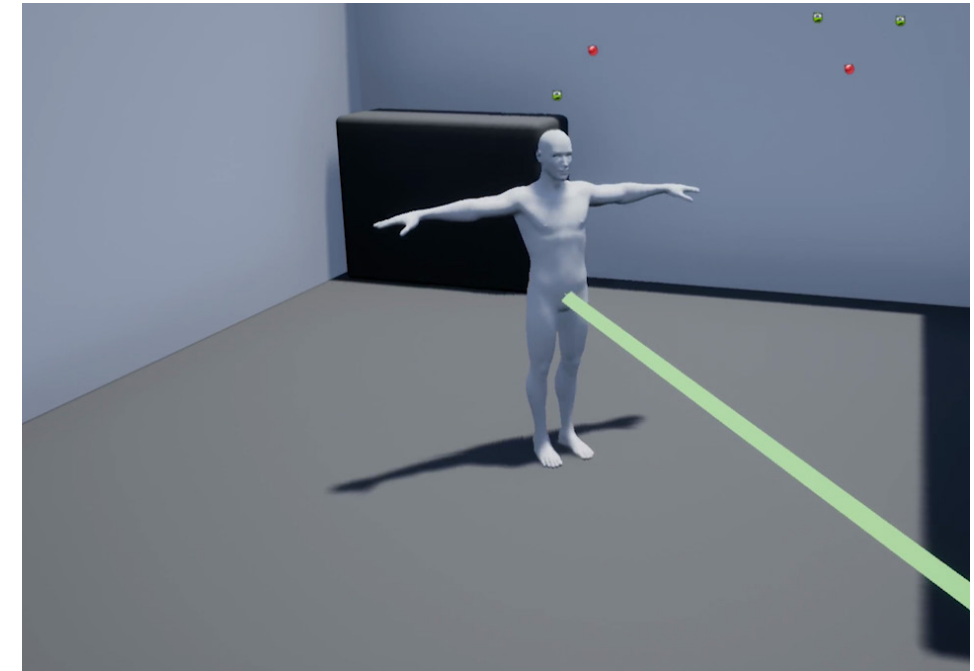
Within UE4, a single animation asset is referred to as an *animation sequence*. These sequences contain “keyframes that specify the position, rotation, and scale of a bone at a specific point in time.”^[18] This allows these assets to drive the movement of the skeleton asset, which in turn drives the movement of the skeletal mesh asset. (Fig. 3.3.43) To implement these animations for the agents, we must procure a library of various animation sequences for every possible scenario. We can then utilize various tools within UE4 to set them up in a way where the agents can determine which animation to execute based on the task that the agent is currently doing.^[19]

The first of these tools is the Blend space. This allows us to blend various animations together based on the value of multiple inputs.^[20] With this, we can combine this collection of animation sequences to create transitional animations between different states, allowing the agent to move in a smooth and realistic fashion without having to use too many hard-coded animation sequences. (Fig. 3.3.44)

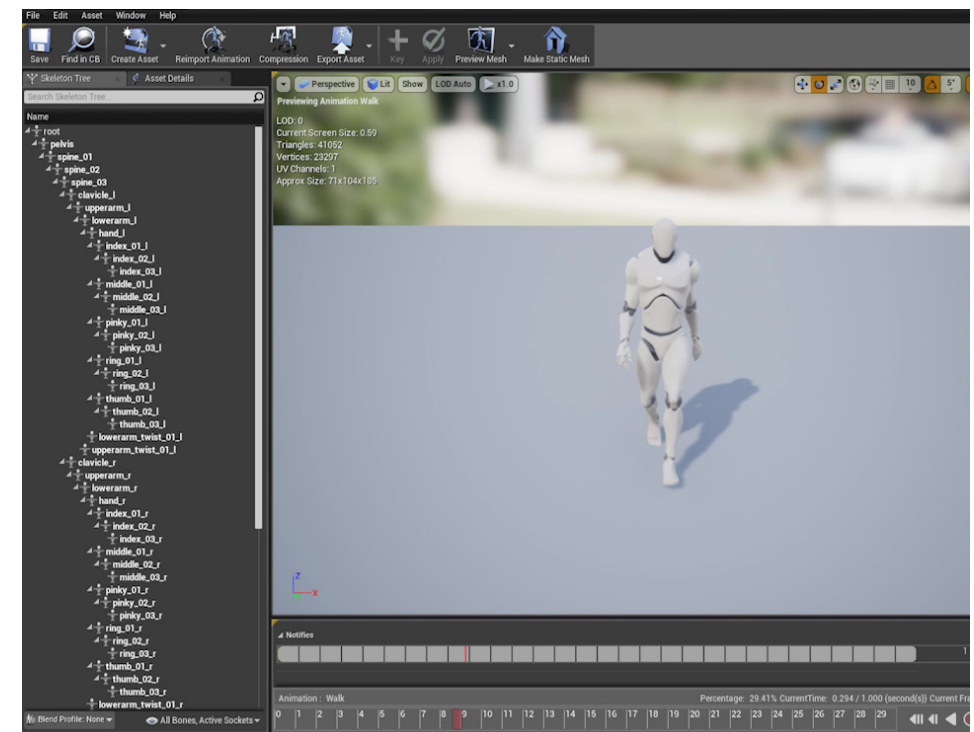
In order to set these states, however, we must also utilize animation blueprints. With this, we can establish various checks to determine if the agent is currently turning, running or falling, etc. Once this is set up, we can produce output variables to set various states within a State Machine. (Fig. 3.3.45) This then allows the agents to execute various animations or Blend Spaces depending on what the agent is physically doing within the simulation. For example, if the agent is currently turning right, the animation blueprint will check for this and set a variable for *AgentTurningRight* to be true. Once this is true, the state machine will go into the *TurningRight* State, which will tell the agent to use the *TurningRight* animation sequence. If the agent is speeding up from a walking speed to a running speed, the animation blueprint can normalize the speed between the two states to tell the Blend Space to generate an animation that smoothly transitions from the walking state to the running state.

Much like the character models, there are many ways to create these animation sequences, which can include manual keyframing or motion tracking.^[21] To simplify this process, however, it is possible to download various animation files from sites such as Adobe Maximo. (Fig. 3.3.46) With this, we can quickly procure the library of animation sequences required to fully animate our agents within their tasks.

- 18 “Animation Sequences | Unreal Engine Documentation,” Unreal Engine Documentation, accessed October 18, 2019, <https://docs.unrealengine.com/en-US/Engine/Animation/Sequences/index.html>.
- 19 “Animation System Overview,” Unreal Engine Documentation, accessed October 18, 2019, <https://docs.unrealengine.com/en-US/Engine/Animation/Overview/index.html>.
- 20 “Blend Spaces,” Unreal Engine Documentation, accessed October 18, 2019, <https://docs.unrealengine.com/en-US/Engine/Animation/Blendspaces/index.html>.
- 21 Unreal Engine, “Real Time Motion Capture in Unreal Engine,” YouTube, 1:05:18, accessed October 18, 2019, <https://youtu.be/jRyq5uPC5UY?t=1066>.



▶ Figure 3.3.42 Agents become lifeless floating objects without animation



▶ Figure 3.3.43 Animation Sequence within UE4 drives the skeletal mesh asset.



▶ Figure 3.3.44 Blend Space

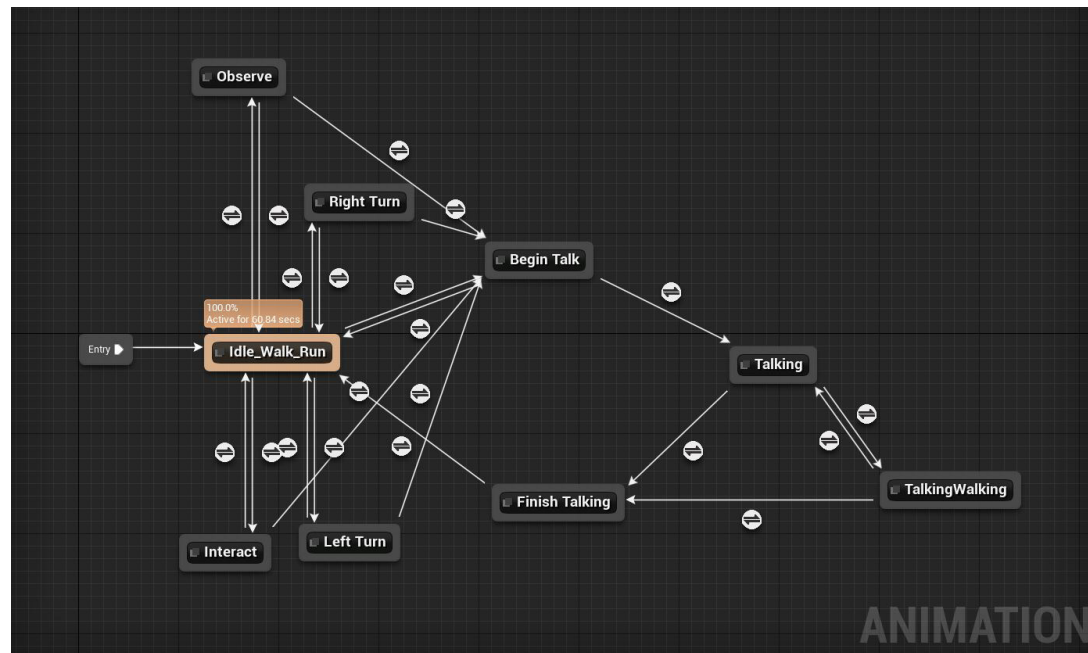
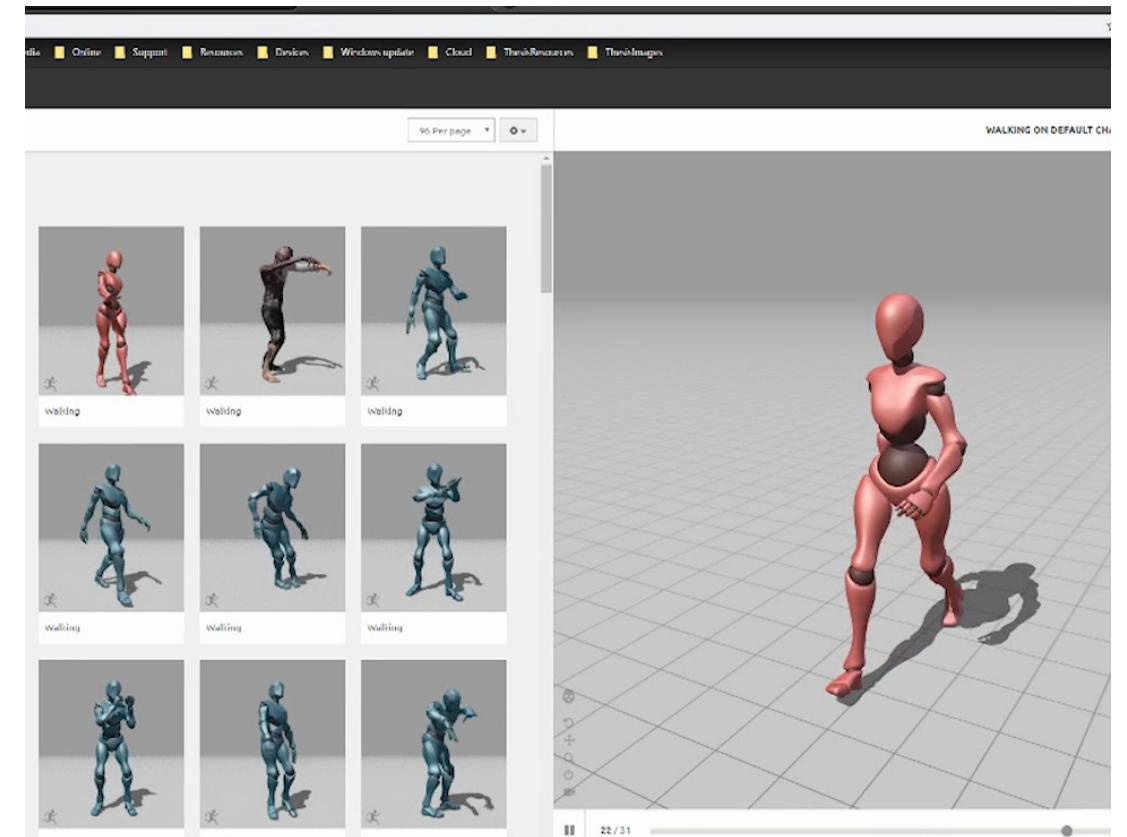


Figure 3.3.45 State Machine



▶ Figure 3.3.46 Adobe Maximo

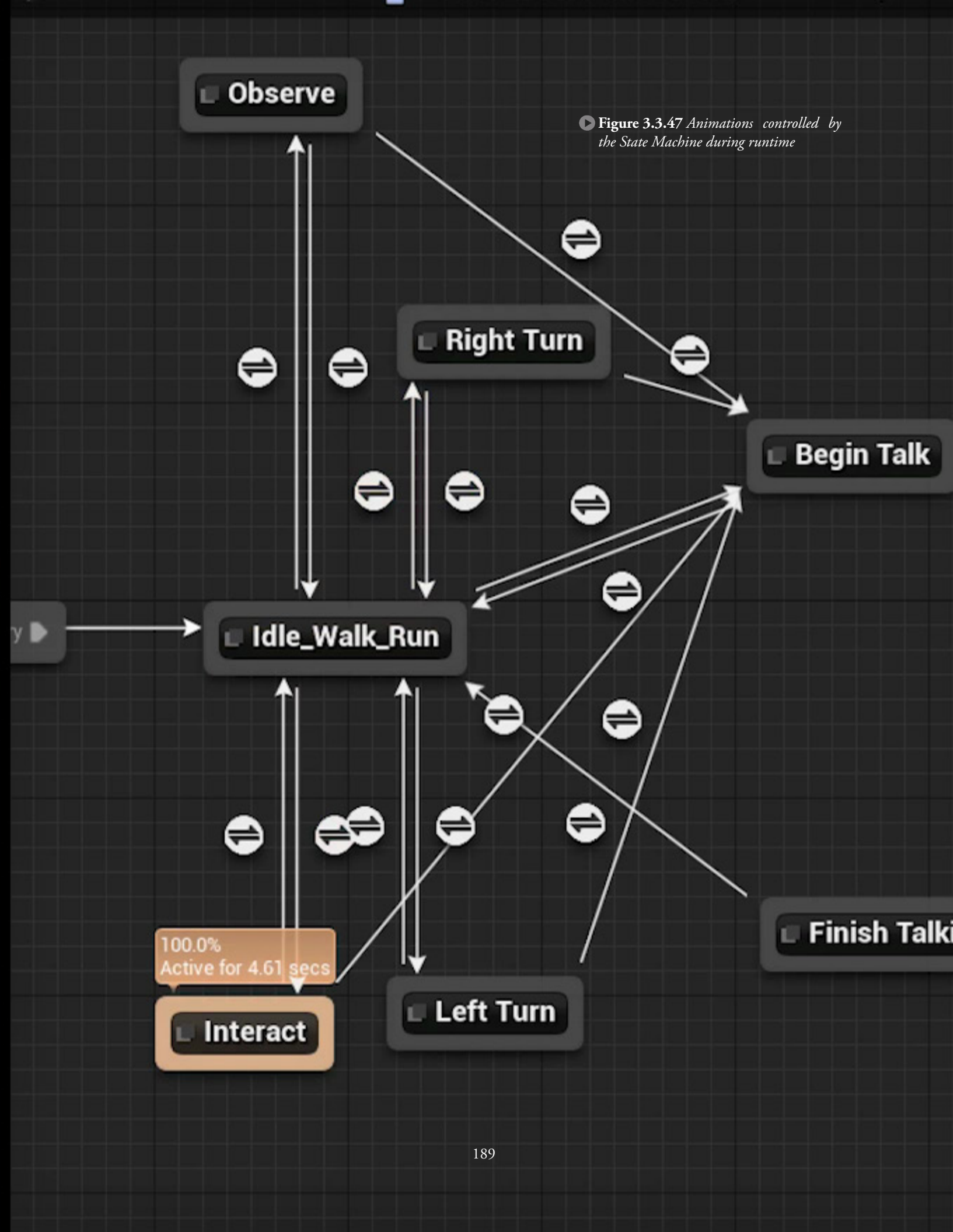


Figure 3.3.47 Animations controlled by the State Machine during runtime

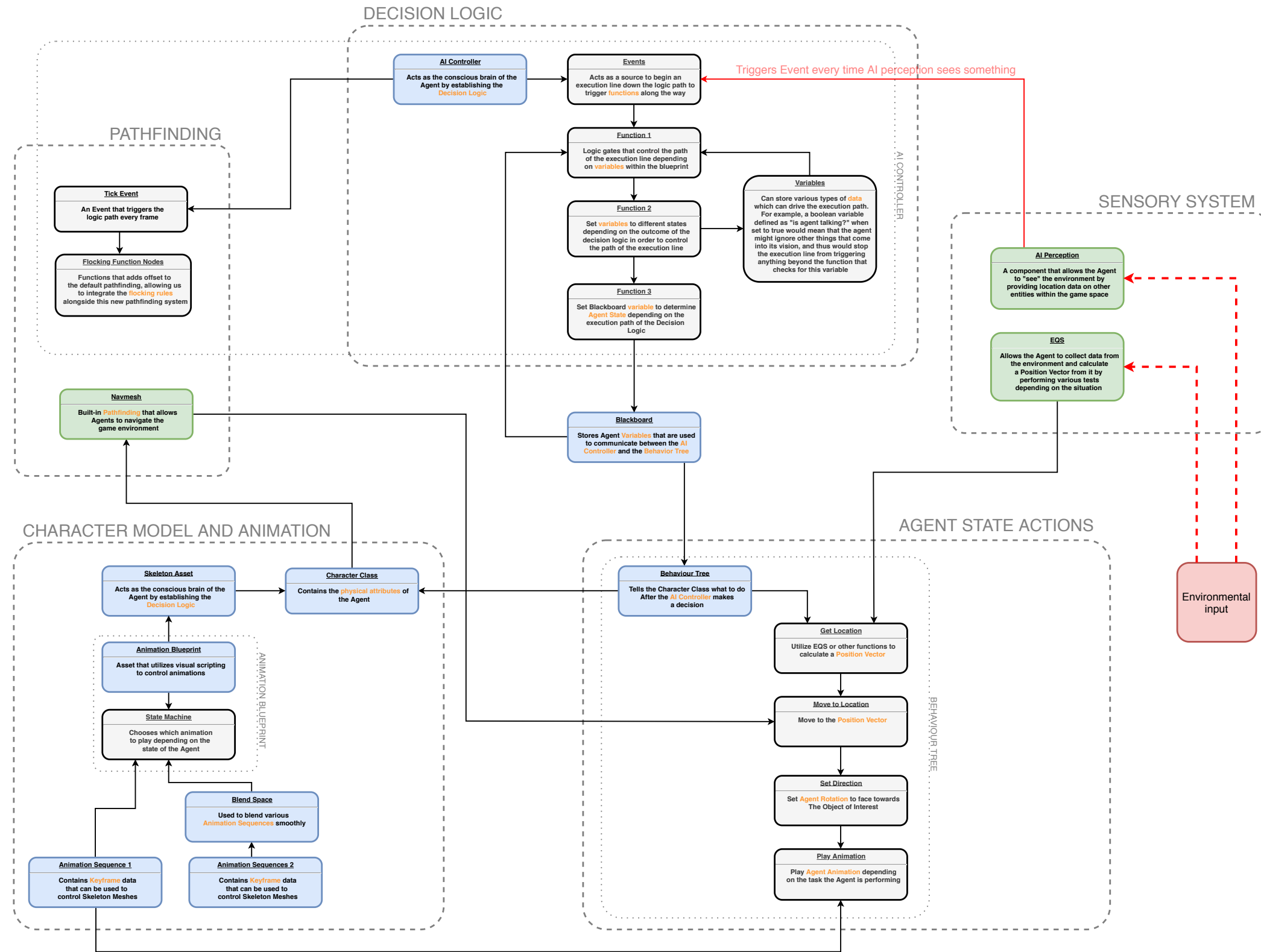


Figure 3.3.48 Flowchart of updated Human Systems within UE4

Chapter 3.4 | Architectural Objects

Now that we have re-established the human agents within UE4, it is time to do the same to the architectural objects. Creating these objects within this software is very similar to creating the human agents. We will continue utilizing a pawn asset, along with its blueprint system, however, we will no longer require some of the other more complex assets such as the blackboard, behavior tree, or AI controller (unless the type of dynamic architectural object is complex enough to require it). On top of this, we will also utilize the construction script, which allows us to create changeable parameters for these objects within the viewport, allowing a much more user-friendly way to set up these generic object assets within the actual visualization workflow. (Fig. 3.4.1 - 2) With this methodology, we can begin recreating these objects that the agents can interact with. While there can be a plethora of architectural objects within a space, to create a basic working simulation, we will need to first establish some basic object typologies in line with the human states we defined in Chapter 3.3.

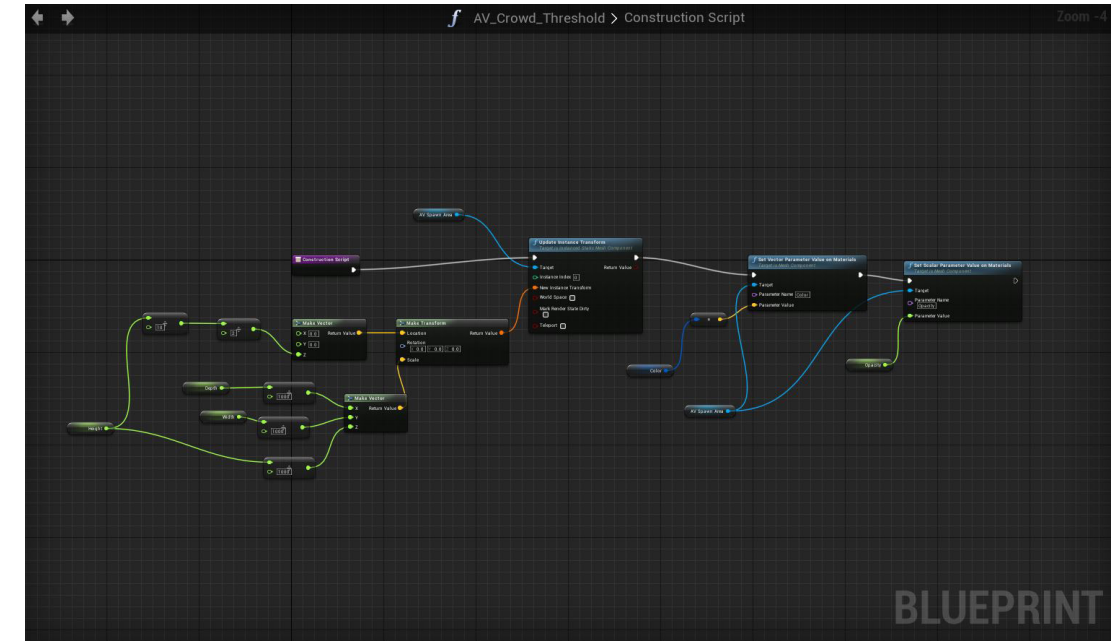


Figure 3.4.1 The Construction Script is visually very similar to the Blueprint

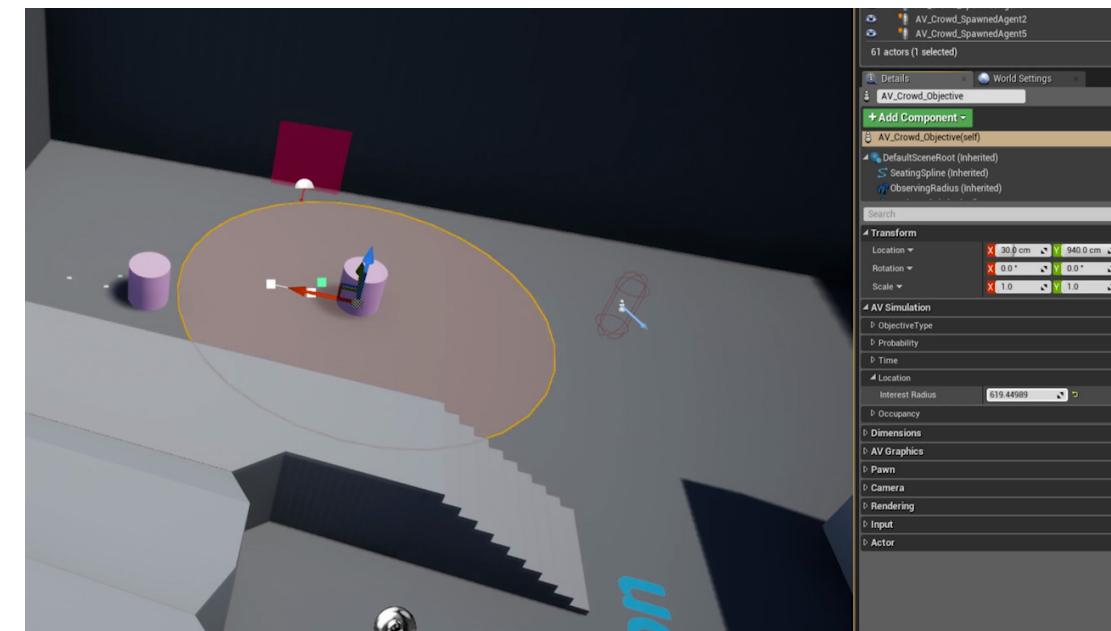


Figure 3.4.2 The Construction Script allows us to sync model attributes within the game space with variables within the Blueprint

Command Center

While this is not an architectural object, the first object we must create is the command center. The purpose of this asset is to handle agent spawning throughout the simulation as well as global parameters such as the number of agents, their spawn rate, and various toggles for analytical and debugging purposes. For this reason, the actual mesh of this object does not matter, as it is simply a container for these variables. (Fig. 3.4.3) As such, we can simply focus on the event graph of this asset, which is required to spawn the agents. To do so, we need to create a list of all the entrances/exits within the environment and begin to spawn human agents from them depending on the preset attraction of the entrances. (Fig. 3.4.4)

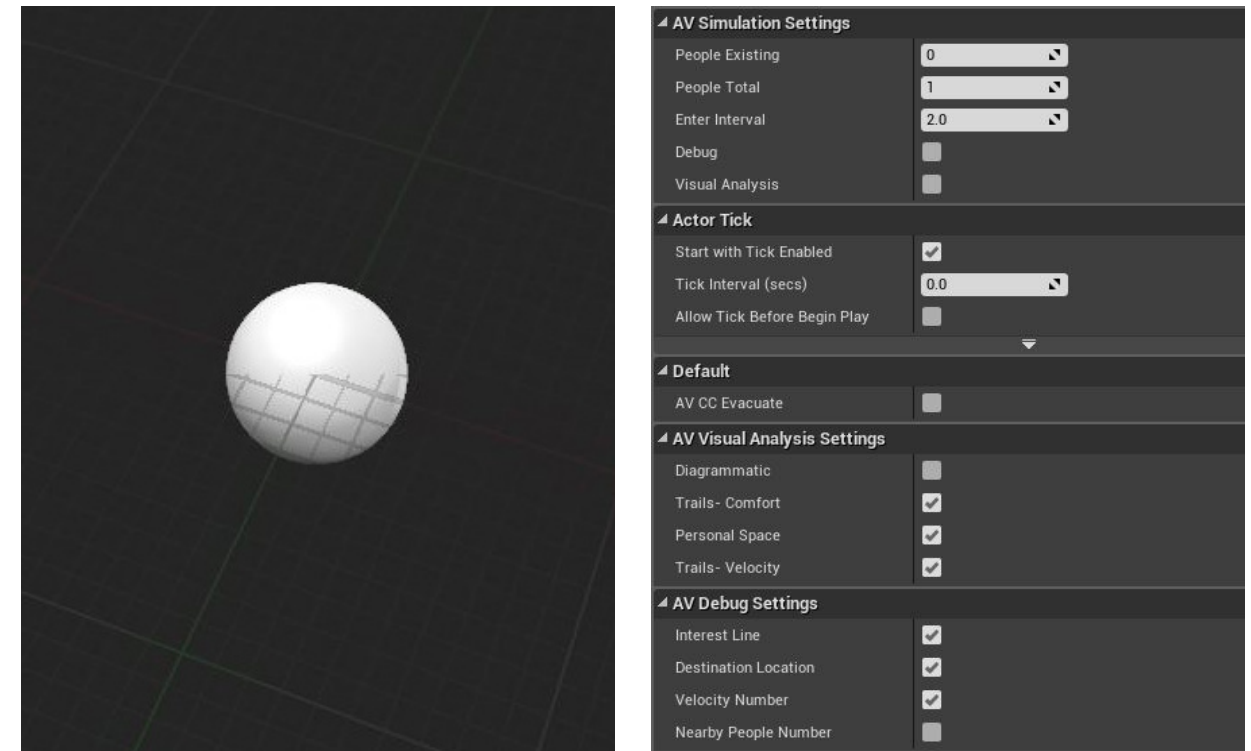


Figure 3.4.3 Command Center viewport and properties

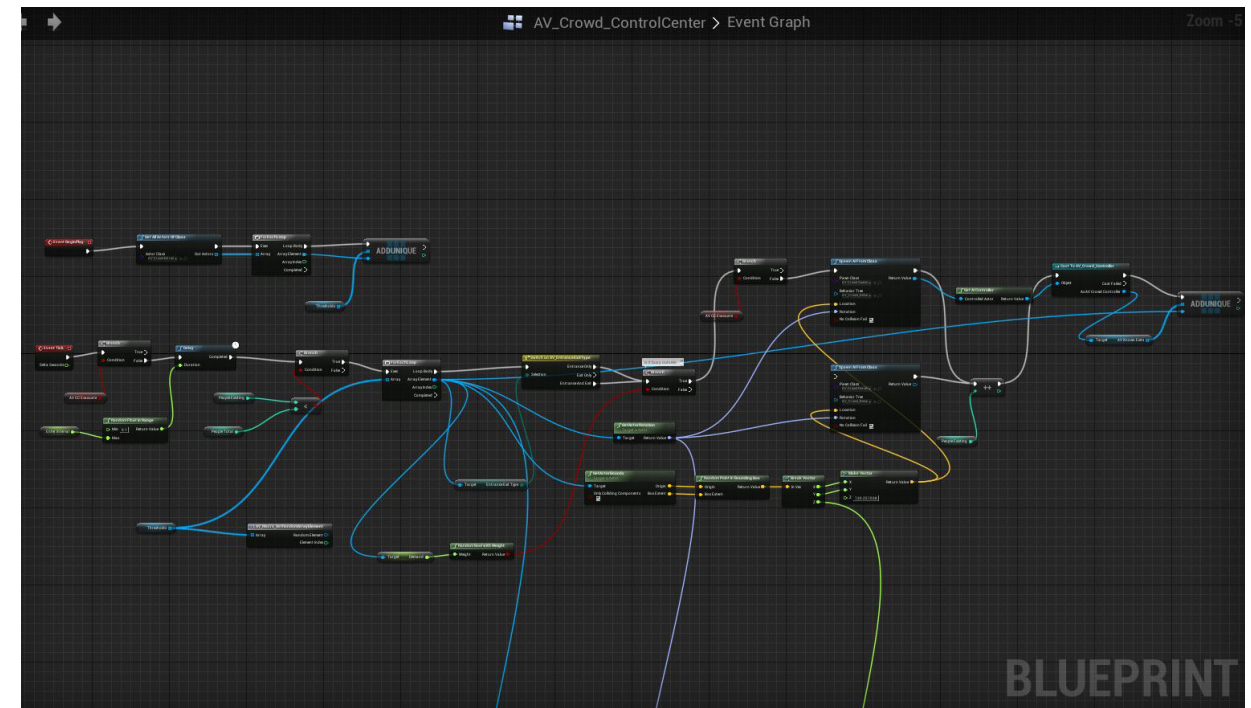


Figure 3.4.4 The Command Center asset contains the blueprint nodes (as defined in the Enter State on pg. 168) to spawn agents at thresholds every game tick during the simulation process

Entrances/Exits

The first of these required architectural objects are the entrances and exits which allow the agents to enter and leave the space. Creating such an asset is straightforward, as we simply require a rectangular volume to symbolize the area in which the door might exist. From here, we need to establish basic parameters such as Width, Height, and Interest. (Fig 3.4.5) We can then use the command center to define a random location within this volume to spawn the agent. Depending on the typology of the object, we can also model-in doors if required.

