AlphaStar: Considerations and Human-like Constraints for Deep Learning Game Interfaces

by

David Choi

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2020

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

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

Statement of Contributions

All chapters of this thesis were written by myself, and not for previous publications, except for text contained within figures and tables and their captions.

This thesis is based on research conducted at DeepMind that led to AlphaStar [92], the authors of which are Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander S. Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom L. Paine, Caglar Gulcerhe, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver.

I am a co-first author on that publication. Aspects I worked on include designing and implementing the architecture, dataset, multi-agent reinforcement learning algorithms, evaluation, and infrastructure. I also wrote or co-wrote significant portions of the publication.

Chapter 3, Chapter 6, and Sections 2.3, 2.4.1, and 2.4.2 of Chapter 2 draw on the aforementioned publication "Grandmaster level in StarCraft II using multi-agent reinforcement learning" [92], but include additional details, motivations, and explanations. Figures and tables are adapted from that publication where indicated. Section 3.7.2 of Chapter 3 is an exception; though it is based on that research, it is an unpublished additional evaluation done afterwards.

The interface design was likely the aspect that I was most involved in, and so forms the other half of the thesis. Chapters 4 and 5 have not been previously published. Along with Chapter 6 they include my analysis, influenced by informal discussions with the AlphaStar co-authors on interface design.

Abstract

Games have historically been a fruitful area for artificial intelligence (AI) research, and StarCraft in particular has been an important grand challenge because of its strategic complexity, multi-agent dynamics, partial observability, large action spaces, delayed rewards, and robust human competitive scene. These complexities mean that approaches common in other game AIs, like Monte-Carlo Tree Search in Go or searching over the action space in Atari, cannot be easily applied to StarCraft. Thus, though there has been significant research, many approaches use handcrafted systems and no approach is competitive with even strong casual players.

In this thesis, we go into detail on AlphaStar, the first AI system to reach the highest tier of human performance in a widely professionally played esport. AlphaStar combines new and existing approaches in imitation learning, reinforcement learning, and multi-agent learning at scale in a general agent with minimal handcrafting. AlphaStar reached a rating above 99.8% of active ranked human players.

In particular, designing an effective interface is an essential component of AI research in games that has historically been under-explored. This thesis lists principles for designing effective interfaces and human-like constraints for deep learning research in games, and explores those principles with AlphaStar as a case study. Though the agent has minimal handcrafting, it needs to interact with the game through an interface that is human-like, expressive enough to capture the game's complexities, and amenable to deep learning in order to produce transferable research insights.

Acknowledgements

I would like to thank my supervisor Prof. Jesse Hoey for his endless help, support, and patience through the long Master's process, and for allowing me complete freedom in my research direction. Thanks to Prof. Peter van Beek and Pascal Poupart for taking the time to read and give feedback on my thesis. Thank you to the members of the CHIL group and other students for making the University of Waterloo such an enjoyable and memorable experience.

I would also like to thank all my co-authors for working with me on the amazing experience that was AlphaStar, and in particular Igor Babuschkin and Oriol Vinyals for reviewing this thesis. Thank you to Blizzard, and in particular Austin Hudelson, Chris Lee, Kevin Calderone, and Tim Morton, for working to make StarCraft a research environment and working with us throughout the process. Thanks to the entire DeepMind team for being a stream of assistance and ideas, and for providing an energizing environment for research.

Finally, thank you to my family and friends for their constant support and encouragement, in this thesis and beyond.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Games have been a core domain for artificial intelligence (AI) research from its inception. In 1959 AI systems could play checkers [72], and decades later computers started to match or exceed top human players in games like Backgammon [86], chess [14], Jeopardy [30], Go [77], and now StarCraft II [92]. Yet, superhuman performance in games is not a goal in and of itself. These domains are a stepping stone to complex, real-world interactions because they provide a way to make progress on key AI problems in simplified environments without dealing with the myriad engineering and research challenges involved in interacting with the real world (from building a robot, to physical performance variations and limited learning time [56]).

That is, games capture discrete problems humans find meaningful and interesting, as otherwise they would not have been made. However, for insights from them to be transferable and meaningful, it's important to approach game interfaces in the correct way. For turn-based board games, this means following the rules and not playing, say, a variant of chess where every piece can move anywhere on the board. Even for chess there are more complexities though. Playing with no time limits turns chess into a much simpler brute-force search problem, while playing with time limits of one microsecond makes it a program optimization problem with minimal opportunity for planning. Giving the chess-playing system access to expert-crafted observations can improve performance [14], but it specializes the approach to a specific game and means that applying the system to a different domain where such observations are not available becomes significantly harder. In modern video-games, designing an interface only becomes more complex as human interaction with the game becomes increasingly complicated. Alterations from how humans interact with a given game distort the game itself, and therefore can shift the focus away from what initially made it of interest to AI.

