Serious Game for Fire Evacuation

by

Dafu Deng

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Mechanical and Mechatronics Engineering

Waterloo, Ontario, Canada, 2019

Abstract

Fire safety for buildings has been of increasing concern due to the increase in occupant density in modern-day infrastructures. Efforts have been made by civil engineers to reduce loss in building fire accidents. For example, building codes have been refined to reduce the potential damage caused by fire by enforcing installation of fire detectors, alarm system, ventilation system, and sprinkler system. In addition, current building codes regulate the number of exits as well as the widths and heights of exits to allow an efficient evacuation process if the fire goes out of control. However the fire evacuation training aspect of fire safety is relatively immature.

The fire evacuation process is still trained by carrying out traditional fire drills. However, the value of traditional fire drills has been questioned. Traditional fire evacuation drills fail to present a realistic fire environment to the participants. Traditional fire drills fail to raise enough seriousness for the participants since in most cases participants are informed about the drills beforehand. The cost of conducting these traditional fire drills can also be very high.

Motivated by the problems faced by traditional fire drills, this research explores a new approach to more effectively and economically train people regarding the fire evacuation process. The new approach is to use a video game to train people for fire evacuation. The whole idea of using games for training and educational purposes falls under the concept of Serious Gaming, which has shown auspicious results in fields of military training, medical training, pilot training, and so on. In the virtual game environment, the fire environment can be simulated and rendered to the players. Doing so can allow the players to experience a more realistic fire environment and hence better prepare them for what to do in response to fire accidents. By setting a proper rewarding system, the game can motivate the players to treat the training more seriously. Also, since the training is carried out in the form of a game, it is more engaging and less costly.

Currently, the game has been developed to render smoke and control the movement of agents. In order to make the game environment more realistic, the smoke is simulated and rendered using fire dynamics, and the agent movement is controlled by appropriate pedestrian models. It is worth mentioning that pedestrian modeling is still a relatively immature field of science and this game also serves as a tool for collecting and analyzing data for pedestrian models.

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Professor Fue-sang Lien. Professor Lien has given not only useful technical support and guidance to me, but also warm encouragement during my time of stress, as well as valuable life lessons.

I would like to thank my co-supervisor, Dr. Steve Gwynne. This thesis involves a field that I had never touched on before. Fortunately, Dr. Gwynne has well-established expertise in the field, and he is always willing to teach me and give me advice. Without his help, the thesis can hardly proceed.

I would also like to thank Game Institute for supporting and funding this project.

My fellow research group members have been a great source of support. Their company has made my two-year program joyful and fulfilling.

Lastly, I want to thank my family for their love and trust.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Nowadays, due to the increasing population, urban infrastructures are now demanded to contain more occupants. Therefore, fire safety for buildings is now of greater importance more than ever. A system of fire codes has been designed by experts to prevent fires from happening in the first place and lower the damage of fire if it goes out of control. For example, in modern buildings, alarm system, sprinkler system, ventilation system, and fire hose are mandatory. Widths of exits are regulated to ensure a fire evacuation goes without congestion. Many other useful regulations [1] also significantly raise the fire safety level for buildings.

Although plenty of regulations have been set, people are still using traditional fire drills to prepare themselves for the evacuation process in fire accidents. There are three apparent problems with traditional fire drills. First of all, traditional fire drills cannot present a realistic fire environment. In a real fire accident, many hazards could emerge, which may change an evacuee's course of action. For example, fire and smoke can block exits, reduce visibility, and intoxicate the evacuee. Oftentimes, under this kind of extreme situation, evacuees' response might be affected by the perceived risk. In other words, since traditional fire drills are usually very idealized and cannot immerse participants into a realistic fire environment, the efficacy of preparing the participants for fire accident is very questionable.

Second of all, traditional fire drills are usually very costly. When the alarm sounds, people would have to stop their work at hand and evacuate the building as soon as possible.

The cost of pausing their work at hand can hardly be estimated but does exist. One can definitely conduct a fire drill where the occupants are informed about it beforehand, and some fire drills are indeed carried out this way so that the occupants would not be doing anything important at the drilling time. A fire drill conducted this way is called an announced fire drill. However, an announced fire drill has another obvious problem: because participants know the fire drill beforehand, they would not take the drill seriously as if there is a real fire happening, which adds up to the problem of being unrealistic. The drill will then fail the task of training participants to act correctly in real fire accidents. In addition, for some special places, like hospitals and nursing homes that usually contain impeded or paralyzed occupants, it is almost impossible to carry out fire drills while training them is still necessary.

Lastly, traditional fire drills are often deemed to be a waste of time, and thus people are not motivated to attend. If fire drills could somehow be made more engaging, people would be more willing to do the drills.

Another extended motivation for this project is related to pedestrian modeling. Briefly speaking, pedestrian modeling is the field of science that describes the behavior, mainly the movement, of pedestrians, which has been successfully used in traffic designs. Pedestrian models can be divided into two categories, large scale pedestrian models and small scale pedestrian models. The large scale pedestrian models are interested in describing the movement of pedestrians as a crowd where the movement of individuals is not important. The most well-known large scale pedestrian model is the flow-based pedestrian model [2, 3, 4, 5], which treats a crowd as fluid and laws that are similar to those of fluid dynamics are used to govern the motion of the crowd. Note that large scale pedestrian models are only suitable for describing the motion of a large and highly dense crowd.

When the scale or the density of crowd is small, large scale pedestrian models can only yield a very coarse and imprecise simulation of the crowd's motion. So when the scale or the density of the crowd is small, pedestrian models need to describe the motion of the crowd on the level of individuals, and this motivates the development of small scale pedestrian models. Small scale pedestrian models focus on the interaction between each pedestrian and their surroundings. In a computational simulation, a pedestrian is represented by a computer agent, so agent-based pedestrian models are oftentimes used to refer to small scale pedestrian models. The majority of agent-based models use fictitious Newtonian forces to govern the motion of individuals.

Note that both large scale pedestrian models and small scale pedestrian models are rather empirical. With the increasing concern of building fire safety, fire safety engineers proposed using pedestrian models to simulate the fire evacuation process in a building design and thus evaluate its fire safety level. The common practice is using third-party pedestrian modeling software to perform the simulation of the evacuation process in the examined building design and fetch important simulation data, for example, the rate of evacuation, to evaluate the building design's fire safety level. Figure 1.1 shows a simulation performed using *Pathfinder*, a pedestrian simulator developed by *ThunderHead Engineering*.

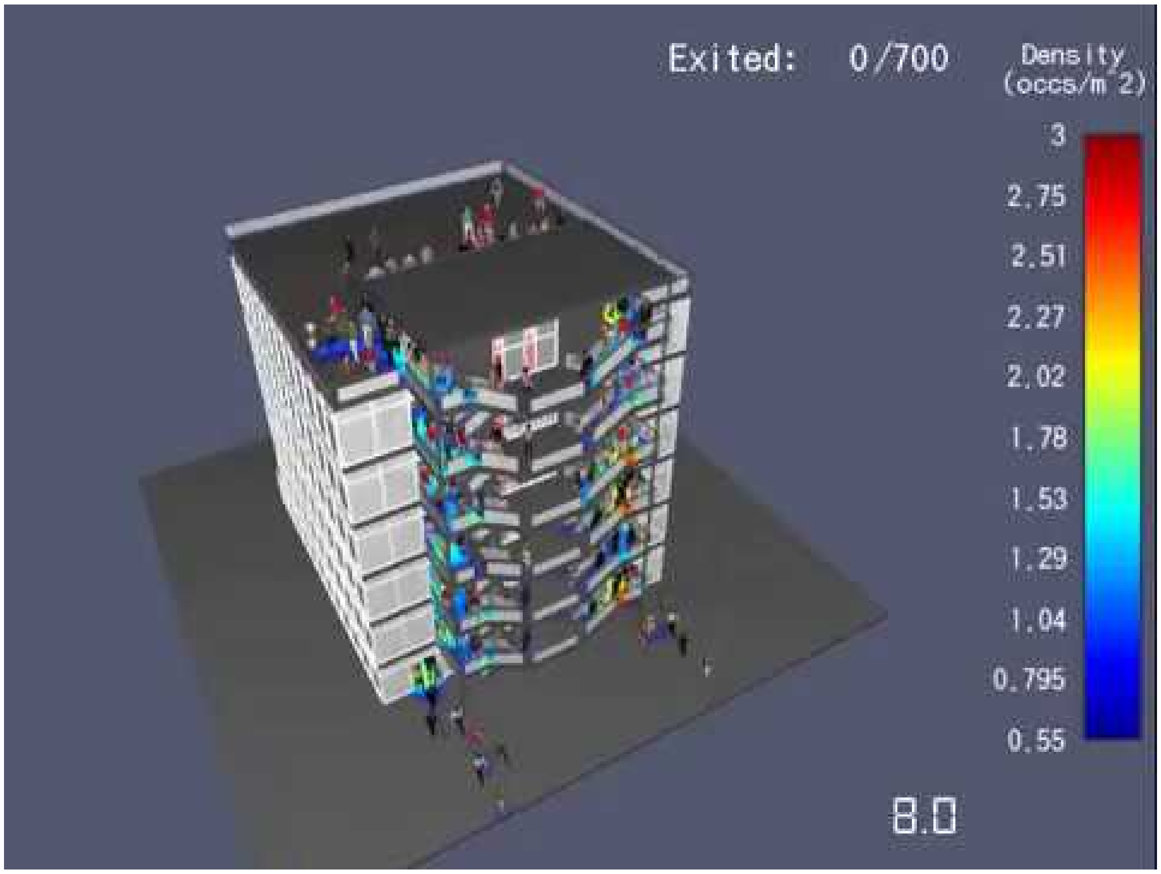


Figure 1.1: An evacuation process simulation by *Pathfinder*.

This method of evaluating the fire safety level of a building design relies on the accuracy of the pedestrian model. It needs to be mentioned that all pedestrian models are relatively empirical due to their fictitious nature. These models are usually constructed by reasoning and verifying on data. Therefore, pedestrian models rely heavily on data, either from experiments or daily observations. Unfortunately, the data is far from sufficient to develop reliable pedestrian models for fire evacuation. This is because experiments for fire evacuation are nearly impossible to conduct in the real world and CCTV footage is either limited or fails to capture representative behaviors. On the whole, using pedestrian modeling in assessing fire safety requires an accurate pedestrian model, but the problem is that a pedestrian model that is able to accurately describe pedestrians' behavior has not been successfully developed due to the lack of data. Ways need to be found to collect such data without putting anybody in danger.

1.2 Objectives

As discussed above, traditional fire drills suffer from three main drawbacks: being unrealistic, costly, and not engaging. In addition, regarding pedestrian modeling, no reliable pedestrian model has been developed to describe human behavior in a fire accident situation due to the lack of data. This project aims to explore a new approach to overcome the three problems faced by traditional fire drills, as well as the hurdle in developing fire evacuation pedestrian models. The new approach is using a virtual game not only to train people to act correctly in a fire accident but also collect useful data for later use. This game is considered a serious game.

Different from ordinary video games, serious games aim primarily to train and educate people. Serious games have shown tremendous success in clinical training, pilot training, classroom education, and so on [6, 7, 8]. The prosperity of the serious game industry inspired the idea of using a video game to train and educate people about fire evacuation.

Using a game (a virtual environment) to train people can directly solve the three main problems faced by traditional fire drills. First of all, traditional fire drills cannot present a realistic fire accident environment. It is believed that people are better trained if the drilling scenario matches the real scenario. In the virtual environment, visual effects can be added to imitate real hazards, like smoke and fire, which can alter the course of action of an evacuee. By adding these visual effects, a more realistic experience can be created for trainees without risking their safety. However, to make the virtual environment realistic, these visual effects need to be convincing. To be more specific, the game designed in this project simulates the spread of smoke and movement of computer-controlled agents based on fire dynamics and pedestrian models respectively.

The second problem of traditional fire drills lies in the cost of conducting them. However, if the fire evacuation drills are conducted in a virtual game, the cost is negligible. People can simply play the game any time they want according to their own schedule. Using a video game for drilling also saves people from physically moving, which makes it possible to train disabled people in places like hospitals and nursing homes. The third problem of traditional fire evacuation drills is that they are non-engaging. In contrast, the whole point of all serious games is making the training and teaching process more fun and engaging and thus yielding a more desirable training and teaching result. This project, *Serious Game for Fire Evacuation*, also aims to more effectively and engagingly train people to act properly in a fire evacuation process.

Last but not least, *Serious Game for Fire Evacuation* can also be used as a tool for collecting data. In the game, the course of action of a player can be recorded; this data can later be used to create or vindicate theories that describe human behavior in the fire evacuation situation. On the premise that the game is made sufficiently realistic, data collected could be of value - especially given the challenges highlighted above. Again, a virtual game environment provides a platform to study human behavior under extreme conditions without putting people in danger.

The training aspect and the research aspect of *Serious Game for Fire Evacuation* are expected to go hand in hand. In the early version of the game, a coarse pedestrian model would be used to control the movement of agents, which means players experience a relatively unrealistic agent movement in the game. Actions of the players would be recorded during the game, which would be used as data to develop a more reliable pedestrian model. Ideally, an agent's course of action in the game can mimic that of a real human. Once a more realistic pedestrian model has been developed, it can be re-implemented into the game and render a more realistic agent flow to players. Then the data collected from players playing in a more realistic virtual environment become more reliable for developing a more accurate pedestrian model. Figure 1.2 demonstrates the circle.

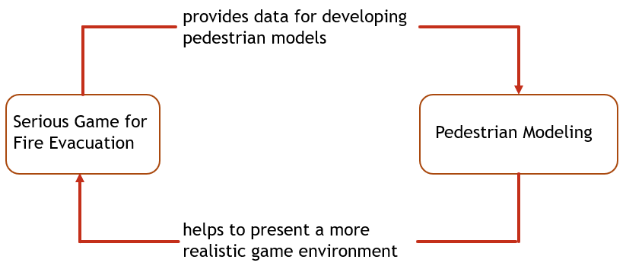

provides data for developing pedestrian models
Serious Game for Fire Evacuation
Pedestrian Modeling
helps to present a more realistic game environment

Figure 1.2: *Serious Game for Fire Evacuation* can be used as a platform to collect data for developing and refining the pedestrian model. The more refined pedestrian model can be implemented in the game to render a more realistic game environment. Again, data collected from a more realistic game environment are more reliable and hence can be used to develop a more accurate pedestrian model.

1.3 Contributions

The contributions of this thesis are summarized as follows:

- This thesis explored the feasibility of using a virtual game to replace tradition fire drills for fire evacuation training. Although *Serious Game for Fire Evacuation* has not been tested on the effectiveness in fire evacuation training, it provides a methodology to implement a virtual environment of such. The game itself can also be used as a foundation where more features can be added to fulfill the purpose of the fire evacuation training.

- This thesis managed to integrate fire simulation into the virtual environment. For most video games, fire and smoke are only rendered as visual effects, which are visually convincing but do not obey physics laws. In contrast, the propagation of the smoke in *Serious Game for Fire Evacuation* is based on strict fire dynamics. The same methodology can be used to integrate other physics simulations into the virtual environment.

- *Serious Game for Fire Evacuation*successfully integrated a simple pedestrian model to control the movement of agents in it. More complicated agent-based pedestrian models can be implemented in the game using the same methodology.

- *Serious Game for Fire Evacuation* provides an interactive platform for testing and developing pedestrian models. In the past, the result of a pedestrian simulation would be determined once the initial condition of the simulation was fixed. In order to study different crowd movements under different conditions, one had to repeatedly change simulation parameters and run simulations, which was very cumbersome. If pedestrian models were implemented to control the movement of agents in real time, like in *Serious Game for Fire Evacuation*, then by changing the condition in real time, we could observe different crowd flows. For example, in this game, agents movement would be constantly updated responding to the position of the player.

1.4 Thesis Organization

Chapter 2 covers the background of serious game and pedestrian modeling. This chapter first introduces the definition of serious game and its applications. After that, some popular pedestrian models are introduced and discussed in terms of their strengths and weaknesses and their suitability for the game.

Chapter 3 covers the general design of *Serious Game for Evacuation*. This chapter discusses the three interfaces of the game, the design interface, the play interface, and the feedback interface.

Chapter 4 covers the implementation of fire simulation. The chapter discusses how the fire scenario is set up, translated into simulation input file, simulated, and visualized in the game.

Chapter 5 covers the implementation of a pedestrian model. The chapter discusses how pedestrian modeling is accomplished in the game engine and some modifications to make the pedestrian model computationally efficient for the game.

Chapter 6 gives the conclusion of the thesis. This chapter also discusses some future work of the project.

Chapter 2

Background and Literature Review

2.1 Serious Game

Serious games are games designed for educational and training purposes. Serious games have yielded profound success in fields like classroom teaching, health care, emergency management, pilot training, and so on. Figure 2.1a shows a game for surgery training, and Figure 2.1b shows a game for pilot training.

(a) A serious game for surgery training.

(b) A serious game for pilot training.

Figure 2.1: Serious game examples. (a) is *Serious Games Développés En Santé*, (b) is *Flight Simulator FlyWings 2014*.

Most of the serious games consist of three main components [9], the learning component, the gaming component, and the simulation component. The concept of serious gaming is well illustrated in Figure 2.2.

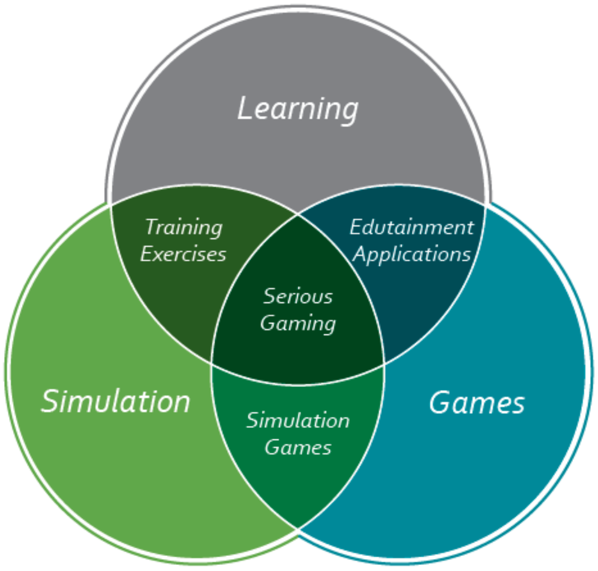


Figure 2.2: Three components of a serious game (source: https://i2.wp.com/www.santacruztechbeat.com/wp-content/uploads/2019/01/serious_game_classification.png).

The learning aspect is what distinguishes a serious game from most other conventional video games. The main objective of a serious game is to educate and train players instead of merely providing entertainment. But then, of course, in order to make the education and training process more engaging, it is done in the form of a video game. Finally, to make a serious game convincing enough for the educational and training purposes, simulation needs to be performed to render a realistic virtual environment.

In the hope of providing a more effective way of training people about fire evacuation using the serious game approach, *Serious Game for Fire Evacuation* was developed. Compared to traditional fire drills, some of the advantages of using the serious game approach for training are: more engaging, capable of rendering a more realistic fire scenario, and cheaper to carry out. Like most other serious games, *Serious Game for Fire Evacuation* also consists of the three main components. The game needs to be built up realistically so that it can effectively prepare players for real fire accidents. For this reason, more work had been done on the simulation part of this game comparing to most other serious games. The simulation of the game carefully incorporates fire dynamics and pedestrian modeling.

2.2 Pedestrian modeling

In order to simulate a fire evacuation process, it is necessary to make the non-human-controlled agents move in a naturalistic manner; *i.e.* to provide a credible social background within which the user-controlled agent moves. Pedestrian modeling is the field of science that describes the movement and behavioral patterns of an individual pedestrian or a crowd of pedestrians. Pedestrian modeling is still a relatively young field of science. The state-of-the-art models all have their own limitations under certain circumstances. Especially when fire-to-human physical and psychological impacts are taken into account, these pedestrian models would give a very imprecise simulation and prediction. Nevertheless, some of these models suffice to convey a sense of reality in the game environment.

Pedestrian models can be separated into two categories, the fluid-based models and the agent-based models. Fluid-based pedestrian models [2, 3, 4, 5] treat pedestrians all together as fluid and use fluid dynamics to describe the motion of the crowd. Because of the computational efficiency of fluid-based models, they are usually used to describe large scale pedestrian crowd movement and traffic flow in urban designs, where individual movement is of no significance. When the scale or the crowd density is small, fluid-based models can only provide a very coarse simulation. On the other hand, agent-based models focus on individual agents thus give a more accurate result for small scale simulation. However, describing each agent individually is computationally difficult or even impossible for some large scale simulations.

There is always a trade-off between precision and computational power. The game developed in this research project aims to immerse players into a fire evacuation scenario, where the density of evacuees is relatively small. Therefore, the game uses an agent-based model, based on the work of Helbing *et al* [10, 11, 12, 13], to control the agent movement. In the following subsections, the theoretical background of this model is introduced. Some modifications of the model is also introduced to make the model better suit the game.

2.2.1 Social-force Model

Social-force model is a popular agent-based model used by pedestrian simulators like *FDS+Evac*. The model introduces a force to keep a reasonable distance between an agent and another agent or an obstacle. This force is interpreted as the "social force", which is meant to describe the phenomenon that pedestrians tend to keep a comfortable distance from each other and from obstacles.

Circle Representation of an Agent

There are two popular ways to represent an agent in a 2-dimensional space, the one-circle representation and three-circle representation. As shown in Figure 2.3, the three-circle representation is more complicated, and its advantages are discussed in [14, 15, 16, 17, 18]. R_t represents the radius of the agent's torso and R_s represents the radius of the shoulder.