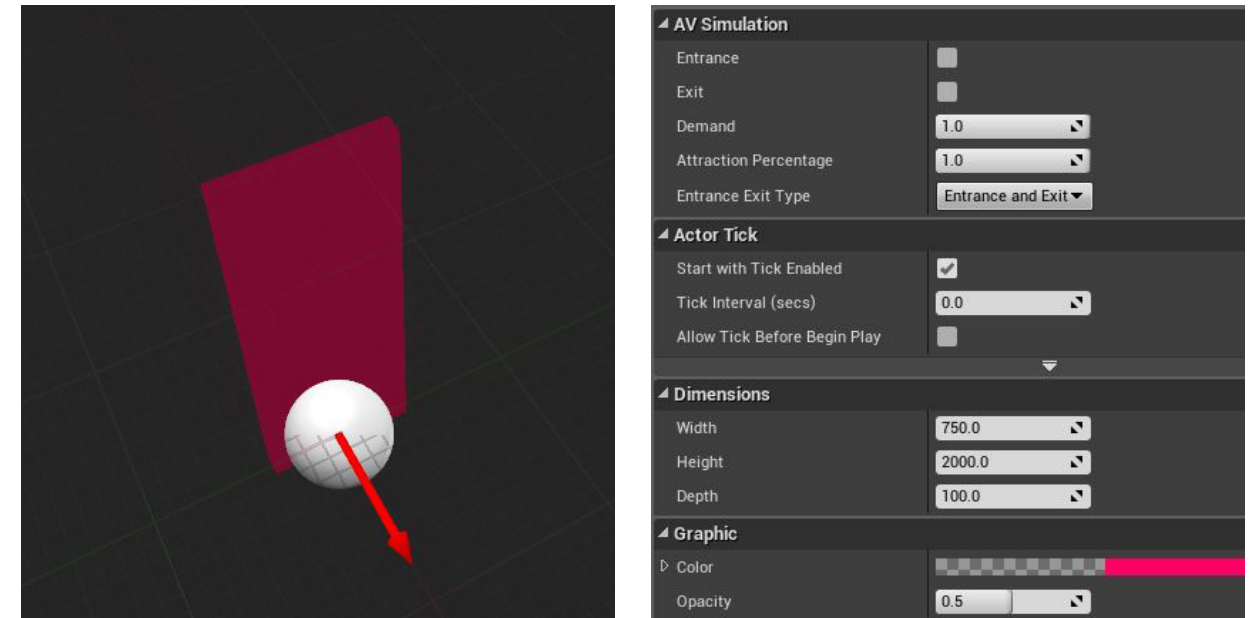


Figure 3.4.5 Entrance/Exit viewport and properties

Thresholds

The next of these required objects are the thresholds. Much like the entrances/exits, we will require a basic volume with an optional door mesh, as well as parameters such as width, height, and interest. (Fig. 3.4.6) The purpose of this object is to allow slightly more control to the crowd flow within the simulation.

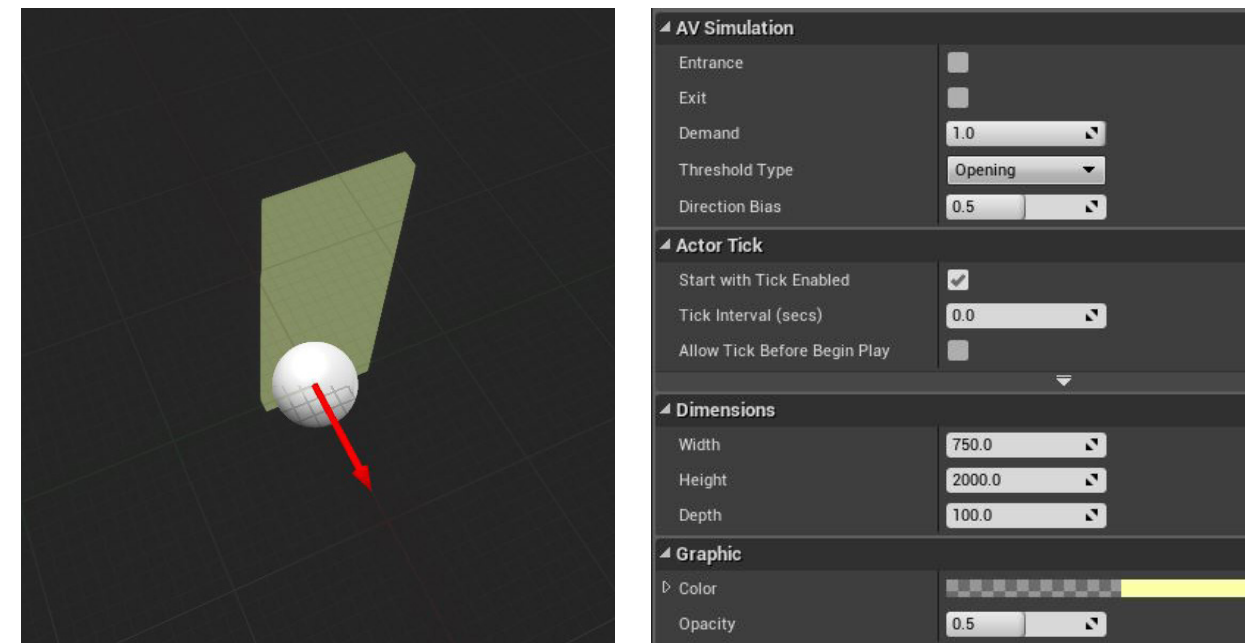


Figure 3.4.6 Threshold viewport and properties

Architectural Objects

The last of these required objects are the actual interactive objects. As already covered in Chapter 2.4, these objects can be quite diverse, therefore, what matters at this stage is to familiarize ourselves with the utility of the object, as well as the tools of this software. In doing so, we can create any type of objects that we desire. With this in mind, we will establish the most basic parameters of these objects in order to work with the *AgentStates* from Chapter 3.3, which are the following: (Fig. 3.4.7 - 8)

Objective Type determines if the object is a normal object or if it can be sat upon or if it requires a lineup.

Probability determines how attractive the object is to passing agents, as well as what ratio of agents might want to look at the object, versus interacting with it. This can be established with a float variable between 0 and 1, where 0 is 0% and 1 is 100%.

Occupancy determines if there is an occupancy limit to the object. For example, a checkout counter may only be able to accommodate 1 person at the same time, in which case a lineup would be required if there are more people than this limit.

Time determines how long an agent might be interested in this object. A complex object such as a book might hold people's attention longer than a simple sculpture.

Location determines the interest radius of the object, which in itself determines how far away the agents can be before they no longer notice the object. A larger object might have a larger radius compared to a smaller object.

Graphics determines both the graphical representation of limits such as interest radius, as well as physical properties such the 3D mesh and the assigned materials of the object.

On top of this, we will also require a basic way for the object to respond if an agent decides to interact with it. To do so, we can use an asset within UE4 called blueprint interface, which allows blueprints to share data with one another.^[1] In doing so, we can establish an event node within the object blueprint that will trigger whenever an agent interacts with it. (Fig. 3.4.9 - 10) We can then add any type of function to this event to simulate their use.

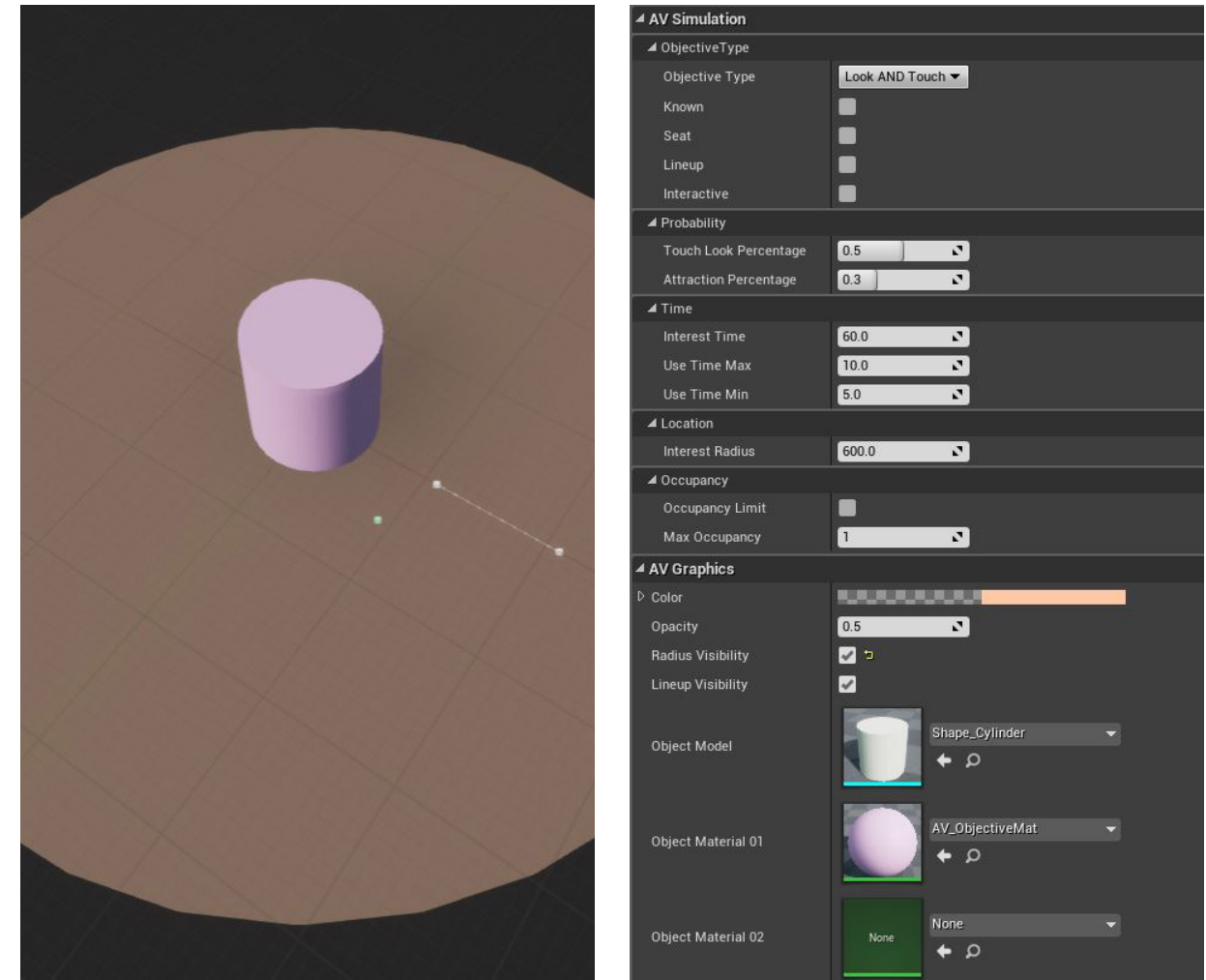


Figure 3.4.7 Architectural Object viewport and properties

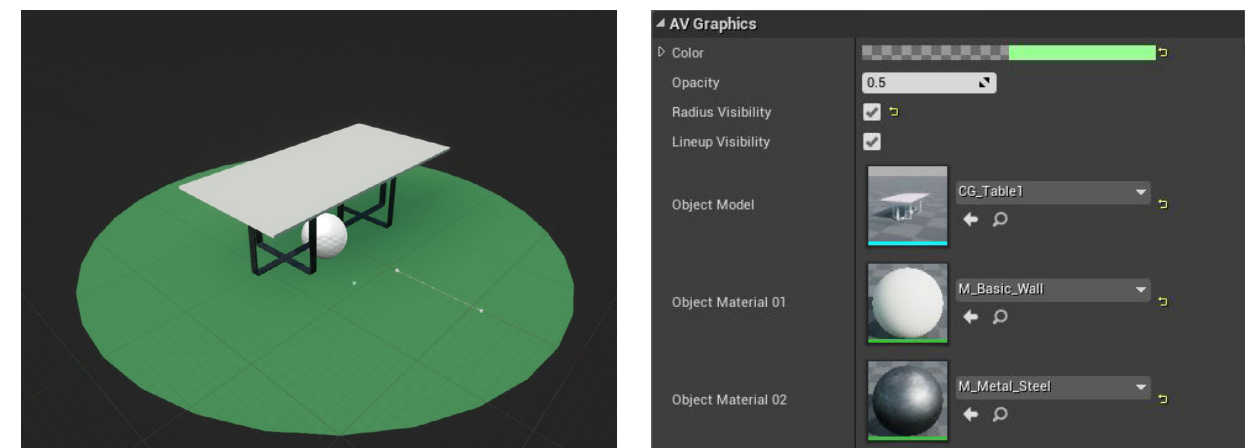


Figure 3.4.8 The 3D mesh and materials of the object can be changed in the properties panel depending on the typology of the actual object

¹ "Blueprint Interface," Unreal Engine Documentation, accessed October 18, 2019, <https://docs.unrealengine.com/en-US/Engine/Blueprints/UserGuide/Types/Interface/index.html>.

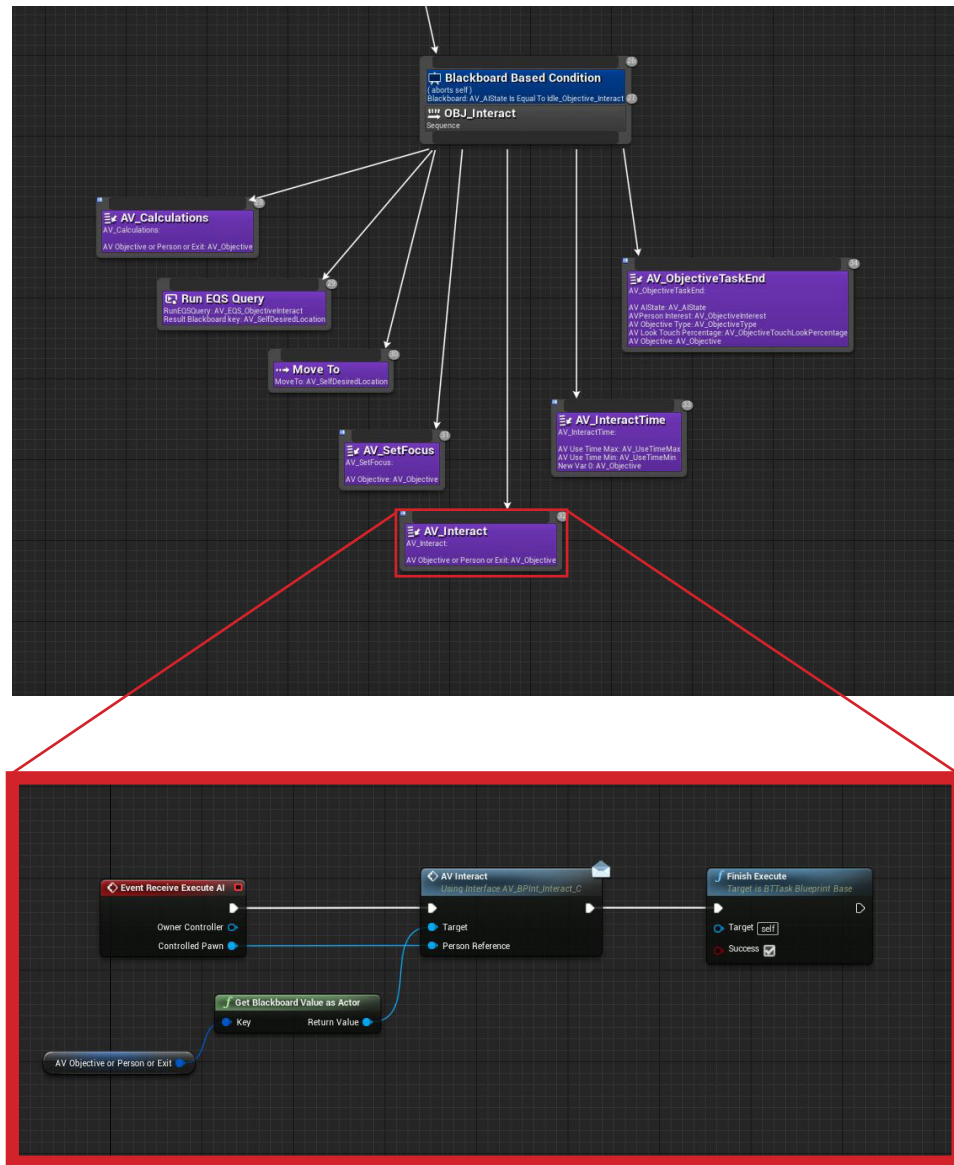


Figure 3.4.9 Blueprint Interface node within the Interact node within the Object Interact State of the Behavior Tree.

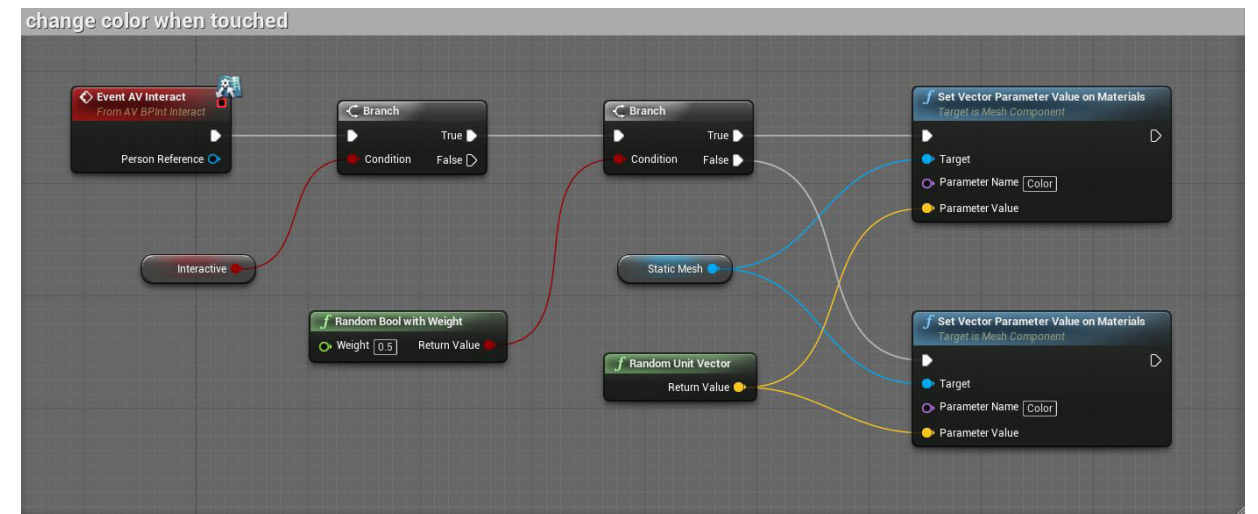


Figure 3.4.10 The Interact Event within the object blueprint allows it to change the color of its texture whenever the Interact node within the behavior tree of the Agent is triggered

Figure 3.4.9 - 3.4.10

The nodes from the Behavior Tree each contains its own blueprint graph. Within the interact node, we can utilize a blueprint interface to communicate with other blueprints. This allows us to use an event node within the object blueprint that is triggered whenever the *Interact* blueprint interface node within the Behavior Tree is triggered.

Chapter 3.5 | Environment Context

Looking back at the start of Chapter 1.1, the world can be broken down into the “firmness and security of a human-made shell” and the “turbulence and uncertainty of nature’s ferocity.” As such, the environment context of the simulation can logically be broken down into nature and architecture.

Architecture

Creating the architectural context of this simulation is as straightforward as importing the building model into Unreal Engine 4. To do so however, we must first consider which software we are importing it from, the file type of the exported model, and the techniques for preserving collision upon import to allow the game engine to generate the pathfinding system.

For this step, I have chosen to utilize Revit as the software to import from, as Building Information Modeling (BIM) seems to be the standard that the architecture industry is heading towards.^[1] In doing so, we can establish a visualization pipeline where the Revit model can be imported for basic scaling and geometry, and the details are added later in UE4 to maximize visualization quality and rendering efficiency. This pipeline can then be described by the following steps: (Fig. 3.5.1 - 2)

1. Export model from Revit as an FBX file. (based on families or textures)
2. Import FBX file into UE4 as a static mesh asset and set the correct scale.
3. Go into the Static Mesh asset and create a physical material and set the collision complexity to “use complex collision as simple.”

From this, the Revit model should be available within UE4 with working collision. (Fig. 3.5.3 - 4) It is then possible to apply various material textures to this model much like in Revit.

Since beginning this thesis, however, Epic Games has also recognized the merit of game engines within architectural visualization. As such, they have created additional tools to work alongside Unreal Engine in a package they call Unreal Studio.^[2] This new software is essentially the Unreal Engine with additional toolsets, templates, and libraries to facilitate architectural visualization. One of the key benefits within this software is *Datasmith*,^[3] which provides a more efficient and seamless way of importing building models into the game development environment.^[4] While this new software is still in beta testing, it further validates the utilization of this game engine for architectural visualization, and is definitely something to investigate beyond this thesis.

¹ “About the National BIM Standard-United States,” National Institute of Building Sciences, accessed October 18, 2019, <https://www.nationalbimstandard.org/about>.

² “Unreal Studio,” Unreal Engine, accessed October 18, 2019, <http://www.unrealengine.com/studio>.

³ “Installing the Datasmith Exporter Plugin for Revit,” Unreal Engine Documentation, accessed October 18, 2019, <https://docs.unrealengine.com/en-US/Studio/Datasmith/SoftwareInteropGuides/Revit/InstallingExporterPlugin/index.html>.

⁴ Unreal Engine, “The Journey from Revit to Unreal Studio | Feature Highlight | Unreal Studio,” YouTube, 1:08:36, accessed October 18, 2019, <https://youtu.be/iuqTvvd16UQ?t=435>.

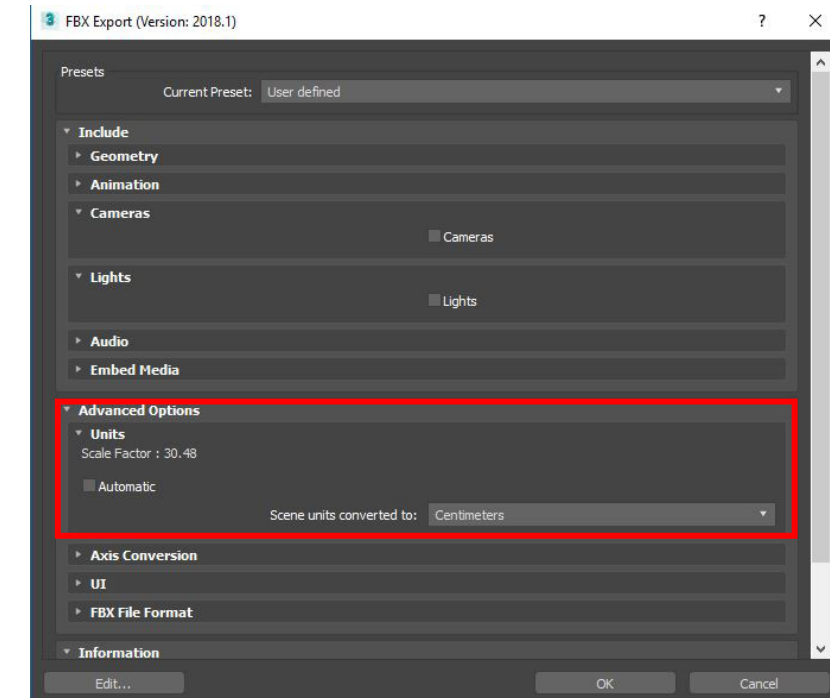


Figure 3.5.1 3ds Max export window

Linking the Revit model to 3ds Max before exporting allows better control in the resulting FBX file. The base units for Revit is in feet, and the base units in UE4 is in Centimeters. This must be taken into account when exporting the FBX file from 3ds Max linked from Revit.

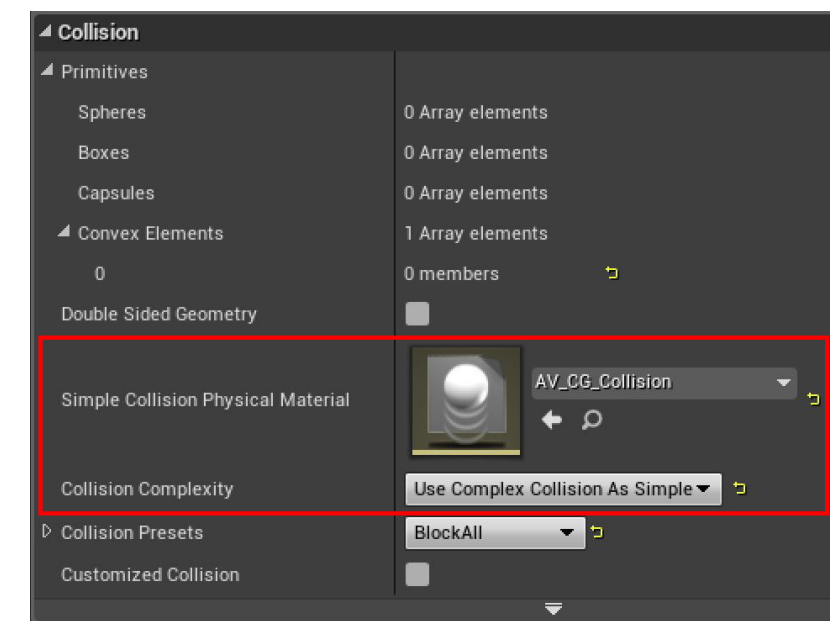


Figure 3.5.2 Static Mesh collision properties

Collision can be auto generated by utilizing a physical material and setting the Collision Complexity of it to “use complex collision as simple”

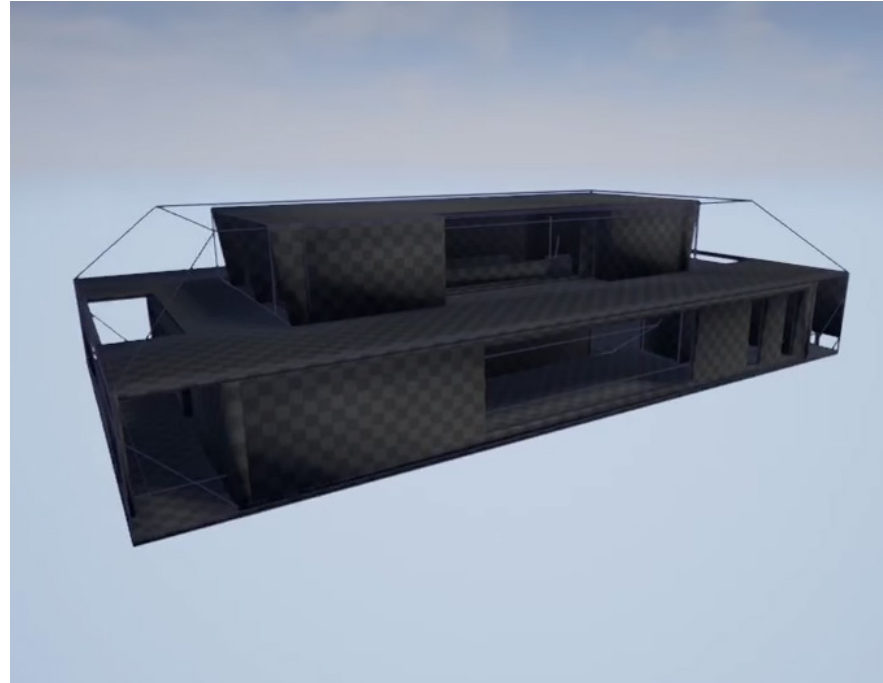


Figure 3.5.3 Collision lines right after Importing from Revit

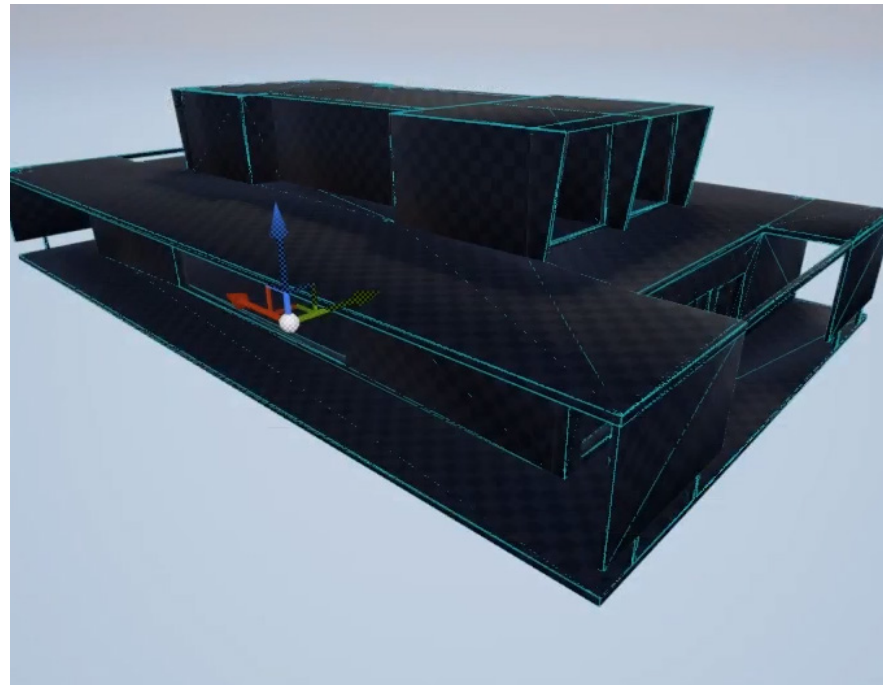


Figure 3.5.4 Fixed collision lines within UE4 with a physical material

Nature

Nature has always been a challenge to model in traditional architectural visualization workflows due to its complexity and randomization. However, UE4 provides basic landscaping tools for an easy setup of such an environment, which is one of the more powerful tools of this pipeline. With this, we can create a basic plane with a contour map and use various brushes and environmental assets to shape and randomly scatter vegetation along this plane, allowing for the fast creation of complex natural landscapes. (Fig. 3.5.5)



Figure 3.5.5 Landscape Creation in UE4

Chapter 3.6 | Application Methodology

Now that we have established the basic assets of this simulation system, as well as a way of importing and creating a context within the virtual environment, we can begin developing a workflow pipeline utilizing this software as a visualization tool. As such, a basic workflow can be described as the following:

1. Import model from Revit (**Fig. 3.6.1 - 2**)
2. Define the entrance and exits (**Fig. 3.6.3 - 4**)
3. Define the architectural elements (**Fig. 3.6.5 - 6**)
4. Simulate. (**Fig. 3.6.7 - 8**)

Beyond this, we can also output various points from each agent to provide various analytics such as crowd flow, densities, and comfort maps. (**Fig. 3.6.9 - 10**) These are useful not only in debugging the simulation but also as a rough analytical tool for architectural visualization. While they are currently a rough approximation, these visualizations will continue to become more accurate as the crowd movement algorithms becomes increasing refined. To further help with the usability of this tool, we can create a GUI (Graphical User Interface) for simpler control of the simulation global parameters, the analytics and debugging visualizations, as well as the sun lighting angles. (**Fig. 3.6.11 - 12**) Ultimately, we can utilize this tool to simulate various types of spatial scenarios, as well as visualize and interact with them in many different ways. (**Fig. 3.6.13 - 18**)

At its basic level, these simulations will be able to convey the capacity of spaces and their ability to accommodate crowds and human movement, as well as pinch points and opportunistic or problematic interaction points. Beyond this, the possibilities are endless.