The game that this thesis focuses on is StarCraft, a domain that involves strategic depth, large state spaces, planning across thousands of timesteps, imperfect information, and a structured action space [93], all properties shared with many other applications. It has a thriving professional esports community with hundreds of tournaments every year, yet has not yielded AI systems of comparable skill to humans despite significant research interest [29].

The main contributions of this thesis are:

1. AlphaStar, the first agent to reach the highest tier of human performance in a widely professionally played esport (StarCraft II) [92]. AlphaStar uses a complex array of new and existing state-of-the-art techniques within deep multi-agent reinforcement learning. The author previously published AlphaStar in collaboration with many others at DeepMind [92].

2. A novel examination of considerations for designing a game interface for fruitful AI deep learning research, with our design of the interface for AlphaStar as a case study.

1.1 Outline

Chapter 2 will describe the background and prior work in StarCraft, deep learning, and reinforcement learning relevant for understanding this thesis.

Chapter 3 gives a description of the entire AlphaStar system and how its various components work together, along with results and analysis.

Chapter 4 presents the considerations and complexities of developing an interface for a complex game like StarCraft II, and discusses how to develop interfaces suitable for learning deep neural networks.

Chapter 5 introduces fairness in games, both various metrics of what it means to be fair and why fairness is an important proxy for researching the correct problems.

Chapter 6 looks more in depth into the interface of AlphaStar, and how it was designed in light of the previous chapters.

Finally, Chapter 7 presents the conclusions of this thesis and possible future work.

Chapter 2

Background and Related Work

2.1 Deep Learning

Within the last decade, deep learning (DL) has arisen as a dominant field within machine learning and AI research as a whole [51]. It has gained great popularity in domains such as image recognition [49], speech recognition [35], recommendation systems [17], healthcare [28], and finance [25]. Other research areas, like sociology [62], and law [15], have adopted deep learning techniques. A full description of deep learning is far beyond the scope of this thesis, and in fact may not be relevant for understanding many sections which are layers of abstraction removed from the underlying mathematics. Therefore, we will only present deep learning, and reinforcement learning, at a high-level.

Fundamentally, deep learning views a complex agent as a graph (called a neural network), where each node of the graph is a real valued number. These nodes are grouped into layers, where each node is a function of the nodes in all previous layers (e.g. the dot product of all nodes in the previous two layers by some vector), the nodes on the first layer are set to observations of the network (e.g. the red, green, and blue (RGB) values of an image), and the nodes on the last layer are the outputs of the agent (e.g. an action the agent will take, encoded as a real number). Some of the functions used to generate the values of each node have trainable parameters (the vector that the previous two layers are taken the dot product with in a previous example), which means we adjust those functions to improve the probability of getting a better output. The most common way of doing this is to evaluate the entire network on a series of examples, and adjust the functions in some small way that shifts the output towards a correct output using gradient descent [9].

Each layer builds on the computations and representations done in previous layers. For example, when classifying an image, each subsequent layer could identify edges and lines, tails and paws, bodies, and finally entire dogs. Deep learning networks are composed of many such layers, each of which relies on work done previously.

Central to deep learning research are:

1. Creating the functions that generate the values for each node. While neural networks with single intermediate layers of sufficient sizes are universal approximators [40], they require infeasible amounts of nodes and often lead to intractable optimization objectives. Instead, research tries to impose priors that bias learning towards solutions that are more correct, while still maintaining expressivity. Notable examples of this include convolutions [53], which only looks at nodes from the previous layer which are nearby in space and reuses the same trainable variables in many functions, transformers [90], where a node weights nodes on a previous layer depending on relative importance for the current computation, and batch normalization [43], which normalizes the inputs to have certain statistics to make the gradients of the network more stable.

2. Defining the inputs and the outputs of the network. Because of how networks are defined and trained, certain inputs and outputs will work better than others. For example, real-valued inputs should all fall within the same range of values, such as in $[-1, 1]$. Additionally, adjusting the inputs and outputs can add memory to a network, like in long short-term memory architectures (LSTMs) [39] which takes the current network's memory as input, transforms that memory through some functions, and outputs the transformed memory.

3. Correctly adjusting the trainable variables within a function based on data. The simplest way is stochastic gradient descent which computes the gradient of the output on a batch of data with respect to each variable, and adjusts each variable to make the output more correct [9]. There are countless extensions possible however. Adam [48] adds momentum to gradient descent and improves convergence by tracking the gradients and their second moments, while second-order methods [57] instead estimate the Hessian and use that to compute updates.

4. Applying the above more fundamental research and generating new insights to implement deep learning on a specific domain. WaveNet [66] for example uses scaled raw audio waveforms (as well as conditioning information, like text if applied to Text-to-Speech) as input, uses functions (including stacked dilated causal convolutions, gated

activation units, softmax, and residual connections) for each node, and outputs new audio samples corresponding to the next sound.