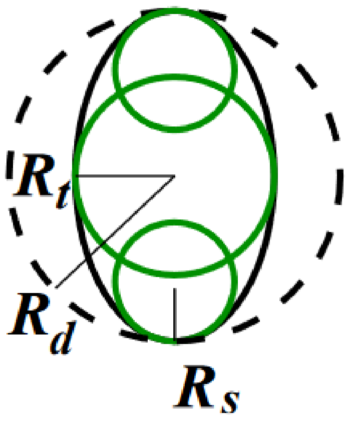


Figure 2.3: Three circle representation.

However, the three circle representation is only mentioned for the sake of completeness of information. The present game only demands a moderate degree of simulation accuracy, so the game uses the one-circle representation, the outer circle R_d, for simplicity and computational efficiency.

Motion Governing Equation

The social-force model is a Newtonian physics model. For agent i, its movement is governed by the equation of motion:

$$m_i \frac{d^2\mathbf{x}_i}{dt^2} = \mathbf{f}_i(t) + \xi_i(t), \tag{2.1}$$

where m_i is the mass of agent i, $\mathbf{x}_i$ is the position, $\mathbf{f}_i$ is the force exerted on it, and ξ_i is a small random fluctuation force.

The force exerted on agent i can be expressed as follow:

$$\mathbf{f}_i(t) = \frac{m_i(v_i^0(t)\mathbf{e}_i^0(t) - \mathbf{v}_i(t))}{\tau_i} + \sum_{j(\neq i)} [\mathbf{f}_{ij}^{soc}(t) + \mathbf{f}_{ij}^{att}(t)] + \sum_b \mathbf{f}_{ib}^{soc}(t) + \sum_k \mathbf{f}_{ik}^{att}(t), \tag{2.2}$$

where the first term on the right hand side of the equation is interpreted as the motivation force that drives agent i to move, $\mathbf{f}_{ij}^{soc}$ is the "social force" between agent i and agent j, $\mathbf{f}_{ij}^{att}$ is the "attraction force" between agent i and agent j, $\mathbf{f}_{ib}^{soc}$ is the "social force" between agent i and boundary b, and the lastly $\mathbf{f}_{ik}^{att}$ can be used to describe some abstract psychological attraction, for example, parents tend to go to their children. Each term will be further explained below.

Motivation Force

The motivation force in equation 2.2, denote it as $\mathbf{f}_m$, is expressed as:

$$\mathbf{f}_m = \frac{m_i(v_i^0(t)\mathbf{e}_i^0(t) - \mathbf{v}_i(t))}{\tau_i}.$$

This is the force that drives agent i towards the desired destination, usually an exit in a fire evacuation scenario. m_i is the mass of pedestrian i. v_i^0 is the desired walking speed and $\mathbf{e}_i^0$ is the desired normalized walking direction, for example, towards an exit. $\mathbf{v}_i$ is the actual walking velocity of pedestrian i. τ_i is interpreted as the "relaxation time constant", which determines how fast the pedestrian can adapt from its actual velocity $\mathbf{v}_i$ to its desired velocity $v_i^0\mathbf{e}_i^0$.

The desired walking speed v_i^0 is set to be a constant based on the statistical data, the "relaxation time constant" τ_i can be tweaked to give the desired simulation. The desired walking direction $\mathbf{e}_i^0$ needs to be determined by an appropriate route selecting algorithm. Different route selecting algorithms will be discussed in subsection 2.2.4.

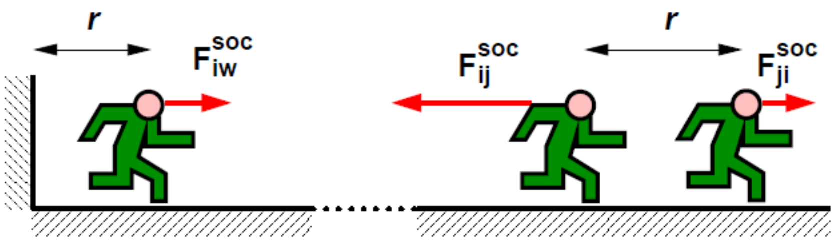


Figure 2.4: Social force concept (source: [19]).

Social Force

Pedestrians tend to keep a distance from each other, and this phenomenon can be described as a repulsive force, or called as a social force $\mathbf{f}_{ij}^{soc}$ (see Figure 2.4). The social force is expressed as:

$$\mathbf{f}_{ij}^{soc}(t) = A_i e^{(r_{ij}-d_{ij})/B_i}\mathbf{n}_{ij}[\lambda_i + (1-\lambda_i)\frac{1+\cos(\varphi_{ij})}{2}]. \tag{2.3}$$

In equation 5.3, A_i denotes the interaction strength and B_i denotes the range of the repulsive interactions. Both A_i and B_i can be adjusted to reflect individual preferences. For example, in cultures where pedestrians tend to keep a larger distance, both A_i and B_i need to be set larger. For a simulation using one-circle pedestrian representation, $r_{ij} = r_i + r_j$ is the sum of radii of the two interacting pedestrians, $d_{ij} = ||\mathbf{x}_i - \mathbf{x}_j||$ is the distance between their centers of circle. In other words, $d_{ij} - r_{ij}$ represents the closest distance between the bodies of the two interacting agents, and

$$\mathbf{n}_{ij}(t) = (n_{ij}^1(t), n_{ij}^2(t)) = \frac{\mathbf{x}_i(t) - \mathbf{x}_j(t)}{d_{ij}(t)} \tag{2.4}$$

is the normalized vector pointing from pedestrian j to pedestrian i. $\lambda_i < 1$ is the anisotropic character of pedestrian interaction. In other words, with λ_i, the model can describe how big the impact from behind is comparing to the impact from ahead. The angle φ_{ij} denotes the angle between the direction of motion $\mathbf{e}_i(t) = \frac{\mathbf{v}_i(t)}{||\mathbf{v}_i(t)||}$ and the opposite direction of force $-\mathbf{n}_{ij}(t)$, *i.e.* $\cos(\varphi_{ij})(t) = -\mathbf{n}_{ij}(t)\mathbf{e}_i(t)$.

Following the same idea of the social repulsive force, the social attraction force can be used to describe pedestrians' tendencies to go towards certain other pedestrians or objects. For example, a mother pedestrian i tends to go to her child pedestrian j, which is described by $\mathbf{f}_{ij}^{att}$, and pedestrian i tends to go a window k for fresh air, which is described by $\mathbf{f}_{ik}^{att}$. The attraction social force is of the same expression as the repulsive social force in equation 5.3, but with an interaction strength constant $A_i < 0$.

2.2.2 Modified Social-force Model - Contact Force

When pedestrians come into direct contact, as pointed out by Langston [14], the original social-force model gives unsatisfactory simulation results. To better describe pedestrians behavior in direct contact, Langston's work add in few more force terms in the original social-force model. The force exerted on pedestrian i is expressed as

$$\mathbf{f}_i(t) = \frac{m_i(v_i^0(t)\mathbf{e}_i^0(t) - \mathbf{v}_i(t))}{\tau_i} + \sum_{j(\neq i)}[\mathbf{f}_{ij}^{soc}(t) + \mathbf{f}_{ij}^{c} + \mathbf{f}_{ij}^{att}(t)] + \sum_b [\mathbf{f}_{ib}^{soc}(t) + \mathbf{f}_{ib}^{c}] + \sum_k \mathbf{f}_{ik}^{att}(t), \tag{2.5}$$

where $\mathbf{f}_{ij}^c$ is the contact force between pedestrian i and pedestrian j and $\mathbf{f}_{ib}^c$ is the contact force between pedestrian i and obstacle b. The concept of the contact force between two three-circle represented pedestrians is shown in Figure 2.5.

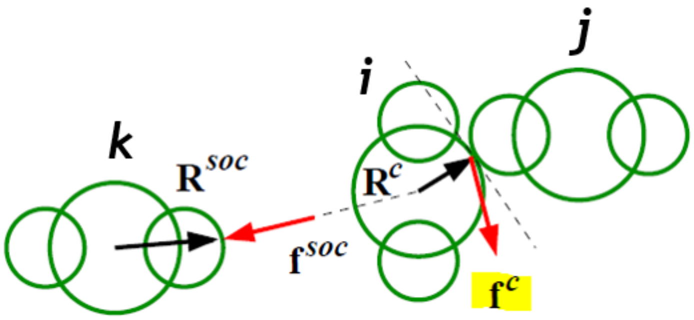


Figure 2.5: f^c is the contact force agent j exerts on agent i, and f^{soc} is the social forece agent i exerts on agent k (source: [19]).

The contact force $\mathbf{f}_{ij}^c$ is expressed as

$$\mathbf{f}_{ij}^c = [k_{ij}(d_{ij} - r_{ij}) + c_d \Delta v_{ij}^n]\mathbf{n}_{ij} + \kappa_{ij}(d_{ij} - r_{ij})\Delta v_{ij}^t \mathbf{t}_{ij}, \tag{2.6}$$

where Δv_{ij}^t is the difference of the tangential velocities of the circles in contact, and vector $\mathbf{t}_{ij}$ is the unit tangential vector of the contacting circles. This force only applies when the circles are in contact, *i.e.*, $r_{ij} - d_{ij} \geq 0$. The radial elastic force strength is given by the constant k_{ij}, and the strength of the fractional force by the force constant κ_{ij}. As described in [14], $c_d \Delta v_{ij}^n \cdot \mathbf{n}_{ij}$ represents a physical damping force, and Δv_{ij}^n is the difference of agent i's and agent j's velocities, $\mathbf{n}_{ij}$ is the normalized vector pointing from agent j to agent i, and c_d is the damping parameter. Note that the parameter c_d reflects the fact that the collision of two humans is not elastic. The boundary-agent (obstacle-agent) contact force, f_{ib}^c, is treated similarly but with different force constants.

2.2.3 Rotation Governing Equation

The motion governing equation for the modified social-force model can be extended to also describe the rotation of each pedestrian, as explained in [19]. The rotation equation for pedestrian i is

$$I_i^z \frac{d^2\varphi_i(t)}{dt^2} = \mathbf{M}_i^z(t) + \eta_i^z(t), \tag{2.7}$$

where I_i^z is the moment of inertia for pedestrian i, $\varphi_i(t)$ is the angle at time t, $M_i^z(t)$ is the torque exerted on it, and $\eta_i^z(t)$ is a small random fluctuation torque.

In correspondence to motivation force, social force and contact force, motivation torque, social torque, and contact torque are defined in a similar manner. Therefore, the total torque exerted on pedestrian i can be expressed as

$$\mathbf{M}_i^z = \mathbf{M}_i^\tau + \mathbf{M}_i^{soc} + \mathbf{M}_i^c, \tag{2.8}$$

where $\mathbf{M}_i^\tau$ is the motivation torque, $\mathbf{M}_i^{soc}$ is the social torque, and $\mathbf{M}_i^c$ is the contact torque.

Motivation Torque

Very similar to the motivation force, the motivation torque makes pedestrian i rotate to the direction of the desired walking velocity $\mathbf{v}_i^0$ as mentioned previously. The motivation torque is expressed as:

$$\mathbf{M}_i^\tau = \frac{I_i^z}{\tau_i^z}((\varphi_i(t) - \varphi_i^0)\omega_i^0 - \omega_i(t)), \tag{2.9}$$

where ω_i^0 is the maximum target angular velocity of a turning pedestrian i, $\omega_i(t)$ is the current angular velocity, φ_i^0 is the target angle (in the direction of $\mathbf{v}_i^0$), and τ_i^z is the angular time relaxation parameter.

Social Torque

The social force exerted on pedestrian i will make it rotate, and this is described by the social torque. The social torque exerted on pedestrian i from pedestrian j is $\mathbf{R}_i^{soc} \times \mathbf{f}_{ij}^{soc}$. Therefore, the total social torque exerted on pedestrian i is expressed as:

$$\mathbf{M}_i^{soc} = \sum_{j \neq i} (\mathbf{R}_i^{soc} \times \mathbf{f}_{ij}^{soc}), \tag{2.10}$$

where $\mathbf{R}_i^{soc}$ is the vector pointing from the center of pedestrian i to the fictitious contact point of the social force (see Figure 2.5), and $\mathbf{f}_{ij}^{soc}$ is the social force exerted by pedestrian j.

Contact Torque

The contact force exerted on pedestrian i will also make it rotate, and this is described by the contact torque. The contact torque exerted on pedestrian i from pedestrian j is $\mathbf{R}_i^c \times \mathbf{f}_{ij}^c$. Therefore, the total contact torque exerted on pedestrian i is expressed as:

$$\mathbf{M}_i^c = \sum_{j \neq i} (\mathbf{R}_i^c \times \mathbf{f}_{ij}^c), \tag{2.11}$$

where $\mathbf{R}_i^c$ is the vector pointing from the center of pedestrian i to the contact point of the contact force (see Figure 2.5), and $\mathbf{f}_{ij}^c$ is the contact force exerted by pedestrian j.

2.2.4 Route Selection Algorithms

One of the premises of using social-force model or modified social-force model is that one has to know the desired walking velocity $\mathbf{v}_i^0$. In other words, these two models or in fact any pedestrian models in general, need an appropriate algorithm to determine the route a pedestrian needs to take to evacuate. In this subsection, three algorithms will be introduced, *FDS+Evac Fluid Algorithm, Game Theory Algorithm, Closest Exit Algorithm.*

FDS+Evac Algorithm

Fire Dynamics Simulator (FDS) [20] is a very powerful simulator for simulating fire and smoke, which is developed by *National Institute of Standards and Technology (NIST).* It mainly simulates fire-driven smoke by using a Computational Fluid Dynamics (CFD) model. It numerically solves the Navier-Stokes equations using the large eddy simulation to describe the evolution of smoke as well as the heat transfer.

Later on, *FDS+Evac* was developed and maintained by *VTT*, Finland. This is a simulator for both fire and pedestrians. The fire simulation part is done by *FDS*, and then the pedestrian simulation is incorporated. Note that the pedestrian simulation takes into account the fire-to-human effect. Ideally, a two-way coupling simulation is expected. It means that fire can influence pedestrian's behavior; for example, fire can block the pedestrian's evacuation route, slow down the pedestrian's moving speed, or make the pedestrian unconscious. On the other hand, pedestrian's behavior can also influence the fire condition; for example, pedestrians may simply put out the fire or open up some windows or doors to create more ventilation. However, since it is almost computationally impossible to do the

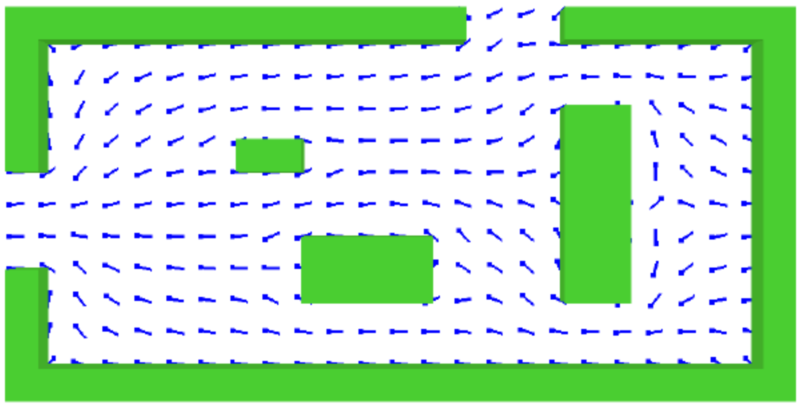


Figure 2.6: A fictitious 2D pedestrian flow field used to guide agents to the exit doors. In this case, only the left exit has an "outflow" boundary condition (source: [19]).

two-way-coupling simulation, *FDS+Evac* only performs the one-way-coupling simulation, where fire influences pedestrian's behavior, but not the other way around.

For the pedestrian simulation part, *FDS+Evac* mainly follows the work of Helbing's group [10, 11, 12, 13]. Recall the motion governing equation:

$$\mathbf{f}_i(t) = \frac{m_i(v_i^0(t)\mathbf{e}_i^0(t) - \mathbf{v}_i(t))}{\tau_i} + \sum_{j(\neq i)} [\mathbf{f}_{ij}^{soc}(t) + \mathbf{f}_{ij}^{att}(t)] + \sum_b \mathbf{f}_{ib}^{soc}(t) + \sum_k \mathbf{f}_{ik}^{att}(t), \tag{2.12}$$

and let $\mathbf{v}_i^0(t) = v_i^0(t)e_i^0(t)$ be the desired walking velocity.

FDS+Evac uses a fluid dynamics approach to get the desired velocity $\mathbf{v}_i^0(t)$. It creates a desired velocity field $\mathbf{v}_i^0$ in the evacuation field as shown in Figure 2.6. This vector field is obtained as an approximate solution of a two-dimensional imcompressible potential flow problem with given boundary conditions, where all walls are inert (*i.e.* slip) and the chosen exit door acts as a fan, which extracts fluid out of the domain. A more thorough description of this algorithm can be found in [19].

Game Theory Algorithm

The game theory approach of describing a pedestrian's decision on the evacuation route is discussed in detail in the literature [21, 22]. In a computational simulation, an agent represents a pedestrian. In short, agents use game theory to choose their routes of evacuation.

The goal of each agent is to choose a route that will grant him the shortest evacuation time while taking other agents' decisions into account. Generally, an agent decides whether or not to go to exit e_k based on these factors: the distance to e_k, the capacity of e_k (maximum crowd flow rate), the congestion (queue) at e_k, and the fire condition at e_k. For the sake of demonstration, a simplified game theory algorithm will be introduced where an agent's decision is based on only the distance to each exit and the potential queuing time at each exit.

In game theory, an agent's decision on which exit to go is called a strategy. Suppose that there are N agents and K exits in an evacuation scenario. The set of all the agents is $\{A_i : i \in \mathcal{N} = \{1, 2, ..., N\}\}$. The set of all available exits is $\{e_k : k \in \mathcal{K} = \{1, 2, ..., K\}\}$. The strategy of an agent is just the exit that it chooses, which is $s_i \in S_i = \{e_1, ..., e_k\}$. The profile of all agents' strategies is denoted as

$$s := (s_1, ..., s_N) \in S_1 \times \cdots \times S_N := S, \tag{2.13}$$

and the notation $s_{-i} := (s_1, ..., s_{i-1}, s_{i+1}, ...s_N) \in S_{-i}$ is used for the strategies of all other agents but agent i.

Denote the location of agent i by $\mathbf{r}_i, i \in \mathcal{N}$, and the location of exit e_k by $\mathbf{b}_k, k \in \mathcal{K}$. The distance between agent i and exit e_k is hence calculated as

$$d(e_k; \mathbf{r}_i) = ||\mathbf{r}_i - \mathbf{b}_k||. \tag{2.14}$$

Let $\mathbf{r} := (\mathbf{r}_1, ..., \mathbf{r}_N)$ be the collection of all the locations of the agents. The payoff function of agent i is the estimated evacuation time, $T_i(s_i, s_{-i}; \mathbf{r})$, which it needs to minimize. Note that the estimated evacuation time is the sum of the time to cover the distance to an exit and the time waiting in queue at the exit. When agent i chooses strategy $s_i = e_k$, T_i is evaluated as

$$T_i(e_k, s_{-i}; \mathbf{r}) = \beta_k \lambda_i(e_k, s_{-i}; \mathbf{r}) + \tau_i(e_k; \mathbf{r}_i), s_{-i} \in S_{-i}, \tag{2.15}$$

where β_k is the exit capacity scalar of exit e_k, $\lambda_i(e_k, s_{-i}; \mathbf{r})$ is the number of other agents that are also heading towards exit e_k and are closer to the exit than agent i, and $\tau_i(e_k; \mathbf{r}_i)$ is the time for agent i to go to exit e_k. The function λ_i is defined by $\lambda_i(e_k, s_{-i}; \mathbf{r}) := |\Lambda_i(e_k, s_{-i}; \mathbf{r})|$ where

$$\Lambda_i(e_k, s_{-i}; \mathbf{r}) := \{j \in \mathcal{N} | s_j = e_k, d(e_k; \mathbf{r}_j) < d(e_k; \mathbf{r}_i)\}, \tag{2.16}$$

and $|\cdot|$ denotes the number of elements in a set. Note that $\Lambda_i(e_k, s_{-i}; \mathbf{r})$ is essentially the set containing all the other agents besides agent i that choose to go to exit e_k ($s_j = e_k$)

and are closer to it than agent i ($d(e_k;\mathbf{r}_j) < d(e_k;\mathbf{r}_i)$). Larger $\lambda_i(e_k, s_{-i};\mathbf{r})$ means longer queuing time at exit e_k. The estimated moving time to an exit is evaluated as

$$\tau_i(e_k;\mathbf{r}_i) := \frac{1}{v_i^0} d(e_k;\mathbf{r}_i), \tag{2.17}$$

where v_i^0 is the moving speed of agent i.

Given the decisions of all other s_{-i}, the "best response" of agent i is expressed as

$$BR_i(s_{-i};\mathbf{r}) := \underset{s_i^{'}\in S_i}{\operatorname{argmin}}\{T_i(s_i^{'}, s_{-i};\mathbf{r})\}, \tag{2.18}$$

which is interpreted as choosing strategy that gives the shortest evacuation time.

With all the necessary tools introduced above, the exit selection of the agents can now be updated periodically. The strategy of agent i on period t is the best response to the other agents' strategies on the previous period:

$$s_i^{(t)} = BR_i(s_{-i}^{(t-1)};\mathbf{r}). \tag{2.19}$$

Note that a Nash equilibrium of the "evacuation game" $\overline{s_i}$ satisfies $\overline{s}_i = BR_i(\overline{s}_{-i};\mathbf{r})$ for all i.

The game theory algorithm is a state-of-the-art algorithm in pedestrian modeling. One version of the game theory algorithm is implemented in *FDS+Evac*. This algorithm works fine for a non-interactive simulation. However, this algorithm is not suitable for an interactive real-time game, where agents have to update their course of actions in accordance with the player's movement. The algorithm is too computationally expensive to be implemented in a real-time game. For this reason, a simpler algorithm, *Closest Exit Algorithm* [23] is used in the game.