Figure 3.6.1 - 3.6.2
 FBX models can be exported from various 3D applications such as Revit, Rhino, 3ds Max, etc. The FBX model can be imported by dragging the file from the windows folder. Textures can then be applied to the imported model by dragging texture assets from the content browser.

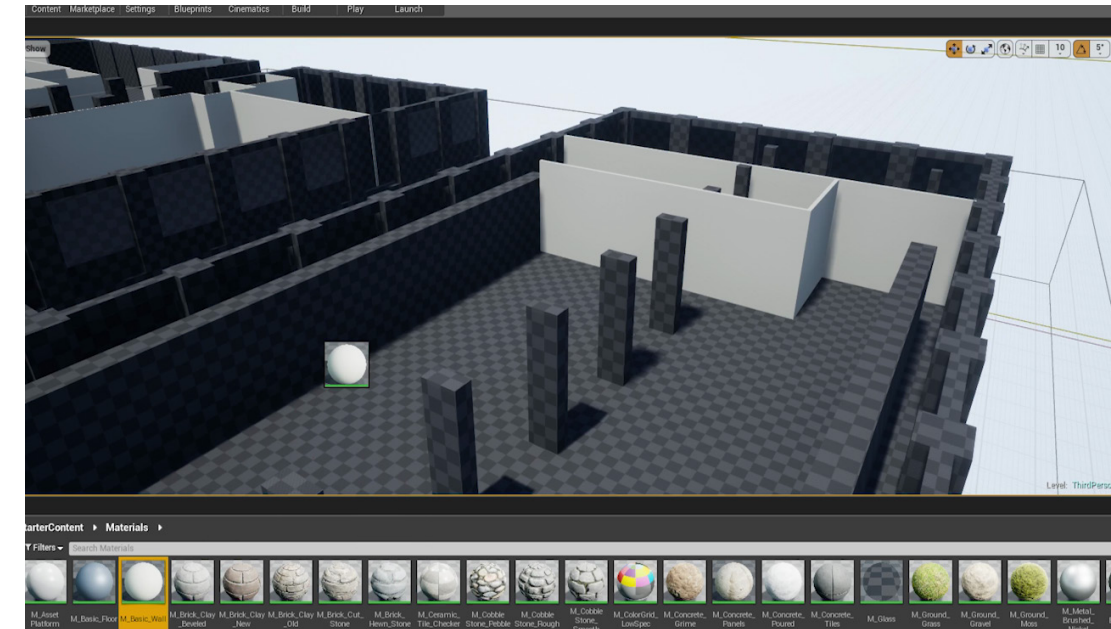


Figure 3.6.1 Step 1: Import FBX model

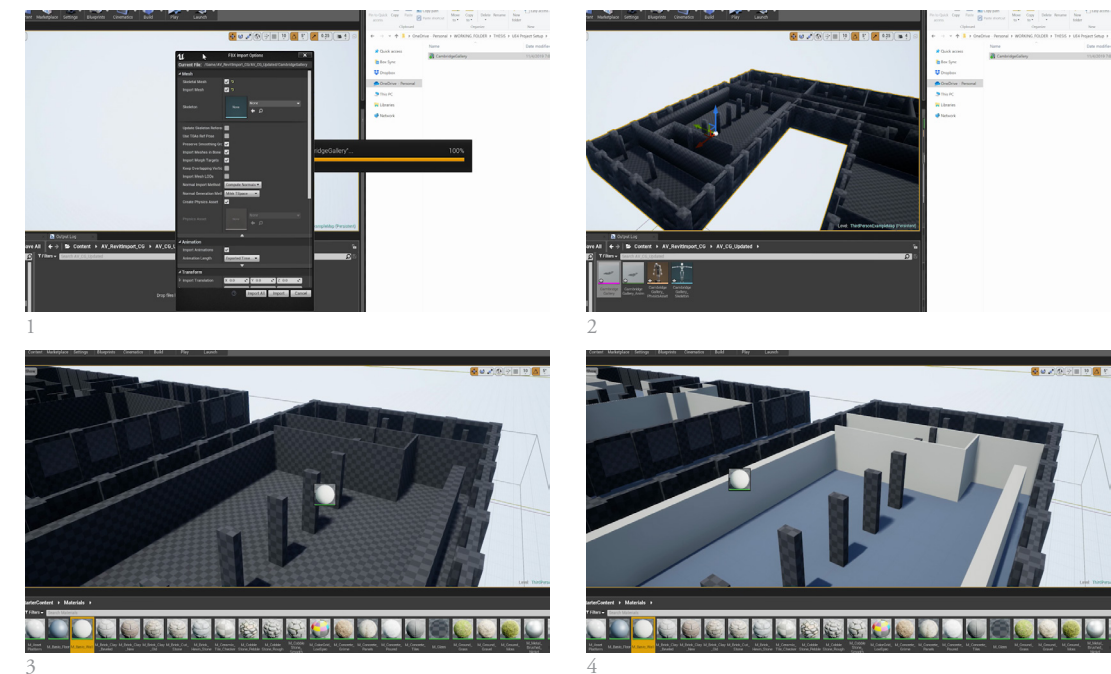
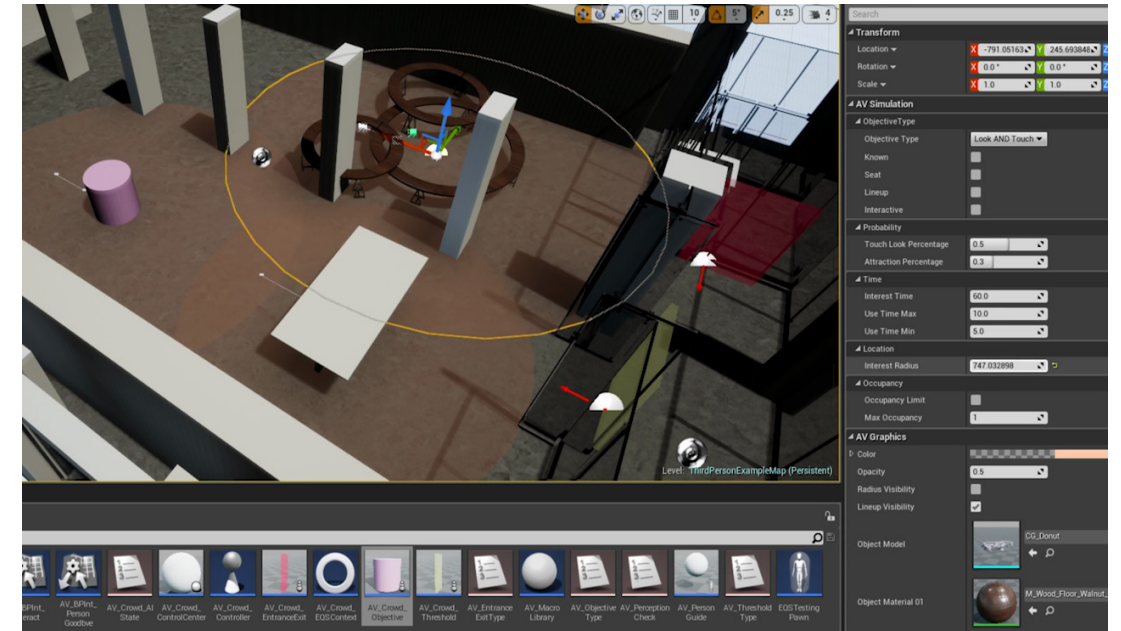


Figure 3.6.2 Step 1 sequential frames



▶ **Figure 3.6.3** Step 2: Define entrances/exits and thresholds



▶ **Figure 3.6.5** Step 3: Define Interactive Elements



Figure 3.6.4 Step 2 sequential frames

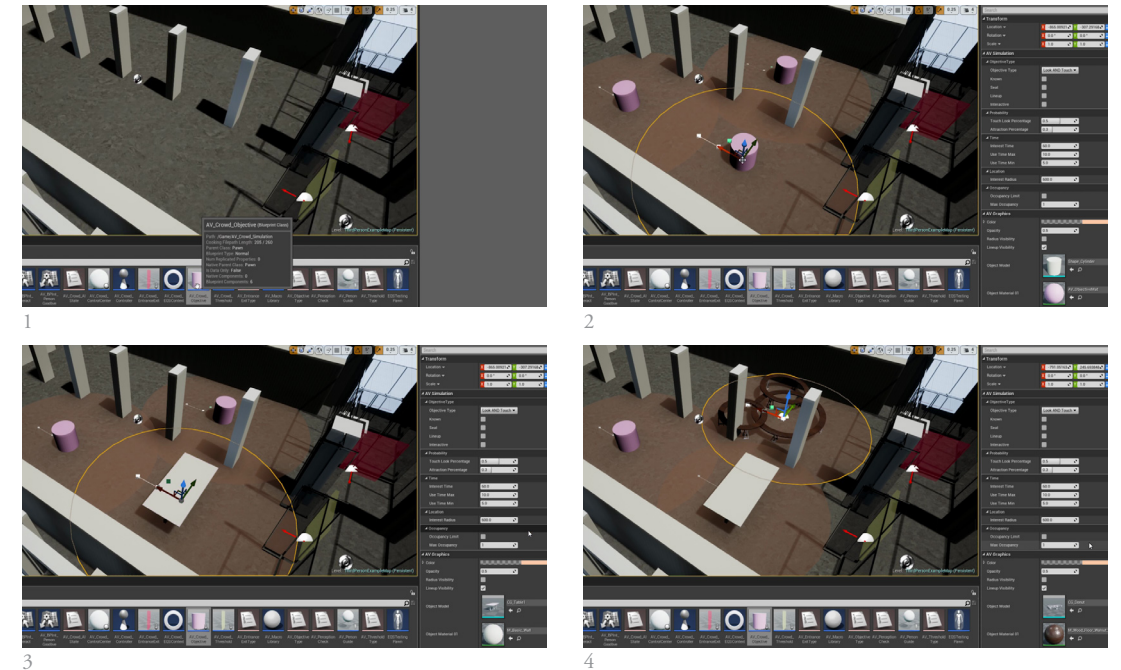


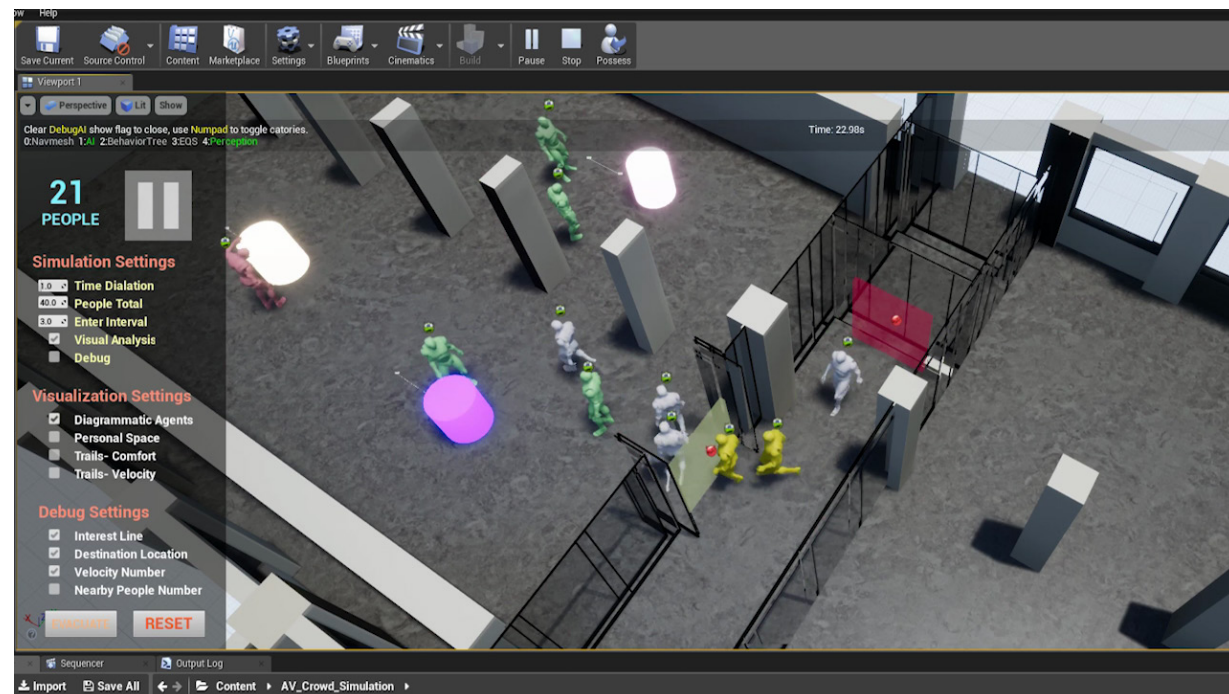
Figure 3.6.6 Step 3 sequential frames

Figure 3.6.3 - 3.6.4

Entrances and exits can be defined within the simulation by dragging assets from the content browser. They can then be customized depending on their typology and location by adjusting their attributes within the properties menu.

Figure 3.6.5 - 3.6.6

Similar to the entrances and exits, interactive elements can be defined by dragging assets from the content browser. They can then be customized depending on their typology by adjusting their attributes within the properties menu.



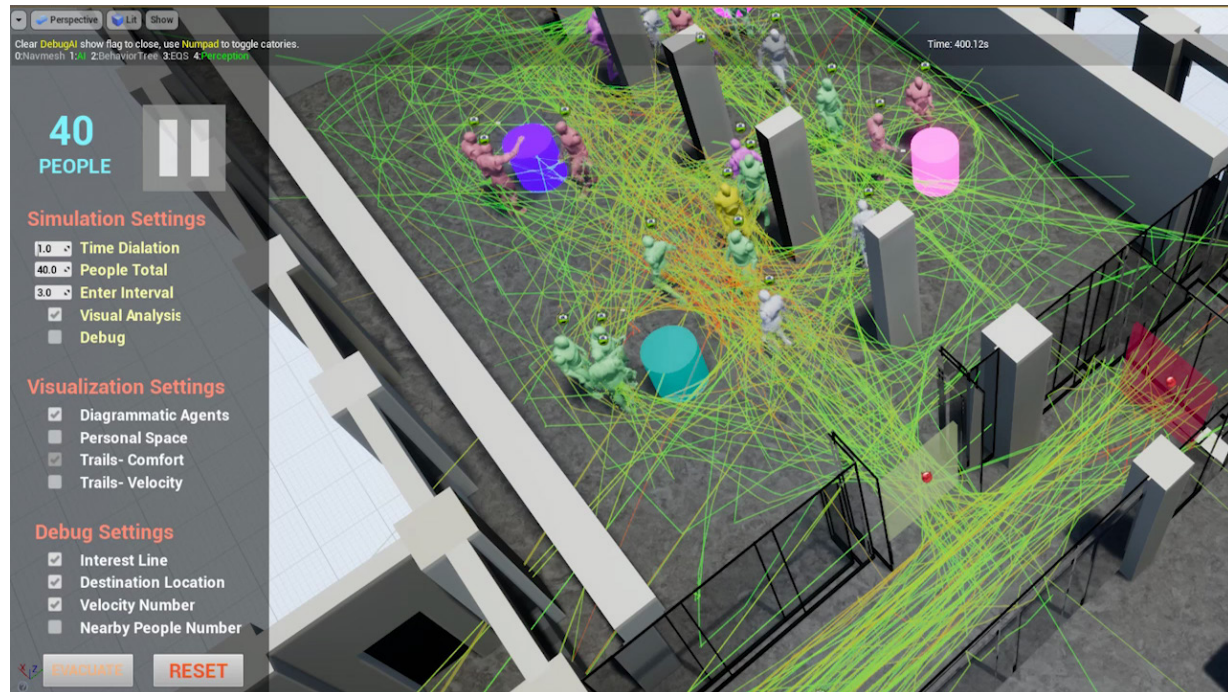
▶ **Figure 3.6.7** Step 4: Initiate the simulation

Figure 3.6.7 - 3.6.8

After establishing the space with our predefined elements, we can begin the simulation. With a simple press of the play button within the GUI (Graphical User Interface), the software begins to generate agents from the entrance/exit thresholds based on global variables such as number of people and spawn rate. This gives the illusion of people flowing from the thresholds and populating the space in real time.



Figure 3.6.8 Step 4 sequential frames



► **Figure 3.6.9** Various forms of data can be visualized and mapped out during the simulation and toggled on or off via the GUI (Graphical User Interface) or hotkeys

Figure 3.6.9 - 3.6.10

Due to the calculated nature of agent-based computer-generated simulations, it becomes straightforward to render out various forms of data during the simulation process. One example of such is the agent's comfort level throughout the space. This is calculated based on the number of other agents within the agent's personal space (visualized by the red bubbles). The more people within this personal space, the less comfort the agent has, and the redder the resulting trail will become. As seen in the figure, areas of discomfort appears in red, which can be used to map out crowd densities where congestion may occur. This map seems to make logical sense in this case since the location of it is by an object of interest as well as being in the most central location of the space.

This is of course, just one example of how data can be mapped throughout the simulation. Depending on the spatial and scenario typologies, multiple layers can be visualized and traced based on varying types of occupancy throughout the space. A hospital might have different paths for the doctors and the patients; a school might have different paths for the teachers, students, and the general public; a store might have different paths for employees and shoppers. As such, by visualizing this data in real time, it becomes possible to not only better design circulation to accommodate everyone during the design phase, but also visualize neglected spaces that can be optimized.

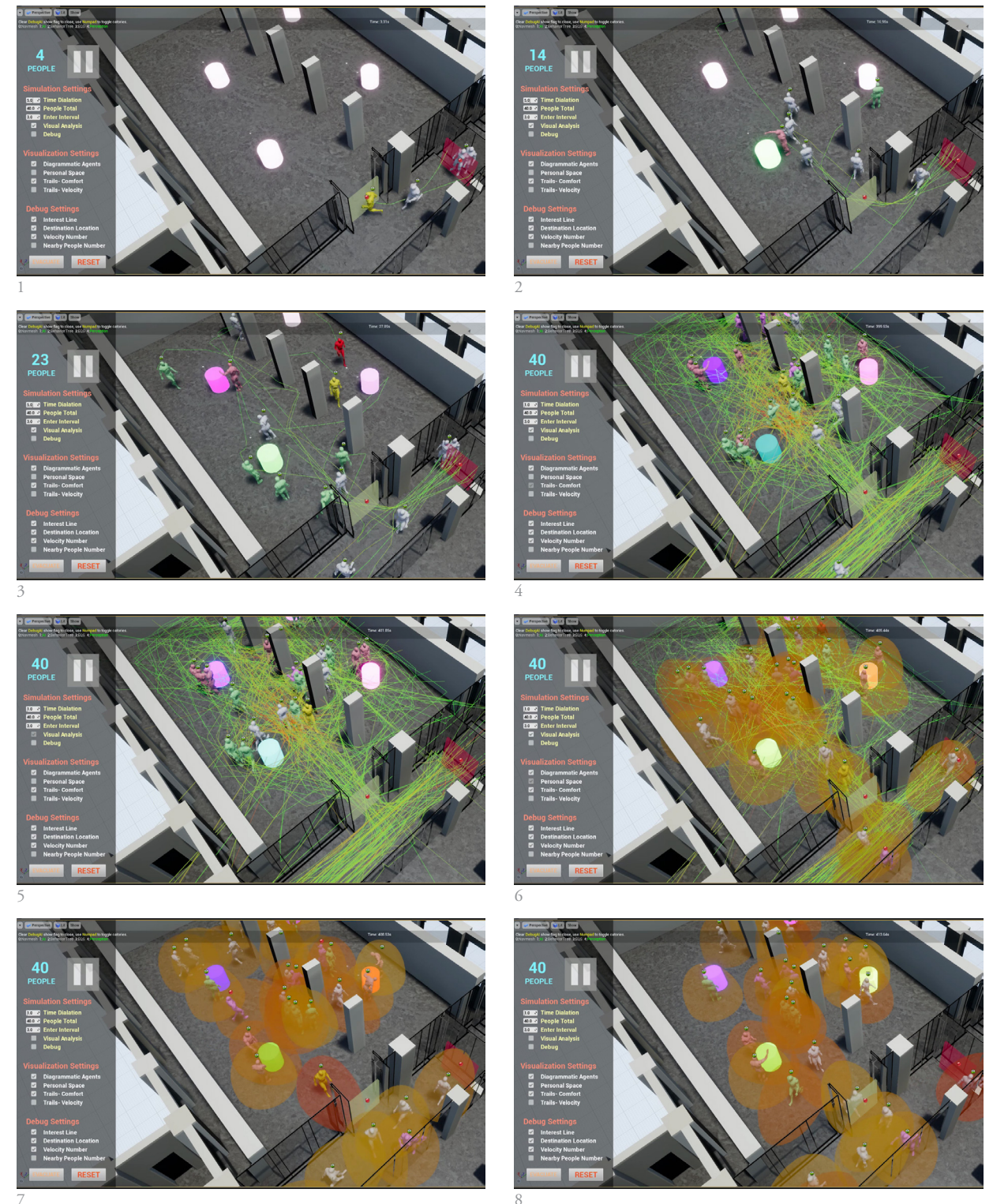


Figure 3.6.10 Data visualization sequential frames



▶ **Figure 3.6.11** GUI controlled Solar Studies

Figure 3.6.11 - 3.6.12

The GUI can also be scripted to control other aspects of the simulation, such as the sun angle and time of day. This interactivity, along with its photorealism capabilities, allows UE4 to become a powerful tool for quickly visualizing photorealistic dynamic lighting conditions to conduct solar studies. As seen from the figure, the sky changes alongside the sun angle, simulating an orange sky during sunset and stars at night. This aspect can be further investigated to interact with the crowd dynamics by making the occupants seek shading and be less likely to walk in the direct sunlight. This can then be further expanded to other environmental factors such as the rain or snow, which can serve to visualize and consider the dynamic nature of environmental factors.

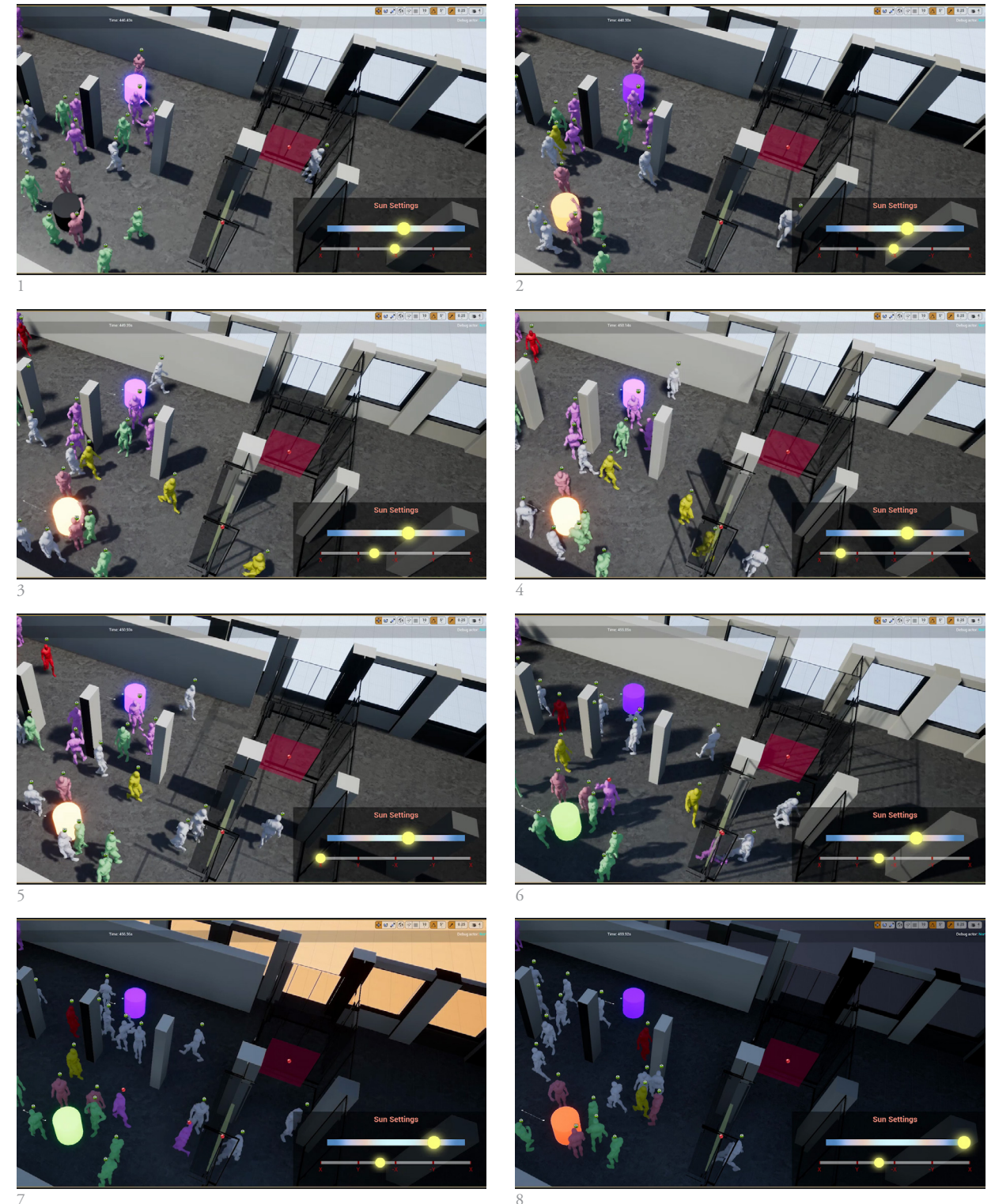
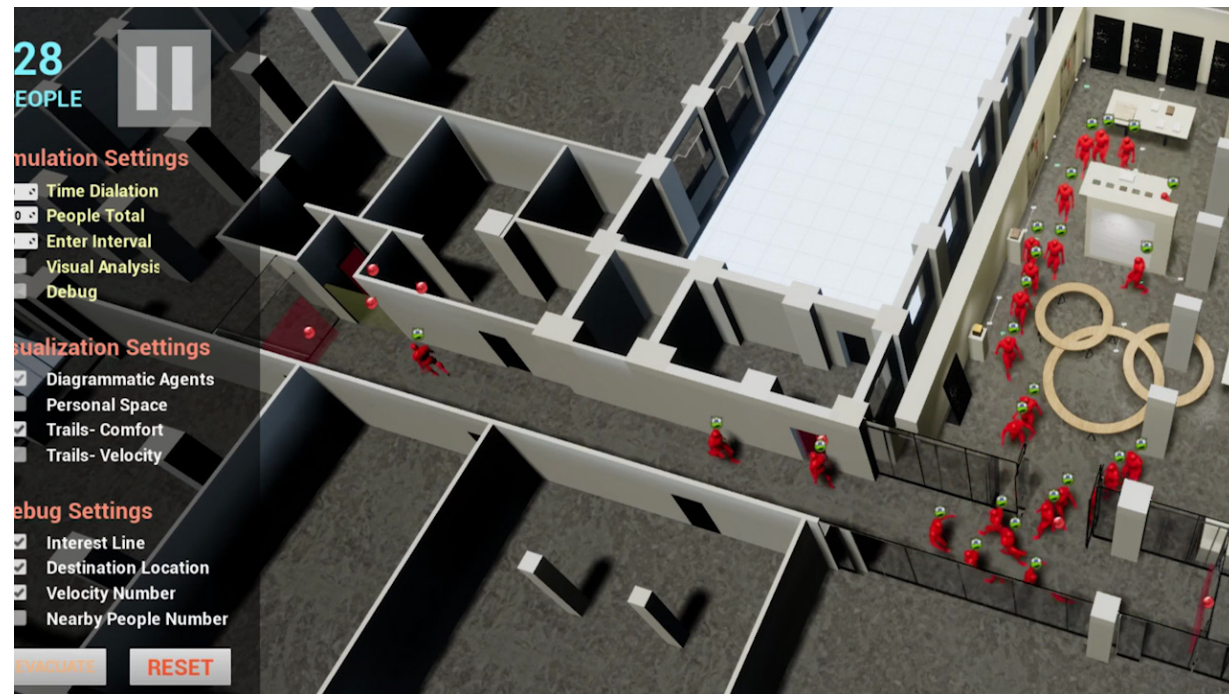


Figure 3.6.12 GUI controlled Solar Studies sequential frames



► **Figure 3.6.13** The GUI can also be used to control various scripted events, such as an evacuation scenario

Figure 3.6.13 - 3.6.14

Various scenarios can be programmed depending on the situation. This figure shows an example of an evacuation scenario, where each agent tries to make their way to the nearest exit. With more substantial scripting depending on the project scenario, it then becomes possible to simulate specific spatial scenarios beyond simple evacuations, such as lectures, outdoor events, gallery talks, performances, New Year's Eve celebrations, etc, as well as more specific spaces that may require additional considerations such as checking into airports, hospitals, and so on.