Closest Exit Algorithm

Closest Exit Algorithm can be seen as a simplified version of *Game Theory Algorithm*, but under certain conditions, it is still able to yield satisfactory simulation results.

Recall that the objective of *Game Theory Algorithm* is to minimize the evacuation time

$$T_i(e_k, s_{-i};\mathbf{r}) = \beta_k\lambda_i(e_k, s_{-i};\mathbf{r}) + \tau_i(e_k;\mathbf{r}_i), s_{-i} \in S_{-i},$$

where the first term $\beta_k \lambda_i(e_k, s_{-i}; \mathbf{r})$ on the right hand side of the equation represents the estimated queuing time at exit e_k and the second term $\tau_i(e_k; \mathbf{r}_i)$ represents the walking time towards exit e_k. The estimated time towards exit e_k is estimated by

$$\tau_i(e_k; \mathbf{r}_i) := \frac{1}{v_i^0} d(e_k; \mathbf{r}_i),$$

$$d(e_k; \mathbf{r}_i) = ||\mathbf{r}_i - \mathbf{b}_k||.$$

In *Closest Distance Algorithm*, the consideration of the queuing time is omitted, and thus each agent's objective is now simply to go the nearest exit. The estimated evacuation time is now simplified as

$$T_i(e_k, s_{-i}; \mathbf{r}) = \tau_i(e_k; \mathbf{r}_i), s_{-i} \in S_{-i}. \tag{2.20}$$

However, a more complicated algorithm is used to determine the distance to an exit. Previously, the distance to a specific exit, say exit e_k, is calculated as $d(e_k; \mathbf{r}_i) = ||\mathbf{r}_i - \mathbf{b}_k||$, which represents the straight-line distance between agent i and exit e_k. However, oftentimes agents have to walk around obstacles to go to an exit. Therefore, a more accurate estimation of the distance is calculated by considering a polygon instead of a straight line as shown in Figure 2.7.

The distance is now calculated as

$$d(e_k; \mathbf{r}_i) = \sum_s d_s, \tag{2.21}$$

where d_s is the length of the line segment s of the distance polygon. In the example shown in Figure 2.7, the actual distance from agent i's position to exit e_k is calculated as $d(e_k; \mathbf{r}_i) = d_1 + d_2 + d_3 + d_4 + d_5$.

The main advantage of *Closest Distance Algorithm* lies in its simplicity. This algorithm is quite straightforward to implement and is also very computationally efficient. For these reasons, this algorithm is used as a starting-point in *Serious Game for Fire Evacuation*.

2.2.5 Fire and Human Interaction

FDS+Evac is used as the template of this project, and the fire and human interaction interpretation in [19] is what we are going to discuss here.

Fire condition can influence pedestrian's behavior in two main aspects. Firstly, smoke reduces visibility and thus impedes the movement of pedestrians. Secondly, smoke can intoxicate pedestrians and cause deaths.

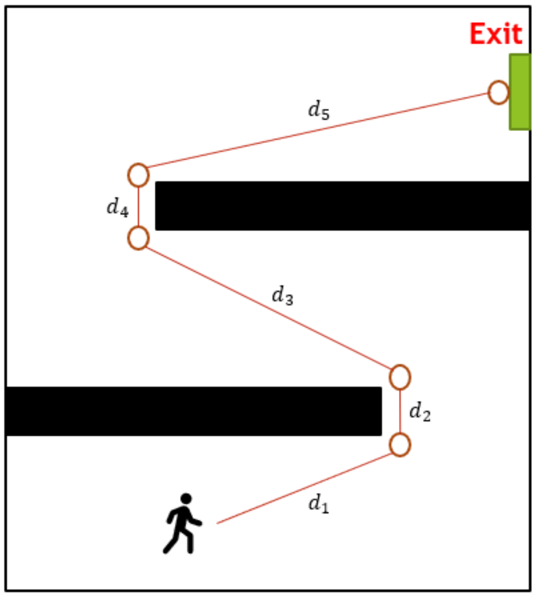


Figure 2.7: A polygon is used to calculate the distance an agent needs to cover to go to an exit.

Impeding Effect

According to literature [24], pedestrian walking speed decreases with increasing smoke concentration by the formula $v(K_s) = \alpha + \beta K_s$, where $\alpha = 0.706$ m s^{-1}, $\beta = -0.057$ m s^{-1} are both empirical coefficient values, while K_s is the extinction coefficient ($[K_s] = \mathrm{m}^{-1}$). For agent i, the walking speed is determined by the formula

$$v_i^0(K_s) = \mathrm{Max}\{v_{i,\min}^0, v_i^0(1 + \frac{\beta}{\alpha}K_s)\}, \tag{2.22}$$

where the minimum walking speed of agent i is $v_{i,\min}^0 = 0.1 \cdot v_i^0$, *i.e.* smoke does not completely stop an agent.

Toxic Effect

The toxic effect of gaseous fire products is incoperated by using Purser's Fractional Effective Dose (FED) concept [25]. The FED value is calculated as

$$\mathrm{FED_{tot}} = (\mathrm{FED_{CO}} + \mathrm{FED_{CN}} + \mathrm{FED_{NO_x}} + \mathrm{FED_{irr}}) \times \mathrm{HV_{CO_2}} + \mathrm{FED_{O_2}}. \tag{2.23}$$

Table 2.1: Coefficients used for the computation of irritant effects of gases.

	HCI	HBr	HF	SO_2	NO_2	C_3H_4O	CH_2O
F_{FLD} (ppm $\times$ min)	111400	11400	87000	12000	1900	4500	22500
F_{FLD} (ppm)	900	900	900	120	350	20	20

The fraction of an incapacitating dose of CO is calculated as

$$\mathrm{FED_{CO}} = \int_0^t 2.764 \times 10^{-5} (C_{\mathrm{CO}}(t))^{1.036} dt, \tag{2.24}$$

where t is time in minutes and C_{CO} is the CO concentration (ppm). The fraction of an incapacitating dose of CN is calculated as

$$\mathrm{FED_{CN}} = \int_0^t (\frac{\exp(\frac{C_{\mathrm{CN}}(t)}{43})}{220} - 0.0045) dt, \tag{2.25}$$

where t is time in minutes and C_{CN} is the concentration (ppm) of HCN corrected for the protective effect of NO_2. C_{CN} is calculated as

$$C_{\mathrm{CN}} = C_{\mathrm{HCN}} - C_{\mathrm{NO_2}}. \tag{2.26}$$

The fraction of an incapacitating dose of NO_x is calculated as

$$\mathrm{FED}_{\mathrm{NO}_x} = \int_0^t \frac{C_{\mathrm{NO_x}}(t)}{1500} dt, \tag{2.27}$$

where t is time in minutes and $C_{\mathrm{NO_x}}$ is the sum of NO and NO_2 concentration (ppm).

The Fraction Lethal Dose (FLD) of irritants is calculated as

$$\mathrm{FLD_{irr}} = \int_0^t (\frac{C_{\mathrm{HCI}}(t)}{F_{\mathrm{FLD,HCI}}} + \frac{C_{\mathrm{HBr}}(t)}{F_{\mathrm{FLD,HBr}}} + \frac{C_{\mathrm{HF}}(t)}{F_{\mathrm{FLD,HF}}} + \frac{C_{\mathrm{SO_2}}(t)}{F_{\mathrm{FLD,SO_2}}} + \frac{C_{\mathrm{NO_2}}(t)}{F_{\mathrm{FLD,NO_2}}} + \frac{C_{\mathrm{C_3H_4O}}(t)}{F_{\mathrm{FLD,C_3H_4O}}} + \frac{C_{\mathrm{CH_2O}}(t)}{F_{\mathrm{FLD,CH_2O}}}) dt, \tag{2.28}$$

where t is time in minutes, the numerators are the instantaneous concentrations (ppm) of each irritant and the denominators are the exposure doses of respective irritants predicted to be lethal to half the population. The lethal exposure doses [15] are given in Table 2.1 .

The fraction of an incapacitating dose of low O_2 hypoxia is calculated as

$$\mathrm{FED}_{\mathrm{O}_2} = \int_0^t \frac{dt}{60 \exp[8.13 - 0.54(20.9 - C_{\mathrm{O}_2}(t))]}, \tag{2.29}$$

where t is time in minutes and C_{O_2} is the O_2 concentration. The hyperventilation factor induced by carbon dioxide is calculated as

$$\mathrm{HV}_{\mathrm{CO}_2} = \frac{\exp(0.1903 C_{\mathrm{CO}_2}(t) + 2.0004)}{7.1}, \tag{2.30}$$

where t is time in minutes and C_{CO_2} is the CO_2 concentration.

An agent in the simulation is considered to be incapacitated if the FED value exceeds unity. An incapacitated agent no longer moves or be affected by social forces from other agents.

Chapter 3

Game Design

Serious Game for Fire Evacuation is a video game that trains players about the fire evacuation process. First of all, an appropriate game engine should be selected as the platform to implement the simulation. In the game engine, the fire simulation is developed using a third-party fire simulator, and then the simulation result is transferred into the game to be visually rendered. Pedestrian models are written as scripts to control the background agents in the game. In addition, for the implementation of this game, a rather simple fire-to-human impact model is used as a starting point. Again, the pedestrian model used in this game is a rather simplified one because the main objective of this project is to explore the feasibility of fire evacuation training through the manner of a video game. More complicated models can later be implemented on this platform following the same methodology.

The general implementation flow chart is demonstrated in Figure 3.1. As shown in the flow chart, the pedestrian model is directly scripted in the game. At the beginning of the project, using a third-party pedestrian simulator, *e.g. Pathfinder*, was considered. However, if the simulation is performed before the game starts, the movement of each agent would be fixed and no longer able to adjust in accordance with the action of the player. For this reason, the pedestrian model is scripted directly into the game, and then the course of action of each agent is able to be updated according to change of the surroundings, including the player, during the run time of the game. However, the downside of implementing the pedestrian models this way is the slowing-down of the game itself; players may experience a certain level of "stuttering" due to the computational power spent on constantly updating agents' actions. Some remedies of the "stuttering" problem will be further discussed in chapter 5.

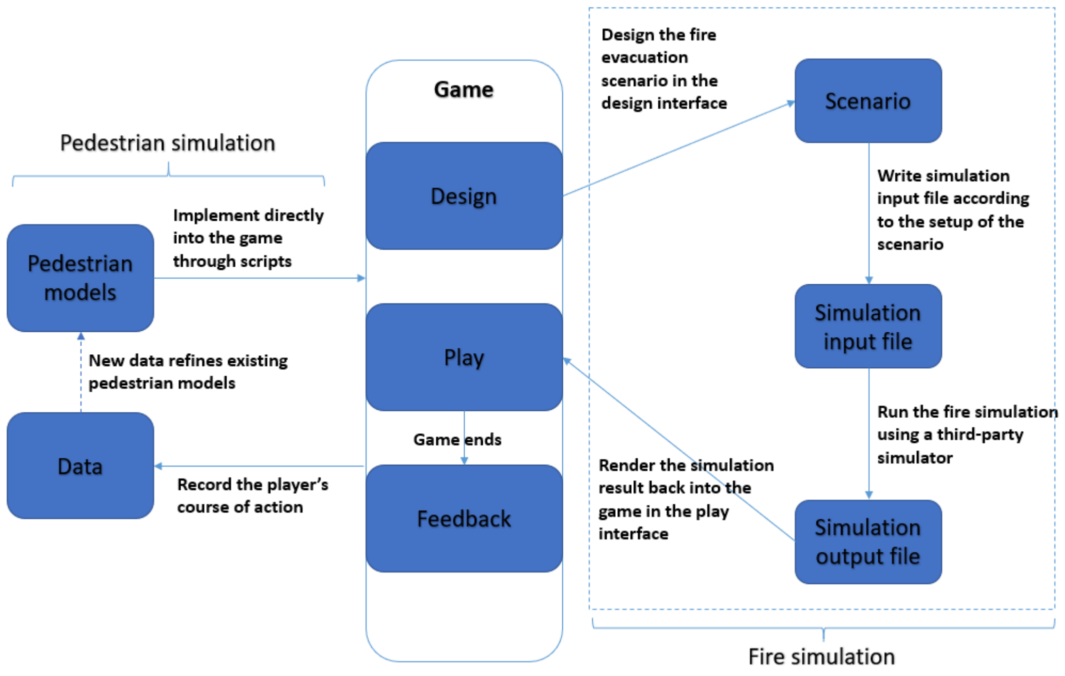

Game
Pedestrian simulation
Design the fire evacuation scenario in the design interface
Scenario
Design
Implement directly into the game through scripts
Pedestrian models
Write simulation input file according to the setup of the scenario
Simulation input file
Play
New data refines existing pedestrian models
Run the fire simulation using a third-party simulator
Game ends
Data
Render the simulation result back into the game in the play interface
Record the player's course of action
Feedback
Simulation output file
Fire simulation

Figure 3.1: The flow chart of implementation.

3.1 Game Interfaces

As shown in Figure 3.1, the game mainly consists of three interfaces, the design interface, the play interface, and the feedback interface.

For designers, the design interface is where they can customize the fire scenario, for example, the locations of the exits, the spawning position of the player, the sizes of the obstacles, the type of the fire, and so on. This interface was created in order to let building designers have a platform to design the game environment that better reflects their own building designs.

After the design is completed in the design interface, the simulation would be performed according to the environment setup of the design, and after the simulation is done, players could then enter the play interface to play the game.

In the play interface, the designed fire scenario would be loaded and the player would spawn at the preset location, and then the game would begin. The objective of the player is not only to escape the fire scene as fast as possible but also to perform a particular series of actions; for example, pressing the alarm soon as realizing there is a fire, helping people in need, and so on. The game would end when the player successfully escapes the building. However, as mentioned previously, the score that the player gets does not depend solely on how fast he/she escapes.

Some data during the game would be recorded, like how long it takes to evacuate, whether or not the player sets off the alarm to warn others, whether or not the player helps others, and so on. Based on these data, the game would then switch to the feedback interface in which feedback would be given to the player on their performance during the game and how they might improve. This aspect of the game is still under development.

3.2 Design Interface

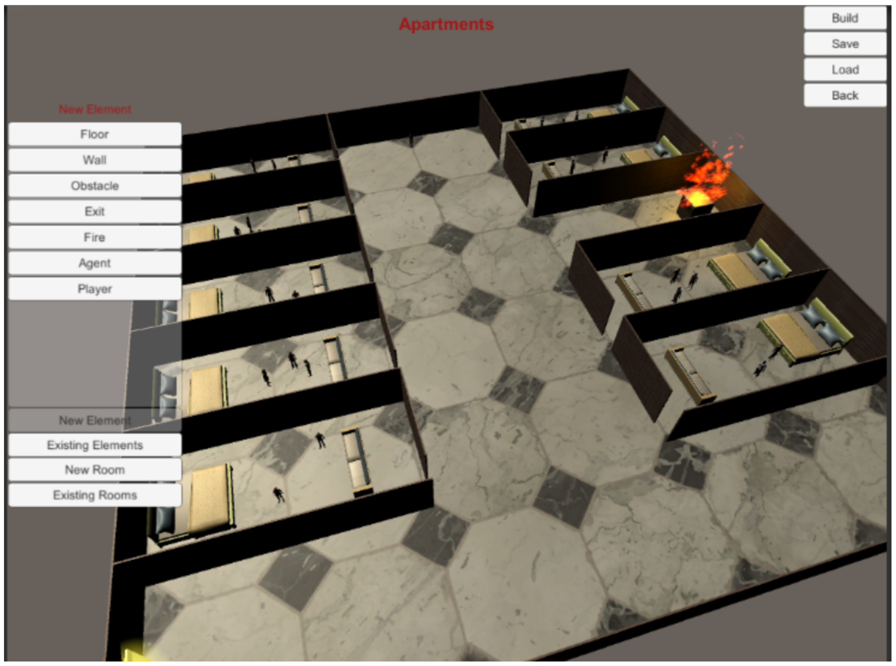
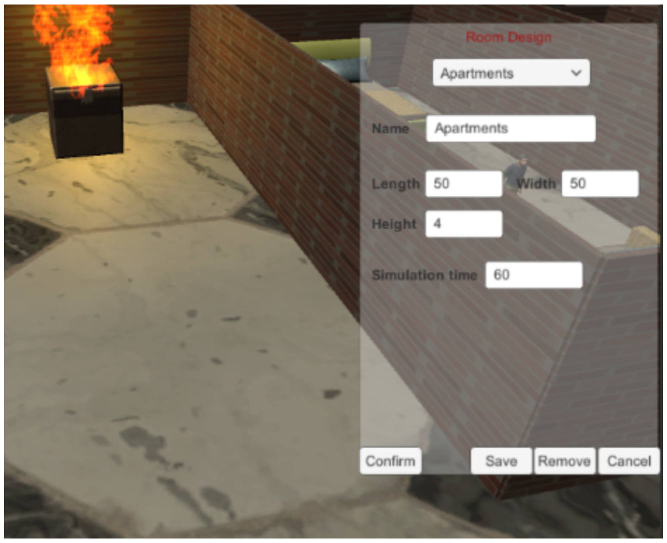


Figure 3.2: Overview of the design interface.

For now, as shown in the element box in Figure 3.2, the game provides seven types of elements for designers to customize; they are floor, wall, obstacle, exit, fire, agent, and player.

3.2.1 Room Design



Apartments

Name: Apartments

Length: 50 Width: 50

Height: 4

Simulation time: 60

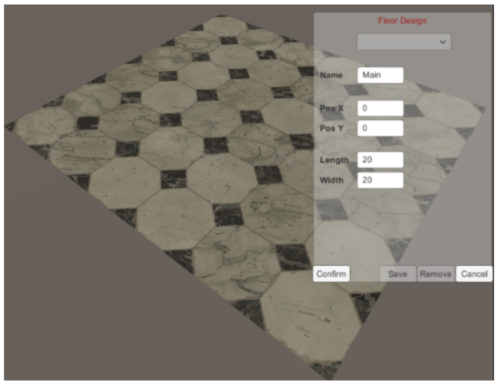
Figure 3.3: Room design panel.

The fire scenario can have different scenes, which are called rooms in the game. Each room consists of the seven fundamental types of elements mentioned in section 3.2. The player can go from a room to another room through a specific exit. A simulation would be run for each room separately, which means that the fire condition of one room would not influence the fire condition of another room. Similarly, agents in one room cannot go into another room. Once an agent enters an exit, it would just disappear. For this reason, designers are expected to place and customize agents separately for each room.

Designers can either build a very large room that includes all the details of a fire evacuation scenario or divide the fire evacuation scenario into several rooms. The apparent advantage of using one large room is that a complete simulation can be done on it. However, if the room scale is too large, the fire simulation will be very slow to generate and also very slow to render in the game. In addition, having a large scale room also means that there could be too many agents running the pedestrian model algorithm during the game run time. Therefore, this kind of design can cause the run time stuttering problem. On the other hand, designers can choose to divide a big scale fire scenario into several smaller rooms, and simulations will then be run and rendered in these smaller rooms separately. Each time, only the fire simulation of the room that the player is in would be rendered. Similarly, only the agents in that room run the pedestrian model algorithm. This design can undoubtedly speed up the game, but the continuity of the fire and pedestrian simulation of the fire scenario would be sacrificed, meaning that the condition of the fire and the pedestrians in one room has nothing to do with that of another room. For now, only three pieces of information are carried from one room to another: the player's health, the time elapsed since the game begins, and the total casualties. Therefore, designers have to be well aware of what level of simulation continuity of the fire scenario is demanded for the training purpose. As shown in Figure 3.2, the "New Room" button creates a new room, and the "Existing Rooms" button displays all the existing rooms.

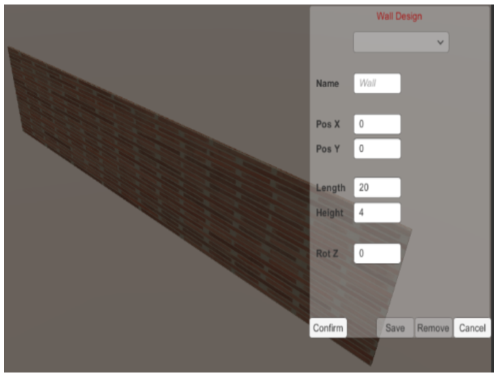
Figure 3.3 shows the room design panel. On the panel, designers need to specify the dimension of the room, and the simulation time of the room, which will later be used to write the fire simulation input file in the simulation phase.

3.2.2 Floor and Wall Design



Name Main
Pos X 0
Pos Y 0
Length 20
Width 20

(a)



Name Wall
Pos X 0
Pos Y 0
Length 20
Height 4
Rot Z 0

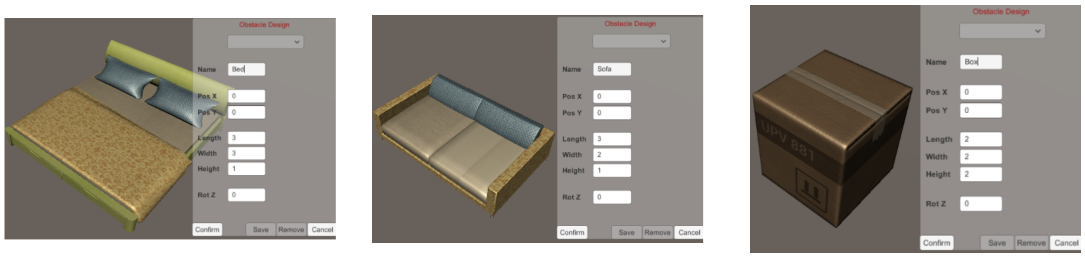
(b)

Figure 3.4: Floor and wall design panels.

As shown in Figure 3.4a, upon creating a floor in the room, designers can customize the new floor's length, width, and position. It is recommended to use only few floor elements in a room for the navigation mesh will later be built on the floor elements. If there are too many floor elements or the floors' alignments are too irregular, the navigation mesh might not be successfully built. The navigation mesh is for the later use of the pedestrian model algorithms, which will be discussed into details in chapter 5. A workaround for building a more complicated and irregular floor is through building one big floor first, and then add obstacles and walls on it to block certain areas out.

As shown in Figure 3.4b, upon creating a wall in the room, designers can customize the new wall's length, height, position, and rotation. Technically, one can use walls to separate the scene (or "Room") into several smaller rooms.

3.2.3 Obstacle Design



(a) Name: Bed; Pos X: 0; Pos Y: 0; Length: 3; Width: 3; Height: 1; Rot Z: 0

(b) Name: Sofa; Pos X: 0; Pos Y: 0; Length: 3; Width: 2; Height: 1; Rot Z: 0

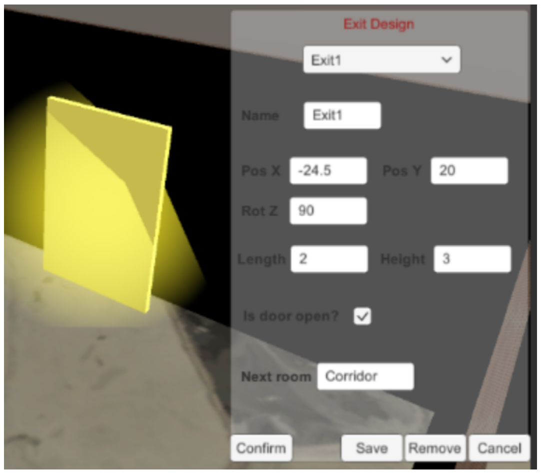
(c) Name: Box; Pos X: 0; Pos Y: 0; Length: 2; Width: 2; Height: 2; Rot Z: 0

Figure 3.5: Obstacle design panels.

Currently, three types of obstacles are available for the designers to put in the room, which are bed, sofa, and box. Only three types of obstacles are implemented for the demonstration purpose. More varieties of obstacles can later be added to the game with ease. As shown in Figure 3.5, upon creating an obstacle, designers can customize its length, width, height, position, and rotation. During the run time of the game, obstacles are static. In other words, agents and players cannot move these obstacles.

3.2.4 Exit Design

Figure 3.6 shows the design panel for exits. Designers have to specify not only the location and dimension of the exit but also whether or not the door is open and which other room the exit would lead the player to. If an exit is set to be not open, then during the run time, this exit will not be considered by the pedestrian model algorithm of agents, and the player cannot pass through this exit. If an exit is open and the room it leads to is specified (in "Next room"), then upon the arrival of the player at the exit, the player would be placed in that new room, and the game would continue in it. However, if "Next room" is specified to be "Outside", then when the player makes it to the exit, the game would end, meaning that the player has successfully finished the game.



Exit1

Name: Exit1

Pos X: -24.5 Pos Y: 20

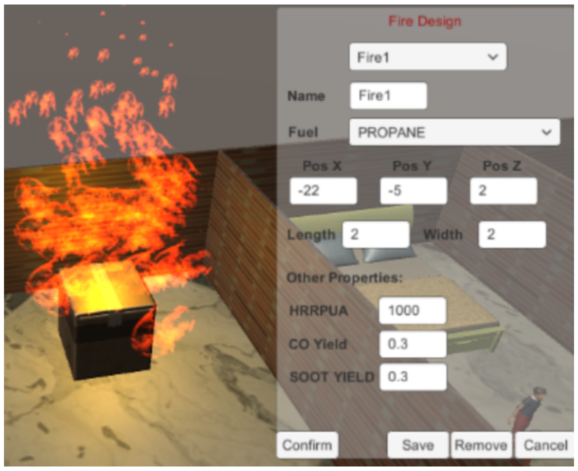
Rot Z: 90

Length: 2 Height: 3

Is door open? ✓

Next room: Corridor

Figure 3.6: Exit design panel.



Fire1

Name: Fire1

Fuel: PROPANE

Pos X: -22 Pos Y: -5 Pos Z: 2

Length: 2 Width: 2

HRRPUA: 1000

CO Yield: 0.3

SOOT YIELD: 0.3

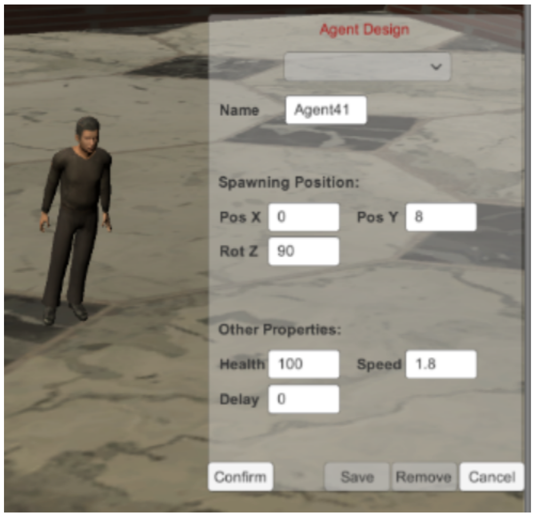
Figure 3.7: Fire design panel.

3.2.5 Fire Design

Figure 3.7 shows the fire design panel. Since *FDS* is the third-party fire simulator used by this game, specifications about the fire are made in accordance with the features implemented in *FDS*. First of all, designers need to specify the location of the fire source. Then one needs to specify the length and width of the fire. In *FDS*, fire source is treated as a heat releasing surface. After that, one needs to specify the fuel of the fire source, for example, propane, hydrogen, methane *etc.* The intensity of the fire is reflected by Heat Release Rate

Per Unit Area (HRRPUA). One can also specify the CO yielding rate and soot yielding rate in the panel. All these specifications would later be translated and written into an *FDS* input file, which can later be run by *FDS*.

3.2.6 Agent Design


Agent Design

Name: Agent41

Spawning Position:

Pos X: 0 Pos Y: 8

Rot Z: 90

Other Properties:

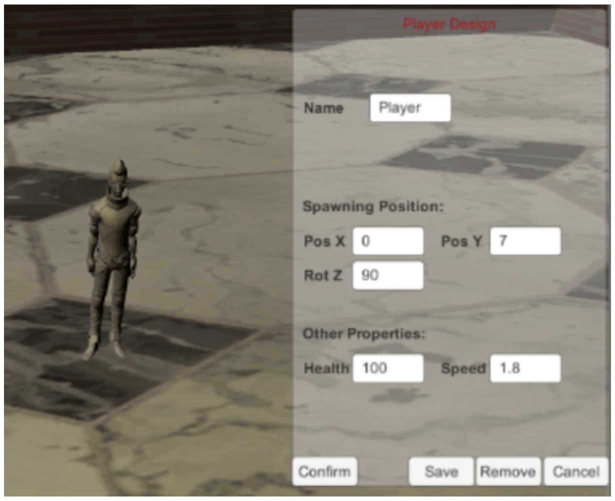
Health: 100 Speed: 1.8

Delay: 0

Figure 3.8: Agent design panel.

Figure 3.8 shows the agent design panel. “Modern People” package of Unity was used as the prototypes of agents. There are twenty different prototypes of agents in the package, including ten males and ten females. All of them differ only in their appearances. Figure 3.8 shows one male prototype. On the agent design panel, one can specify the spawning location of the agent. Some additional specifications for agents are the health, walking speed, and the delay time. These three specifications are all related to the pedestrian model algorithm used in this game. For the current development, the fire-to-human impact is reflected by the intoxicating effect of gas. The health of an agent will be reduced according to the density of the toxic gases. Therefore, by adjusting the health of an agent, one can, in turn, determine how long an agent can sustain in the fire scenario. One can use “Speed” to specify the normal walking speed of an agent, say agent i. This value would be used as the desired walking speed term of agent i, v_i^0, in the social-force model. The delay time, also called pre-evacuation time, is a new concept. Literature [26, 27, 28] suggests that the evacuation time of an agent is calculated as $t_{ev} = t_p + t_m$, where t_p is the pre-evacuation time, and t_m is the time of an agent moving to an exit (including the queuing time at the exit). The pre-evacuation time t_p reflects the phenomenon that people tend not to immediately take action even after realizing the fire. Designers can use “Delay Time” to set how long an agent should wait until it starts to move. If unspecified, by default, the agents start as soon as the room has been loaded.

3.2.7 Player Design



Name: Player

Spawning Position:

Pos X: 0
Pos Y: 7
Rot Z: 90

Other Properties:

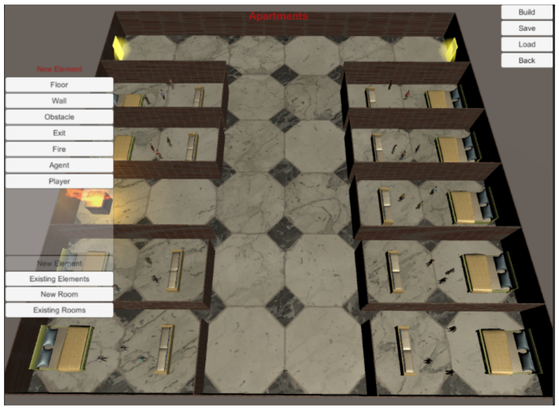
Health: 100
Speed: 1.8

Figure 3.9: Player design panel.

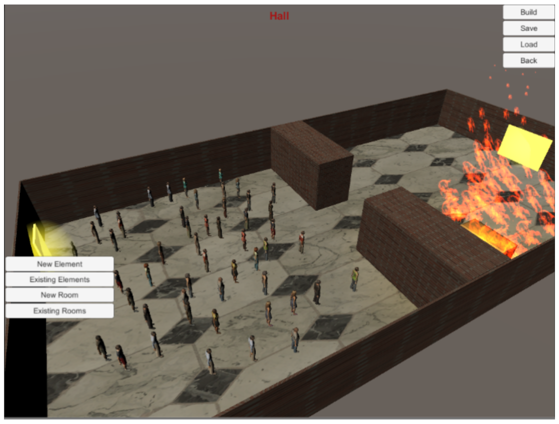
Figure 3.9 shows the player design panel. The Unity-built-in third-person-controlled character is used as the prototype of the player to differ from the prototypes of other agents. Besides the usual specifications, like all other agents, designers can also specify the health of the player, and once the health runs out due the intoxicating effect of the gas, the player dies, and the game ends. Also, the speed can be specified to determine how fast the player moves. In contrast to the other agents, the player is the only character with genuine agency in the game, so it is not restricted by the pre-evacuation time (delay time) and is able to influence the agents’ actions.

3.2.8 Design Example

Figure 3.10 shows the design of a fire evacuation scenario. The room “Apartments” (Figure 3.10a) contains several apartments that people live in. Fire is placed at the position



(a) Apartments



(b) Hall

Figure 3.10: A design example.

shown in the figure. Furniture is placed in each apartment, which is treated as obstacles. As fire spreads out, people need to escape from the scene. The exit at the top-left corner is set to be unavailable due to the fire condition, while the exit at the top-right corner connects to the second room "Hall" (Figure 3.10b). After entering the top-right exit in room "Apartments", the player would spawn at the left exit in "Hall". Agents are scattered in the left half of "Hall", and a fire is burning behind a wall. Note that the passage narrows down in the middle, where congestion is expected to be observed. The exit on the right of "Hall" leads to a safe place, meaning that, game ends upon entering this exit.

This simple design example meant to demonstrate that one can build more complicated fire evacuation scenarios using the very fundamental tools provided in the design interface.

3.3 Play Interface

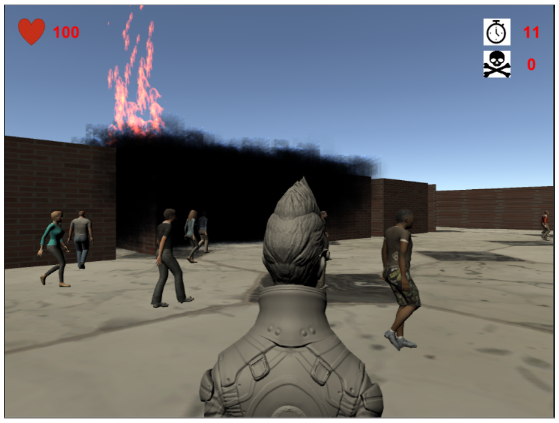
After the design of a fire evacuation scenario is finished, one can click the "Build" button on the top-right of the screen (see Figure 3.10) to perform the fire simulation based on the design and then play in the designed scenario in the "Play" interface.

For the moment, the game is only available on PCs. In the "Play" interface, the player can move freely using the standard "WASD" keyboard control, and the view rotation is controlled by the movement of mouse.

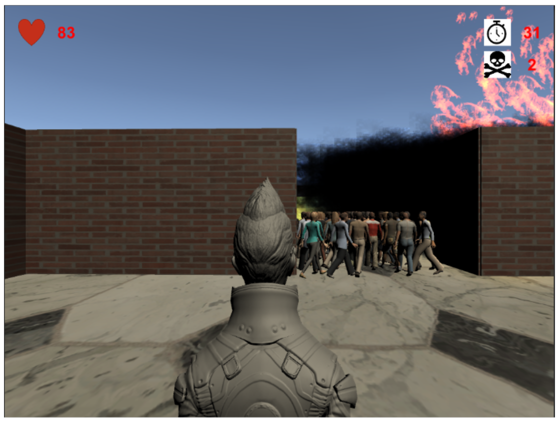
Figure 3.11a shows what it looks like in the "Play" interface in the "Apartments" room of the example discussed above. Agents evacuate and the smoke spreads based on the pedestrian model and the fire simulation respectively in real time. The details of pedestrian model implementation and fire simulation implementation will be discussed in later chapters.

The top-left corner of the monitor indicates the health of the player, which is influenced by the smoke. If the player's health drops to 0, it means that the player has been incapacitated by the smoke, and the game is over. The fire-to-human interaction is part of the pedestrian model and will be discussed in chapter 5. On the top-right corner of the monitor are a clock and a casualty indicator. The clock indicates the time elapsed from the start of the game and the casualty indicator shows the number of casualties that have occurred.

Figure 3.11b shows the scene in the "Hall" room. One can notice that two casualties have already occurred, the health of the player has dropped to 83, and congestion has formed at the narrow passage as expected.



(a) Play in room "Apartments".



(b) Play in room "Hall".

Figure 3.11: Overview of the play interface.

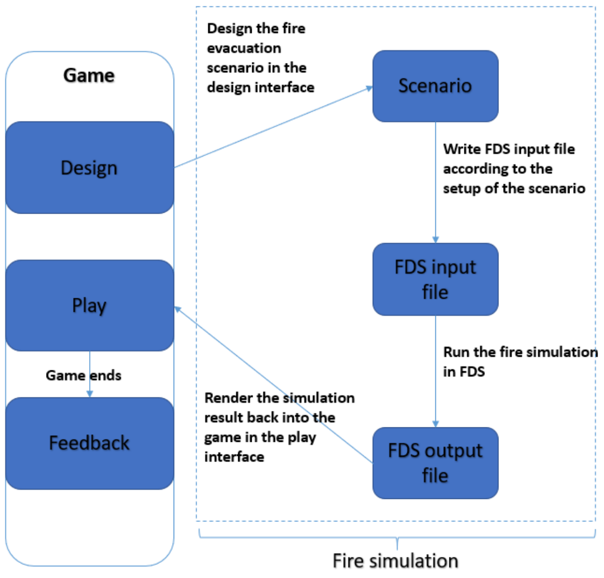
3.4 Chapter Summary

Serious Game for Fire Evacuation consists of three interfaces, the design interface, the play interface, and the feedback interface. The design interface allows designers to design the fire scenario to carry out the evacuation. The design interface can be seen as a GUI of some third-party fire simulators, like *FDS*, but with some additional features. The play interface is where the player gets to play in the designed fire scenario. In the play interface, the fire simulation and the pedestrian simulation of the fire scenario will be rendered. The player is expected to perform a certain course of actions as if in a real fire evacuation situation. After the player finishes in the fire evacuation scenario, the game will switch to the feedback interface where an evaluation on the performance of the player will be given. The feedback is meant to instruct the player how to improve in the fire evacuation. The feedback interface has not been implemented into the game.

Chapter 4

Fire Simulation Implementation

Fire dynamics simulation is too complicated to be implemented into the game manually. Therefore, this game uses a third-party fire simulator to do the simulation. *Fire Dynamics Simulator (FDS)* is chosen because it is very well developed and is an open source.



Game
Design
Play
Game ends
Feedback
Design the fire evacuation scenario in the design interface
Scenario
Write FDS input file according to the setup of the scenario
FDS input file
Run the fire simulation in FDS
Render the simulation result back into the game in the play interface
FDS output file
Fire simulation

Figure 4.1: Flow chart of fire simulation.

Once the design phase is complete, designers can click the "Build" button in the design interface to build the scenario. The flow chart of the fire simulation is shown in Figure 4.1. An *FDS* input file will be generated for each room according to its setup. Then through scripts, *FDS* would run these *FDS* input files to perform the simulation. Once the simulation is done, a simulation output file would be created, and all the requested simulation result is stored in the output file. The last step is to read the output file and render the simulation result back into the game. These steps briefly conclude the implementation of the fire simulation.

4.1 Write Simulation Input File

After the design of the scenario is completed and the “Build” button is pressed, a folder will be created through scripts for each room to store the design and the simulation result of that room. First of all, for each room, an *FDS* input file (.fds) will be generated according to the design of that room. A typical *FDS* input file is shown as follows:

```
&HEAD CHID =' Hall', TITLE =' Hall'/
&MESH IJK = 20, 40, 4, XB = −10, 10, −20, 20, 0, 4/
&TIME T_END = 60/
&OBST XB = −10, −2.5, 1.45, 1.55, 0, 4/
&OBST XB = −10, −2.5, −1.55, −1.45, 0, 4/
&OBST XB = −2.55, −2.45, −1.5, 1.5, 0, 4/
&OBST XB = 2.5, 10, 1.45, 1.55, 0, 4/
&OBST XB = 2.5, 10, −1.55, −1.45, 0, 4/
&OBST XB = 2.45, 2.55, −1.5, 1.5, 0, 4/
&OBST XB = 5.5, 8.5, 2.5, 5.5, 0, 2/
&REAC ID =' HallReaction'
FUEL =' PROPANE'
CO_YIELD = 0.4
SOOT_YIELD = 0.4/
&SURF ID =' HallReaction', HRRPUA = 1000/
&VENT XB = 5.5, 8.5, 2.5, 5.5, 2, 2, SURF_ID =' CorridorReaction'/
&SLCF PBZ = 0.5, QUANTITY =' DENSITY', SPEC_ID =' SOOT'/
&SLCF PBZ = 1.5, QUANTITY =' DENSITY', SPEC_ID =' SOOT'/
&SLCF PBZ = 2.5, QUANTITY =' DENSITY', SPEC_ID =' SOOT'/
&SLCF PBZ = 3.5, QUANTITY =' DENSITY', SPEC_ID =' SOOT'/
&DEVCXYZ = −25, −25, 1.5, QUANTITY =' EXTINCTIONCOEFFICIENT'/
&TAIL/
```

The details of the fire scenario specified previously in the design interface are now translated and written into the simulation input file. The commands in the input file are explained below [20].

- The namelist group &HEAD on the first line specifies the name of the simulation. CHID is a string of 30 characters or less used to tag the output files. TITLE is a string of 60 characters or less that describes the simulation. Here, the game will generate the input file for a room with both CHID and TITLE being the name of the room.

- The &MESH namelist group on the second line specifies the alignment and size of the simulation meshes. For a simulation mesh alignment that counts i meshes going from $\mathrm{x_0}$ to $\mathrm{x_1}$ in the x direction, j meshes from $\mathrm{y_0}$ to $\mathrm{y_1}$ in the y direction, and k meshes from $\mathrm{z_0}$ to $\mathrm{z_1}$ in the z direction, it is written as $\mathrm{\&MESH\ IJK = i, j, k,\ \ XB = x_0, x_1, y_0, y_1, z_0, z_1/}$. Note that meshes are of length $(\mathrm{x_1} - \mathrm{x_0})/i$, width $(\mathrm{y_1} - \mathrm{y_0})/j$, and height $(\mathrm{z_1} - \mathrm{z_0})/k$. In our example, mesh size is set to be 1 m, and thus the &MESH namelist group is set accordingly to be so.