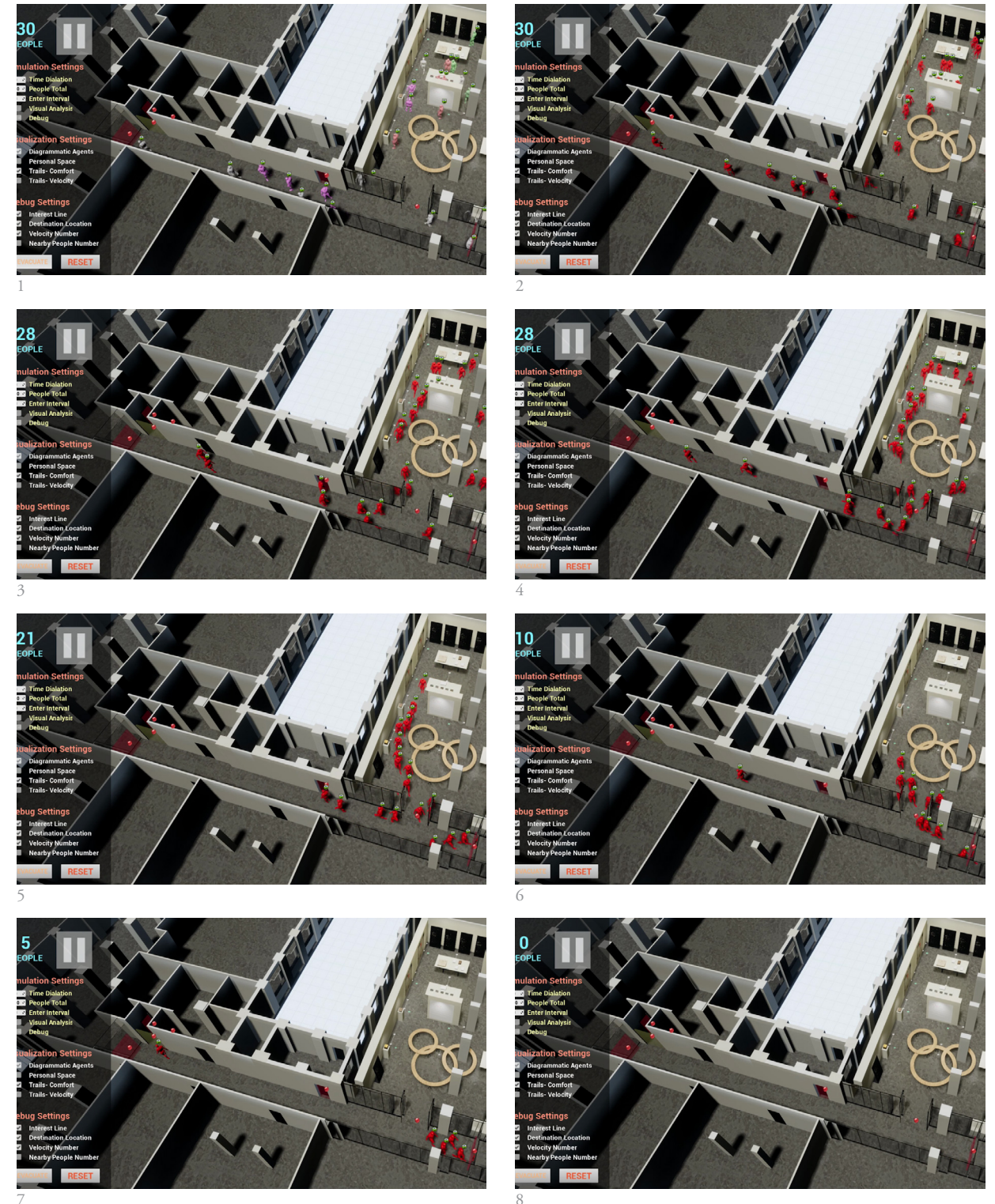


Figure 3.6.14 Scenario programming sequential frames



► **Figure 3.6.15** The simulation can also be visualized and interacted from different perspectives to better visualize the space

Figure 3.6.15 - 3.6.16

This software environment also allows us to utilize various forms of interaction and visualization mediums beyond just visualizing the space from the top down. This opens the possibilities of interacting with the space via mouse and keyboard inputs, VR and AR headsets, as well as body tracking technologies. This figure shows an example of a 3rd person perspective where someone can control an agent to walk around the space, visualizing the simulation from a new perspective.

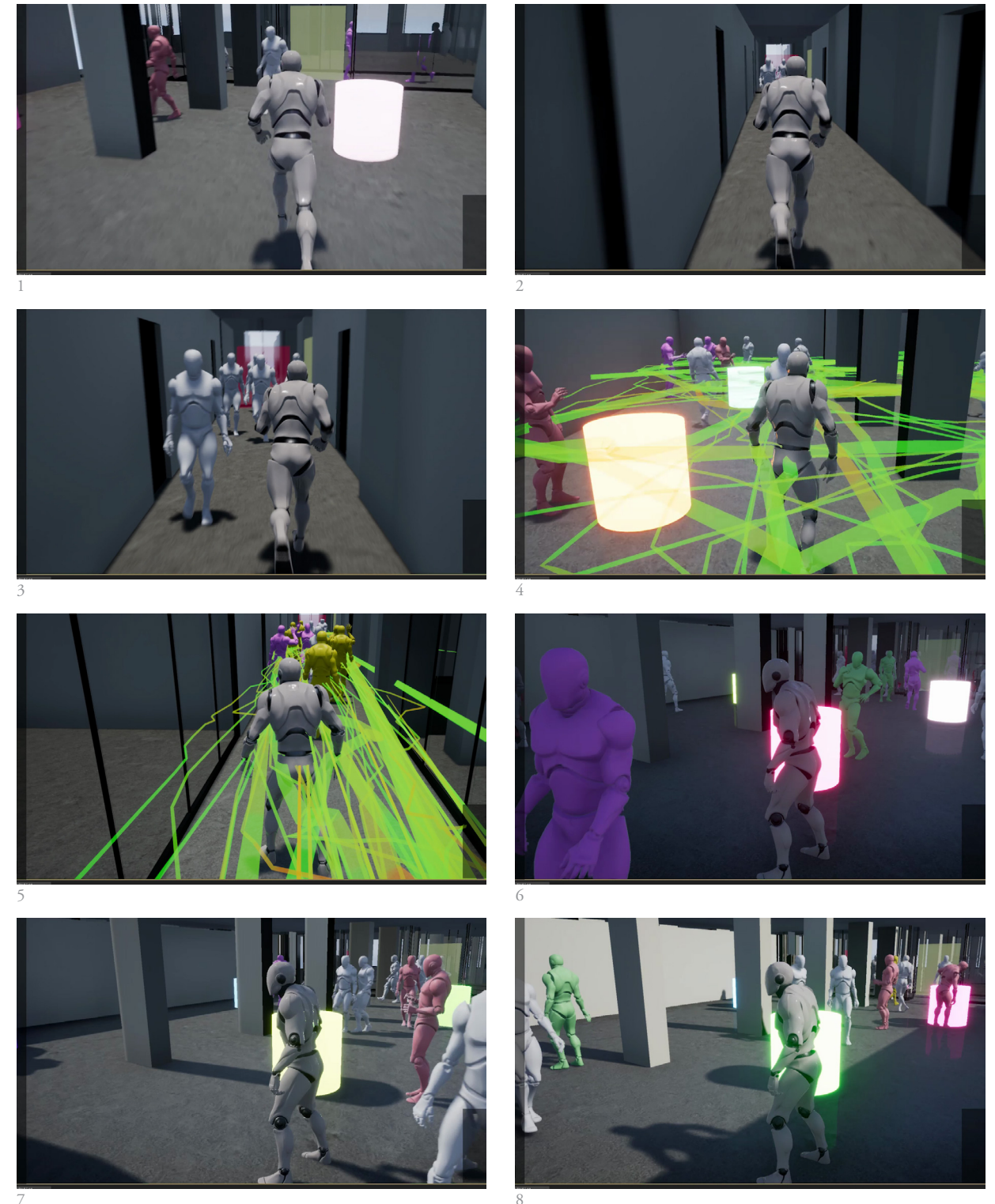
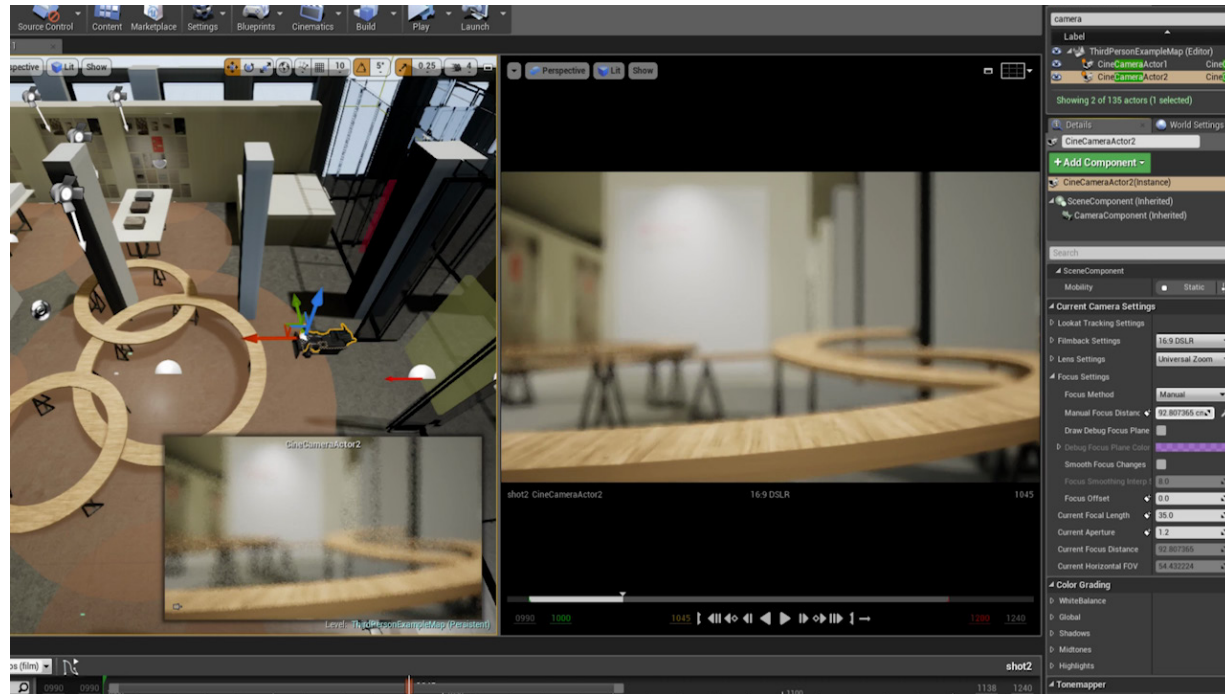


Figure 3.6.16 Interactive perspective variation sequential frames



▶ **Figure 3.6.17** The virtual camera can be utilized to simulate a real camera to produce cinematic footage



Figure 3.6.18 Virtual camera sequential frames

Figure 3.6.17 - 3.6.18

These figures show how we can move the camera to produce cinematic visualizations by using techniques such as truck and dolly movements. Much like a physical camera, we can also adjust the DOF (depth of field) of the camera by controlling the aperture value within the properties. This camera control along with its real-time rendering capabilities allows us to produce cinematic visualizations that are comparable to the film industry but also rendered at a much faster pace. This faster pace of rendering will allow designers to utilize more realistic visualizations that can represent the space in real life even during the iterative process of the design phase, which is substantially more useful as a tool than simply being used for client pitches that are made after the design, such as the examples shown in Chapter 1.2: Inadequacy of Current Visualization Methods.

Part 4 | Tool Evaluation

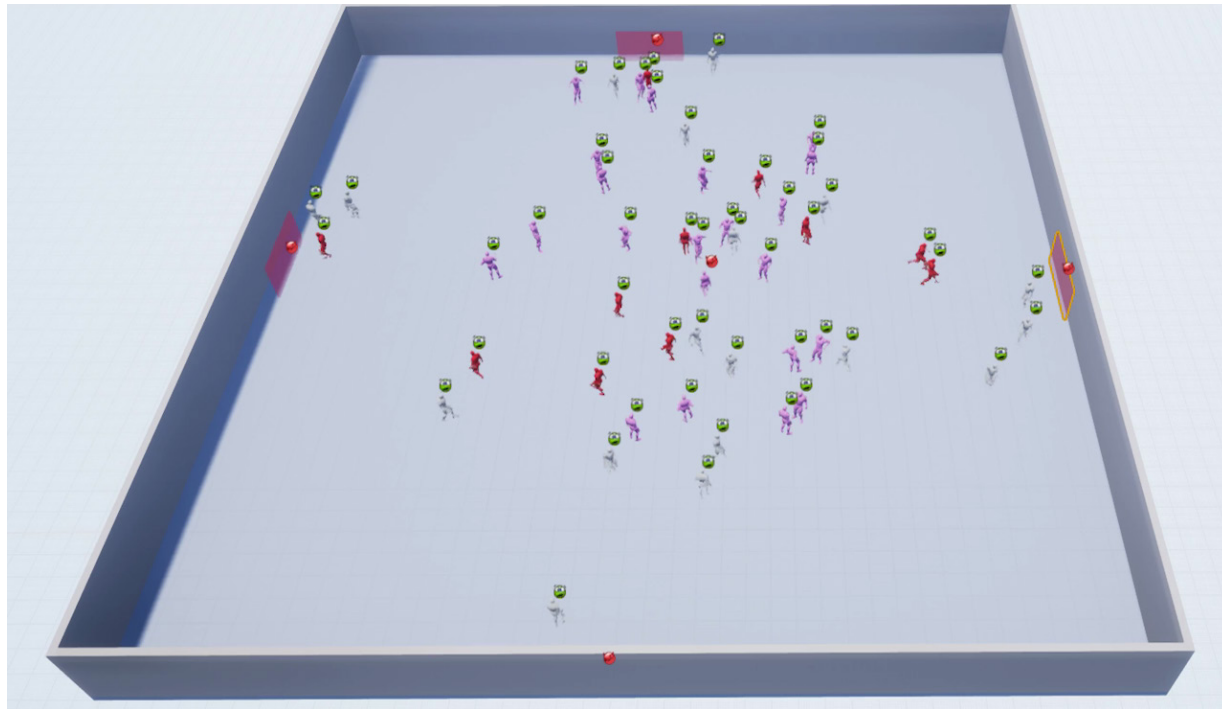
Simulations of Real-World Spaces

Simulations of Real-World Spaces

This section will evaluate the capabilities of the toolset that was established in *Part 3: Tool Creation* by investigating various architectural applications that this simulation framework could be used for. It will do so by first examining generic spatial conditions with the toolset to establish a limit of operation potential. It will then utilize the toolset to simulate a variety of architectural spaces starting from the smallest scale and working its way up to larger scenarios.

Chapter 4.1 | Spatial Conditions

The first step is to test this tool within some generic conditions to get a sense of the crowd flow in various spatial typologies. In doing so, we can confirm the agent's ability to adapt to different spatial scenarios, as well as establish a range of operation potential to better understand the limits of this framework in its current form.



► **Figure 4.1.1** *Open Space Condition*

Figure 4.1.1 - 4.1.2

The purpose of this scenario is to investigate agent behaviors as well as the overall crowd dynamics within an open space without the presence of interactive elements. With this, we can observe the agent behaviors in a predominantly agent to agent interactive space and fine tune its agent interaction percentage parameters to better resemble real world crowds. It can be seen here that a percentage of the agents will begin talking while others will explore the space before finding an exit and leaving. In doing this, we can also observe a rough limit of 100 agents before the simulation starts to slow down with current hardware, and a limit of 50 agents if a screen capture software is running alongside it.

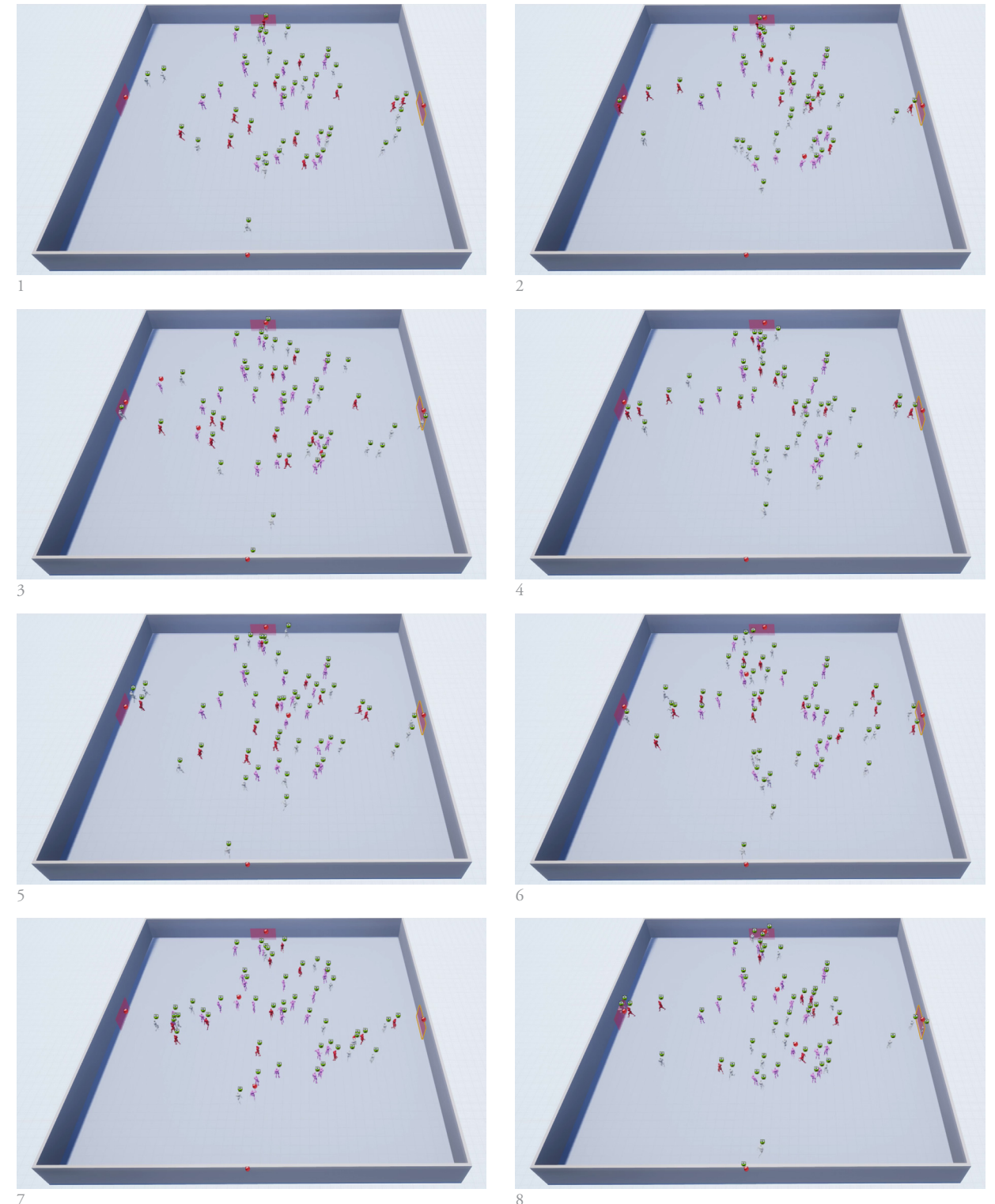
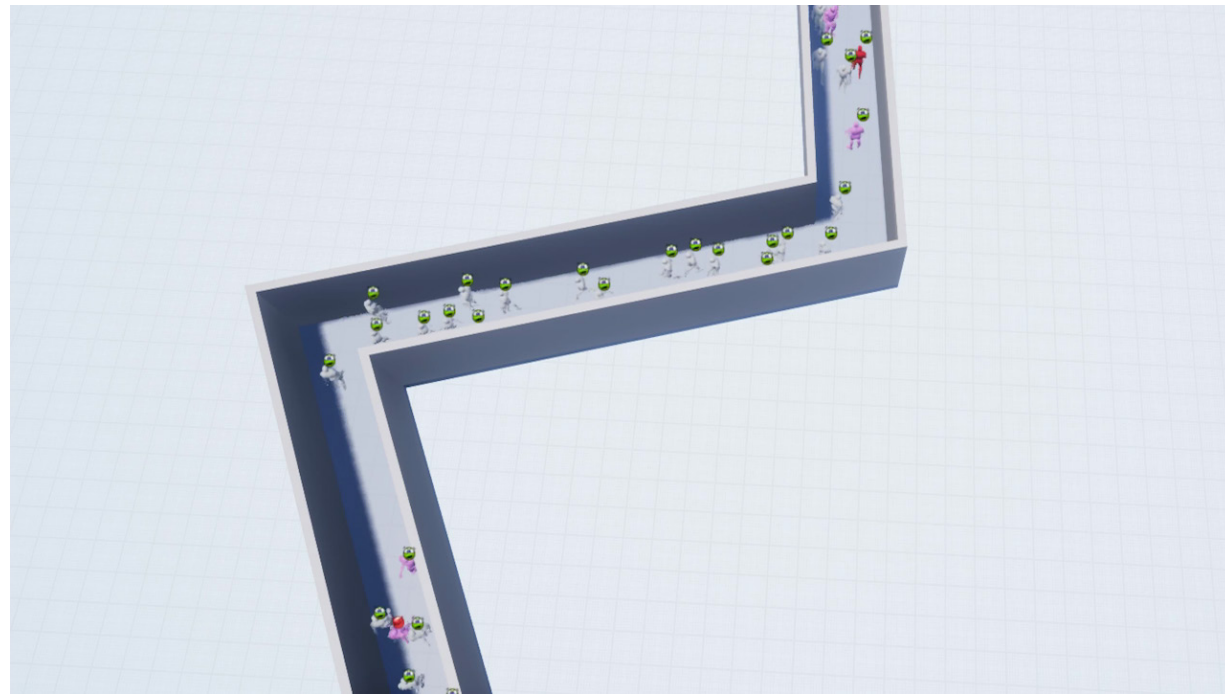


Figure 4.1.2 *Open Space analytical frames*



► **Figure 4.1.3** *Corridor Condition*

Figure 4.1.3 - 4.1.4

These figure looks at how the agents behave within a corridor condition. It can be observed that the agents will walk along the corridor geometry, with a much smaller percentage of them talking due to the lower amount of agent to agent interactions from the smaller more constricted space. It is also interesting to note how the crowd dynamics begin to form lanes similar to the social forces model briefly mentioned from chapter 2, which can be speculated to be caused by the alignment forces that causes people to keep pace with their surroundings.

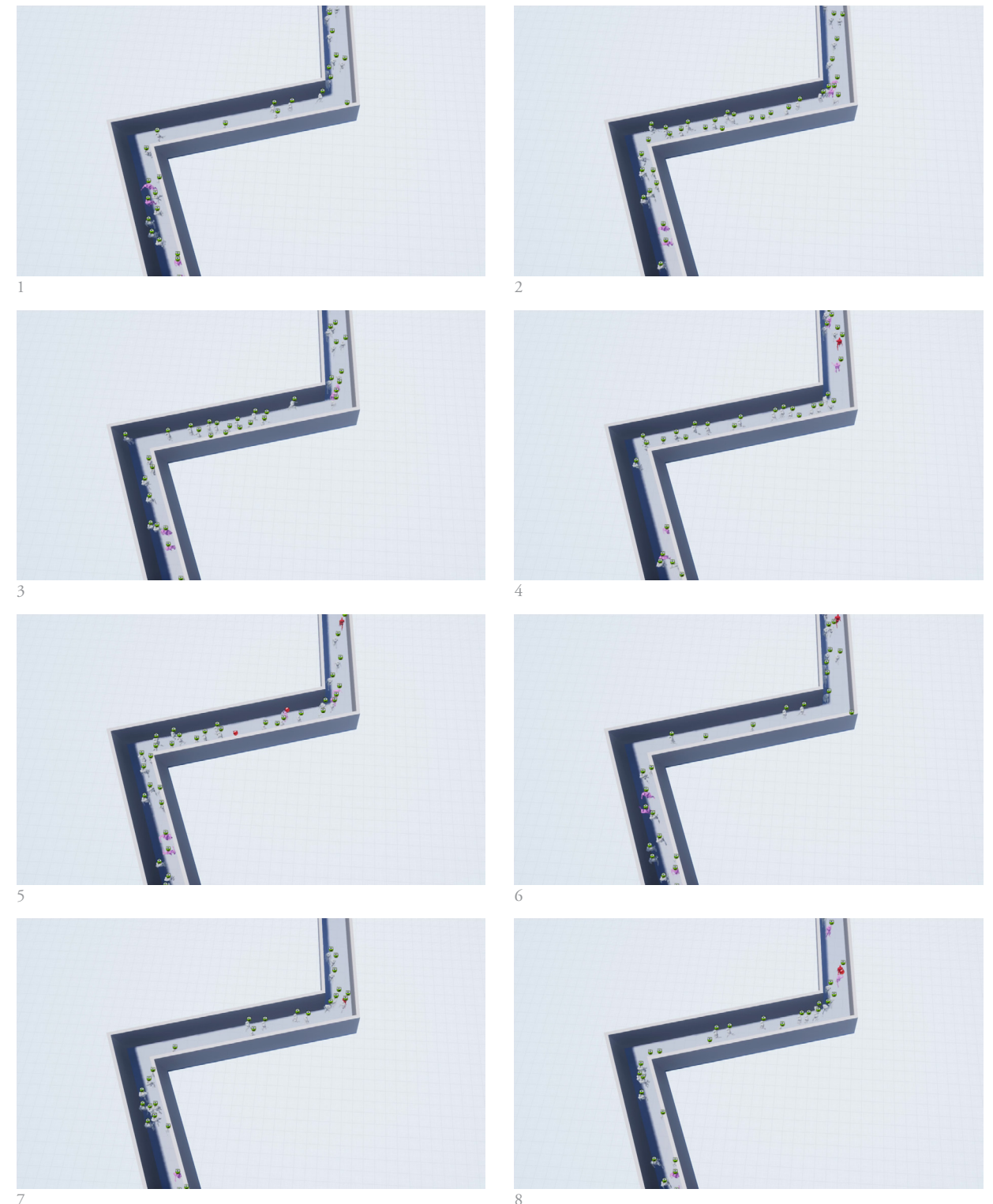
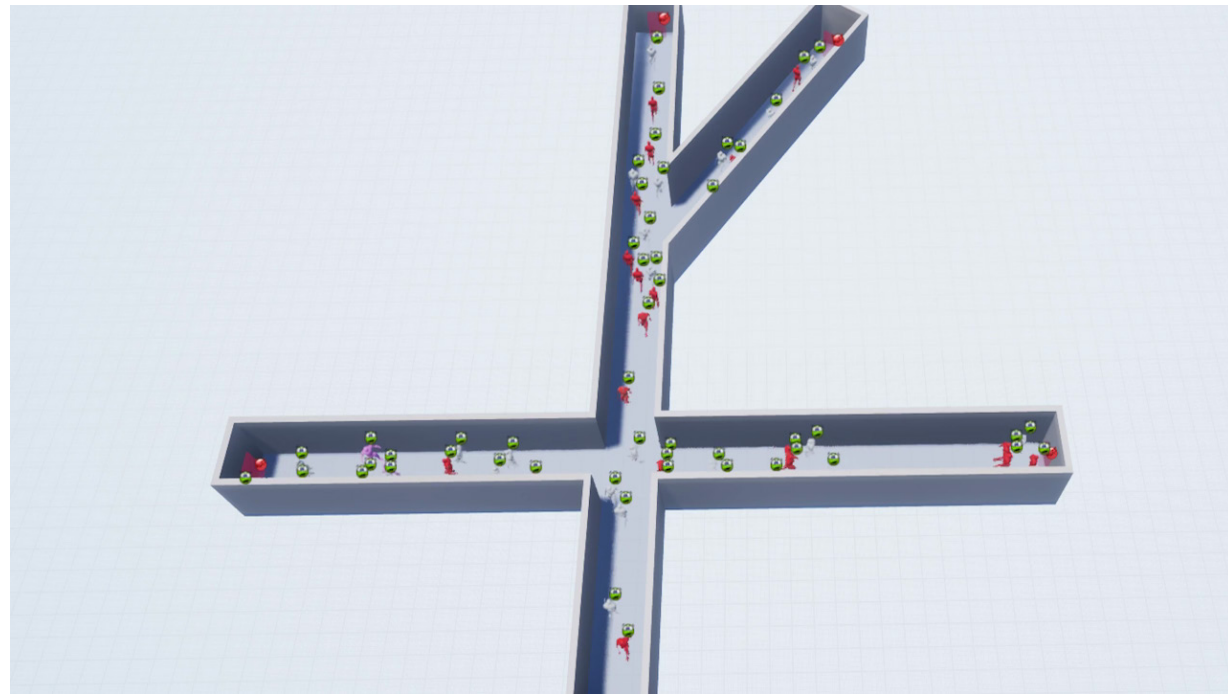


Figure 4.1.4 *Corridor analytical frames*



► **Figure 4.1.5** *Intersection Condition*

Figure 4.1.5 - 4.1.6

This scenario tests how the agents maneuver through intersections. The agents moving through the intersection will subtly change their path to avoid colliding into other agents. When the hallway divides into 2, a percentage of the agents will choose 1 path while another percent will choose the other path.