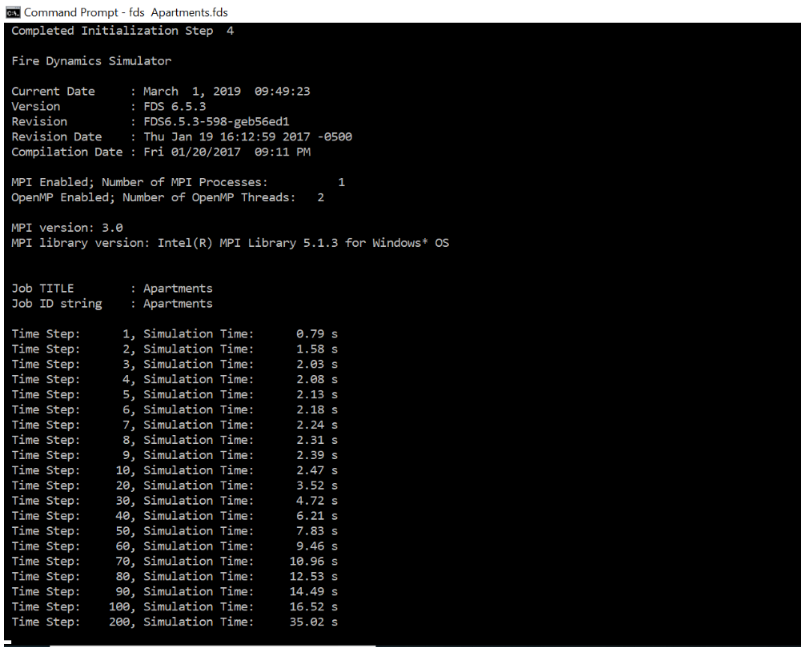
- On the third line, the &TIME namelist group specifies the simulation time and time step. For a simulation that starts from $\mathrm{t_0}$ and ends at $\mathrm{t_1}$, it is written as $\mathrm{\&TIME\ T\_BEGIN = t_0\ T\_END = t_1/}$. In our example, the simulation covers up to 60 s, and since the beginning time of the simulation is not specified, it starts from 0 s by default. In the same line, the initial time step can be specified to be dt using the command DT = dt. The initial time step, DT, is not specified in the simulation of this game. If the initial time step DT is not specified, then by default, it will be automatically set to be the value of dividing the size of a mesh by the characteristic velocity of the flow. To be more specific, the default value of DT is $5(\delta x \delta y \delta z)^{\frac{1}{3}}/\sqrt{gH}$ s, where δx, δy, and δz are the dimensions of the smallest mesh cell, H is the height of the height of the computational domain, and g is the acceleration of gravity. In our example, since all the meshes are of dimensions $\delta x = 1$ m, $\delta y = 1$ m, $\delta z = 1$ m, and the domain height is $H = 4$ m, the initial time step is thus DT = 0.79 s. During the simulation, the time step is adjusted so that the CFL (Courant, Friedrichs, Lewy) condition is satisfied. This is an advanced topic, one can just let *FDS* take care of it.

In the design interface, one has to specify the dimensions of all the obstacles. Note that all walls are also treated as obstacles when writing the *FDS* input file. For obstacles, only the dimensions need to be specified in the *FDS* input file. Line 4 to line 10 of the example input file show the specification of all obstacles of the example scenario.

Line 11 to line 16 specify the fire. The &REAC namelist group specifies the reaction of the fire. When extracting the setup of the game into the input file, the name of the reaction would be automatically set to be the name of the room combined with keyword "Reaction". In the design phase, one needs to specify the fire as shown in Figure 3.7, and these details will be translated and written into the input file as shown.

Line 17 to line 20 specify four slices at height 0.5 m, 1.5 m, 2.5 m, and 3.5 m to collect the soot density data. Later four soot density profiles will be generated at these four different heights. Line 21 specifies a device that records the FED value profile at the specified position. There are actually many more command lines like line 21 in the actual input file, which put a device recording the FED value profile at each mesh point at height 1.5 m (covers from 1 m to 2 m in z direction, where a human's head should be). Note that the current design of the game is to render smoke according to the soot density, and use FED value for the intoxicating effect. But for later development, one can use other commands [20] to extract other data of interest.

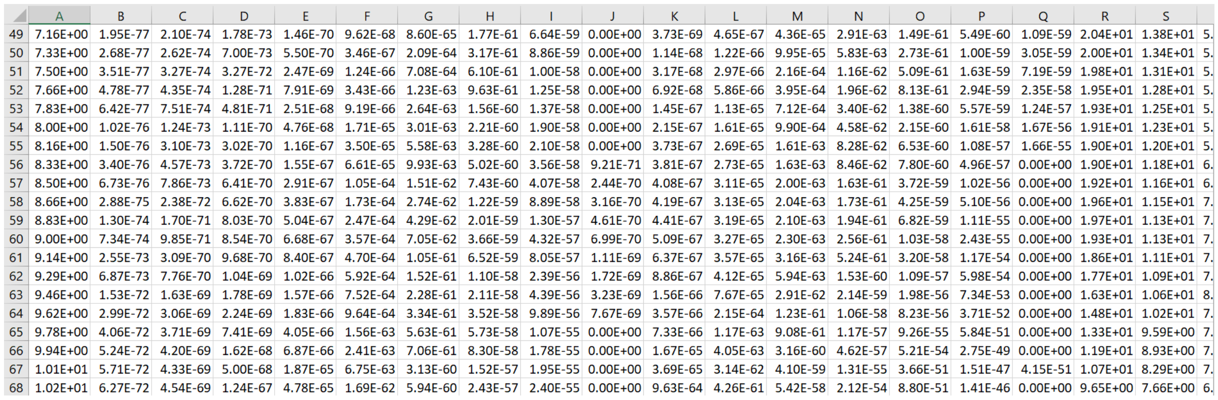
4.2 Run Simulation



```
Command Prompt - fds  Apartments.fds
Completed Initialization Step  4

Fire Dynamics Simulator

Current Date     : March  1, 2019  09:49:23
Version          : FDS 6.5.3
Revision         : FDS6.5.3-598-geb56ed1
Revision Date    : Thu Jan 19 16:12:59 2017 -0500
Compilation Date : Fri 01/20/2017  09:11 PM

MPI Enabled; Number of MPI Processes:         1
OpenMP Enabled; Number of OpenMP Threads:   2

MPI version: 3.0
MPI library version: Intel(R) MPI Library 5.1.3 for Windows* OS


Job TITLE        : Apartments
Job ID string    : Apartments

Time Step:      1, Simulation Time:      0.79 s
Time Step:      2, Simulation Time:      1.58 s
Time Step:      3, Simulation Time:      2.03 s
Time Step:      4, Simulation Time:      2.08 s
Time Step:      5, Simulation Time:      2.13 s
Time Step:      6, Simulation Time:      2.18 s
Time Step:      7, Simulation Time:      2.24 s
Time Step:      8, Simulation Time:      2.31 s
Time Step:      9, Simulation Time:      2.39 s
Time Step:     10, Simulation Time:      2.47 s
Time Step:     20, Simulation Time:      3.52 s
Time Step:     30, Simulation Time:      4.72 s
Time Step:     40, Simulation Time:      6.21 s
Time Step:     50, Simulation Time:      7.83 s
Time Step:     60, Simulation Time:      9.46 s
Time Step:     70, Simulation Time:     10.96 s
Time Step:     80, Simulation Time:     12.53 s
Time Step:     90, Simulation Time:     14.49 s
Time Step:    100, Simulation Time:     16.52 s
Time Step:    200, Simulation Time:     35.02 s
```

Figure 4.2: *FDS* running simulation based on the input file.

After the *FDS* input file has been created, the game will externally use *FDS* to run the *FDS* input file (see Figure 4.2). *FDS* will then generate output files which contain data requested, like the FED value data shown in Figure 4.3. Note that the simulation process is hidden from the users in the game.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	
49	7.16E+00	1.95E-77	2.10E-74	1.78E-73	1.46E-70	9.62E-68	8.60E-65	1.77E-61	6.64E-59	0.00E+00	3.73E-69	4.65E-67	4.36E-65	2.91E-63	1.49E-61	5.49E-60	1.09E-59	2.04E+01	1.38E+01	5.
50	7.33E+00	2.68E-77	2.62E-74	7.00E-73	5.50E-70	3.46E-67	2.09E-64	3.17E-61	8.86E-59	0.00E+00	1.14E-68	1.22E-66	9.95E-65	5.83E-63	2.73E-61	1.00E-59	3.05E-59	2.00E+01	1.34E+01	5.
51	7.50E+00	3.51E-77	3.27E-74	3.27E-72	2.47E-69	1.24E-66	7.08E-64	6.10E-61	1.00E-58	0.00E+00	3.17E-68	2.97E-66	2.16E-64	1.16E-62	5.09E-61	1.63E-59	7.19E-59	1.98E+01	1.31E+01	5.
52	7.66E+00	4.78E-77	4.35E-74	1.28E-71	7.91E-69	3.43E-66	1.23E-63	9.63E-61	1.25E-58	0.00E+00	6.92E-68	5.86E-66	3.95E-64	1.96E-62	8.13E-61	2.94E-59	2.35E-58	1.95E+01	1.28E+01	5.
53	7.83E+00	6.42E-77	7.51E-74	4.81E-71	2.51E-68	9.19E-66	2.64E-63	1.56E-60	1.37E-58	0.00E+00	1.45E-67	1.13E-65	7.12E-64	3.40E-62	1.38E-60	5.57E-59	1.24E-57	1.93E+01	1.25E+01	5.
54	8.00E+00	1.02E-76	1.24E-73	1.11E-70	4.76E-68	1.71E-65	3.01E-63	2.21E-60	1.90E-58	0.00E+00	2.15E-67	1.61E-65	9.90E-64	4.58E-62	2.15E-60	1.61E-58	1.67E-56	1.91E+01	1.23E+01	5.
55	8.16E+00	1.50E-76	3.10E-73	3.02E-70	1.16E-67	3.50E-65	5.58E-63	3.28E-60	2.10E-58	0.00E+00	3.73E-67	2.69E-65	1.61E-63	8.28E-62	6.53E-60	1.08E-57	1.66E-55	1.90E+01	1.20E+01	5.
56	8.33E+00	3.40E-76	4.57E-73	3.72E-70	1.55E-67	6.61E-65	9.93E-63	5.02E-60	3.56E-58	9.21E-71	3.81E-67	2.73E-65	1.63E-63	8.46E-62	7.80E-60	4.96E-57	0.00E+00	1.90E+01	1.18E+01	6.
57	8.50E+00	6.73E-76	7.86E-73	6.41E-70	2.91E-67	1.05E-64	1.51E-62	7.43E-60	4.07E-58	2.44E-70	4.08E-67	3.11E-65	2.00E-63	1.63E-61	3.72E-59	1.02E-56	0.00E+00	1.92E+01	1.16E+01	6.
58	8.66E+00	2.88E-75	2.38E-72	6.62E-70	3.83E-67	1.73E-64	2.74E-62	1.22E-59	8.89E-58	3.16E-70	4.19E-67	3.13E-65	2.04E-63	1.73E-61	4.25E-59	5.10E-56	0.00E+00	1.96E+01	1.15E+01	7.
59	8.83E+00	1.30E-74	1.70E-71	8.03E-70	5.04E-67	2.47E-64	4.29E-62	2.01E-59	1.30E-57	4.61E-70	4.41E-67	3.19E-65	2.10E-63	1.94E-61	6.82E-59	1.11E-55	0.00E+00	1.97E+01	1.13E+01	7.
60	9.00E+00	7.34E-74	9.85E-71	8.54E-70	6.68E-67	3.57E-64	7.05E-62	3.66E-59	4.32E-57	6.99E-70	5.09E-67	3.27E-65	2.30E-63	2.56E-61	1.03E-58	2.43E-55	0.00E+00	1.93E+01	1.13E+01	7.
61	9.14E+00	2.55E-73	3.09E-70	9.68E-70	8.40E-67	4.70E-64	1.05E-61	6.52E-59	8.05E-57	1.11E-69	6.37E-67	3.57E-65	3.16E-63	5.24E-61	3.20E-58	1.17E-54	0.00E+00	1.86E+01	1.11E+01	7.
62	9.29E+00	6.87E-73	7.76E-70	1.04E-69	1.02E-66	5.92E-64	1.52E-61	1.10E-58	2.39E-56	1.72E-69	8.86E-67	4.12E-65	5.94E-63	1.53E-60	1.09E-57	5.98E-54	0.00E+00	1.77E+01	1.09E+01	7.
63	9.46E+00	1.53E-72	1.63E-69	1.78E-69	1.57E-66	7.52E-64	2.28E-61	2.11E-58	4.39E-56	3.23E-69	1.56E-66	7.67E-65	2.91E-62	2.14E-59	1.98E-56	7.34E-53	0.00E+00	1.63E+01	1.06E+01	8.
64	9.62E+00	2.99E-72	3.06E-69	2.24E-69	1.83E-66	9.64E-64	3.34E-61	3.52E-58	9.89E-56	7.67E-69	3.57E-66	2.15E-64	1.23E-61	1.06E-58	8.23E-56	3.71E-52	0.00E+00	1.48E+01	1.02E+01	7.
65	9.78E+00	4.06E-72	3.71E-69	7.41E-69	4.05E-66	1.56E-63	5.63E-61	5.73E-58	1.07E-55	0.00E+00	7.33E-66	1.17E-63	9.08E-61	1.17E-57	9.26E-55	5.84E-51	0.00E+00	1.33E+01	9.59E+00	7.
66	9.94E+00	5.24E-72	4.20E-69	1.62E-68	6.87E-66	2.41E-63	7.06E-61	8.30E-58	1.78E-55	0.00E+00	1.67E-65	4.05E-63	3.16E-60	4.62E-57	5.21E-54	2.75E-49	0.00E+00	1.19E+01	8.93E+00	7.
67	1.01E+01	5.71E-72	4.33E-69	5.00E-68	1.87E-65	6.75E-63	3.13E-60	1.52E-57	1.95E-55	0.00E+00	3.69E-65	3.14E-62	4.10E-59	1.31E-55	3.66E-51	1.51E-47	4.15E-51	1.07E+01	8.29E+00	7.
68	1.02E+01	6.27E-72	4.54E-69	1.24E-67	4.78E-65	1.69E-62	5.94E-60	2.43E-57	2.40E-55	0.00E+00	9.63E-64	4.26E-61	5.42E-58	2.12E-54	8.80E-51	1.41E-46	0.00E+00	9.65E+00	7.66E+00	6.

Figure 4.3: FED output data.

4.3 Render Simulation

Smokeview is the visualization program that renders the simulation result of *FDS*. The way that *Smokeview* determines the opacity of smoke is described in section 17.10.2 [29]. The intensity of monochromatic light passing a distance L through smoke is attenuated according to

$$I/I_0 = e^{-KL}, \tag{4.1}$$

where K is the light extinction coefficient (see [30] for more information).

The intensity of monochromatic light, I/I_0 can be used to determine the opacity of smoke. Note that, not digging into details, the light extinction coefficient K is calculated as $K = K_m \rho Y_S$, where K_m is a constant, and ρY_s is soot density. Then one can see that a higher soot density leads to a lower intensity of monochromatic light, meaning less light can go through and thus more opaque. *Smokeview* uses the conventional RBGA color, the opacity is calculated as $255 \cdot (I/I_0)$. Figure 4.4 demonstrates different smoke of different opacities. Note that, different from the RBGA color convention where 255 is the most opaque and 0 is the most transparent, *Smokeview* interprets the opacity reversely with 255 being the most transparent and 0 being the most opaque.

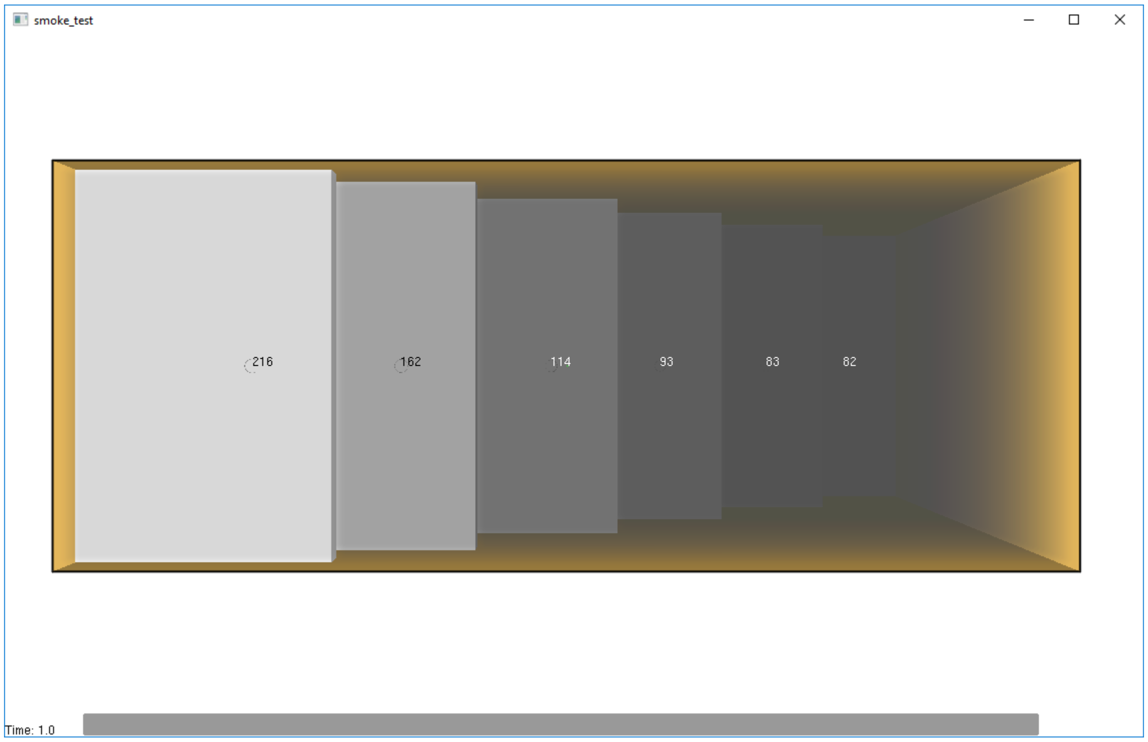


Figure 4.4: Smoke of different opacities (source: https://github.com/firemodels/fds/issues/5118).

For this project, in the game engine Unity, the smoke is rendered using the built-in particle system [31]. The particle system is essentially an animation that runs repeatedly. The texture of the smoke can be achieved using the appropriate design of the particle system. One can certainly use more advanced tools to more realistically render the smoke, but for the moment, the particle system suffices to do the job. Figure 4.5 shows one design of smoke using the particle system. The smoke opacities $255, 196, 137, 78, 20$ go from left to right respectively. Unity uses the conventional RBGA color with 255 alpha value being the most opaque and 0 being the most transparent.

The data required to update the opacity of the smoke particle system is light extinction coefficient K, which can be directly extracted from the simulation. The time profile of opacity, A, of a particle system is then calculated by

$$A = 255 \cdot (1 - \frac{I}{I_0}) = 255 \cdot (1 - e^{-KL}), \tag{4.2}$$

where $(1 - e^{-KL})$ accounts for the different interpretation of RGBA alpha value in Unity

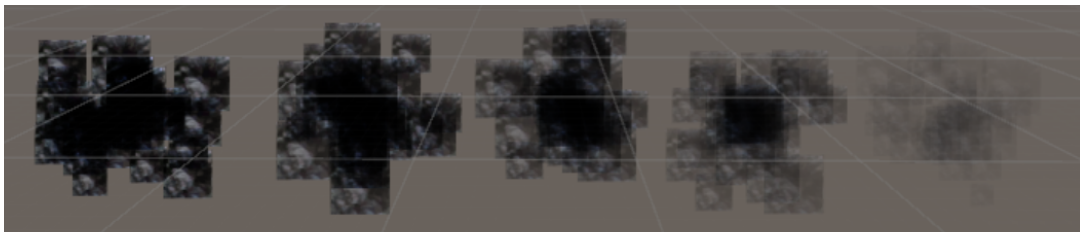


Figure 4.5: Different opacities of smoke rendered using the particle system in Unity. The alpha values from left to right are: $255, 196, 137, 78, 20$.

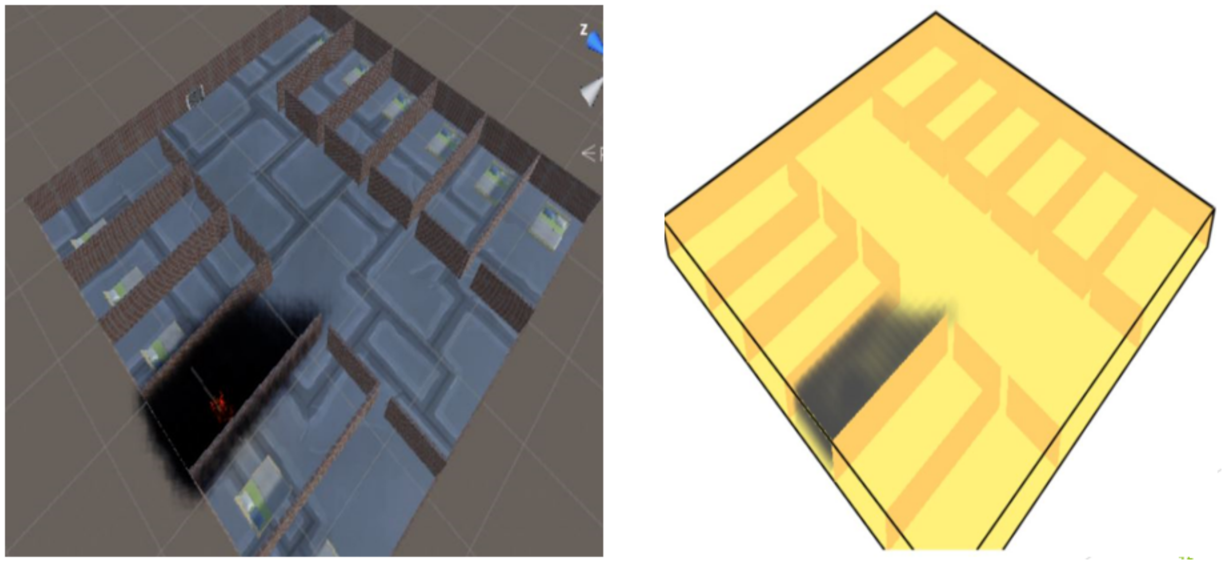
from *Smokeview*, and L is set to be 1 since in this project meshes are of size 1 m.