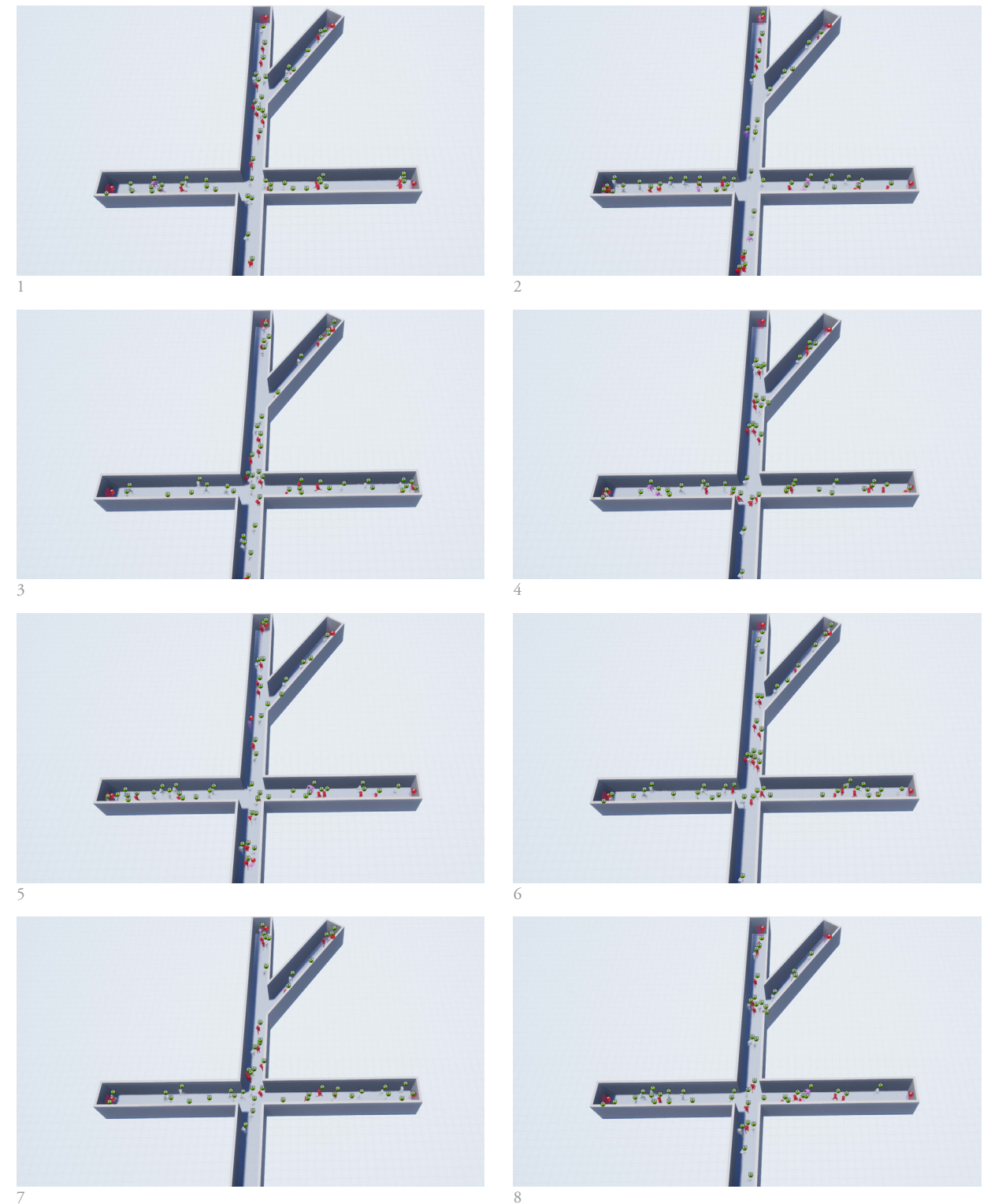
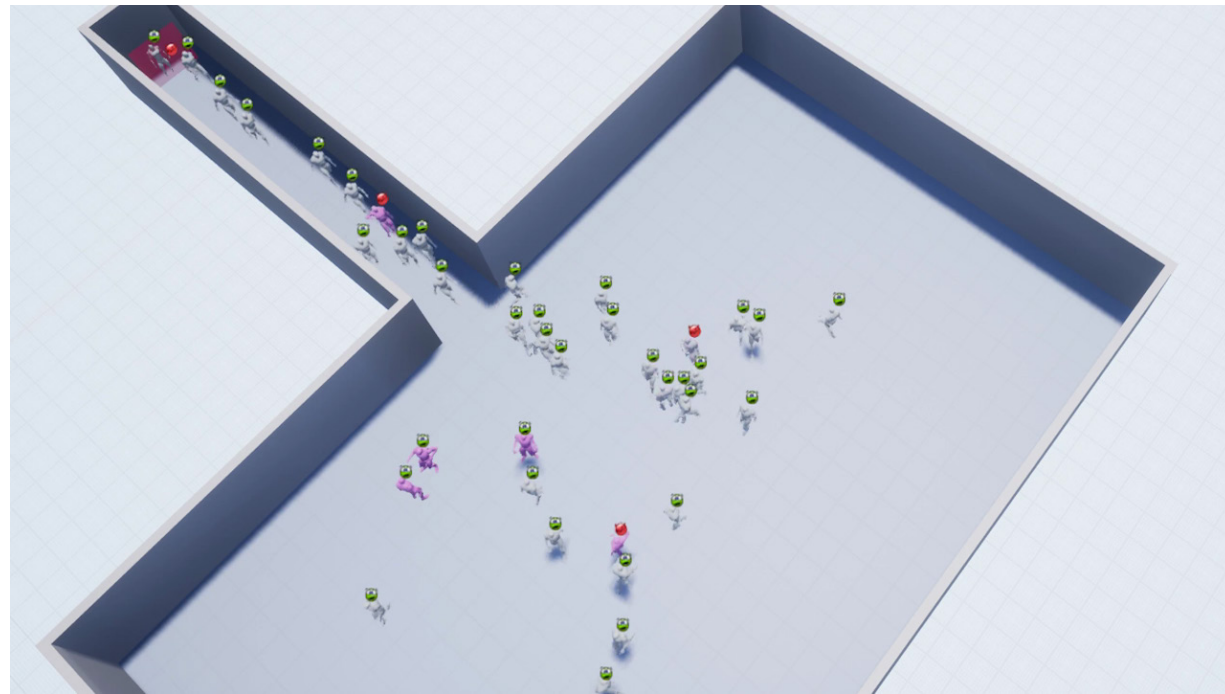


Figure 4.1.6 *Intersection analytical frames*



► **Figure 4.1.7** *Expansion Condition*

Figure 4.1.7 - 4.1.8

This scenario investigates how the agents move from a constricted space to an open space. From these figures, we can observe how the agents fan out to explore as they spill into the open space. This shows how the crowd dynamics of these autonomous agents can change to accommodate the spatial conditions, as the agents conform to the spatial limits of the environment.

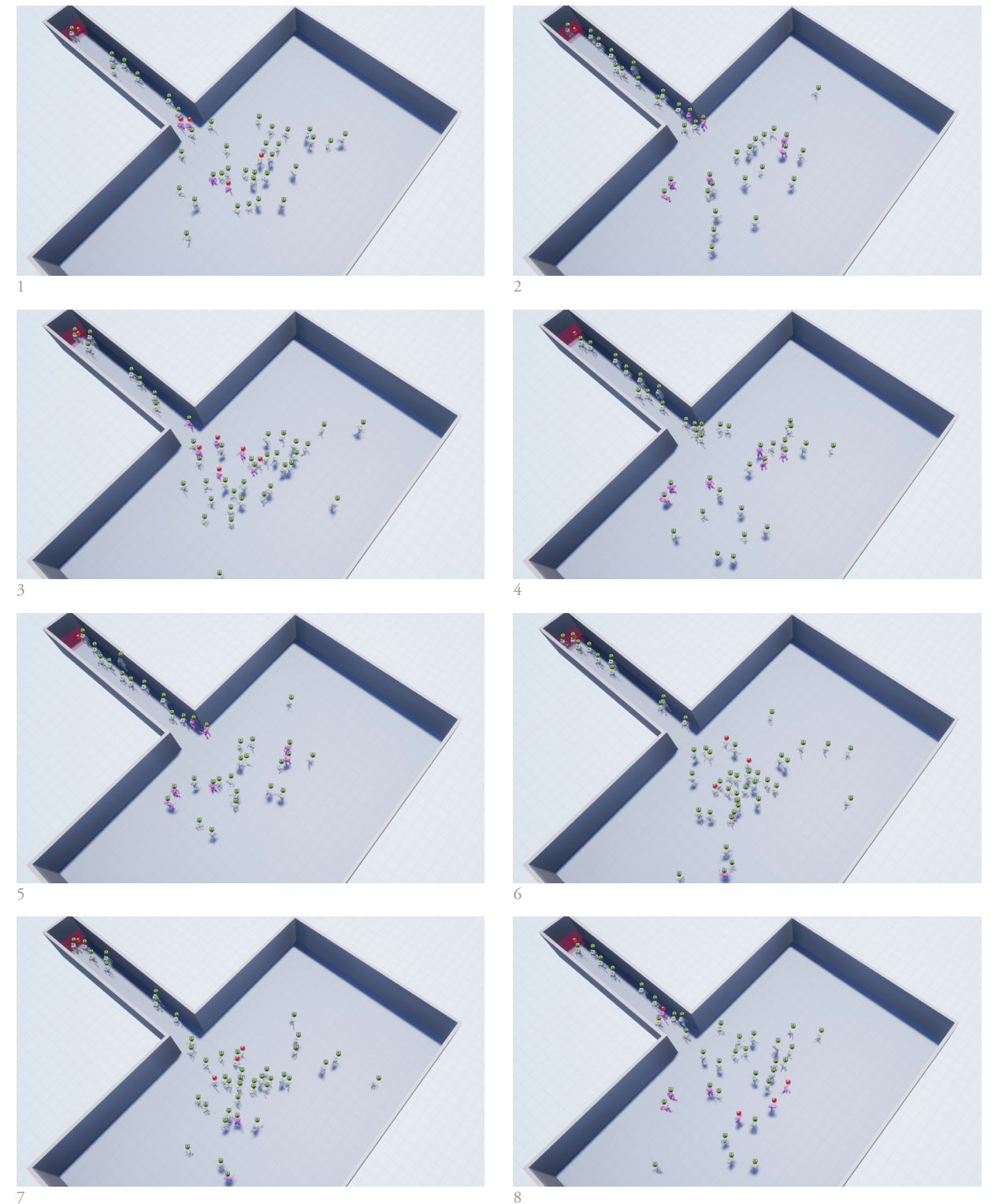


Figure 4.1.8 *Expansion analytical frames*



▶ **Figure 4.1.9** *Open space condition sped-up 20x*

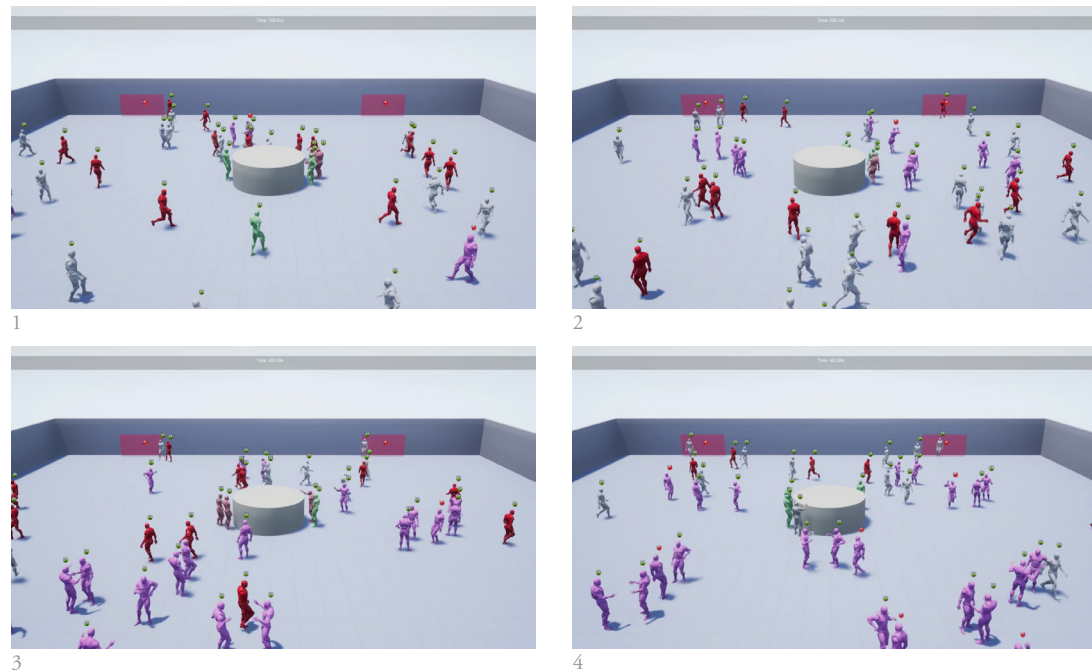


Figure 4.1.10 *Open space condition sped-up analytical frames*

Figure 4.1.9 - 4.1.12

After confirming the Agent's ability to adapt to different spatial scenarios, we can create a rough mock-up of a real-world space to gage how well this crowd simulation can resemble it. Grand Central Station in NYC was chosen for its relatively simple spatial typology of being an open box with thresholds along its edges and a centralized object of interest. As such, these 2 figures compare the generated crowd movements of a virtual animation with a documented time-lapse video of the Grand Central Station. Although the scale of the space and the number of people of the simulation is not exact, the emergence of similar movement patterns can be seen from both examples. The waves of high and low-density crowd movements alongside agent interactions with the centralized object allows the simulated crowd to resemble the real-world crowd dynamics of the space.



▶ **Figure 4.1.11** *Grand Central Station time-lapse*

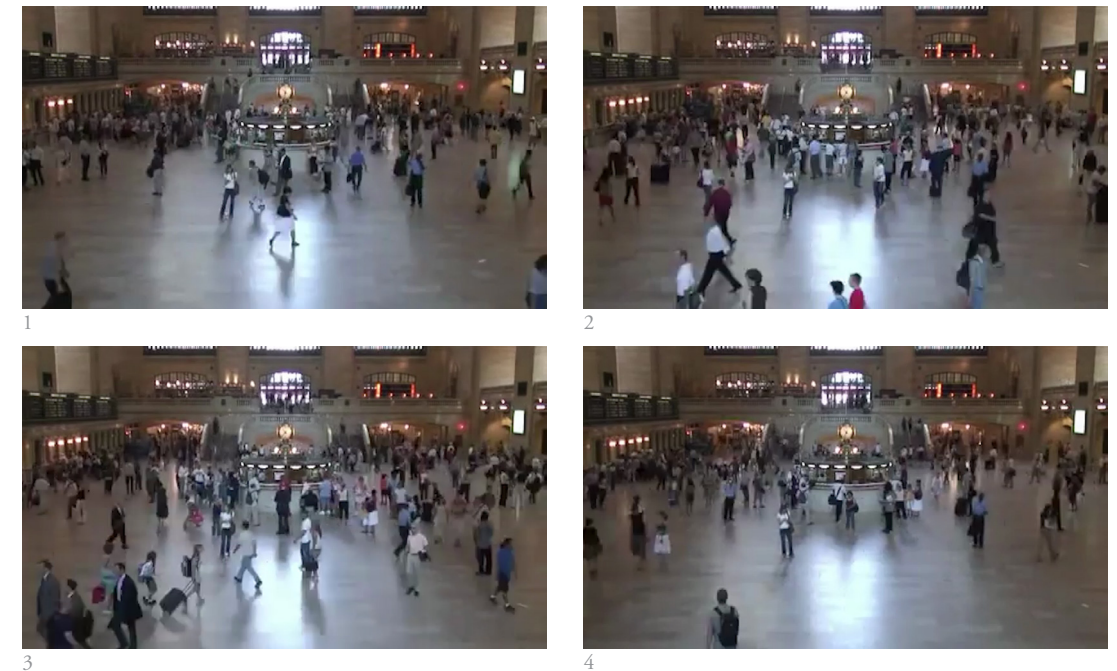
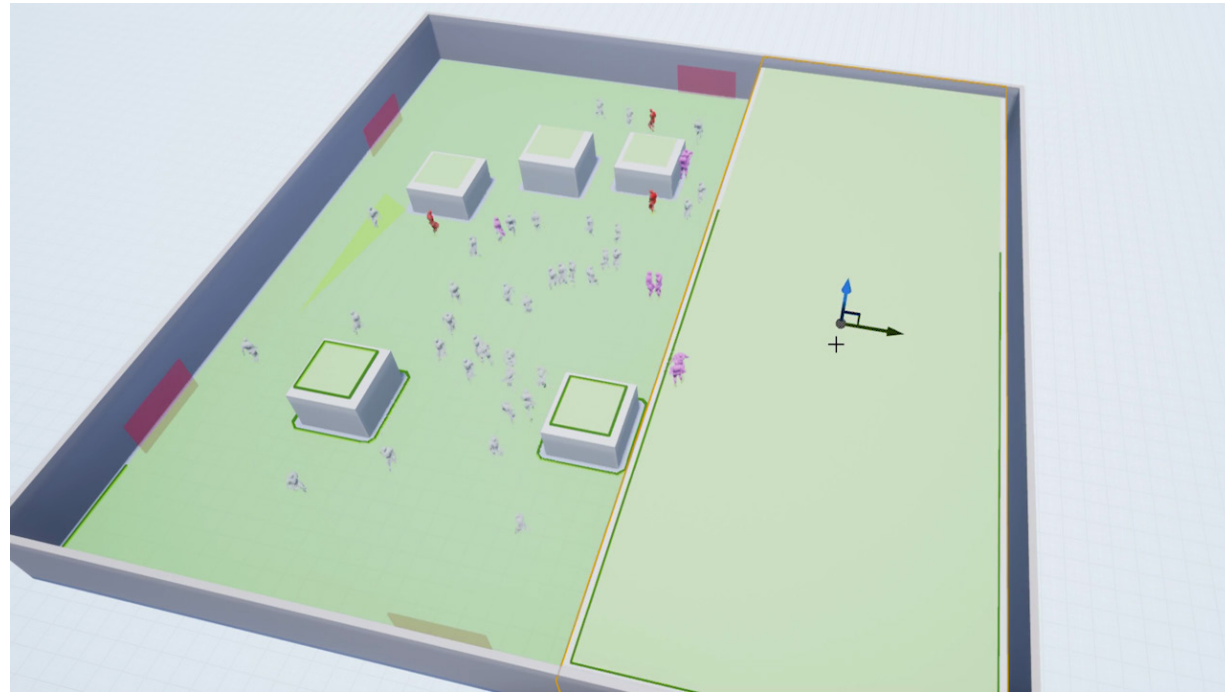


Figure 4.1.12 *Grand Central Station time-lapse analytical frames*



► **Figure 4.1.13** *The simulated agents will react and adapt to a changing environment in real time.*
Note how the green Navmesh area is updated in real time as the space is manipulated.

Figure 4.1.13 - 4.1.14

Since these agents operate as autonomous agents with sight and decision making, they have the ability to react to a changing environment. This not only allows us to test different design scenarios and sizes in real time, but also allows the agents themselves to interact with scripted dynamic objects within the space, which in turn allows us to visualize how these interactive and dynamic spaces may impact the occupants during the iterative process of the design phase. With this, the simulation has the potential to create an interplay of dynamic systems—people interacting with architecture, and architecture interacting with people, which allows us to consider different geometries and scenarios based on these interactions during the design phase.

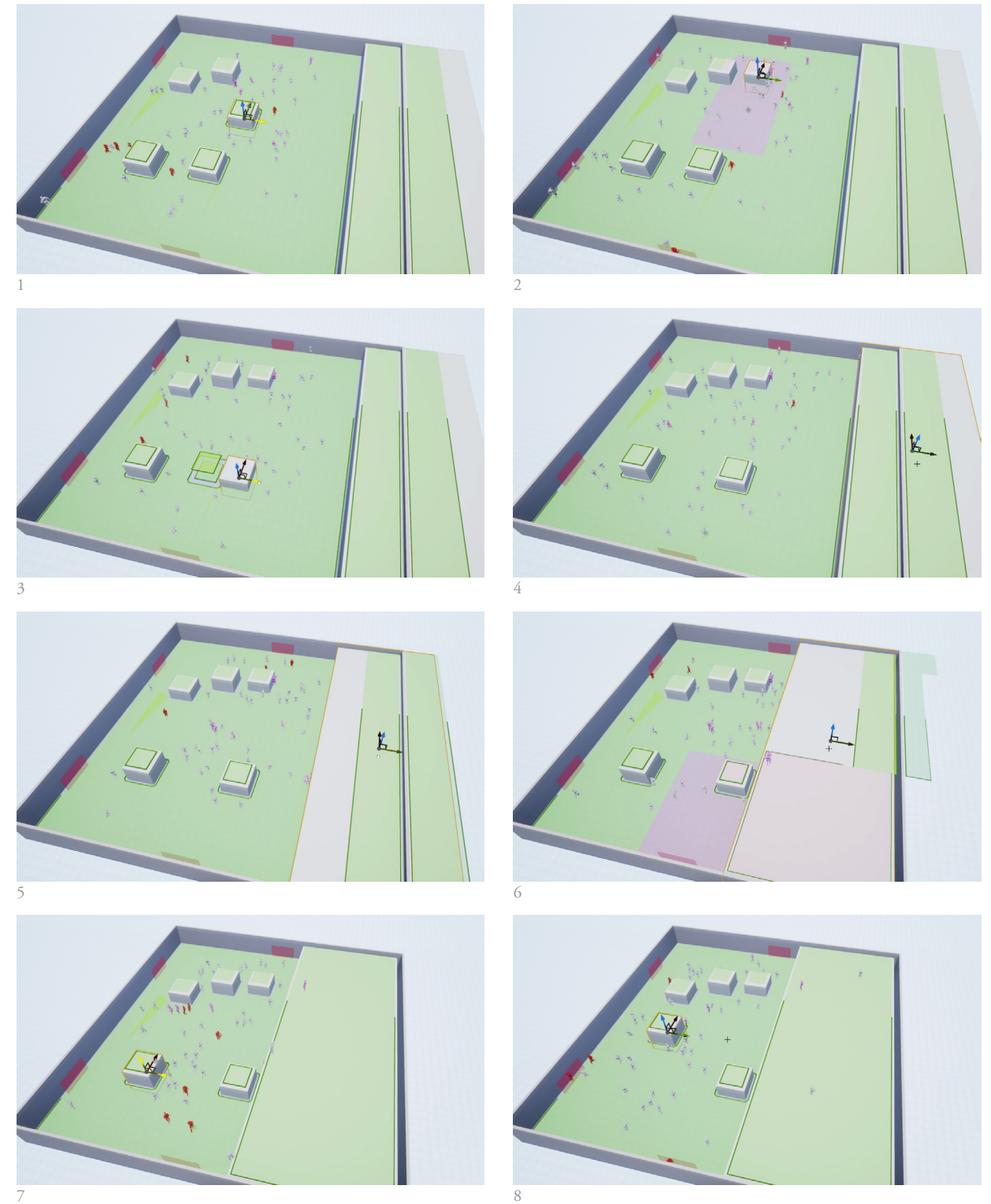


Figure 4.1.14 *Adaptive agents sequential frames*

Chapter 4.2 | Nuit Blanche Cushion

After testing the agents within various spatial conditions and establishing their limits, we can evaluate this simulation tool for architectural visualization use by first simulating architectural spaces at the smallest of scales. To do so, we can re-simulate *Cushion*—the Nuit Blanche Toronto installation space we prototyped in chapter 2.5—which allows us to compare our simulation tool with our prototype to see its progression. This space also fits our criteria for providing a scenario that is both small in scale and inherently involves crowds of people interacting with dynamic elements.

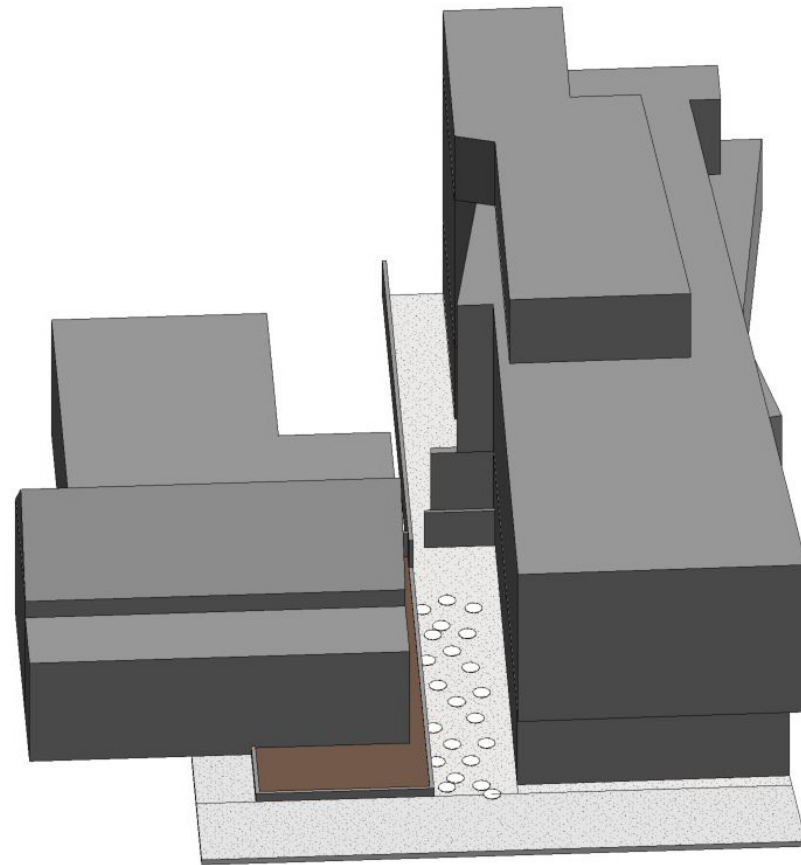


Figure 4.2.1 *Cushion Revit model*

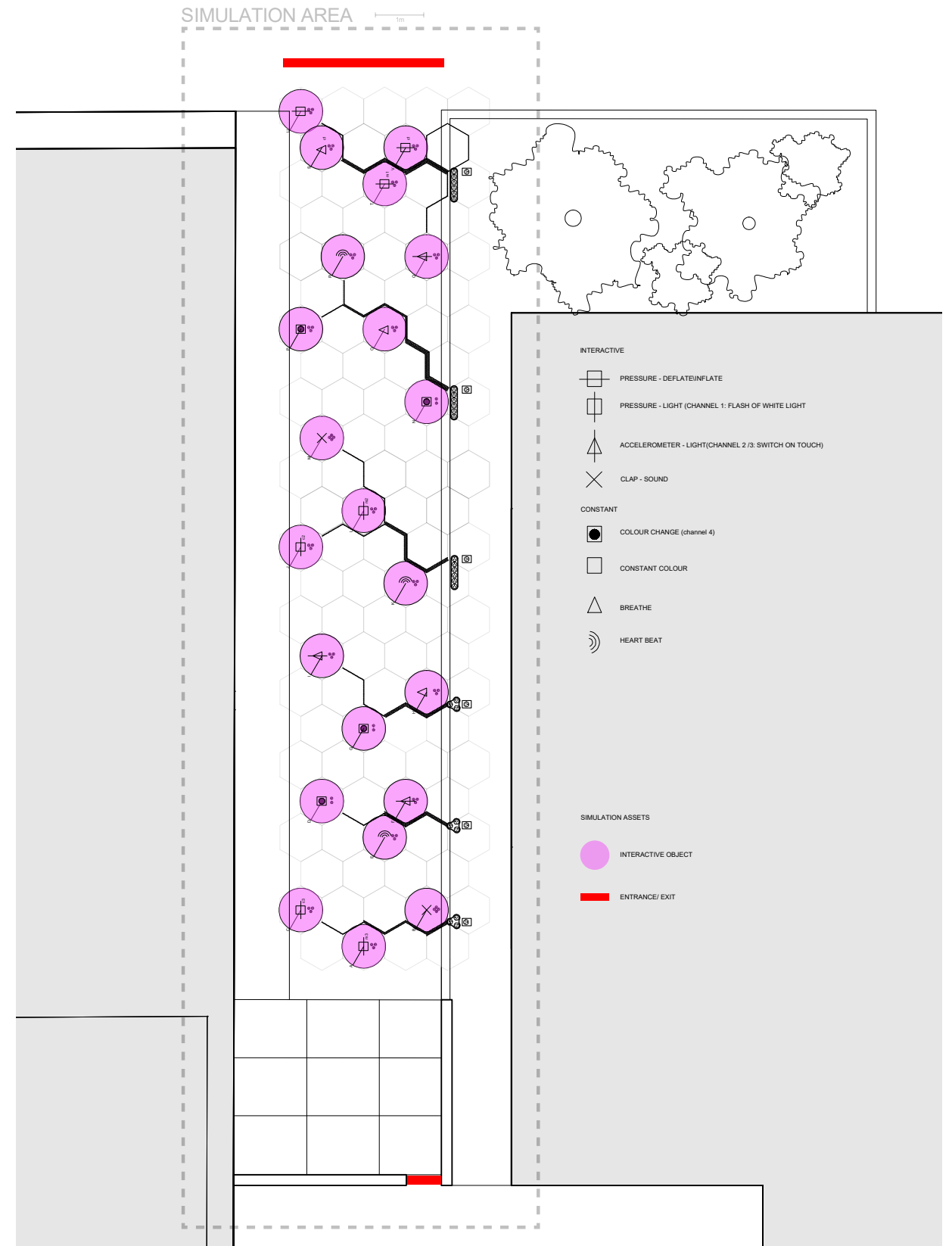


Figure 4.2.2 *Cushion Floor Plan*



► **Figure 4.2.3** *Cushion crowd simulation*

Figure 4.2.3 - 4.2.4

Within this first simulation, we can observe that the agents are behaving as anticipated. They are first generated at one end where they can come into the space and explore the installation. As they move through the alleyway, they will interact with the various cushion objects within the space, which in turn causes the objects to change in color. Once they have reached the end, they will exit the space. It is also interesting to note how the more open space at the back allows for more agent to agent interactions, which in turn facilitates an area of conversation within the space.

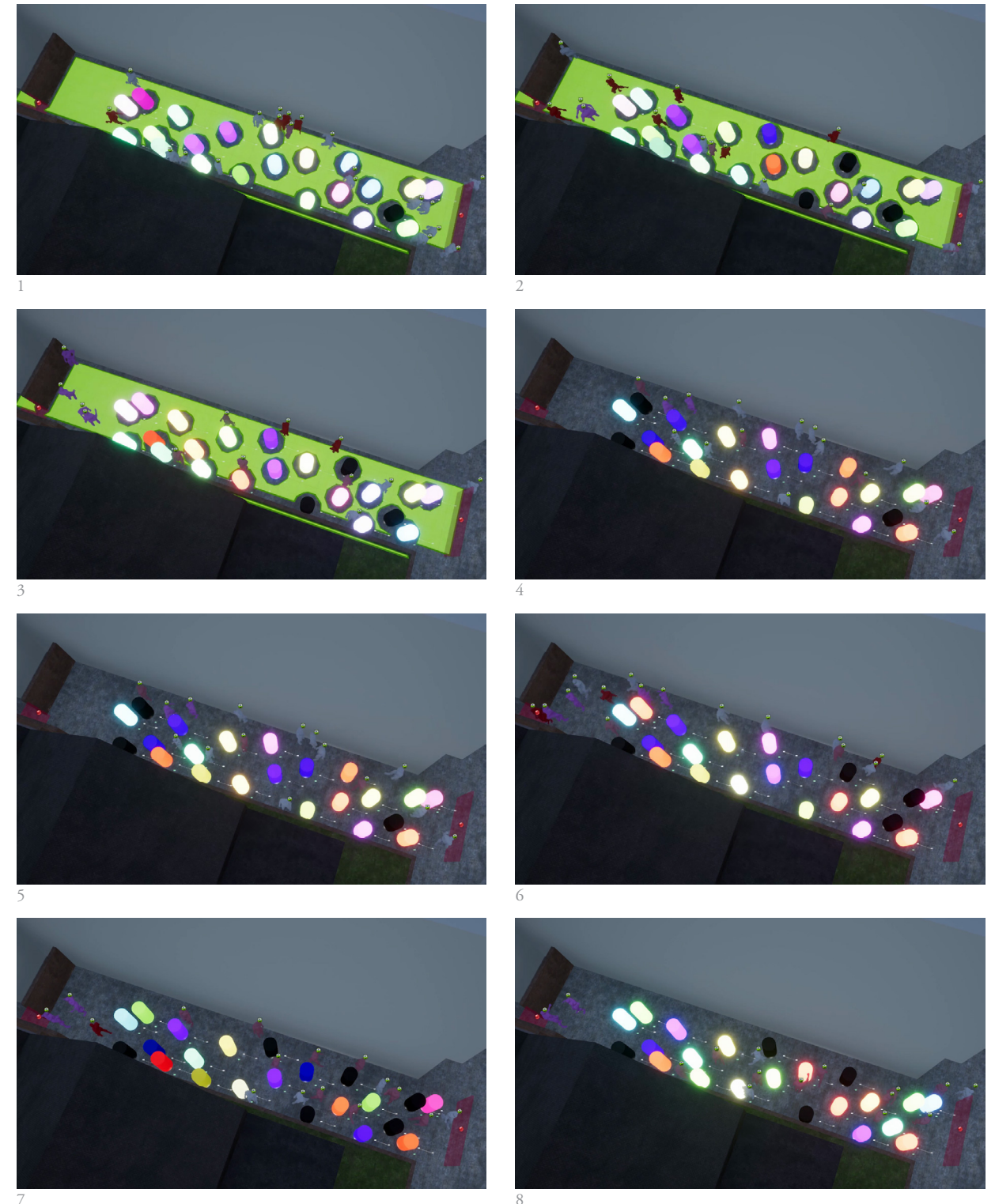
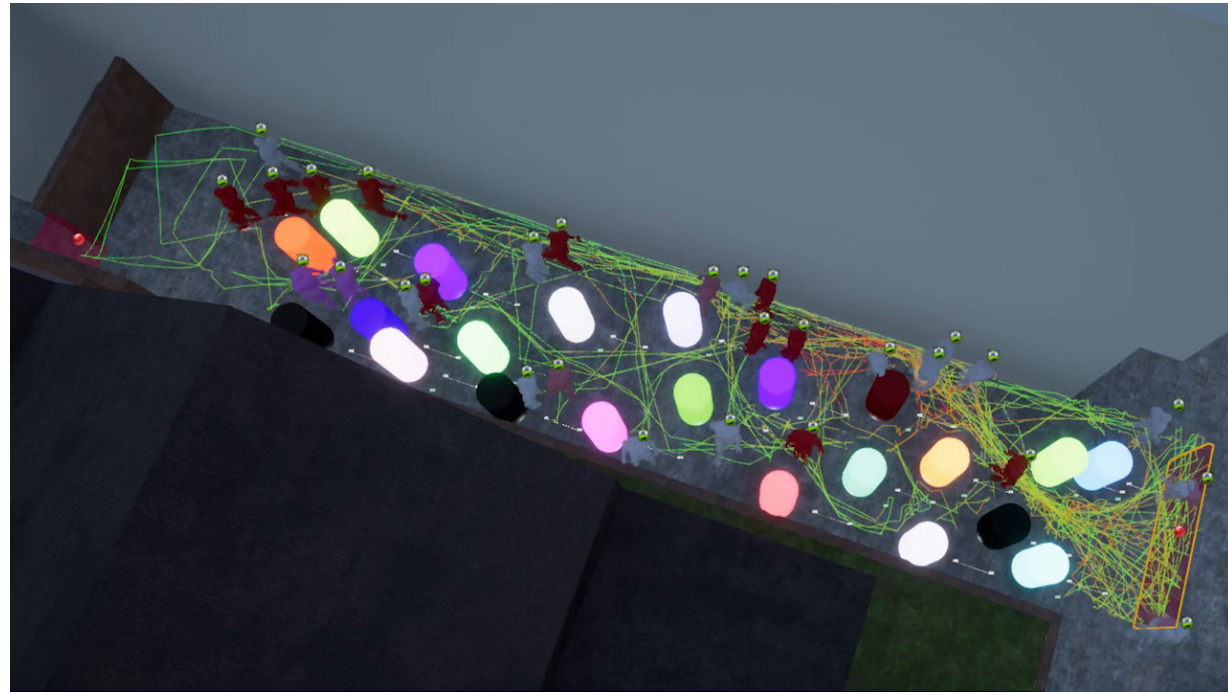


Figure 4.2.4 *Cushion crowd simulation analytical frames*



► **Figure 4.2.5** *Cushion agent-comfort map*

Figure 4.2.5 - 4.2.6

These figures shows how the comfort trails can highlight a potential area of congestion. This result makes sense as the congestion area occurs at one of the first bottlenecks of the installation plan when entering the space. As such, it can be assumed that placing an object of high interest here will worsen the effect.



Figure 4.2.6 *Cushion agent-comfort map analytical frames*



▶ **Figure 4.2.7** *Cushion real world footage*



Figure 4.2.8 *Cushion real world footage analytical frames*



▶ **Figure 4.2.9** *Cushion rendered visualization*

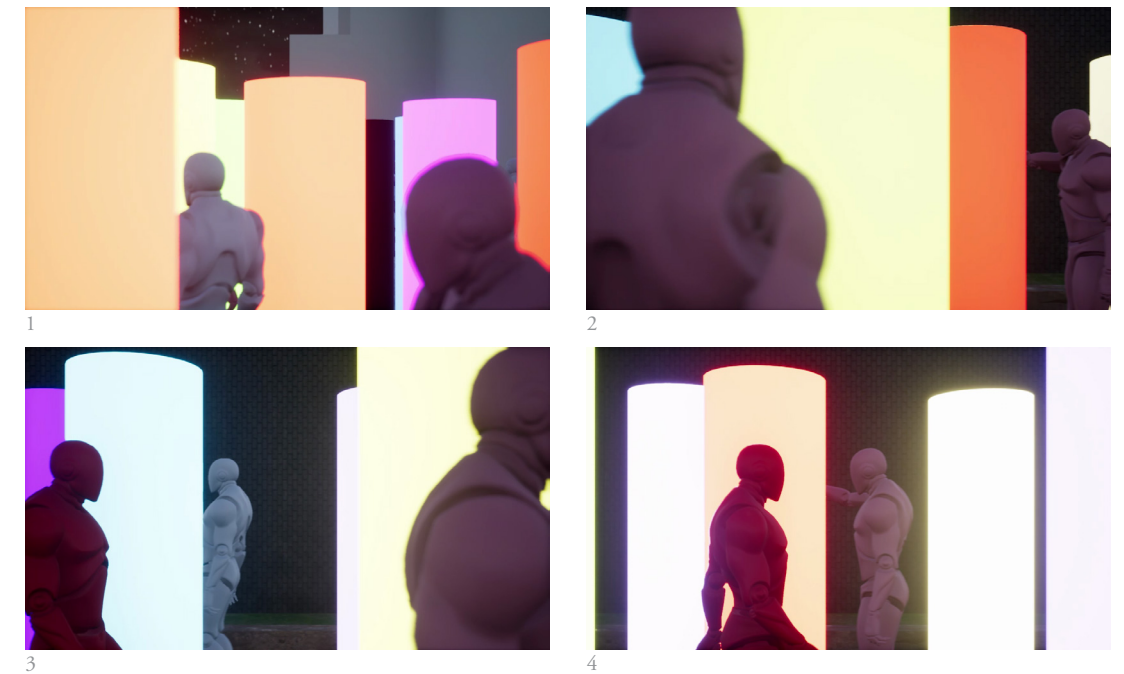


Figure 4.2.10 *Cushion rendered visualization analytical frames*

Figure 4.2.7 - 4.2.10

Utilizing virtual cinematic cameras allows us to render out animations in real time. In doing so, we can match our visualization to real world footage, which allows us to compare the experiential qualities of the installation versus the simulation. From this, we can see that while the simulation is not an exact replica, it is still much better than no agents at all at conveying the usability and experiential qualities of the space.



▶ **Figure 4.2.9** *Cushion rendered visualization*



Figure 4.2.10 *Cushion rendered visualization analytical frames*

Figure 4.2.9 - 4.2.12

While it can be argued that the simulated agents are not fully accurate when comparing to the real world, comparing them further to the same visualization without the agents reveals how much impact the addition of these simulated agents can bring to the resulting spatial visualization. Without the agents, not only does the visualization convey less about the experience of the space, but the dynamic interactive elements within the simulation also become subdued without the agents providing a source of input to interact with them.



▶ **Figure 4.2.11** *Cushion rendered visualization without simulated crowds*

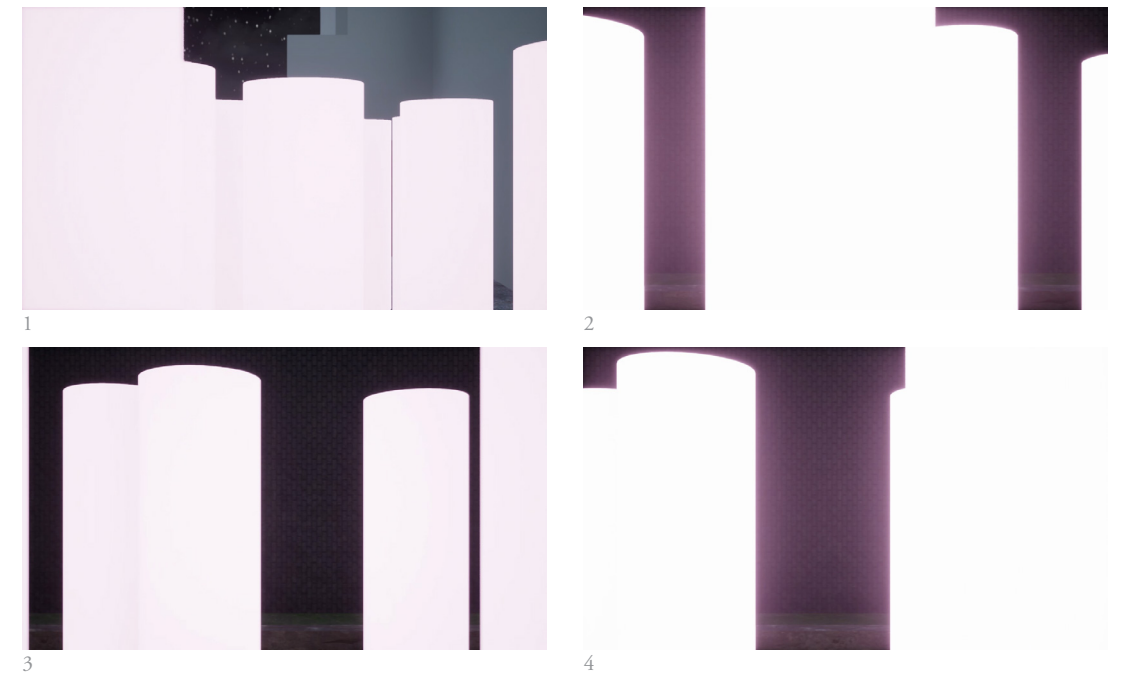


Figure 4.2.12 *Cushion rendered visualization analytical frames without simulated crowds*

Chapter 4.3 | Riverside Gallery

After investigating the installation space, we can utilize this tool further within a larger and more practical context. As such, the Riverside Gallery in Cambridge, Ontario was chosen. It is a space that not only fits these criteria of being larger and more 'real world' than an installation, but its location within the architecture school also allows an ease of documentation and translation of this space into the virtual environment.

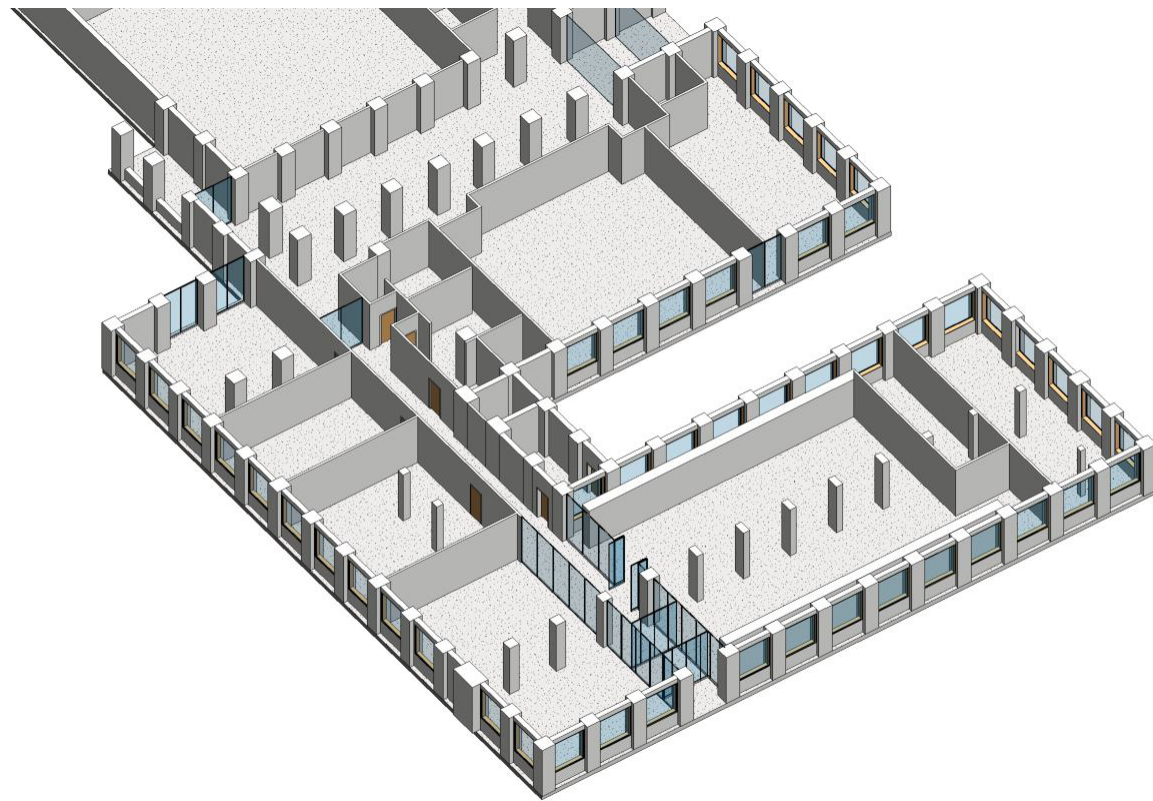


Figure 4.3.1 Riverside Gallery Revit model

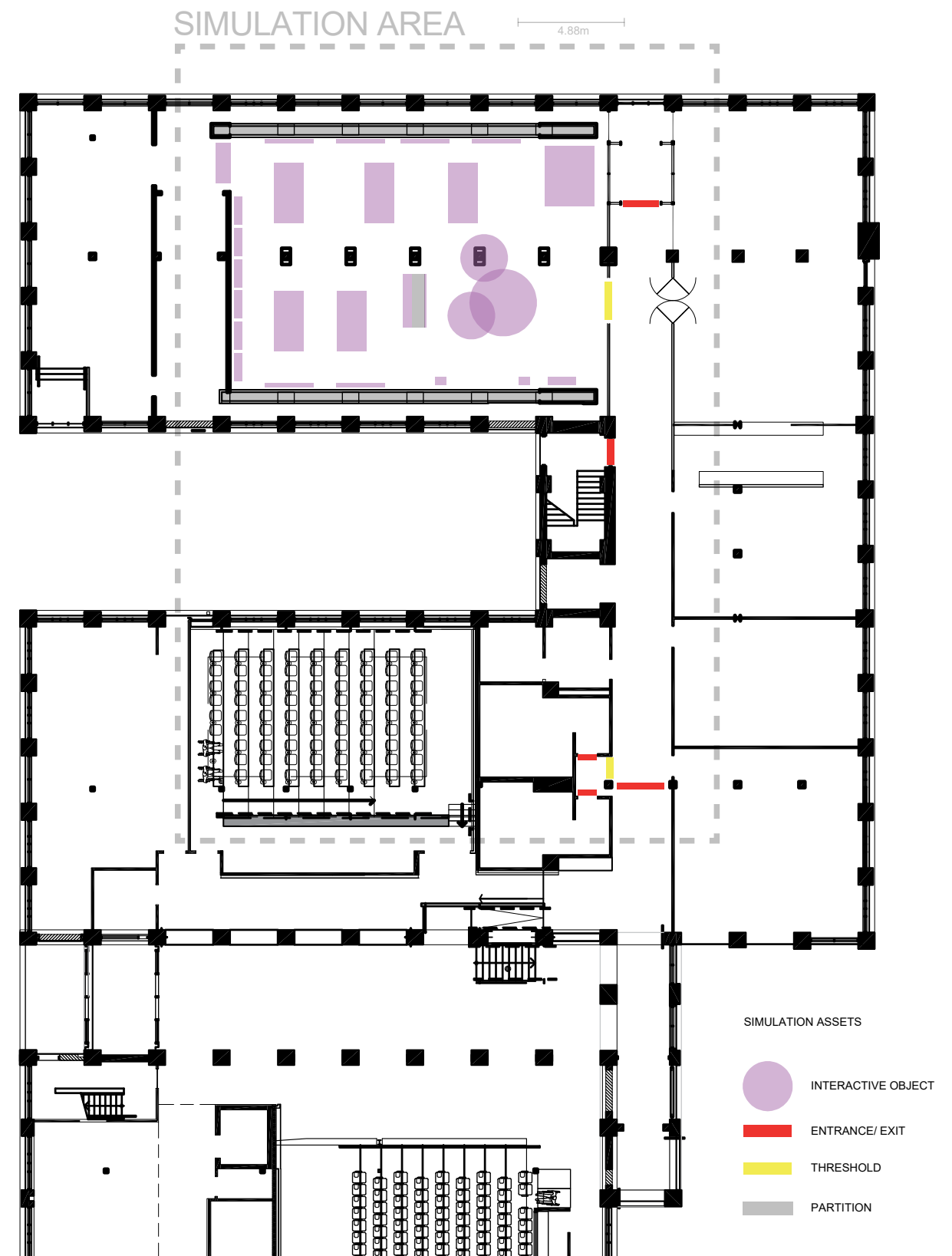


Figure 4.3.2 Riverside Gallery Floor Plan

It should be noted that the simulation area does not need to encompass the entire building. The entrance/exit assets can be used to define the boundaries of a space as long as it is a closed system.



► **Figure 4.3.3** *Riverside Gallery crowd simulation*

Figure 4.3.3 - 4.3.4

This simulation shows how the agents move around within more traditional spaces such as an art gallery. While the furniture elements within this gallery—such as tables and displays—are not dynamically interactive like the ones seen in cushion, they can still be considered interactive objects within the space, since the agents can manipulate and move them. This figure shows how the agents are drawn towards the gallery as they enter the building, with some of them heading down the hallway to the bathrooms or the school. As they walk around the gallery, they will observe and interact with the displays within the space, or begin talking with other agents.

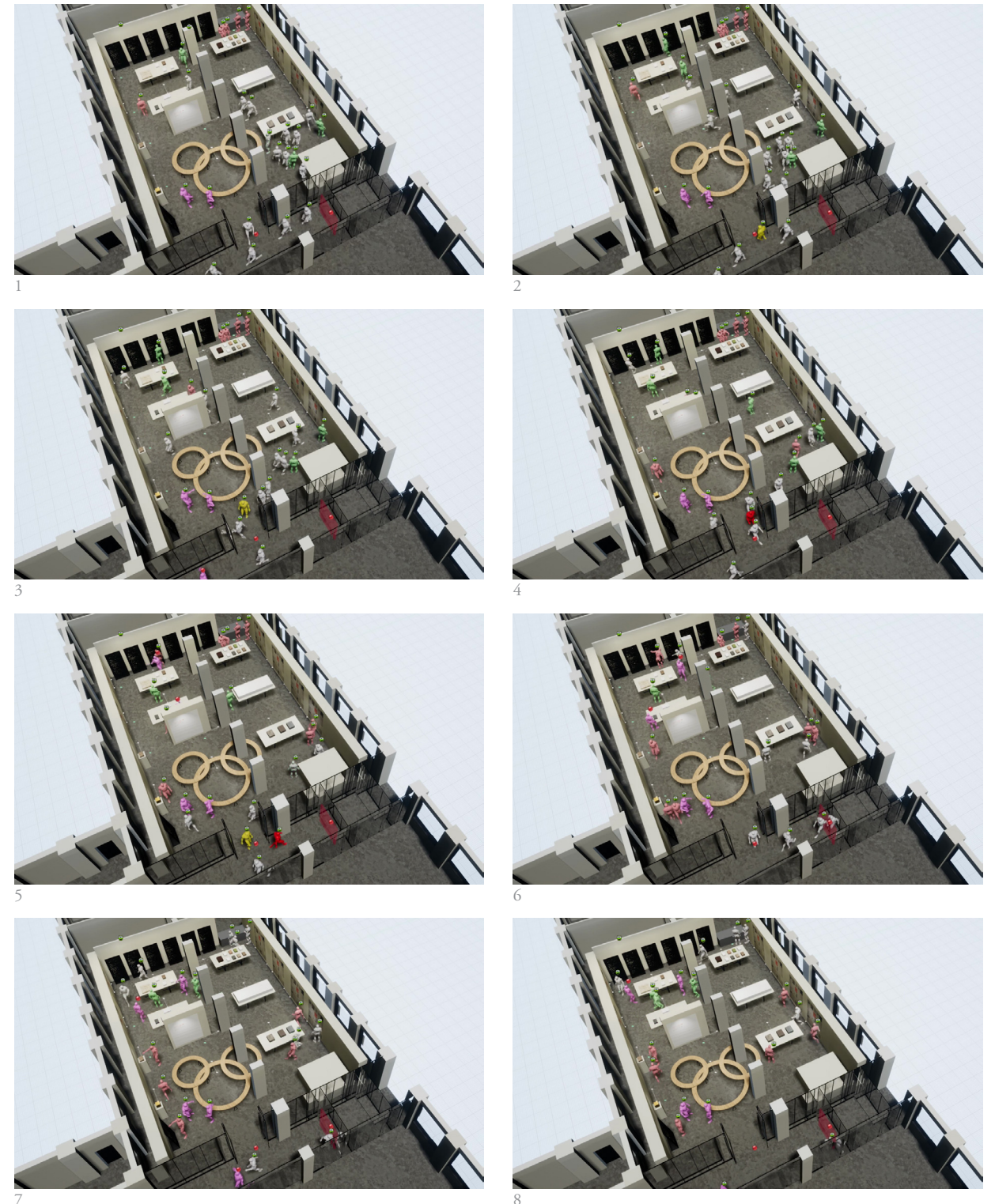
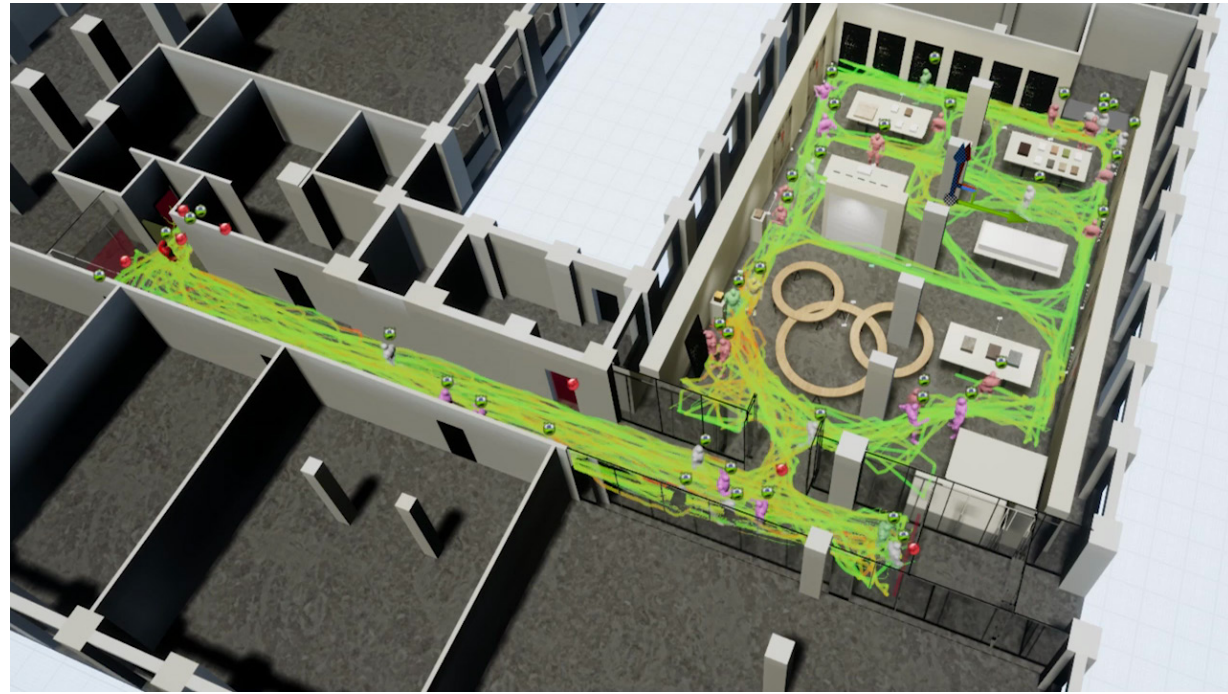


Figure 4.3.4 *Riverside Gallery crowd simulation analytical frames*



▶ **Figure 4.3.5** Riverside Gallery agent-comfort map

Figure 4.3.5 - 4.3.6

The generated comfort trails here highlight a possible area of congestion by the display, which makes sense as it is a physical bottleneck around an object of interest near the entrance of the gallery space.

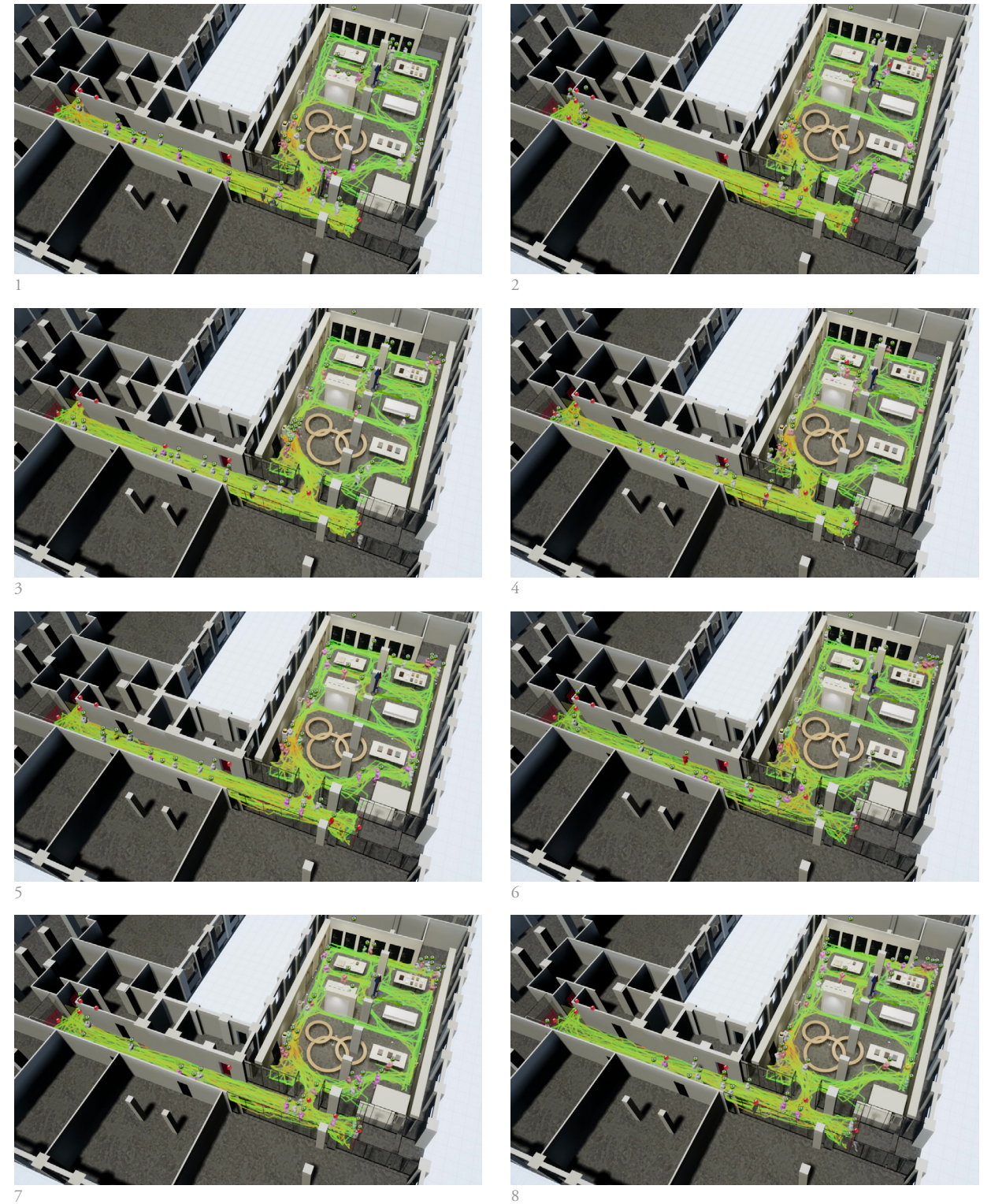


Figure 4.3.6 Riverside Gallery agent-comfort map analytical frames



▶ **Figure 4.3.7** Riverside Gallery real world footage

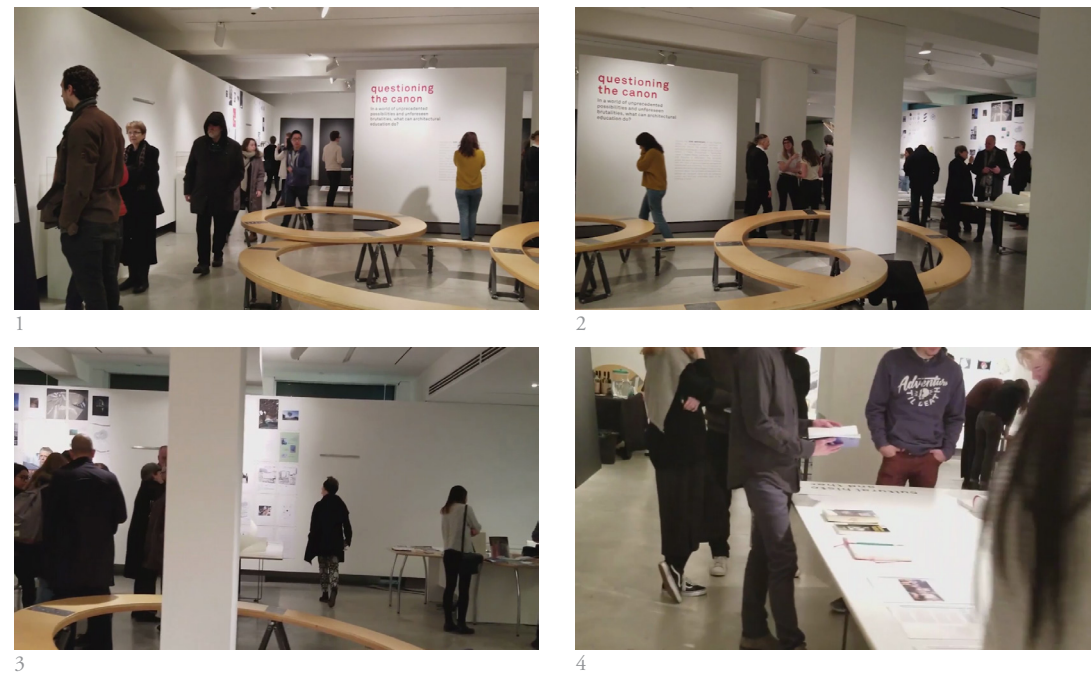


Figure 4.3.8 Riverside Gallery real world footage analytical frames



▶ **Figure 4.3.9** Riverside Gallery rendered visualization



Figure 4.3.10 Riverside Gallery rendered visualization analytical frames

Figure 4.3.7 - 4.3.10

Utilizing the virtual cinematic cameras allows us to once again render out animations in real time. This comparison between the real-world footage and the simulation is somewhat more exact, mainly due to the increased model detail of the interactive elements within the space.