In the building phase of the game, an empty game object will be created at each mesh point, calling it mesh-object. In Unity, an empty game object has no physical appearance, which can be simply considered as a reference. One smoke particle system will then be attached to each mesh object. Through scripts, each mesh object also contains essential data profiles at the corresponding mesh point in the simulation. For example, the opacity profile for smoke rendering, the fractional effective dose (FED) profile for human-fire interaction, and any other profiles for other purposes. Because the time step of this game is set to be 1 s, the opacity of each smoke particle system is updated every 1 s based on the opacity pre-stored in its attached mesh object. Note that in Figure 4.5, if the alpha value is smaller than 20, the smoke is almost transparent. Therefore, if the opacity profile of a mesh-object contains no opacity bigger than 20 throughout the simulation time, then no smoke particle system will be created for that mesh object. This simplification saves up a lot of computational power.

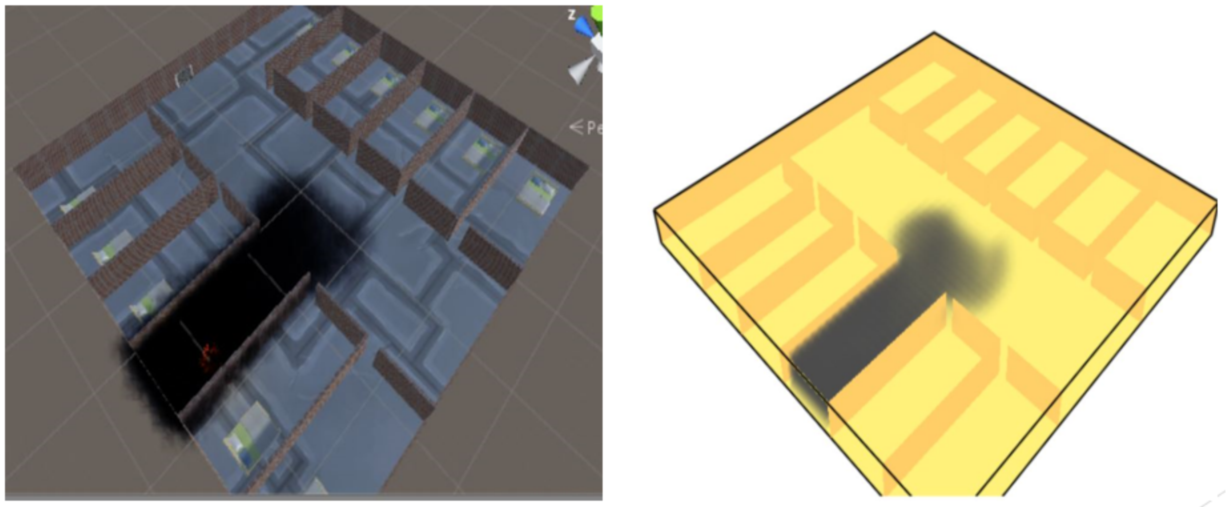
4.4 Result and Discussion

Figure 4.6 and Figure 4.7 compare the smoke visualization in the game and in *Smokeview* of the example scenario. Despite the textures, the propagations of the smoke are identical because they are based on the same simulation result.

Figure 4.8 is how it looks like standing in the midst of the smoke. The overall visualization of the smoke in the game is slightly crude due to the fact that the particle system representation might not be the best tool to render the smoke, and it should also be remembered that the simulation/rendering mesh size is set to be 1 m, which is rather coarse. Later on, some advanced animation tools, like *Blender* [32], can be used to improve the smoothness of the visualization. Also, if system permits, a more elaborate fire simulation can be done using smaller simulation mesh sizes, which would result in a more continuous and smooth smoke visualization in the game.

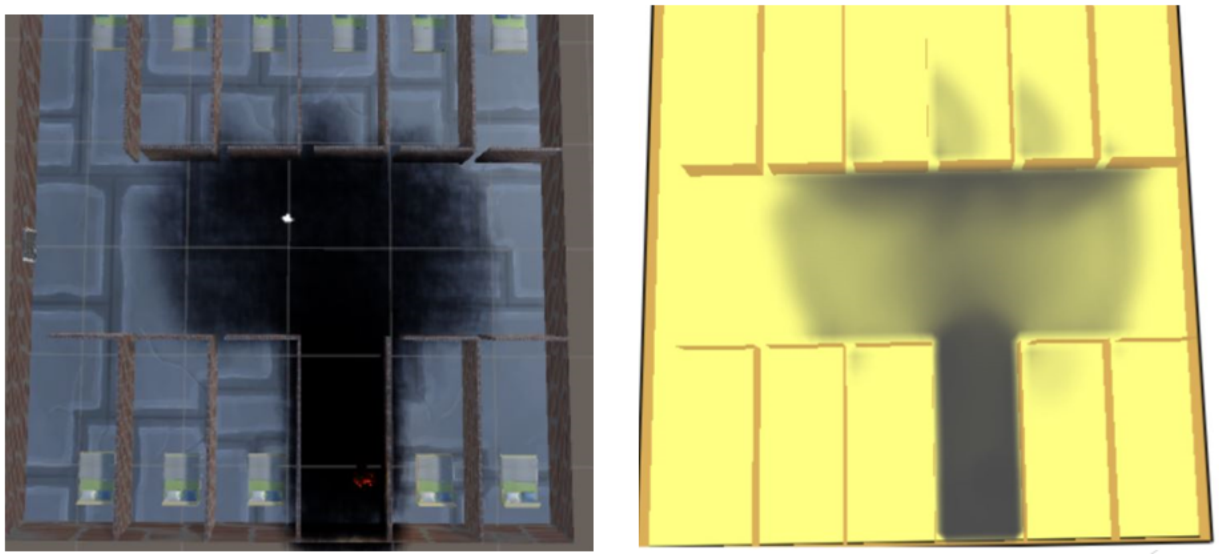


(a) Simulation at 10 s.



(b) Simulation at 20 s.

Figure 4.6: Comparison of visualization of smoke in the game and *Smokeview* at 10 s and 20 s.


(a) Simulation at 40 s.

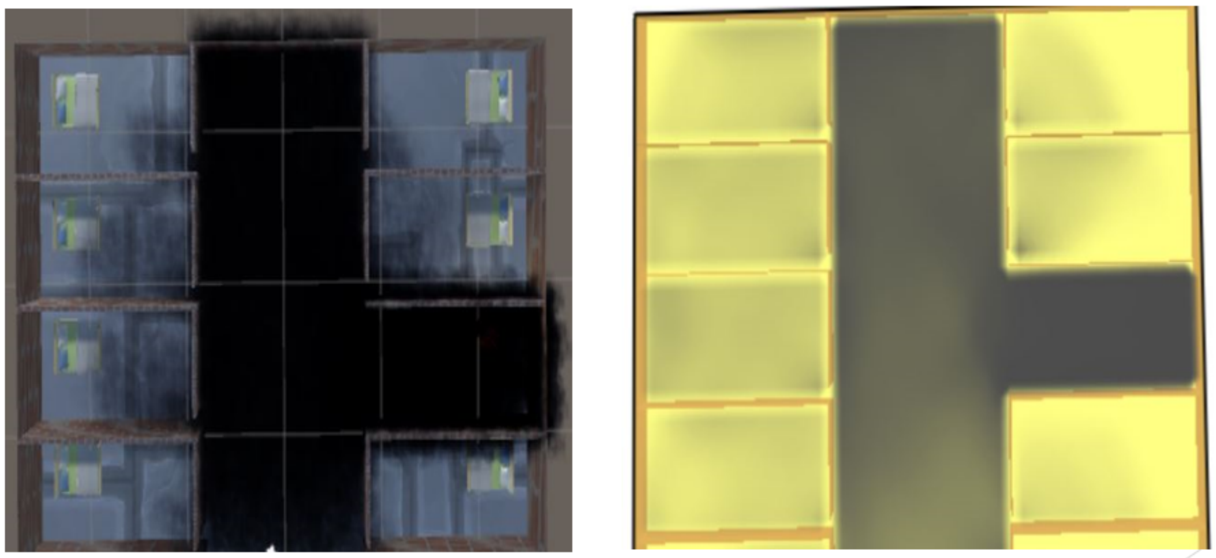

(b) Simulation at 80 s.

Figure 4.7: Comparison of visualization of smoke in the game and *Smokeview* at 40 s and 80 s.

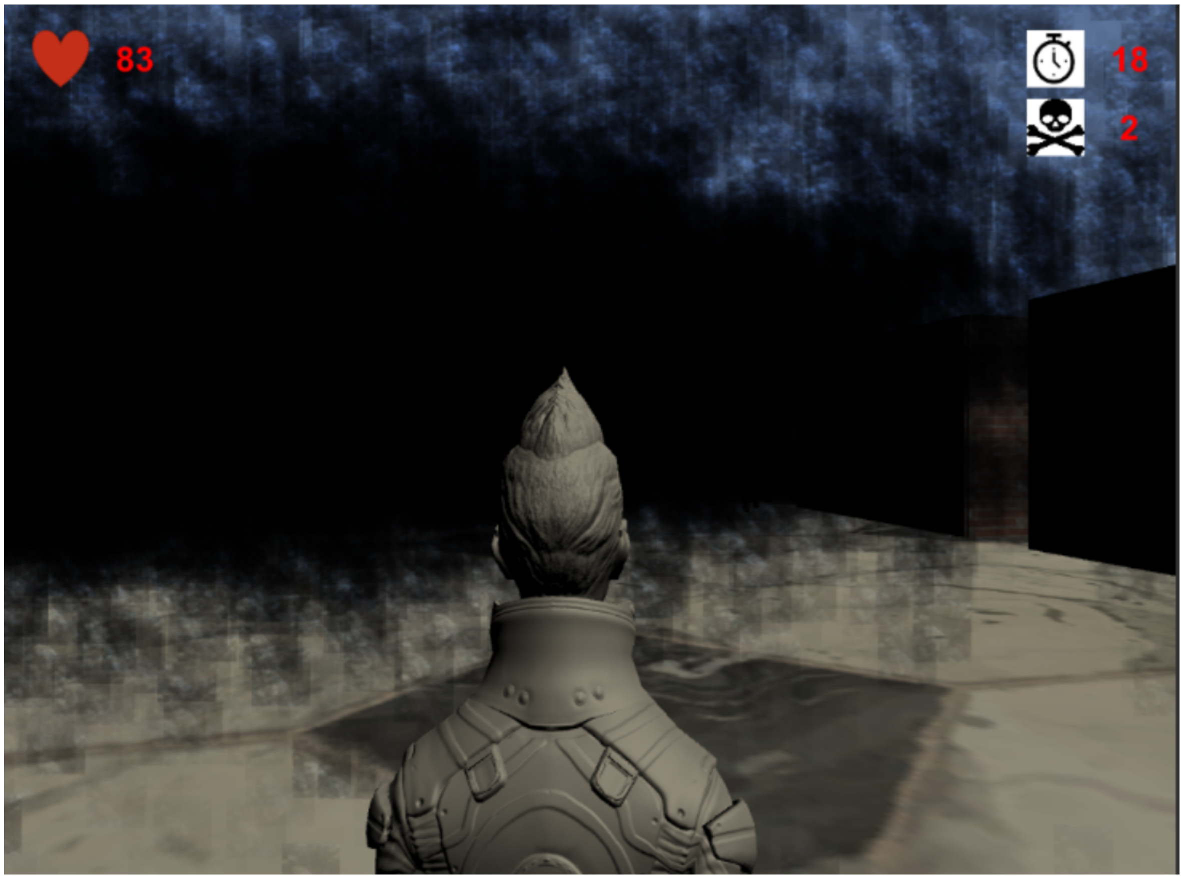


Figure 4.8: The look of the smoke from the player's view in the game.

4.5 Chapter Summary

Serious Game for Fire Simulation uses *FDS* to perform the fire simulation. The fire scenario is first translated and written into an *FDS* input file, and then the input file is run by *FDS*.

There is a trade-off between simulation accuracy and the game run time performance. It takes more computational power to render simulations that are more accurate. For the purpose of this game, the simulation is done with a 1 m mesh size and 1 s time step, which results in a satisfactory accuracy and game run time experience.

For the moment, the soot density data and the FED value data are extracted from the simulation. The soot density data is used directly for the visualization of smoke in the game. The particle system in the game engine Unity is used to visualize the smoke. The particle system is essentially an animation tool, which makes the smoke visualization look smoother, compensating the coarseness of the simulation. FED value data is used by the fire-to-human interaction part of the pedestrian model.

In the field of pedestrian modeling, the two-way coupling of fire and pedestrian has not been accomplished. In this game, fire affects pedestrians, but not the other way around.

Chapter 5

Pedestrian Modeling Implementation

In this chapter, the implementation of a pedestrian model will be introduced. This part of the implementation includes the one-circle representation of agents, route selection algorithm, agent-to-agent interaction algorithm, fire-to-agent interaction algorithm, and tricks to enhance the smoothness of the game.

The project mainly focused on the modified social-force model, see section 2.2 for the theoretical background. Here's a brief review of the model. The equation governing the movement of each agent is

$$m_i \frac{d\mathbf{x}_i}{dt^2} = \mathbf{f}_i(t) + \xi_i(t), \tag{5.1}$$

where $\mathbf{f}_i(t)$ is expressed as

$$\mathbf{f}_i(t) = \frac{m_i(v_i^0(t)\mathbf{e}_i^0(t) - \mathbf{v}_i(t))}{\tau_i} + \sum_{j(\neq i)} [\mathbf{f}_{ij}^{soc}(t) + \mathbf{f}_{ij}^c] + \sum_b [\mathbf{f}_{ib}^{soc}(t) + \mathbf{f}_{ib}^c]. \tag{5.2}$$

Note that the attractive forces $f_{ij}^{att}, f_{ik}^{att}$ in the original equation 2.5 are omitted due to their vagueness. Therefore, the task was to implement all the other forces. The implementation of the pedestrian model in this game mostly followed the setup of the pedestrian model used in *FDS+Evac*, and thus used the same parameter values in the equation.

In 5.2, recall that $\frac{m_i(v_i^0(t)\mathbf{e}_i^0(t) - \mathbf{v}_i(t))}{\tau_i}$ represents the motivation force, which leads agent i towards an exit. τ_i is the relaxation time parameter, which is set to be uniformly distributed from $0.8s$ to $1.2s$. For simplicity, τ_i was set to be the average value, 1 s.

Recall that the agent-to-agent social force,, is expressed as:

$$\mathbf{f}_{ij}^{soc}(t) = A_i e^{(r_{ij} - d_{ij})/B_i} \mathbf{n}_{ij} [\lambda_i + (1 - \lambda_i) \frac{1 + \cos(\varphi_{ij})}{2}], \tag{5.3}$$

where A_i and B_i describe the strength and spatial extent of the force, respectively. $A_i = 2000$ N and $B_i = 0.04$ m were used in this game. The parameter λ_i controls anisotropy of the social force, meaning that if $\lambda_i = 1$, the force is symmetric, and if $0 < \lambda_i < 1$, the force is larger in front of the agent than behind. $\lambda_i = 0.5$ was used in this game. For the boundary-to-agent social force f_{ib}^c, $A_b = 2000$ N, $B_b = 0.08$ m, and $\lambda_b = 0.2$ were used.

Recall that the contact force is expressed as:

$$\mathbf{f}_{ij}^c = [k_{ij}(d_{ij} - r_{ij}) + c_d \Delta v_{ij}^n]\mathbf{n}_{ij} + \kappa_{ij}(d_{ij} - r_{ij})\Delta v_{ij}^t \mathbf{t}_{ij}, \tag{5.4}$$

where the force constant k_{ij} represents the radial elastic force strength, the force constant κ_{ij} represents the frictional force strength, and c_d is a damping parameter of the physical damping force. Values $k_{ij} = 12 \times 10^4$ kg m^{-2}, $\kappa_{ij} = 4 \times 10^4$ kg s^{-1}m^{-1}, and $c_d = 500$ kg s^{-1} were used for both agent-to-agent and boundary-to-agent interactions.

The rotation aspect of the modified social-force model was omitted for simplicity. An agent will always face the direction of the path that leads it to the closest exit, which will be discussed more in subsection 5.3.

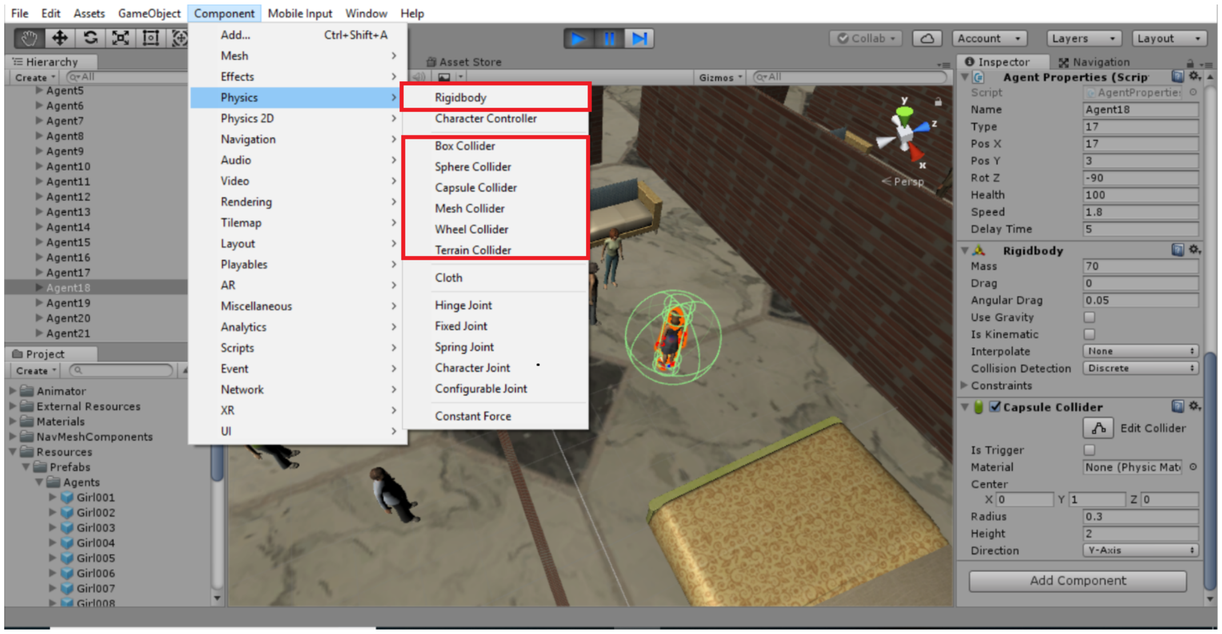
5.1 Useful Unity Built-in Tools

Physics system

Unity provides a very powerful physics system under which the object's motion can be governed by Newtonian forces, namely by the classic force equation $f = ma$. As shown in Figure 5.1, Unity provides a lot of interesting and powerful tools under the physics system. The two that this project had utilized were the rigid body component and the collider component.

The rigid body component is the most fundamental component that an object must have to use the physics system. The rigid body component allows the user to specify mass and some other physics properties to an object. In this project, all the obstacle object were made static, so there was no need to assign them rigid bodies. For each agent, a rigid body was assigned, and its mass was set to be 70 kilograms by default. For simplicity, the mass of an agent is not accessible in the design phase of the game, but the feature of changing the mass of an agent could be easily added in the future.

Once an agent possesses a rigid body, forces of pedestrian models could be applied on it through scripts. This game mainly used the physics system to simulate the interaction forces, *i.e.* the agent-to-agent interaction force and the obstacle-to-agent interaction force.



Agent Properties (Script)
Script AgentProperties
Name Agent18
Type 17
Pos X 17
Pos Y 3
Rot Z -90
Health 100
Speed 1.8
Delay Time 5

Rigidbody
Mass 70
Drag 0
Angular Drag 0.05
Use Gravity ☐
Is Kinematic ☐
Interpolate None
Collision Detection Discrete
Constraints

Capsule Collider
Edit Collider
Is Trigger ☐
Material None (Physic Mat
Center X 0 Y 1 Z 0
Radius 0.3
Height 2
Direction Y-Axis

Figure 5.1: The built-in physics system of Unity.

The collider component is essential for obstacle and agent avoidance. With the collider component, an agent is not allowed to go through other agents and obstacles. From a relatively large distance, agents interaction is governed by forces of pedestrian models. Ideally, once two agents are relatively closed to each other, both the contact force and the colliders are effective to take care of the interaction of the two agents. However, to enhance the computational performance of the game, when an area contains too many agents, which means congestion has formed, then the contact force script is turned off and only the collider is effective to take care of the bumping between every two agents. This implementation significantly eases the calculation and yields a result not much different from that of also considering the contact force effect.

Navigation System

Unity also provides a very useful navigation system, Figure 5.2. The navigation system allows agents to navigate in the game world. Using the navigation system, agents can go to target positions, choosing the shortest path and walking around obstacles. Note that the target position does not need to be fixed; it can be altered during the run time of the game.