▶ **Figure 4.3.9** Riverside Gallery rendered visualization



▶ **Figure 4.3.11** Riverside Gallery rendered visualization without simulated crowds



Figure 4.3.10 Riverside Gallery rendered visualization analytical frames



Figure 4.3.12 Riverside Gallery rendered visualization analytical frames without simulated crowds

Figure 4.3.9 - 4.3.12

Once again, comparing this animated visualization to one without agents at all, highlights how even generic humanoid agents can better convey the usability and experience of the space. Without these agents, the space feels a lot less authentic when compared to the real world.

Part 5 | Next Steps

What the Future Holds

What the Future Holds

This section concludes this exploration by acknowledging potential directions to investigate and various aspects that can be improved upon in the future. Since the initial aim was to create a *foundational* framework for visualizing dynamic spaces, it makes sense that there are many elements that can be improved upon and developed beyond the scope of this thesis. As such, improvements to this framework in the near future can be broken down into the following categories: *Simulation Improvements*, *Photorealism*, *Virtual Reality*, and *Workflow Refinements*.

Chapter 5.1 | Simulation Improvements

The first of these improvements is the simulation itself. The last section (*Part 4: Simulation Applications*) investigated various applications that this simulation framework could be used for; starting with generic basic conditions to small scale installations and working its way up to larger ‘real world’ spaces such as galleries. From these investigations, it can be seen that the agents are still somewhat unsophisticated with jerky movements when compared to people in the physical world. While this is largely unnoticeable when more agents are generated within the simulation, it still shows that there is room for improvement. At the same time, there also seems to be a limit to the number of occupants that can be generated before the entire system slows down significantly, which can limit the variety of spaces that this tool can be used for. As such, it is evident that this model can be improved both in terms of optimization and accuracy. To do so, we must first finalize and organize the simulation model. From here, we can optimize performance by utilizing C++ and increase accuracy with additional considerations to the simulated agents. We can then use this new model to establish new features that can be used for architectural visualization.

Organization: Finalizing the Simulation Model

Before even considering these improvements, however, it is important to first organize and simplify the simulation model. At its current state from *Chapter 3.3: Human Agents*, these assets are messy with leftover ‘experimental logic’ due to the prototyping process, which leaves a lot of unnecessary variables and functions to wade through when modifying the logic.

While this was deliberate in learning this software, it becomes inefficient in the process of adding additional functionalities. Therefore, the first step that should be taken beyond this thesis is to go through the entire logic of the simulation model and determine the necessary portions of code. From there, it would then become possible to organize the logic into macros and functions in order to create a library of reusable nodes for future modifications. Doing this will solidify the foundational aspect of this framework, which will facilitate an easier process to add additional considerations and features to the crowd simulation tool.

Optimization: Blueprints vs C++

Within UE4, when a blueprint is executed, it is calling back to the C++ code that was written for it. As such “there is an overhead cost associated with executing blueprints that isn’t present with purely native code.”^[1] This lack of an overhead can allow the native C++ code to outperform blueprints by up to 10 times, which becomes significant as we begin adding more people to the simulation. (Fig. 5.1.1)

¹ Irascible, comment on “[Twitch] Fortnite Developers Discussion - Apr. 17, 2014,” Unreal Engine Forums, accessed October 18, 2019, <https://forums.unrealengine.com/unreal-engine/events/3192-twitch-fortnite-developers-discussion-apr-17-2014/page2>.

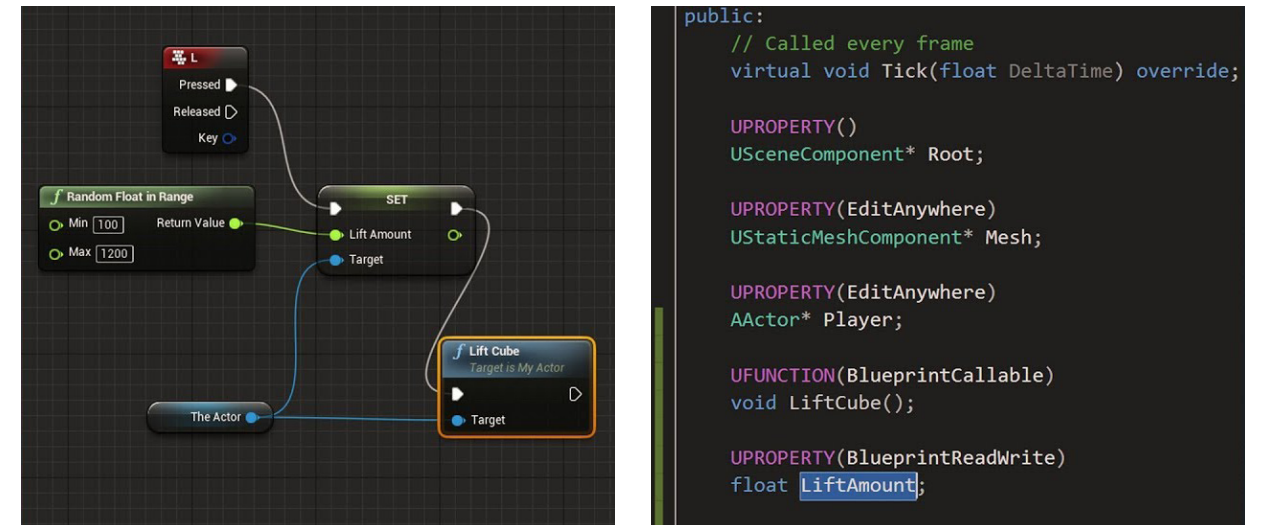


Figure 5.1.1 Blueprint scripting vs C++ scripting

A common approach to this within UE4 is to build prototypes in blueprint first and then move some or all of the functionalities to C++ once a “refractor point” is reached in which the base functionalities are proven and it becomes favorable to solidify the code for other people to use.^[2] While this translation may take significantly longer than prototyping with blueprints, the increased performance will allow the simulation of increasingly bigger crowds, which can expand the range of applications of this tool depending on the architectural project.

With our current hardware and setup from Part 3, we have a limitation of around 150 people before the system begins to slow down substantially. Therefore, by optimizing the logic, it may be possible to increase this limitation and increase the range of this application from simulating smaller spaces such as galleries and installations to larger spaces such as train stations and malls. In doing so, it also becomes possible to utilize more detailed agent Skeletal Mesh assets to portray a more realistic variance within the crowd.

Accuracy: Additional Simulation Model Considerations

As already mentioned in *Chapter 2.2: Establishing Model Methodology*, additional variables can be introduced beyond the concept of personal space to introduce increased complexity to these agents. For example, variables can be created for each agent that states his or her energy, hunger, interest, comfort, memory, emotion, and task list. This produces a pseudo-personality for each agent, where their task may be altered based on their current physical and mental state. An agent that does not have enough energy may be too tired and will try to seek seating, whereas an agent who has achieved his or her goal within the space may no longer have any interest and choose to seek an exit. These can then be mapped during the simulation to visualize areas where people may become stressed, hungry, tired, happy, etc. Beyond this, additional parameters can also be considered from higher complexity models (also mentioned in *Chapter 2.2*) such as *HiDAC*, where factors such as pushing, falling and panic behaviors can be introduced. This new simulation model can then be fine tuned further by doing a series of real world crowd studies in different locations, which can introduce other factors to influence crowd movements such as geography, culture, average height, population age, etc. The foundational framework we developed throughout this thesis then allows us to slowly build upon it by adding these features, allowing the simulation model to become increasingly capable and accurate over time.

² “Balancing Blueprint and C++,” Unreal Engine Documentation, accessed October 18, 2019, <https://docs.unrealengine.com/en-US/Resources/SampleGames/ARPG/BalancingBlueprintAndCPP/index.html>.

Utility: Additional features for the simulation tool

These improvements in the simulation model can also allow the implementation of additional features within the simulation tool—many of which were already briefly mentioned in *Chapter 3.6: Application Methodology*—which can include additional ways to visualize solar and thermal comfort, occupancy responses and reactions, as well as the ability to layer different types of agent trails depending on the persona, typology and groupset of the existing occupants. We can also recreate more accurate and additional scenarios to offer unique insight for architects during design. Beyond just evacuation we can simulate events such as holiday openings, conference lectures, and performances, as well as simulate more specific spaces that may require additional considerations such as checking into airports, hospitals, event gatherings, and so on. (**Fig. 5.1.2**) These features further contribute to the utility of this tool as an analytical and visualization tool for not only interactive dynamic spaces, but also occupied static spaces.



Figure 5.1.2 An airport is one example of a dynamic space that requires multiple layers of queuing, and many groupings of occupants

Chapter 5.2 | Photorealism

The next set of improvements to explore for this pipeline would be photorealistic rendering outputs. With hardware becoming increasingly powerful and software becoming increasingly sophisticated, more photorealistic rendering methods are becoming increasingly accessible. While the default visuals of UE4 provided adequate results, it is only the beginning of what this game engine is capable of. This step of improvements at the base would require investigating various ways of material creation, but to fully utilize this software, we would also need to investigate more advanced photorealism methods such as photogrammetry and real-time ray tracing. In doing so, we can further solidify this framework as an efficient way to both analyze and visualize the space within the same workflow.

Photogrammetry

Photogrammetry can be defined as “the art, science and technology of obtaining reliable information about physical objects and the environment through the process of recording, measuring and interpreting photographic images and patterns of electromagnetic radiant imagery and other phenomena.”^[1] Simply put, this allows us to create exceptionally detailed 3D models by taking multiple images of the subject from different angles. **(Fig. 5.2.1)** While this concept is not new, commercial photogrammetry software has only recently become easily accessible. By utilizing software such as *Agisoft Photoscan*, *Reality Capture*, and *Pix4d*, we can relatively easily create realistic assets that can in turn be used to create visualizations that are almost visually on par with traditional ray-traced methods at a fraction of the rendering time.^[2] **(Fig. 5.2.2 - 3)** These visualizations are approaching a level of photorealism that is hard to distinguish from real life, and as such, the implications of this within architectural visualization are undoubtedly significant and will be worth investigating.

Real-time ray tracing

As already mentioned in *Chapter 1.3: Advent and Progression of the Game Engine*, ray tracing is a rendering method where the paths of simulated “light rays” bouncing throughout the environment are traced back to the source of the camera.^[3] This technique is what allows traditional architectural rendering methods to outperform game engines in terms of photorealistic representation as they have the luxury of simulating the lighting based on real-world physics instead of faking it with textures. *Real-time* ray tracing then becomes somewhat self-explanatory in its benefits within architectural visualization.

This new technology allows us to output renderings at a much faster pace compared to traditional methods, and as such has the potential to further blur the line between virtual interactive visualizations and reality. While this technology is still relatively new, more and more software—such as UE4, albeit this feature is still in beta—are beginning to support it, with demos already showing the capabilities of this new technology.^[4] **(Fig. 5.2.4)** This, along with Nvidia’s recent introduction of ray tracing specific RTX graphics cards shows that the future of real-time ray tracing is right around the corner and will only become more powerful as technology improves.

⁴ Unreal Engine, “Reflections Real-Time Ray Tracing Demo | Project Spotlight | Unreal Engine,” YouTube, 1:04, accessed October 18, 2019, <https://www.youtube.com/watch?v=j3ue35ago3Y>.

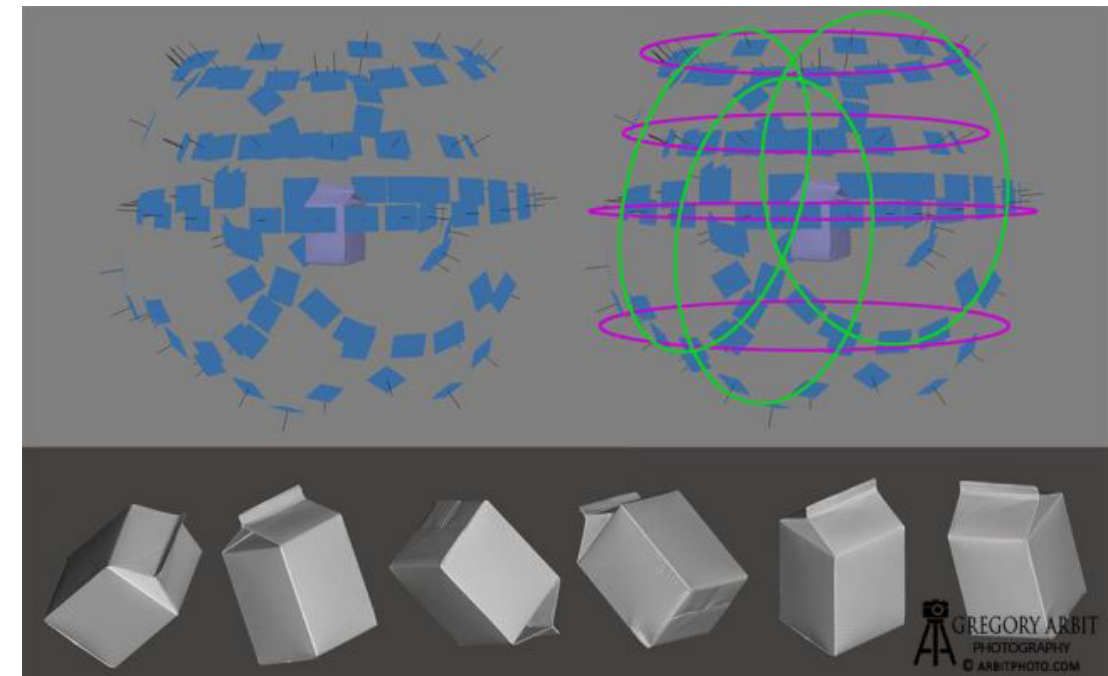


Figure 5.2.1 Photogrammetry recreates an object by capturing multiple images of the object in various angles

¹ James S. Bethel et al., *Manual of Photogrammetry* (Bethesda, MD: American Society for Photogrammetry and Remote Sensing, 2004), 2.

² Joseph Azzam, “Everything You Need to Know about Photogrammetry I Hope,” Gamasutra, January 10, 2017, accessed October 18, 2019, https://www.gamasutra.com/blogs/JosephAzzam/20170110/288899/Everything_You_Need_to_Know_about_Photogrammetry_I_hope.php.

³ Appel, “Some Techniques for Shading Machine Renderings of Solids.”



▶ **Figure 5.2.2** *Star Wars Battlefront Photogrammetry Mod*



▶ **Figure 5.2.3** *Rebirth photorealism within UE4 demo*



▶ **Figure 5.2.4** *Star Wars realtime Ray-tracing demo*

Chapter 5.3 | Virtual Reality

The real-time rendering and interaction capabilities of this UE4 pipeline naturally opens the potential of VR integration, which allows us to synchronize the virtual world to the physical.^[1] Through VR headsets, we can achieve levels of immersion that are unparalleled compared to static or even dynamic interactive visualizations on a traditional screen. There are multiple levels of immersion to this, ranging from simple rotationally-tracked headgear to full body positional tracking, many of which already have some sort of support within UE4.^[2]

This introduces possibilities of expanding both immersion and production aspects within architectural visualization. Simple positionally tracked controllers have already been used for applications that allow people to draw within a virtual 3D VR environment.^[3] (Fig. 5.3.1) Optical trackers such as Kinect^[4] and leap motion^[5] allows us to easily and cheaply track the human body and hands, which can be used for both immersive movement control as well as animation asset creation.^[6] (Fig. 5.3.2 - 3) More advanced rigs can also be used to simulate camera movements within the virtual environment that are comparable to holding a real camera in the physical world. (Fig. 5.3.4)

By exploring these multiple ways of facilitating physical virtual interactions, we will be able to realize a plethora of different options that can be used with this software. The benefits of this medium are further justified by Ronald Tang's recently defended M.Arch thesis *Step into the Void: A study of spatial perception in Virtual Reality*, in which he rationalizes the utility of VR as a medium within architectural visualization.^[7]

Eventually with enough hardware and software improvements, both the environment as well as the human agents^[8] (Fig. 5.3.5) will become indistinguishable from reality, allowing us to fully represent spaces, possibly making this the future of architectural visualization.

1 "Unreal Engine for AR, VR & MR," Unreal Engine, accessed October 18, 2019, <https://www.unrealengine.com/en-US/vr>.

2 "Virtual Reality Development," Unreal Engine Documentation, accessed October 18, 2019, <https://docs.unrealengine.com/en-US/Platforms/VR/index.html>.

3 "Painting from a New Perspective," Tilt Brush by Google, accessed October 18, 2019, <https://www.tiltbrush.com/>.

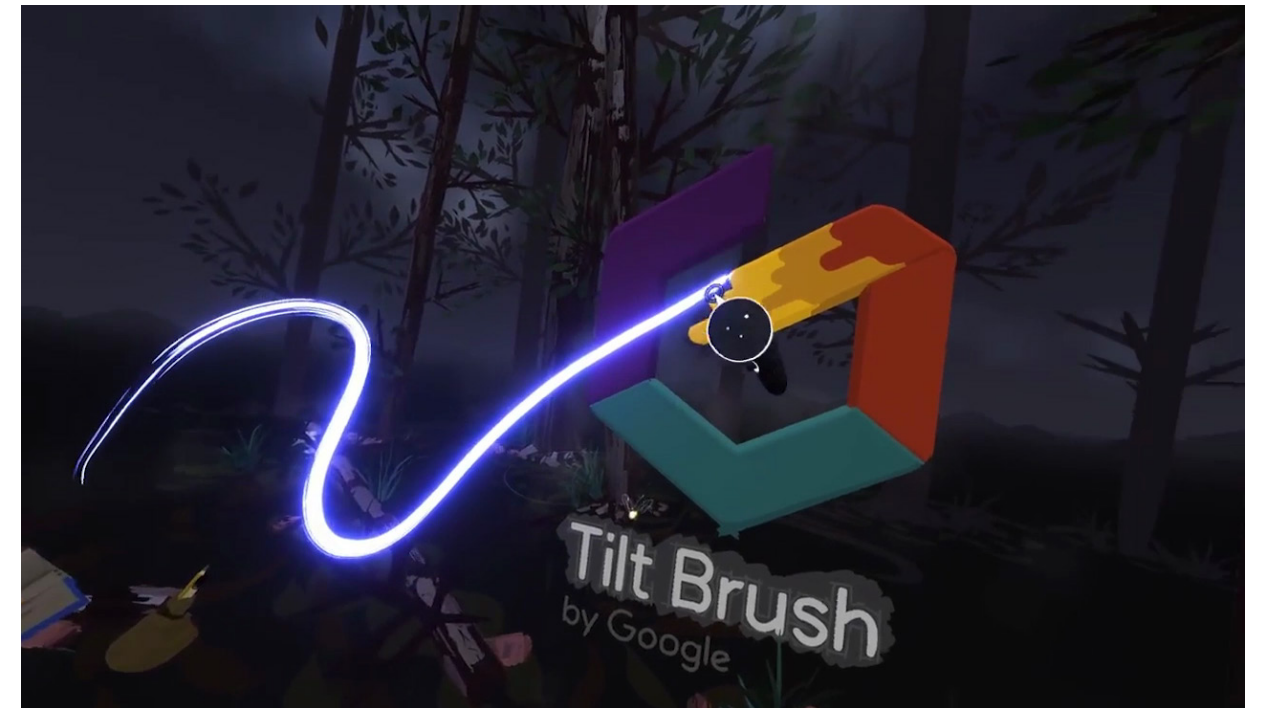
4 Kevin Carbotte, "You Can Use A Kinect For Full Body Tracking In SteamVR; Here's How," Tom's Hardware, September 16, 2017, <https://www.tomshardware.com/news/driver4vr-kinect-full-body-vr-tracking,35476.html>.

5 "Reach into the Future of Virtual and Augmented Reality," Leap Motion, accessed October 18, 2019, <https://www.leapmotion.com/>.

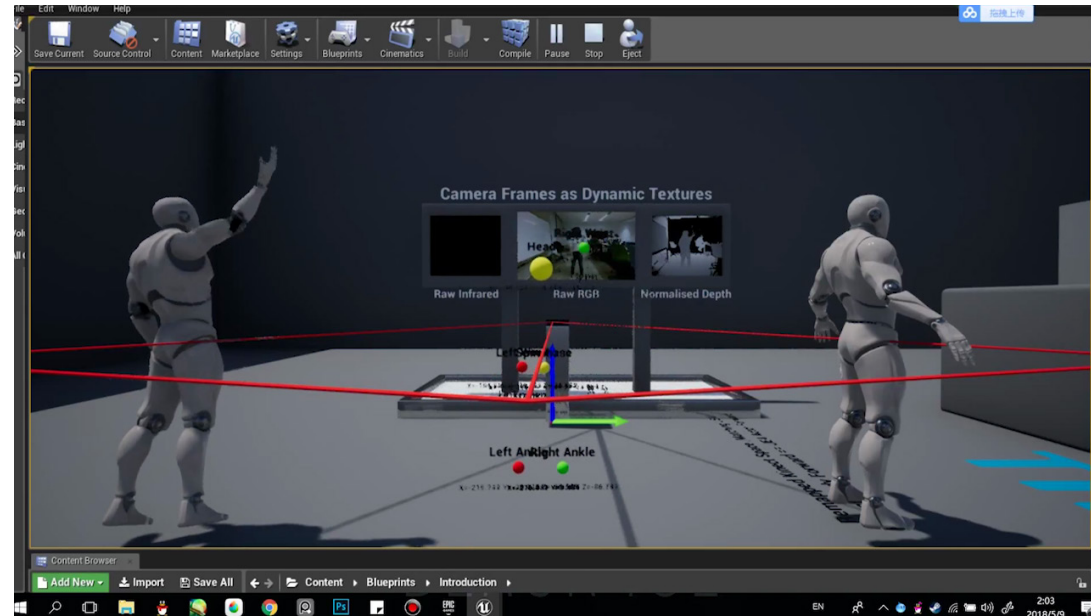
6 "Virtual Reality Motion Tracking Technology Has All the Moves," *Virtual Reality Society* (blog), May 5, 2017, <https://www.vrs.org.uk/virtual-reality-gear/motion-tracking/>.

7 Ronald Tang, "Step into the Void: A Study of Spatial Perception in Virtual Reality" (Master's thesis, University of Waterloo, Waterloo, 2019), <http://hdl.handle.net/10012/14468>.

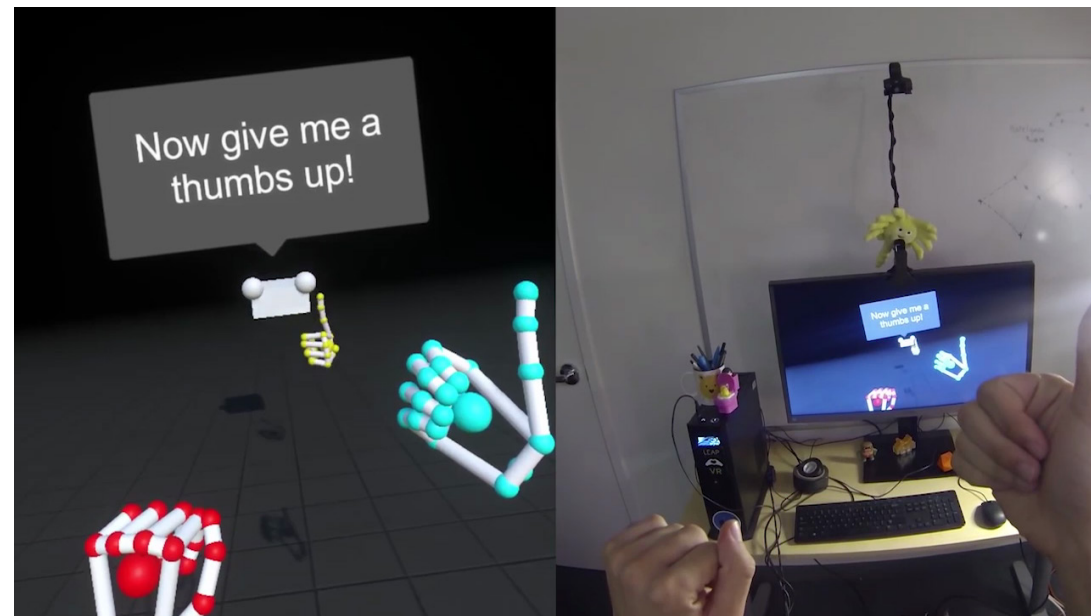
8 "Siren Real-Time Performance | Project Spotlight | Unreal Engine," YouTube, 0:41, accessed October 18, 2019, <https://www.youtube.com/watch?v=9owTAISsvwk>.



► Figure 5.3.1 Tiltbrush is one example of an VR painting application



▶ **Figure 5.3.2** *Kinect Body Tracking within UE4*



▶ **Figure 5.3.3** *Leap Motion Hand Tracking*



▶ **Figure 5.3.4** *Utilizing a VR camera rig in the making of the short film Rebirth in UE4*



▶ **Figure 5.3.5** *Project Siren demonstrates real-time face and body tracking alongside photorealistic human rendering within UE4*

Figure 5.3.2 - 5.3.5

Here we can see various levels of body tracking, which can be as precise as our fingers and faces and as inclusive as our whole body. Tracking can also be used on objects, which introduces the potential for syncing other tools, objects, and even architectural spaces. These new methods of tracking can allow us to both create new custom animation assets to further advance the movements of the crowds, as well as give us the ability to better interact with the simulation itself. This in turn allows us to fully simulate and test new interactive designs in virtual space before actually building them.

Chapter 5.4 | Workflow Refinements

The last of these aspects worth investigating in the future are workflow refinements, which investigates ways to improve the efficiency and potential uses of the workflow that we have established in chapter 3.6. One way to do so is to consider Python interoperability between applications. Python is a programming language that is supported amongst many 3D applications within the media and entertainment industries.^[1] UE4 is of course no exception, as such, it offers a Python API (Application Programming Interface) to help with scripting and automating within Unreal Editor.^[2] (Fig. 5.4.1) This allows the scripting of various management systems to automate workflows to optimize production pipelines in the future. The Datasmith workflow toolkit that was mentioned in chapter 3.5 is one example of such use, where its goal is to make “moving data into unreal engine as frictionless as possible.”^[3] (Fig. 5.4.2) Building upon this, it then becomes theoretically possible to automate the establishment of interactive objects and thresholds within UE4 depending on how the families are defined within Revit. (Fig. 5.4.3) This will be an important aspect to consider as the tool becomes more refined and better utilized for real world practical applications. In doing so, it becomes possible to further increase the efficiency of the potential visualization pipeline of this framework that this thesis has now established.

1 “Scripting the Editor Using Python,” Unreal Engine Documentation, accessed October 18, 2019, <https://docs.unrealengine.com/en-US/Engine/Editor/ScriptingAndAutomation/Python/index.html>.
 2 “Unreal Python API Introduction,” Unreal Engine Documentation, accessed October 18, 2019, <https://docs.unrealengine.com/en-US/PythonAPI/introduction.html>.
 3 Ken Pimentel, “Technology Sneak Peek: Python in Unreal Engine,” Unreal Engine, November 15, 2017, accessed November 20, 2019, <https://www.unrealengine.com/en-US/tech-blog/technology-sneak-peek-python-in-unreal-engine>.

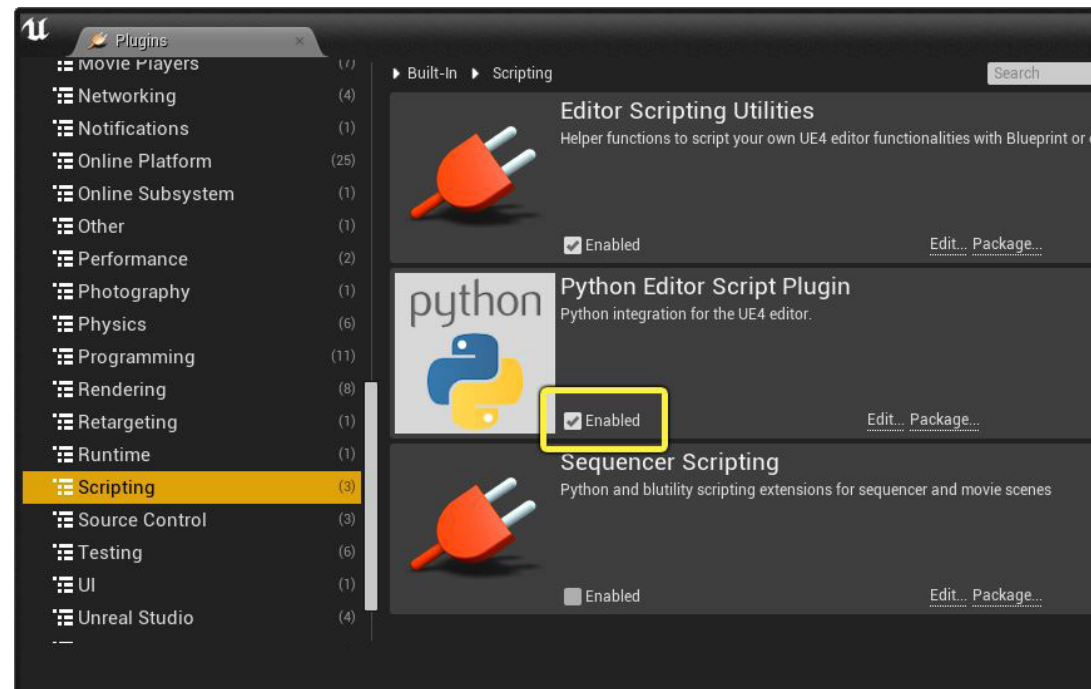


Figure 5.4.1 Python Editor Script Plugin within UE4

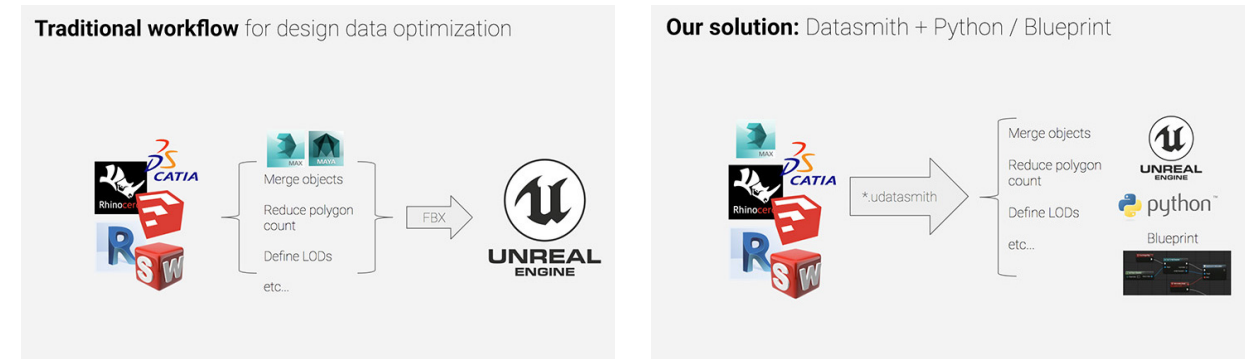


Figure 5.4.2 Datasmith is a collection of tools and plugins that automates various tasks from traditional workflows

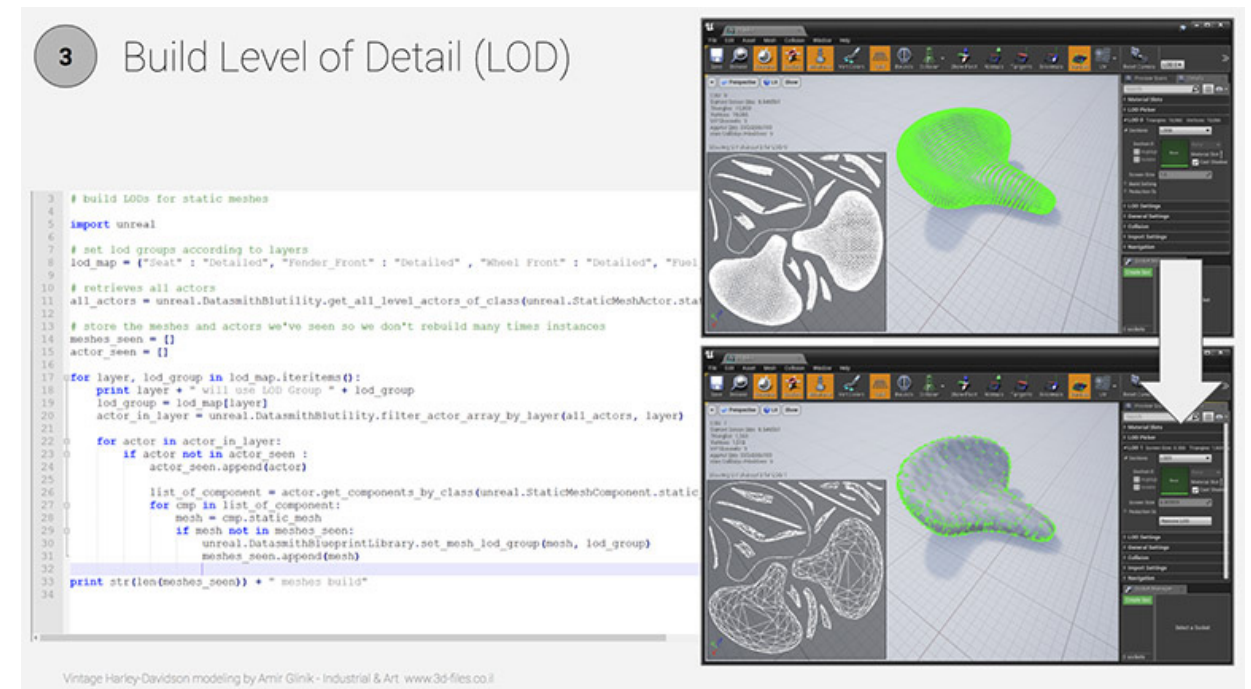


Figure 5.4.3 An example of a Python script used to automatically generate a LOD from a higher complexity mesh

As demonstrated in this thesis, this virtual software environment is fully capable of both simulating and rendering anything we can imagine in real time. As long as we can establish a purpose, and a methodology, this crowd simulation is just a foundation to what is possible to create in the future.

Bibliography

- ACADIA. "About ACADIA." Accessed October 16, 2019. <http://acadia.org/>.
- Addington, D. Michelle, and Daniel L. Schodek. *Smart Materials and New Technologies: For the Architecture and Design Professions*. London: Routledge, 2016.
- Akkerman, Abraham. "Urban Void and the Deconstruction of Neo-Platonic City-Form." *Ethics, Place & Environment* 12, no. 2 (2009): 205–18. <https://doi.org/10.1080/13668790902863416>.
- Alberti, Leon Battista. *On Painting*. Translated with an Introduction and Notes by John R Spencer. New Haven, 1966.
- Alexander, R. McNeill. "Energetics and Optimization of Human Walking and Running: The 2000 Raymond Pearl Memorial Lecture." *American Journal of Human Biology* 14, no. 5 (2002): 641–48. doi:10.1002/ajhb.10067.
- Appel, Arthur. "Some Techniques for Shading Machine Renderings of Solids." *Proceedings of the April 30--May 2, 1968, Spring Joint Computer Conference on - AFIPS 68 (Spring)*, 1968. <https://doi.org/10.1145/1468075.1468082>.
- Arduino. "What Is Arduino?" Accessed July 23, 2019. <https://www.arduino.cc/en/Guide/Introduction>.
- Arnheim, Rudolf. *The Dynamics of Architectural Form*. Berkeley: University of California Press, 2009.
- Autodesk Support & Learning. "Example: Using Populate." Accessed October 17, 2019. <https://knowledge.autodesk.com/support/3ds-max/learn-explore/caas/CloudHelp/cloudhelp/2017/ENU/3DSMax/files/GUID-BE-A89C57-3A7B-4AB5-AAF7-02494AA01CFA-htm.html>.
- Autodesk. "Autodesk Character Generator." Accessed October 18, 2019. <https://charactergenerator.autodesk.com/>.
- Azzam, Joseph. "Everything You Need to Know about Photogrammetry I Hope." Gamasutra, January 10, 2017. Accessed October 18, 2019. https://www.gamasutra.com/blogs/JosephAzzam/20170110/288899/Everything_You_Need_to_Know_about_Photogrammetry_I_hope.php.
- Baharlou, Ehsan. *Generative Agent-Based Architectural Design Computation: Behavioral Strategies for Integrating Material, Fabrication and Construction Characteristics in Design Processes*. Stuttgart: Institute for Computational Design and Construction, 2017.
- Banks, Jerry, John S. Carson, Barry L. Nelson, and David M. Nicol. *Discrete-Event System Simulation*. Upper Saddle River, NJ: Prentice Hall, 2001.
- Barnes, Michael, and Michael Dickson. *Widespan Roof Structures*. London: Telford, 2000.
- Basefount Company. "Miarmy." Accessed October 17, 2019. <http://www.basefount.com/miarmy.html>.
- Berg, Jur Van Den, Ming Lin, and Dinesh Manocha. "Reciprocal Velocity Obstacles for Real-Time Multi-Agent Navigation." *2008 IEEE International Conference on Robotics and Automation*, (May 2008): 1928–35. <https://doi.org/10.1109/robot.2008.4543489>.
- Bethel, James S., Edward M. Mikhail, J. Chris. McGlone, and Roy Mullen. *Manual of Photogrammetry*. Bethesda, MD: American Society for Photogrammetry and Remote Sensing, 2004.
- Carbotte, Kevin. "You Can Use A Kinect For Full Body Tracking In SteamVR; Here's How." Tom's Hardware, September 16, 2017. <https://www.tomshardware.com/news/driver4vr-kinect-full-body-vr-tracking,35476.html>.
- Cardinal, David. "How Nvidia's RTX Real-Time Ray Tracing Works." ExtremeTech, August 21, 2018. Accessed October 17, 2019. <https://www.extremetech.com/extreme/266600-nvidias-rtx-promises-real-time-ray-tracing>.

- Caulfield, Brian. "What's the Difference Between Ray Tracing, Rasterization?" The Official NVIDIA Blog, April 11, 2019. Accessed October 16, 2019. <https://blogs.nvidia.com/blog/2018/03/19/whats-difference-between-ray-tracing-rasterization/>.
- CGTrader. "UASSET 3D models - download UnrealEngine (UASSET) file format 3D assets." Accessed October 18, 2019. <https://www.cgtrader.com/3d-models/ext/uasset>.
- Chapanis, A. "Review of the Human Senses." *Psychological Bulletin* 51, no. 1 (01, 1954): 100-101. doi:<http://dx.doi.org.proxy.lib.uwaterloo.ca/10.1037/h0050962>.
- Charrieras, Damien, and Nevena Ivanova. "Emergence in Video Game Production: Video Game Engines as Technical Individuals." *Social Science Information* 55, no. 3 (September 2016): 337-56. <https://doi.org/10.1177/0539018416642056>.
- Cimbala, John M. "Descriptions of Fluid Flows." Penn State Engineering. Accessed August 3, 2019. https://www.mne.psu.edu/cimbala/Learning/Fluid/Introductory/descriptions_of_fluid_flows.htm.
- City of Toronto. "Nuit Blanche." Accessed October 10, 2019. <https://www.toronto.ca/explore-enjoy/festivals-events/nuitblanche/>.
- Codecademy. "What Is an IDE?" Accessed October 18, 2019. <https://www.codecademy.com/articles/what-is-an-ide>.
- Crang, Mike, and Stephen Graham. "Sentient Cities Ambient Intelligence and the Politics of Urban Space." *Information Communication and Society* 10, no. 6 (2007): 789-817. <https://doi.org/10.1080/13691180701750991>.
- Dadova, Jana. "Cellular Automata Approach for Crowd Simulation." Master's thesis, Comenius University, Bratislava, 2012. Accessed August 3, 2019. http://www.sccg.sk/~dadova/phd/rigorozka_dadova_final.pdf.
- Dagnelie, Gislin. *Visual Prosthetics: Physiology, Bioengineering and Rehabilitation*. New York: Springer, 2011.
- Dalin, Cheng. "The Great Wall of China," in *Borders and Border Politics in a Globalizing World*, edited by Paul Ganster and David E. Lorey, 11-20. Lanham, MD: SR Books, 2005.
- Delaney, William, and Erminia Vaccari. *Dynamic Models and Discrete Event Simulation*. New York: M. Dekker, 1989.
- Donelan, J. Maxwell, Rodger Kram, and Arthur D. Kuo. "Mechanical Work for Step-to-Step Transitions Is a Major Determinant of the Metabolic Cost of Human Walking." *The Journal of Experimental Biology* 205 (August 2002): 3717-27.
- Emshoff, James R., and Roger L. Sisson. *Design and Use of Computer Simulation Models*. New York: MacMillan Etc., 1976.
- Fortmeyer, Russell, and Charles D. Linn. "Abu Dhabi Investment Council Headquarters" in *Kinetic Architecture: Designs for Active Envelopes*, 176-183. Mulgrave: Images Publishing, 2014.
- Fox, Michael. *Interactive Architecture: Adaptive World*. New York: Princeton Architectural Press, 2016.
- Frazer, Andrew H. "Design Considerations for Retractable-roof Stadia." Master's thesis, 2005. Accessed July 23, 2019. <https://dspace.mit.edu/handle/1721.1/31119>.
- Glynn, Ruairi. "Fearful Symmetry." Accessed October 18, 2019. <http://www.ruairiglynn.co.uk/portfolio/fsymmetry/>.
- Golaem. "Digital Extras at Your Fingertips." Accessed October 17, 2019. <http://golaem.com/>.
- Gregory, Jason. *Game Engine Architecture*. Boca Raton; London; New York: CRC Press, 2019.
- Hall, Edward T. *The Hidden Dimension*. Garden City, NY: Doubleday, 1966.
- Helbing, Dirk, and Péter Molnár. "Social Force Model for Pedestrian Dynamics." *Physical Review E* 51, no. 5 (1995): 4282-286. doi:10.1103/PhysRevE.51.4282.
- Herwig, Andrian, and Philip Paar. "Game Engines: Tools for Landscape Visualization and Planning?" (November 2014): 1-10. Accessed October 16, 2019. https://www.researchgate.net/publication/268212905_Game_Engines_Tools_for_Landscape_Visualization_and_Planning.
- Hesham, Omar, and Gabriel Wainer. "Centroidal Particles for Interactive Crowd Simulation." *2016 Summer Computer Simulation Conference (SCSC 2016)*, (2016): <https://doi.org/10.22360/summersim.2016.scsc.012>.
- Irascible. Comment on "[Twitch] Fortnite Developers Discussion - Apr. 17, 2014." Unreal Engine Forums. Accessed October 18, 2019. <https://forums.unrealengine.com/unreal-engine/events/3192-twitch-fortnite-developers-discussion-apr-17-2014/page2>.
- ITU. "Internet of Things Global Standards Initiative." Accessed July 23, 2019. <https://www.itu.int/en/ITU-T/gsi/iot/Pages/default.aspx>.
- Johnson, Steven. *Emergence: The Connected Lives of Ants, Brains, Cities, and Software*. New York: Scribner, 2004.
- Khan Academy. "Early Applications of Linear Perspective." Accessed July 26, 2019. <https://www.khanacademy.org/humanities/renaissance-reformation/early-renaissance1/beginners-renaissance-florence/a/early-applications-of-linear-perspective>.
- Kitchin, Rob, and Matin Dodge. *Code/Space: Software and Everyday Life*. Software Studies. MIT Press, 2011. <https://books.google.ca/books?id=ZHez2BXgleQC>.
- Klüpfel, Hubert. "A Cellular automaton model for crowd movement and egress simulation." (July 2003): 1-136. Accessed December 26, 2019. https://www.researchgate.net/publication/29800160_A_Cellular_automaton_model_for_crowd_movement_and_egress_simulation.
- Kretzer, Manuel, and Ludger Hovestadt. *ALIVE: Advancements in Adaptive Architecture*. Basel: Birkhäuser, 2014.
- Lachambre, Sébastien, Sébastien Lagarde, and Cyril Jover. *Photogrammetry Workflow*, 2017. Accessed October 17, 2019. https://unity3d.com/files/solutions/photogrammetry/Unity-Photogrammetry-Workflow_2017-07_v2.pdf.
- Leap Motion. "Reach into the Future of Virtual and Augmented Reality." Accessed October 18, 2019. <https://www.leapmotion.com/>.
- Levine, Neil. "The Template of Photography in Nineteenth-Century Architectural Representation." *Journal of the Society of Architectural Historians* 71, no. 3 (January 2012): 306-31. <https://doi.org/10.1525/jsah.2012.71.3.306>.
- Levine, Robert V., and Ara Norenzayan. "The Pace of Life in 31 Countries." *Journal of Cross-Cultural Psychology* 30, no. 2 (1999): 178-205. doi:10.1177/0022022199030002003.
- Lewis, John, and William Loftus. *Java Software Solutions: Foundations of Program Design*. Boston: Addison-Wesley, 2012.
- Lewis, Michael, and Jeffrey Jacobson. "Game Engines in Scientific Research." *Communications of The ACM* 45, no. 1 (January 2002): 27-31. Accessed October 16, 2019. <https://www.cse.unr.edu/~sushil/class/gas/papers/GameAIP27-lewis.pdf>.
- Massive Software. "What Is Massive?" Accessed October 17, 2019. <http://www.massivesoftware.com/applications.html>.
- Miller, John H., and Scott E. Page. *Complex Adaptive Systems: An Introduction to Computational Models of Social Life*. Princeton, NJ: Princeton University Press, 2007.

- Minetti, A. E. "The three modes of terrestrial locomotion." In *Biomechanics and Biology of Movement*, edited by Benno Maurus Nigg, Brian R. MacIntosh, and Joachim Mester, 67–78. Human Kinetics, 2000.
- Mohler, Betty J., William B. Thompson, Sarah H. Creem-Regehr, Herbert L. Pick, and William H. Warren. "Visual Flow Influences Gait Transition Speed and Preferred Walking Speed." *Experimental Brain Research* 181, no. 2 (2007): 221–28. <https://doi.org/10.1007/s00221-007-0917-0>.
- Moore, Gordon E. "Cramming More Components onto Integrated Circuits." *Proceedings of the IEEE* 86, no. 1 (1998): 82–85. <https://doi.org/10.1109/jproc.1998.658762>.
- Moore, Gordon E. "Progress in Digital Integrated Electronics [Technical Literature, Copyright 1975 IEEE. Reprinted, with Permission. Technical Digest. International Electron Devices Meeting, IEEE, 1975, Pp. 11-13]." *IEEE Solid-State Circuits Society Newsletter* 11, no. 3 (2006): 36–37. <https://doi.org/10.1109/n-ssc.2006.4804410>.
- NASA. "Displacement, Velocity, Acceleration." Accessed August 04, 2019. <https://www.grc.nasa.gov/www/k-12/airplane/disvelac.html>.
- NASA. "Newton's Second Law." Accessed August 04, 2019. <https://www.grc.nasa.gov/www/k-12/airplane/newton2.html>.
- NASA. "Scalars and Vectors." Accessed August 04, 2019. <https://www.grc.nasa.gov/www/k-12/airplane/vectors.html>.
- National Institute of Building Sciences. "About the National BIM Standard-United States®." Accessed October 18, 2019. <https://www.nationalbimstandard.org/about>.
- Nilson, Björn, and Martin Söderberg. "Game Engine Architecture," (May 26, 2007): 1-18. Accessed October 16, 2019. <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.459.9537&rep=rep1&type=pdf>.
- Oasys. "Crowd Simulation Software: MassMotion." Accessed October 17, 2019. <https://www.oasys-software.com/products/pedestrian-simulation/massmotion/>.
- OED Online*. "Dynamic, Adj. and n." In Oxford University Press. Accessed October 18, 2019. <http://www.oed.com/view/Entry/58818>.
- Pelechano, Nuria, Jan M. Allbeck, & Norman I. Badler. "Controlling Individual Agents in High-Density Crowd Simulation." *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, (2007): 99-108. <http://repository.upenn.edu/hms/210>.
- Pimentel, Ken. "HOK on Architectural Visualization: Aggregate, Iterate, Communicate." Unreal Engine, March 13, 2019. Accessed October 17, 2019. <https://www.unrealengine.com/en-US/spotlights/hok-architectural-visualization-aggregate-iterate-communicate>.
- Pimentel, Ken. "Technology Sneak Peek: Python in Unreal Engine." Unreal Engine, November 15, 2017. Accessed November 20, 2019. <https://www.unrealengine.com/en-US/tech-blog/technology-sneak-peek-python-in-unreal-engine>.
- Pita, Pierre. "List of Full Body VR Tracking Solutions." Virtual Reality Times, February 21, 2017. Accessed October 17, 2019. <https://virtualrealitytimes.com/2017/02/21/list-of-full-body-vr-tracking-solutions/>.
- Raspberry Pi. "Raspberry Pi Blog - News, Announcements, and Ideas." Accessed July 23, 2019. <https://www.raspberry-pi.org/blog/>.
- Reynolds, Craig W. "Steering Behaviors For Autonomous Characters." Reynolds Engineering & Design. Accessed October 17, 2019. <http://www.red3d.com/cwr/steer/gdc99/>.
- Roman, Alex. "The Third & The Seventh." Uploaded November 24, 2009. Vimeo, 12:29. Accessed July 26, 2019. <https://vimeo.com/7809605>.
- Schindler, Christoph. "Information-Tool-Technology: Contemporary digital fabrication as part of a continuous development of process technology as illustrated with the example of timber construction." PhD diss., 2007. Accessed June 26, 2019. http://www.caad.arch.ethz.ch/wiki/uploads/Organisation/2007_Schindler_Information-tool-technology.pdf.
- Sennett, Richard. *The Craftsman*. London: Penguin, 2009.
- Shiffman, Daniel. "Chapter 1. Vectors." In *The Nature of Code*. United States: D. Shiffman, 2012. Accessed October 17, 2019. <https://natureofcode.com/book/chapter-1-vectors/>.
- Shiffman, Daniel. "Chapter 2. Forces." In *The Nature of Code*. United States: D. Shiffman, 2012. Accessed October 17, 2019. <https://natureofcode.com/book/chapter-2-forces/>.
- Shiffman, Daniel. "Chapter 6. Autonomous Agents." In *The Nature of Code*. United States: D. Shiffman, 2012. Accessed October 17, 2019. <https://natureofcode.com/book/chapter-6-autonomous-agents/>.
- Sokolowski, John A., and Catherine M. Banks. *Principles of Modeling and Simulation: A Multidisciplinary Approach*. Hoboken, NJ: John Wiley, 2009.
- Sonoff. "DIY A Temperature Controlled Smart Lock." Accessed July 23, 2019. <https://sonoff.ithead.cc/en/news/266-diy-a-temperature-controlled-smart-lock>.
- Statistics Canada. "Journey to Work, 2016 Census of Population." Uploaded November 29, 2017. Accessed October 17, 2019. <https://www150.statcan.gc.ca/n1/pub/11-627-m/11-627-m2017038-eng.htm>.
- Statistics Canada. "Labour Force Characteristics, Monthly, Seasonally Adjusted and Trend-Cycle, Last 5 Months." Accessed October 17, 2019. <https://www150.statcan.gc.ca/t1/tbl1/en/cv.action?pid=1410028701#timeframe>.
- Steiner, Hadas A. *Beyond Archigram: The Structure of Circulation*. New York: Routledge, 2009.
- Substance Academy. "The PBR Guide - Part 1." Accessed October 17, 2019. <https://academy.substance3d.com/courses/the-pbr-guide-part-1>.
- Sud, Avneesh, Russell Gayle, Erik Andersen, Stephen Guy, Ming Lin, and Dinesh Manocha. "Real-time Navigation of Independent Agents Using Adaptive Roadmaps." *ACM SIGGRAPH 2008*, (2008): doi:10.1145/1401132.1401207.
- Tang, Ronald. "Step into the Void: A Study of Spatial Perception in Virtual Reality." Master's thesis, University of Waterloo, Waterloo, 2019. <http://hdl.handle.net/10012/14468>.
- The British Library. "Invention of Photography." Accessed July 28, 2019. <https://www.bl.uk/learning/timeline/item106980.html>.
- Tilt Brush by Google. "Painting from a New Perspective." Accessed October 18, 2019. <https://www.tiltbrush.com/>.
- Tudor-Locke, Catrine, and David R Bassett. "How Many Steps/Day Are Enough?" *Sports Medicine* 34, no. 1 (2004): 1–8. <https://doi.org/10.2165/00007256-200434010-00001>.
- Unity. "Optimizing Graphics Performance." Accessed October 17, 2019. <https://docs.unity3d.com/Manual/OptimizingGraphicsPerformance.html>.
- Unreal Engine Documentation. "AI Perception." Accessed October 18, 2019. <https://docs.unrealengine.com/en-US/Engine/ArtificialIntelligence/AIPerception/index.html>.
- Unreal Engine Documentation. "Animation Sequences | Unreal Engine Documentation." Accessed October 18, 2019. <https://docs.unrealengine.com/en-US/Engine/Animation/Sequences/index.html>.

Unreal Engine Documentation. "Animation System Overview." Accessed October 18, 2019. <https://docs.unrealengine.com/en-US/Engine/Animation/Overview/index.html>.

Unreal Engine Documentation. "Assets and Packages." Accessed October 18, 2019. <https://docs.unrealengine.com/en-US/Engine/Basics/AssetsAndPackages/index.html>.

Unreal Engine Documentation. "Balancing Blueprint and C++." Accessed October 18, 2019. <https://docs.unrealengine.com/en-US/Resources/SampleGames/ARPG/BalancingBlueprintAndCPP/index.html>.

Unreal Engine Documentation. "Behavior Tree Overview." Accessed October 18, 2019. <https://docs.unrealengine.com/en-US/Engine/ArtificialIntelligence/BehaviorTrees/BehaviorTreesOverview/index.html>.

Unreal Engine Documentation. "Behavior Tree Quick Start Guide." Accessed October 18, 2019. <https://docs.unrealengine.com/en-US/Engine/ArtificialIntelligence/BehaviorTrees/BehaviorTreeQuickStart/index.html>.

Unreal Engine Documentation. "Blend Spaces." Accessed October 18, 2019. <https://docs.unrealengine.com/en-US/Engine/Animation/Blendspaces/index.html>.

Unreal Engine Documentation. "Blueprint Class." Accessed October 18, 2019. <https://docs.unrealengine.com/en-US/Engine/Blueprints/UserGuide/Types/ClassBlueprint/index.html>.

Unreal Engine Documentation. "Blueprint Editor Reference." Accessed October 18, 2019. <https://docs.unrealengine.com/en-US/Engine/Blueprints/Editor/index.html>.

Unreal Engine Documentation. "Blueprint Interface." Accessed October 18, 2019. <https://docs.unrealengine.com/en-US/Engine/Blueprints/UserGuide/Types/Interface/index.html>.

Unreal Engine Documentation. "Blueprint Variables." Accessed October 18, 2019. <https://docs.unrealengine.com/en-US/Engine/Blueprints/UserGuide/Variables/index.html>.

Unreal Engine Documentation. "Components." Accessed October 18, 2019. <https://docs.unrealengine.com/en-US/Programming/UnrealArchitecture/Actors/Components/index.html>.

Unreal Engine Documentation. "Environment Query System Quick Start." Accessed October 18, 2019. <https://docs.unrealengine.com/en-US/Engine/ArtificialIntelligence/EQS/EQSQuickStart/index.html>.

Unreal Engine Documentation. "EventGraph." Accessed October 18, 2019. <https://docs.unrealengine.com/en-US/Engine/Blueprints/UserGuide/EventGraph/index.html>.

Unreal Engine Documentation. "Events." Accessed October 18, 2019. <https://docs.unrealengine.com/en-US/Engine/Blueprints/UserGuide/Events/index.html>.

Unreal Engine Documentation. "Functions." Accessed October 18, 2019. <https://docs.unrealengine.com/en-US/Engine/Blueprints/UserGuide/Functions/index.html>.

Unreal Engine Documentation. "Installing the Datasmith Exporter Plugin for Revit." Accessed October 18, 2019. <https://docs.unrealengine.com/en-US/Studio/Datasmith/SoftwareInteropGuides/Revit/InstallingExporter-Plugin/index.html>.

Unreal Engine Documentation. "Navmesh Content Examples." Accessed October 18, 2019. <https://docs.unrealengine.com/en-US/Resources/ContentExamples/NavMesh/index.html>.

Unreal Engine Documentation. "Physically Based Materials." Accessed October 17, 2019. <https://docs.unrealengine.com/en-US/Engine/Rendering/Materials/PhysicallyBased/index.html>.

Unreal Engine Documentation. "Realistic Rendering." Accessed October 18, 2019. <https://docs.unrealengine.com/en-US/Resources/Showcases/RealisticRendering/index.html>.

Unreal Engine Documentation. "Scripting the Editor Using Python." Accessed October 18, 2019. <https://docs.unrealengine.com/en-US/Engine/Editor/ScriptingAndAutomation/Python/index.html>.

Unreal Engine Documentation. "Setting Up a Character." Accessed October 18, 2019. <https://docs.unrealengine.com/en-US/Engine/Animation/CharacterSetupOverview/index.html>.

Unreal Engine Documentation. "Skeletal Meshes." Accessed October 18, 2019. <https://docs.unrealengine.com/en-US/Engine/Content/Types/SkeletalMeshes/index.html>.

Unreal Engine Documentation. "Unreal Engine 4 Documentation." Accessed October 18, 2019. <https://docs.unrealengine.com/en-US/index.html>.

Unreal Engine Documentation. "Unreal Engine 4 Terminology." Accessed October 18, 2019. <https://docs.unrealengine.com/en-US/GettingStarted/Terminology/index.html>.

Unreal Engine Documentation. "Unreal Python API Introduction." Accessed October 18, 2019. <https://docs.unrealengine.com/en-US/PythonAPI/introduction.html>.

Unreal Engine Documentation. "Virtual Reality Development." Accessed October 18, 2019. <https://docs.unrealengine.com/en-US/Platforms/VR/index.html>.

Unreal Engine Marketplace. "Characters." Accessed October 18, 2019. <https://www.unrealengine.com/marketplace/en-US/content-cat/assets/characters>.

Unreal Engine. "Real Time Motion Capture in Unreal Engine." YouTube, 1:05:18. Accessed October 18, 2019. <https://youtu.be/jRyq5uPC5UY?t=1066>.

Unreal Engine. "Reflections Real-Time Ray Tracing Demo | Project Spotlight | Unreal Engine." YouTube, 1:04. Accessed October 18, 2019. <https://www.youtube.com/watch?v=J3ue35ago3Y>.

Unreal Engine. "Siren Real-Time Performance | Project Spotlight | Unreal Engine." YouTube, 0:41. Accessed October 18, 2019. <https://www.youtube.com/watch?v=9owTAISsvwk>.

Unreal Engine. "The Journey from Revit to Unreal Studio | Feature Highlight | Unreal Studio." YouTube, 1:08:36. Accessed October 18, 2019. <https://youtu.be/iuqTvd16UQ?t=435>.

Unreal Engine. "Unreal Engine for AR, VR & MR." Accessed October 18, 2019. <https://www.unrealengine.com/en-US/vr>.

Unreal Engine. "Unreal Studio." Accessed October 18, 2019. <http://www.unrealengine.com/studio>.

Veracode. "What Is an Integrated Development Environment (IDE)?" May 9, 2019. Accessed October 18, 2019. <https://www.veracode.com/security/integrated-development-environment>.

Vimeo. "Unreal Engine Tutorial in Videos on Vimeo." Accessed October 18, 2019. <https://vimeo.com/search/page:2?q=unreal+engine+tutorial>.

Virtual Reality Society. "Virtual Reality Motion Tracking Technology Has All the Moves," May 5, 2017. <https://www.vrs.org.uk/virtual-reality-gear/motion-tracking/>.

Wardman, Mark. "Public Transport Values of Time." *Institute of Transport Studies, University of Leeds, Working Paper 564* (2001): 1–56. Accessed October 17, 2019. http://eprints.whiterose.ac.uk/2062/1/ITS37_WP564_uploadable.pdf.

YouTube. "Unreal Engine." Accessed October 18, 2019. <https://www.youtube.com/channel/UCBobmJyZsJ6Ll7Ubf-hI4iwQ>.

Appendix A | Multimedia Figures

This appendix is a zip folder containing various animated video files to accompany the corresponding figures throughout this thesis.

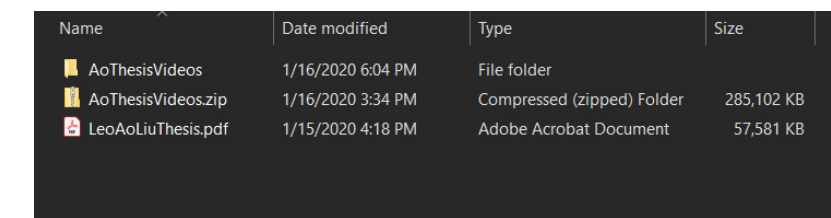
The file name of this folder is “AoThesisVideos.zip”, which contains various video files titled in the format of “Fig-0-0-0_FigureName.mp4”.

In order to open these video files within the PDF, download this zip folder from UWSpace and extract it so that “AoThesisVideos” is in the same folder as the PDF file. (**Refer to Fig. A.1**)

Once extracted to the proper location, you can open these videos by clicking on the corresponding figure with an “▶” symbol in front of the figure number. This will launch the video in your media player of choice.

Alternatively, these videos can be opened manually by referring to their filenames, which will match the figure numbers in the work.

If you accessed this thesis from a source other than the University of Waterloo, you may not have access to this file. You may access it by searching for this thesis on <https://uwspace.uwaterloo.ca>.



Name	Date modified	Type	Size
AoThesisVideos	1/16/2020 6:04 PM	File folder	
AoThesisVideos.zip	1/16/2020 3:34 PM	Compressed (zipped) Folder	285,102 KB
LeoAoLiuThesis.pdf	1/15/2020 4:18 PM	Adobe Acrobat Document	57,581 KB

Figure A.1 *The relative folder structure should look like this after extraction*