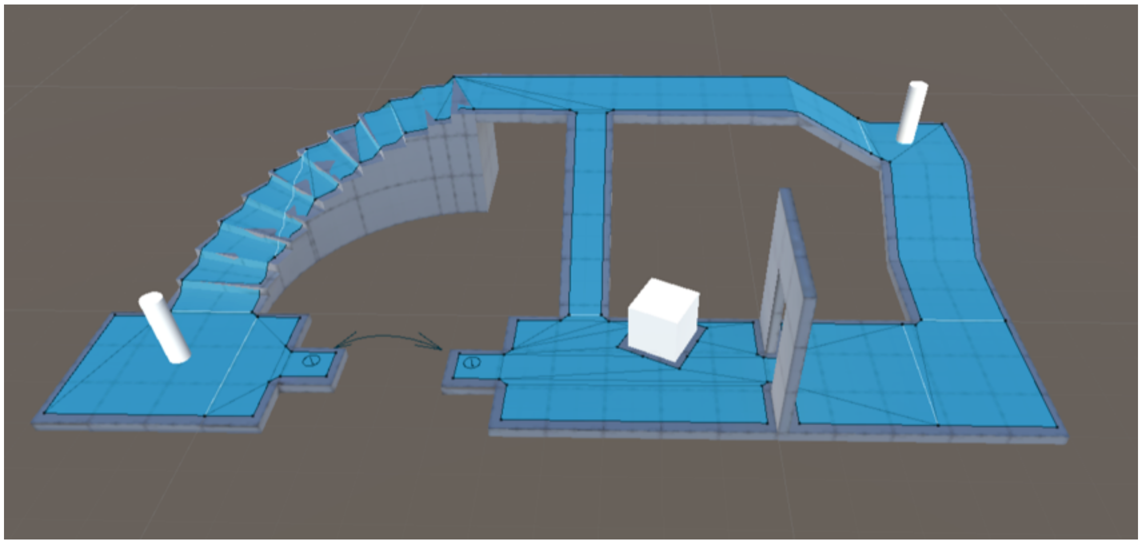
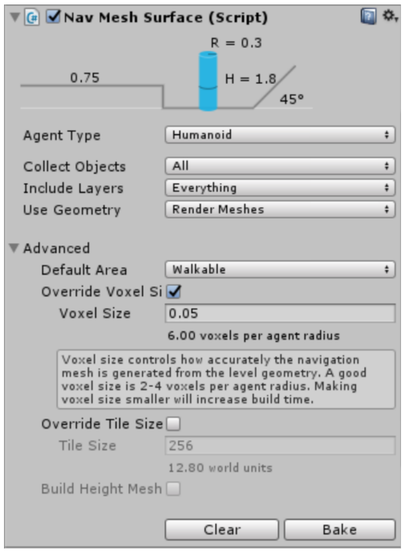


Figure 5.2: Navigation system of Unity (source: https://docs.unity3d.com/Manual/Navigation.html).

In order to use the navigation system, one needs to assign the *Navigation Mesh Surface* component (script), as shown in Figure 5.3, to surfaces that agents are allowed to walk on. At the beginning of the game, meshes would be created on these surfaces, and these meshes would be used by the navigation algorithm to control the movement of agents. Note that no mesh is created on places covered by obstacles.

One also needs to assign the *Navigation Mesh Agent* component, as shown in Figure 5.4, to agents that are expected to be controlled by the navigation algorithm. Once the surface meshes are available, these agents would be controlled by the navigation algorithm to move to the target destination.

A brief introduction of the navigation system algorithm will be given in the exit selection algorithm section. A more detailed documentation of the navigation system can be found in [33].



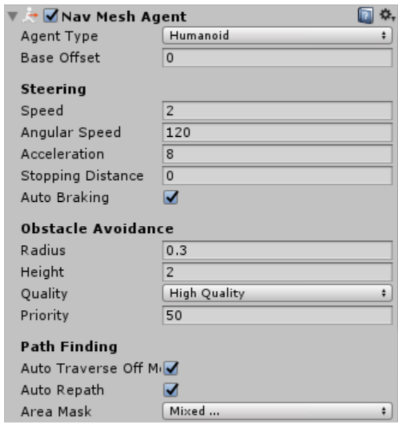
Nav Mesh Surface (Script)

Agent Type	Humanoid
Collect Objects	All
Include Layers	Everything
Use Geometry	Render Meshes
Advanced	
Default Area	Walkable
Override Voxel Si	✓
Voxel Size	0.05
	6.00 voxels per agent radius
Override Tile Size	
Tile Size	256
	12.80 world units
Build Height Mesh	

Voxel size controls how accurately the navigation mesh is generated from the level geometry. A good voxel size is 2-4 voxels per agent radius. Making voxel size smaller will increase build time.

Clear Bake

Figure 5.3: Navigation mesh surface component (script).



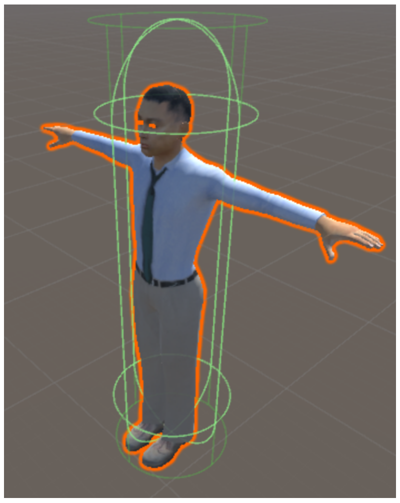
Nav Mesh Agent

Agent Type	Humanoid
Base Offset	0
Steering	
Speed	2
Angular Speed	120
Acceleration	8
Stopping Distance	0
Auto Braking	✓
Obstacle Avoidance	
Radius	0.3
Height	2
Quality	High Quality
Priority	50
Path Finding	
Auto Traverse Off M	✓
Auto Repath	✓
Area Mask	Mixed ...

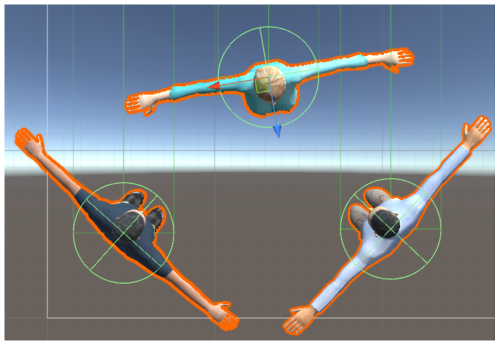
Figure 5.4: Navigation mesh agent component (script).

5.2 Circle Representation

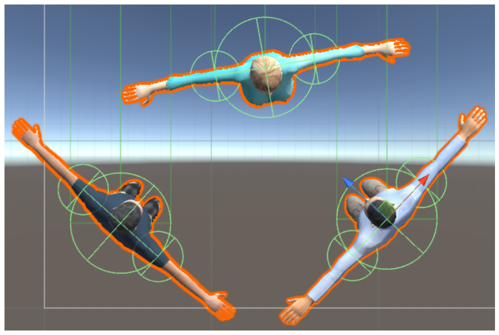
Unity provides built-in colliders in the physics system. Objects with collider cannot go through each other in the game; in other words, objects collide when in contact. Colliders come in different shapes, the collider used in this game is the capsule collider as shown in Figure 5.5a. Here the one-circle-representation is equivalent to using one capsule collider to represent an agent's physical body. Figure 5.5b shows the top view of the one-circle-



(a) Capsule collider.



(b) One-circle-representation using one capsule collider.

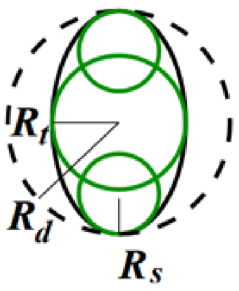


(c) Three-circle-representation using three capsule colliders.

Figure 5.5: Using collider for circle representation of human body.

representation of an agent. Roughly based on Table 5.1, for simplicity, the capsule radius is set to be 0.255 m for all types of agents including the player.

The three-circle-representation has not been fully implemented, due to the complication it brings to the pedestrian model relating to the contact force. If we only consider he physical shape, then the three-circle-representation can be implemented using three capsules of different radii as shown in Figure 5.5c. One capsule represents the torso, two other capsules represent the shoulder, and the radii of the three capsules can be obtained from Table 5.1.



Body type	R_d (m)	R_t/R_d (-)	R_s/R_d (-)	d_s/R_d (-)	Speed (m/s)
Adult	0.255±0.035	0.5882	0.3725	0.6275	1.25±0.30
Male	0.270±0.020	0.5926	0.3704	0.6296	1.35±0.20
Female	0.240±0.020	0.5833	0.3750	0.6250	1.15±0.20
Child	0.210±0.015	0.5714	0.3333	0.6667	0.90±0.30
Elderly	0.250±0.020	0.6000	0.3600	0.6400	0.80±0.30

Table 5.1: Properties of different types of pedestrian.

5.3 Exit Selection Algorithm

In order to use the modified social-force model, one has first to determine the desired walking velocity $\mathbf{v}_i^0 = v_i^0\mathbf{e}_i^0$. An agent's desired walking speed v_i^0 is set in the design phase of the game; data in Table 5.1 is a good reference for designers to set the speed. The main challenge lies in finding an appropriate moving direction $\mathbf{e}_i^0$.

As discussed in subsection 2.2.4, there are several ways to determine the desired walking velocity $\mathbf{v_i^0}$ of agent i, which are the fluid algorithm, the game theory algorithm, and the closest exit algorithm. So far, the closest exit algorithm has been implemented into the game.

The Unity built-in navigation system is used to implement the closest exit algorithm. The route for an agent to go to an exit is a polygon. The navigation system is efficient to find the shortest polygon which allows the agent to walk around all the obstacles on its way. The polygon generated for an agent is shown in Figure 5.6. The distance between an agent and an exit is calculated as the sum of the lengths of the segments of the polygon.

At each frame, a route polygon of such will be generated for each available exit. The distance of each route (the length of the polygon) will then be calculated and the closest exit can be found. The desired route of agent i is the route polygon that leads it to the closest exit. Therefore, the desired walking direction $\mathbf{e}_i^0$ is the direction of the segment of the shortest polygon that agent i is on at that frame. Once the desired walking direction $\mathbf{e}_i^0$ is determined, together with the predetermined walking speed v_i^0, the motivation force $\frac{m_i(v_i^0(t)\mathbf{e}_i^0(t)-\mathbf{v}_i(t))}{\tau_i}$ can be calculated.

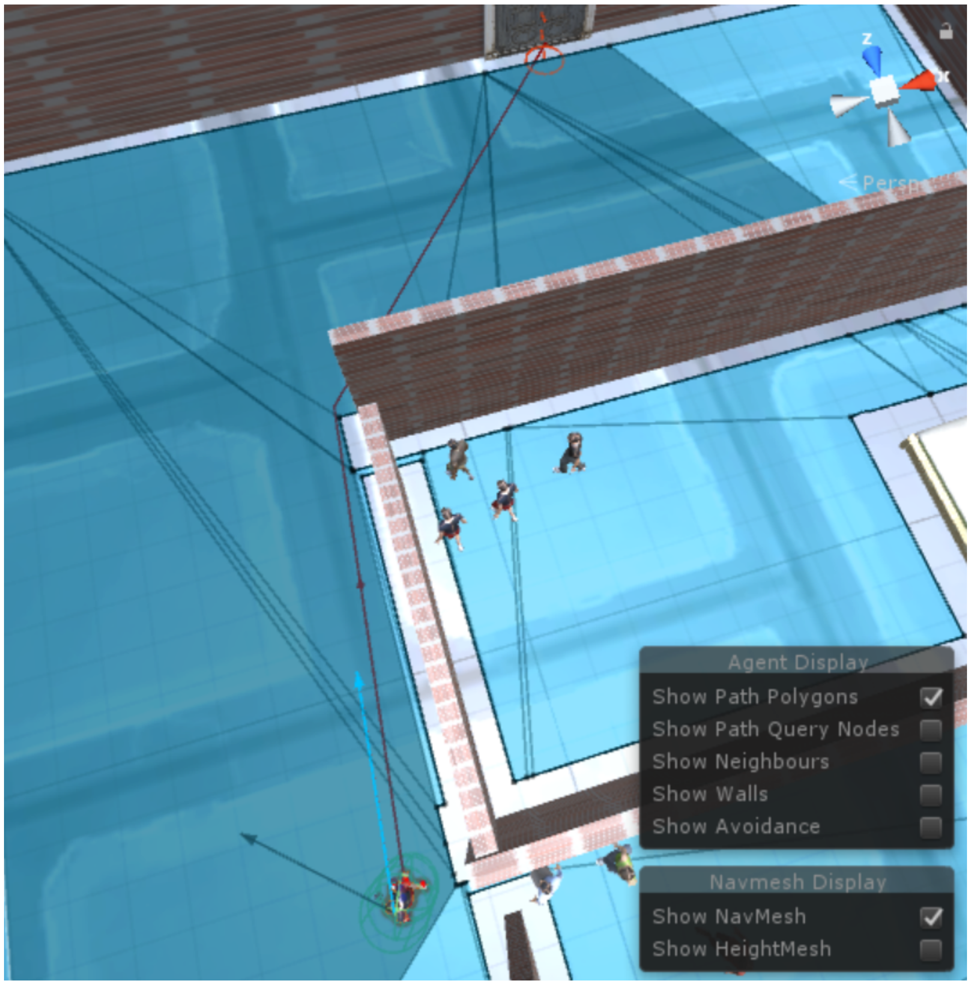


Figure 5.6: Navigation algorithm generates a polygon that connects the agent to its destination.

5.4 Agent-to-agent and Obstacle-to-agent Interaction

Agent i detects all the other agents and obstacles within 5 meters. The calculation of the social force and the contact force from each of the other agents and obstacles is done based on their positions. Note that the force exerted from another agent or obstacle is negligibly small if it is far away from agent i. Therefore, by only considering the interactions between an agent and its surroundings within a certain range, the calculation of can be significantly sped up without causing any noticeable difference. For this reason, an active range has

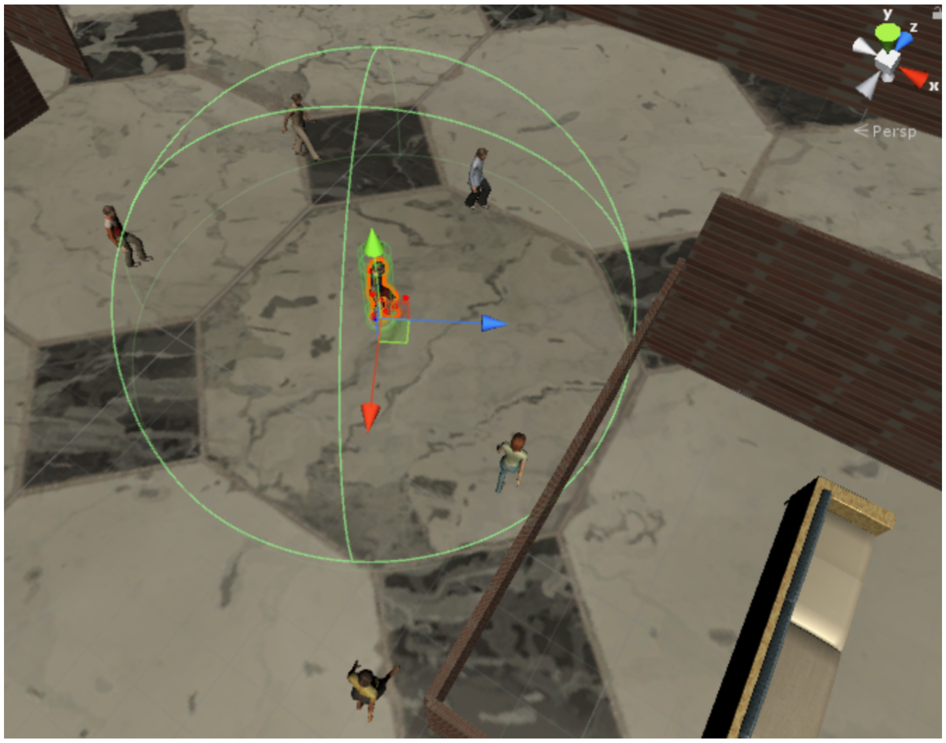


Figure 5.7: This figure shows the active range implementation. The focused agent only reacts to other agents and obstacles within the circle.

been implemented. The active range was set to be 5 meters, which is adjustable. Forces exerted from agents and obstacles from beyond 5 meters are negligible. For example, as shown in Figure 5.7, the agent focused, namely agent i, is only affected by the two other agents and the wall within its active range. The three agents, all the other walls, and the couch make no impact since they are out of the active range.

Figure 5.8 shows part of modified social-force model script that calculates the social force exerted from other agents and obstacles. All the other forces and other kinds of interactions are scripted in the same manner. These forces can be applied to an agent through the physics system of Unity.

At each frame of the game, the calculation of forces and other interactions needs to be done for all the agents, so if there are too many active agents in the room, the game will stutter. After testing, it is suggested that the room should contain no more than five hundred agents. The number may vary for different computer systems. Stuttering of the game can sometimes occur when too many agents are packed within the active range. A

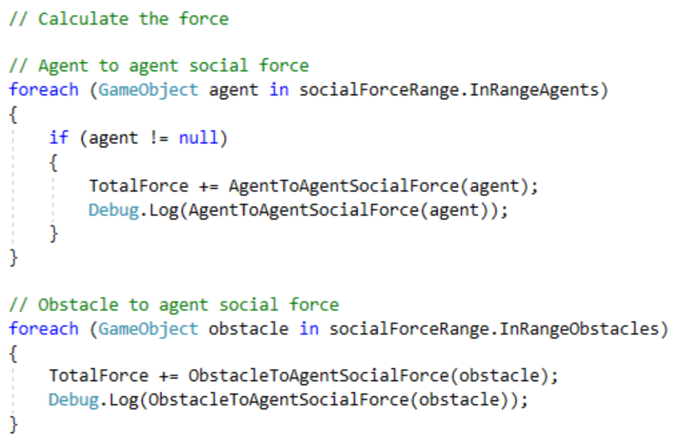

```
// Calculate the force

// Agent to agent social force
foreach (GameObject agent in socialForceRange.InRangeAgents)
{
    if (agent != null)
    {
        TotalForce += AgentToAgentSocialForce(agent);
        Debug.Log(AgentToAgentSocialForce(agent));
    }
}

// Obstacle to agent social force
foreach (GameObject obstacle in socialForceRange.InRangeObstacles)
{
    TotalForce += ObstacleToAgentSocialForce(obstacle);
    Debug.Log(ObstacleToAgentSocialForce(obstacle));
}
```

Figure 5.8: Partial script of the modified social-force model.

trick that solved this problem is introduced in section 5.6.

5.5 Fire-to-agent Interaction

The implementation of the fire-to-agent interaction was planned to follow the theoretical basis described in section 2.2.5. For the moment, the implementation of the toxic effect has been finished while the impeding effect is still under development.

Recall that an agent is incapacitated if the FED value it has exposed to accumulates up to unity. The FED profile for each mesh point can be extracted from the simulation result and put into the mesh object in Unity as mentioned in section 4.3. The time step of the simulation is 1 s, so the FED value is recorded every 1 s. To synchronize the change of the FED value, the toxic effect is applied on agents on a second-to-second basis as well.

The way this game accounts for the toxic effect goes as follows. In the design phase, one can specify an agent's health, and by default, an agent's health is set to be 100. At each second when the game is in the play interface, agent i's health will be deducted by $100 \cdot \text{FED}_i(x, y, t)$, where $\text{FED}_i(x, y, t)$ is the FED value read from the mesh object that agent i is in at time t. Note that the z location value of the mesh object is not considered. Only the meshes on the height level $z = 1.5$ m (from 1 m to 2 m) are used because this is where the agents' heads are. Also for this reason, only the mesh objects on the height level $z = 1.5$ m contain the FED value profiles.

Once an agent's health drops down to 0, it is considered incapacitated. During the

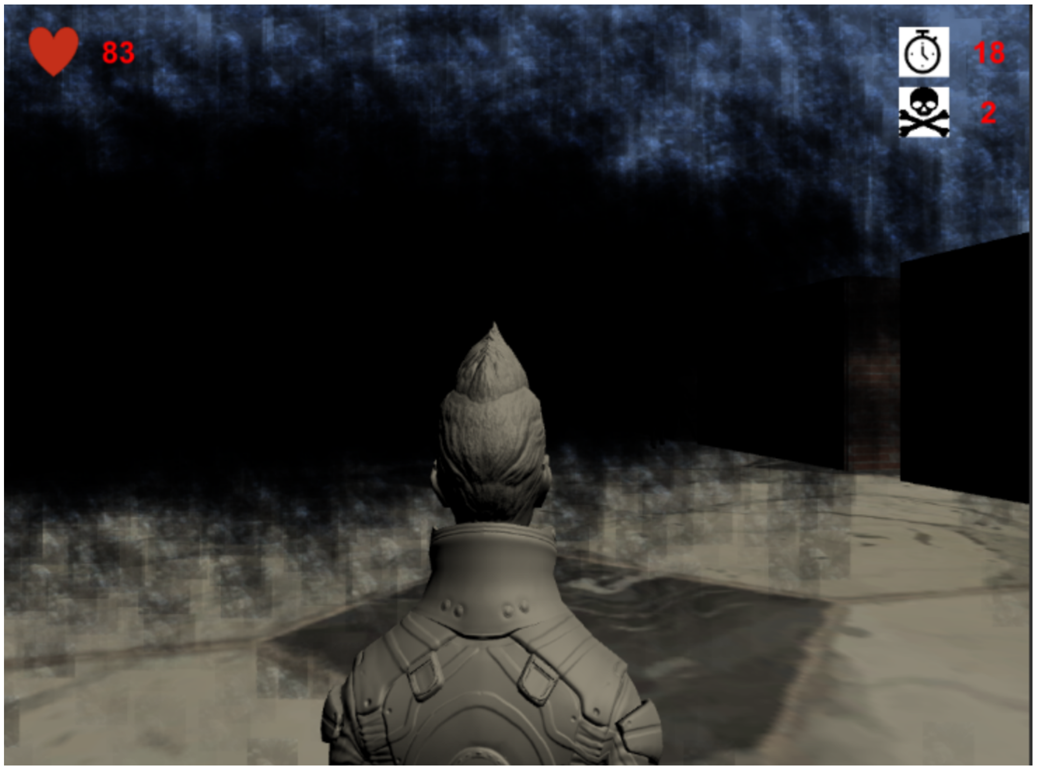


Figure 5.9: This is a screenshot of playing in the play interface in the example scenario. The left top corner shows the health of the player, the right top corner shows time elapsed and the casualties.

design phase, one can adjust how susceptible an agent is by changing its health value, with obviously a lower health indicating a higher susceptibility. When an agent is incapacitated, it can either be treated as a static obstacle which is no longer affected by forces exerted from the surrounding but still able to exert agent-to-agent force to other active agents; or simply be removed from the game, not leaving a physical carcass. For simplicity, in this game, when an agent is incapacitated, it is removed from the game.

The player character is also influenced by the toxic effect the same way as all other agents. If the player's health drops down to 0, the player fails, and the game ends. The player will then be given some feedback and given the options to either restart the game or quit. A game interface is shown in Figure 5.9. In the game, the top left corner shows the health of the player, and the top right corner shows the time elapsed and the casualties.

5.6 Enhancing Game Performance

One common measure of the smoothness of a game is its FPS (frames-per-second) rate during run time. An article online [34] gives a very concise introduction about FPS. FPS essentially measures the number of images a GPU (graphics processing unit) is able to render on a monitor each second. For example, if a game was running at 1 frame per second, the game is no much different from a slideshow, and the game is pretty much unplayable in this state.

FPS is rounded up to 30, 60, 120, and 240. Note that the stuttering feeling of a game is only noticeable at under 20 FPS, so anything above 20 FPS can be considered playable. Running at 30 FPS, a game is targeted to be played on consoles and lower-end PCs; a 30-FPS game is usually good for single player and for a more cinematic experience. 60 FPS is the target goal for most gaming PCs; on consoles, only the less demanding or better-optimized games can manage a stable 60 FPS. 60 FPS is what competitive multiplayer games should aim for as it can improve reaction time. 120 FPS is a framerate that is only possible on monitors with 144-165 Hz refresh rates, and it is only achievable for some powerful high-end gaming PCs. 120 FPS is not achievable by consoles, and it is only aimed by some more serious competitive gamers. 240 FPS is the very highest framerate only possible with the most advanced 240 Hz monitors and the most powerful hardware.

Although the main purpose of *Serious Game for Fire Evacuation* is training and education instead of competitive gaming, in order to give players a smooth game experience, it still aims to maintain an at least 30 FPS framerate during run time on most PCs. Ideally, updates need to be made at each frame for accurate fire and pedestrian simulations. However, updating at each frame significantly drops the framerate of the game, so a compromise updating rate needs to be found. For this reason, this game currently performs some necessary calculations and updates every certain period, irrespective of the number of frames have run by in the period. To be more specific, in the game, a smoke particle updates its opacity according to its density every one second; the intoxicating effect is applied on agents and the player every one second. The interaction forces and navigation changes on each agent are still updated every frame.

A trick has been implemented to enhance the smoothness of the game when congestion has occurred in the game. As shown in Figure 5.10, congestion has formed. When agent i detects that there are more than 20 (an adjustable threshold) other agents in its active range (a circle of 5 meters in radius), it knows there is a congestion around. Agent i will then turn on the congestion mode, in which it is no longer affected by the fictitious forces, the social force and the contact force, but it is still driven by the navigation system to move to

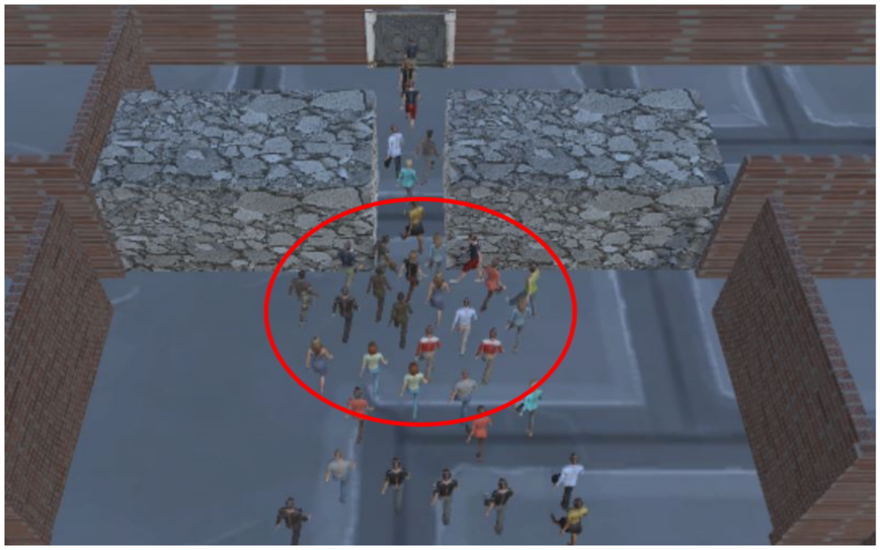


Figure 5.10: Congestion forms at the narrow passage.

the exit. The rigid body component of agents prevent them from going through each other, so in the congestion, agents are basically bumping and squeezing together to go through a narrow passage. The apparent benefit of the implementation of the congestion mode is that no calculation of pedestrian model forces needs to be performed, which significantly saves up the computational power. Moreover, this implementation does not cause a noticeable difference from the simulation that strictly follows the pedestrian model used.

5.7 Result Comparison

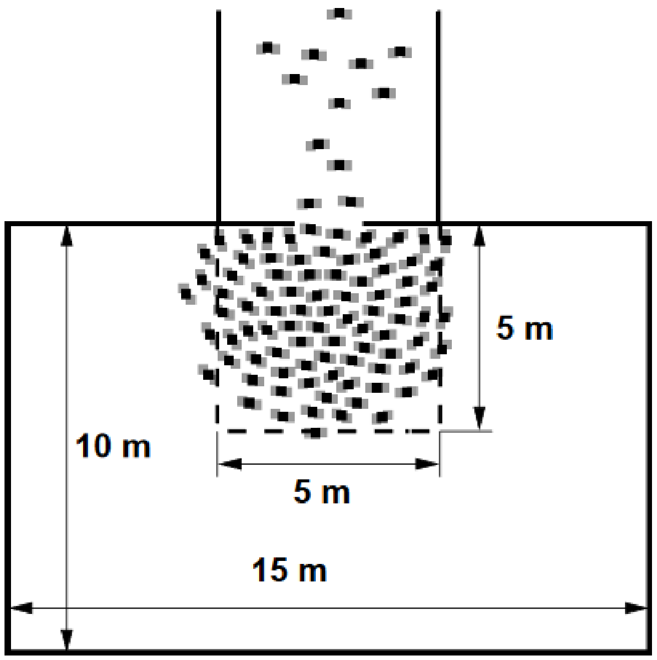
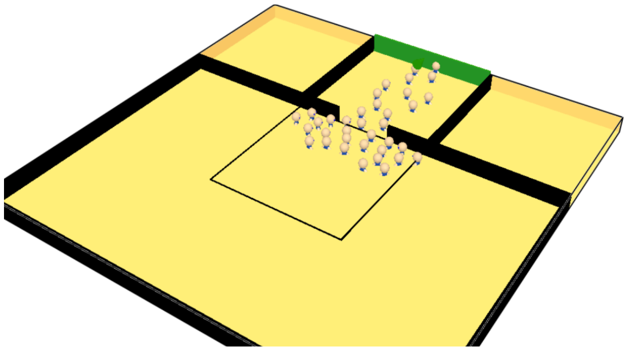


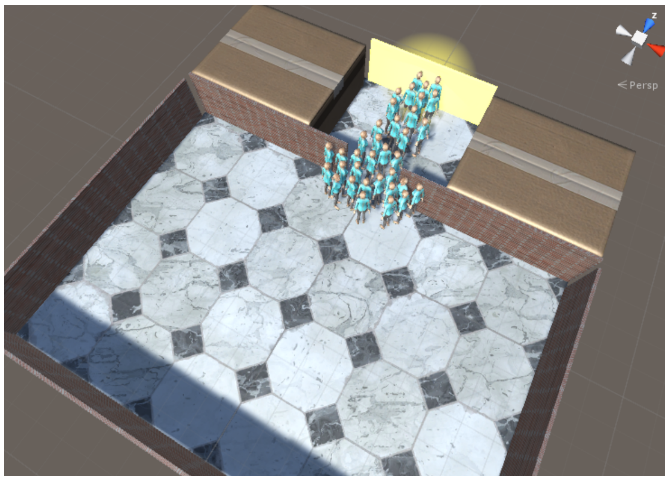
Figure 5.11: A test geometry used to verify the pedestrian model of *FDS+Evac*.

A test geometry was used to compare the pedestrian simulation results of the implementation of the game and *FDS+Evac*. The test geometry is demonstrated in Figure 5.11, there are 100 agents in the room. The simulated evacuation processes of *FDS+Evac* and the game are shown in Figure 5.12a and Figure 5.12b respectively. This test geometry was also used by FDS+Evac to verify their model, details about this test is in [19]. Note that the pre-determine time of agents have been changed to be 0.

The chart 5.13 shows the simulation data comparison. The x-axis is the time elapsed while the y-axis is the number of agents in the room. As expected, without the intervention of the player, the pedestrian model implemented in the game yielded a very close simulation as the simulation yielded by FDS+Evac for this test. However, this is not to say that the pedestrian model implemented in the game is good enough. More tests need to be conducted to verify other features of the implementation; for example, the route selection algorithm and the fire-to-human interaction implementation.



(a) Test simulation in *FDS+Evac*.



(b) Test simulation in *Serious Game for Fire Evacuation*.

Figure 5.12: Simulation comparison between *FDS+Evac* and *Serious Game for Fire Evacuation*.

The validity of the pedestrian model used in this game is questionable, so are the many other pedestrian models used in *FDS+Evac* and other third-party simulators. However, the validity of the pedestrian model is not of the primary concern for the moment, for this project mainly wanted to explore the feasibility of implementing and running pedestrian models in an interactive virtual game environment. In future development, pedestrian models that are more complicated can be implemented following the same methodology.

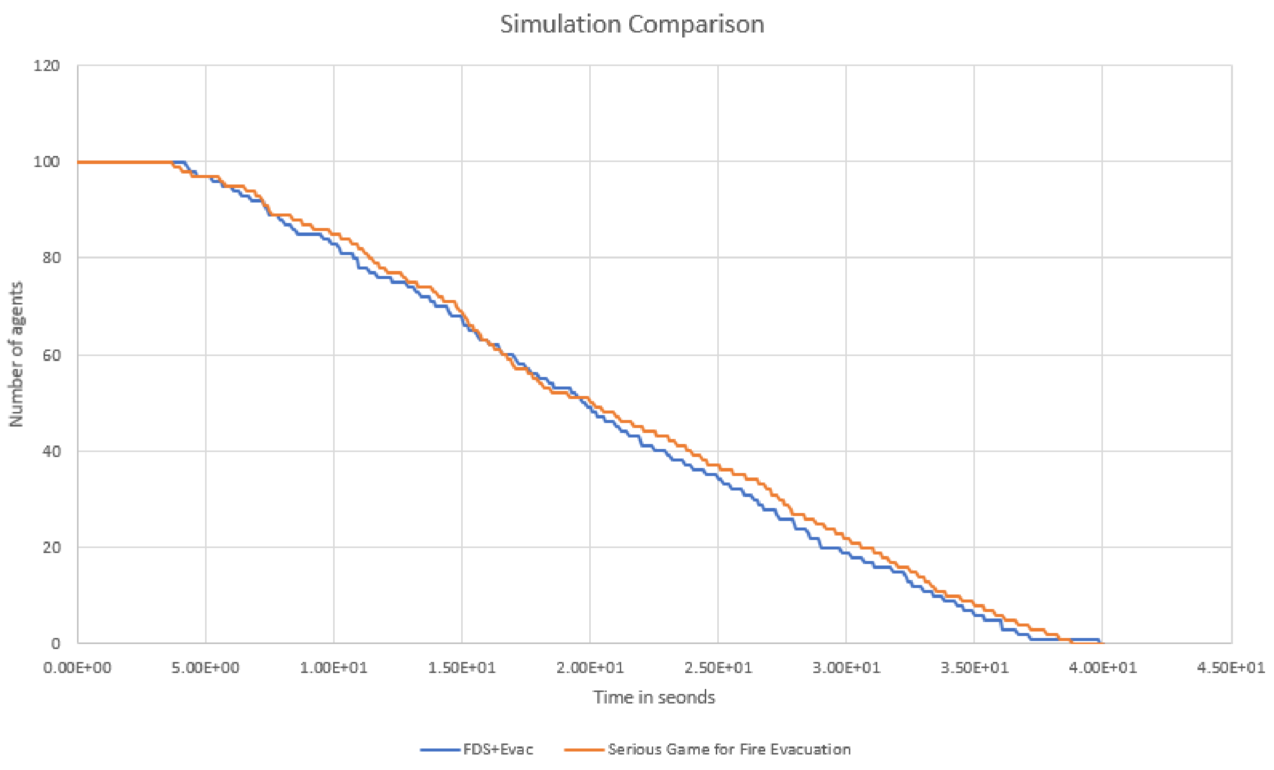


Figure 5.13: Simulation comparison.

Chapter 6

Concluding Summary

6.1 Conclusions

This project explored the feasibility of using a virtual game to replace conventional fire drills to train people about fire evacuation. Training in the virtual game environment has the advantages of rendering hazards, easy and cheap conduction, and being more engaging. The game developed in this project, *Serious Game for Fire Evacuation*, serves mainly for training and research purposes; therefore, it is regarded as a serious game. The game is able to provide a relatively realistic game experience, which shows that the idea of training people for fire evacuation in a virtual game is promising.

The game was designed to firstly provide a platform for designers to customize the fire evacuation scenario. Secondly, it provides a virtual environment that is set up based on the designed scenario for players to play in. Lastly, it gives feedback to help players to learn. The design interface and the play interface have been developed while the feedback interface has not. The feedback interface and some other instructive features need to be designed by people with expertise in fire drilling.

The game is built on the game engine Unity. To make the virtual environment realistic, the game incorporates fire simulation and pedestrian modeling. The fire simulation is based on fire dynamics, and is done by a third-party fire simulator, *FDS*. After the simulation is done, the result will be visualized in the game engine. Since the simulation is performed before the actual game starts, its result cannot be altered during run time, which means there is no human-to-fire interaction in the game.

A pedestrian model is directly implemented into the game through scripts, which control the movement of background agents. The pedestrian model implemented in the game is a simplified version of the social-force model [10, 11, 12, 13]. Since agents are controlled by scripts, they are interactive with the surroundings. The agents' states and actions will be constantly updated in the run time of the game. The game's pedestrian model yields a similar pedestrian simulation as the pedestrian simulation done by *FDS+Evac*, a third-party pedestrian simulator, for a simple fire evacuation scenario. However, more tests need to be done to verify the game's pedestrian model.

6.2 Future Work

- **Instructive features:** More instructive features need to be implemented into the game. To meet the goal of training, features like giving instruction to the player on what to do should be implemented. Also, the feedback interface needs to be designed. These instructive features need to be designed by people with expertise in fire drilling for they have the insight of what is crucial for people to do in fire evacuation.

- **Pedestrian models**: More complicated pedestrian models can be implemented and tested in this game. Testing a pedestrian model in the game tells how sensitive the model is to a character of genuine agency (the player).

- **Smoke visualization:** The graphic of the game can be improved by using some animation tools, for example *Blender*. After the simulation is done, one can first create an animation of the simulation, and then put the complete animation into the game. Doing this can ease the game from updating the smoke visualization in the run time, which can save up a tremendous amount of computational power and thus largely improve the game performance. In addition, doing this can potentially give a much better visualization.

- **Multi-player gaming:** For now, the game only allows one player to play in. In the future, the game can be developed to allow multiple players to play together. With less algorithm-controlled agents and more human players, the game will better reflect a real fire evacuation.

- **Artificial intelligence:** The movement of the player can be recorded as data. The data can be used to tweak the parameters in pedestrian models. To think out of the box, the data can be directly used to let agents mimic the player's course of actions instead of relying on some vague and fictitious models. This idea taps into the field of artificial intelligence and has been mentioned in [35].

- **Virtual reality and augmented reality:** The idea and methodology of bringing simulation into an interactive virtual environment can be incorporated in virtual reality and augmented reality as well. [36] has shown that using augmented reality in evacuation training is possible and promising.

References

[1] *Fire Safety Design in Buildings*. Canadian Wood Council, 1996.

[2] D. Helbing *et al. A Fluid Dynamic Model for the Movement of Pedestrians*. Complex Systems, volume 6: pp. 391-415, 1998.

[3] R.L. Hughes *The Flow of Human Crowds*. Annual Review of Fluid Mechanics 35: pp. 169-182, 2003.

[4] M. Moussad, *The Collective Dynamics of Human Crowd Motion: Where Physics Meets Cognitive Science*. 2011.

[5] S. Ali, M. Shah. *A Lagrangian Particle Dynamics Approach for Crowd Flow Segmentation and Stability Analysis*.

[6] *CityOne*, by IBM, 2010.

[7] *A Force More Powerful: The Game of Nonviolent Strategy*, by the International Center on Nonviolent Conflict (ICNC) and York Zimmerman Inc., 2006.

[8] *Ship Simulator*, by VSTEP, 2006.

[9] Frederick Joseph F.. *Defining Serious Games*. 2016.

[10] D. Helbing, P. Molnr. *Social Force Model for Pedestrian Dynamics*. Physical Review E51: pp. 4282-4286, 1995.

[11] D. Helbing, I. Farkas, T. Vicsek. *Simulating Dynamical Features of Escape Panic*. Nature 407: pp. 487-490, 2000.

[12] D. Helbing, I. Farkas, P. Molnr, T. Vicsek. *Simulating of Pedestrian Crowds in Normal and Evacuation Situations*. Pedestrian and Evacuation Dynamics, Schreckenberg, M. and Sharma, S.D.(eds.), Springer, Berlin, pp. 2158, 2002.

[13] T. Werner, D. Helbing. *The Social Force Pedestrian Model Applied to Real Life Scenarios*. Pedestrian and Evacuation Dynamics Proceedings of the Second International Conference, University of Greenwich, London, pp. 17-26, 2006.

[14] P.A. Langston, R. Masling, B.N. Asmar. *Crowd Dynamics Discrete Element Multi-circle model*. Safety Science 44: pp. 395-417, 2006.

[15] T. Korhonen, S. Hostikka, S. Helivaara, H. Ehtamo, H., K. Matikainen. *Integration of an Agent Based Evacuation Simulation and the State-of-the-Art Fire Simulation*. Proceedings of the 7th Asia-Oceania Symposium on Fire Science and Technology, 2007.

[16] T. Korhonen, S. Hostikka, S. Helivaara, H. Ehtamo, K. Matikainen. *FDS+Evac: Evacuation Module for Fire Dynamics Simulator*. Proceedings of the Interflam: 11th International Conference on Fire Science and Engineering, Interscience Communications Limited, London, UK, pp. 1443-1448, 2007.

[17] T. Korhonen, S. Hostikka, S. Helivaara, H. Ehtamo. *FDS+Evac: Modelling Social Inter-actions in Fire Evacuation*. Proceedings of the 7th International Conference on Performance-Based Codes and Fire Safety Design Methods, Auckland, New Zealand, 2008.

[18] T. Korhonen, S. Hostikka, S. Helivaara, H. Ehtamo. *FDS+Evac: An Agent Based Fire Evacuation Model*. Proceedings of the 4th International Conference on Pedestrian and Evacuation Dynamics, Berlin, Heidelberg, 2008.

[19] T. Korhonen. *Fire Dynamics Simulator with Evacuation: FDS+Evac Technical Reference and Users Guide*. 2018.

[20] K. McGrattan, S. Hostikka, R. McDermott, J. Floyd, C. Weinschenk, K. Overholt *Fire Dynamics Simulator Users Guide*. 2013.

[21] H. Ehtamo, S. Heliovaara, T. Korhonen, S. Hostikka. *Game Theoretic Best-Response Dynamics for Evacuees Exit Selection*. 2010.

[22] S. Bouzat, M. N. Kuperman. *Game Theory in Models of Pedestrian Room Evacuation*. 2013.

[23] T. Kretzp, A. Grobep, S. Hengstp, L. Kautzschp, A. Pohlmannp, P. Vortisch. *Quickest Paths in Simulations of Pedestrians*. 2011.

[24] H. Frantzich, D. Nilsson. *Evacuation in Dense Smoke: Behaviour and Movement*. 2003.

[25] D. A. Purser. *Toxicity Assessment of Combustion Products.* in SFPE Handbook of Fire Protection Engineering, 3rd ed., pp. 2/83-2/171, 2003.

[26] M. Spearpoint. *The Effect of Pre-evacuation on Evacuation Times in the Simulex Model.* Journal of Fire Protection Engineering 14(1): pp. 33-53, 2004.

[27] J.D. Sime. *Crowd Psychology and Engineering.* Safety Science, Volume 21, Issue 1, 1995.

[28] S. M. V. Gwynne, E. Kuligowski, M. Spearpoint, E. Ronchi *Bounding Defaults in Egress Models.* Safety Science, Volume 21, Issue 1, 1995.

[29] G.P. Forney. *Smokeview, A Tool for Visualizing Fire Dynamics Simulation Data — Volume I: Users Guide.* 2019.

[30] G.W. Mulholland. *SFPE Handbook of Fire Protection Engineering, chapter Smoke Production and Properties.*

[31] Unity User Manual: Particle Systems,
`https://docs.unity3d.com/Manual/ParticleSystems.html`

[32] Blender Website:
`https://www.blender.org/`

[33] Unity User Manual: Navigation and Pathfinding,
`https://docs.unity3d.com/Manual/Navigation.html`

[34] S. Stewart. *What Is The Best FPS For Gaming?.*
`https://www.gamingscan.com/best-fps-gaming/`

[35] Y. Ma, E.W.M. Lee, R.K.K Yuen. *An Artificial Intelligence-Based Approach for Simulating Pedestrian Movement.* IEEE Transactions on Intelligent Transportation Systems 17(11): pp. 1-12, 2016.

[36] H. Mitsuhara, M. ShishiboriJunya, K. Iguchi. *Game-Based Evacuation Drills Using Simple Augmented Reality.* IEEE 16th International Conference on Advanced Learning Technologies, 2016.