2.2 Reinforcement Learning

Reinforcement learning (RL) is a sub-field of AI where an agent acts within an environment. More formally, reinforcement learning is typically modeled as a Markov decision process (MDP) that consists of (S, A, P_a, R_a) [82]:

- A set of environment states S.
- A set of possible actions A that an agent can take.
- A transition function $P_a(s, s') = \Pr(s_{t+1} = s' | s_t = s, a_t = a)$ that defines the probability of transitioning to a state $s' \in S$ given that at time $t \in \mathbb{N}$ an agent is in state $s \in S$ and takes action $a \in A$.
- A reward function $R_a(s, s') = \mathbb{E}[r_{t+1} | s_{t+1} = s', s_t = s, a_t = a]$ that defines the expected immediate reward $r_{t+1} \in \mathbb{R}$ from transitioning from state s to state s' by the action a.

In the case where the agent cannot observe the true complete environment state, like Atari, StarCraft II, or the real world, this becomes a Partially observable Markov decision process (POMDP) that consists of $(S, A, P_a, R_a, \Omega, O)$ [82] where S, A, P_a, R_a are defined as above and:

- Ω is the set of possible observations.
- $O(s', a) = \Pr(o | s', a)$ is the probability of the agent receiving the observation $o \in \Omega$ given the environment state s' and taken action a. A history is then $h_t = (o_1, a_1, o_2, a_2, ..., o_t, a_t)$, the combination of observations and actions taken since the beginning of the environment.

An agent is then a policy function $\pi(a_t | h_t) = \Pr(a_t | h_t)$ that selects which actions to take at each timestep. The objective of the agent is the maximize the expected future rewards $R_t = \sum_{i=t}^{\infty} r_i$, or in some cases expected discounted future rewards $R_t = \sum_{i=t}^{\infty} \gamma^{i-t} r_i$ to weight rewards that happen earlier more heavily by applying a discount factor γ multiplicatively. When applied to deep learning, the agent policy function becomes $\pi_\theta(a_t | h_t)$ defined by a neural network with parameters θ. This framework is, perhaps surprisingly, applicable to almost all cases where an agent acts within an environment.

2.2.1 Reinforcement Learning Updates

There are many ways of updating the parameters θ, creating the objectives to adjust the network outputs towards in order to improve the expected future rewards. These include Monte Carlo methods [82], Deep Q-Learning [61], Proximal Policy Optimization [75], and IMPALA [27]. Under reasonable assumptions, with sufficient exploration, and assuming the policy function π_θ is a universal function approximator, these methods are guaranteed to converge to the optimal policy eventually [74, 27, 82], however they differ in their behavior while converging, assumptions, and the rate of convergence. Because AlphaStar mainly uses IMPALA, we will describe it in more detail.

IMPALA combines Asynchronous Advantage Actor-Critic (A3C) [60] with an off-policy correction method called V-trace [27]. Actor critic methods combine the policy function π_θ with a value function $V_\theta(h_t) = \mathbb{E}[R_t|h_t, \pi_\theta]$ whose goal is to estimate the expected future rewards the agent would get from its current state if it acts according to the function π_θ. By running the policy π_θ on an environment many times, we can generate a trajectory of experience and train V_θ to produce the correct output reward on that trajectory, and then use the value function as a critic to have the actor policy take actions that produce better than average rewards more often, and worse than average rewards less often.

In more detail, the parameters θ are updated to minimize the temporal difference errors of the value function: $r_t + \gamma * V_t - V_{t-1}$ where $V_i = V_\theta(h_i)$ which is a more stable objective than the approximating the total rewards directly [82]. Parameters θ also update the reward through $\Delta_\theta \log \pi_\theta(a_t|h_t)(R_t - V_t)$, which is an unbiased estimate of $\Delta_\theta \mathbb{E}[R_t - V_t]$, where $R_t - V_t$ is the advantage of taking the chosen action over the expected policy action (using the advantage instead of the reward directly decreases the variance of the objective).

IMPALA modifies this slightly by weighting the updates of the value and policy by importance sampling weights, which takes into account how different the behavior policy μ that generated a trajectory is from the current policy. This is necessary for distributed computation and using a single trajectory for multiple training steps [82, 95]. These importance sampling weights are $c_t = \min(\bar{c}, \frac{\pi(a_t|h_t)}{\mu(a_t|h_t)}), \rho_t = \min(\bar{\rho}, \frac{\pi(a_t|h_t)}{\mu(a_t|h_t)})$ for the value and policy respectively, where $\bar{\rho} \geq \bar{c} \geq 1$.

2.3 StarCraft

StarCraft is a domain that consists of two real-time strategy video-games, StarCraft: Brood War and StarCraft II, created by Blizzard Entertainment and first released in 1998. The

Figure 2.1: Screenshot of a StarCraft II game in progress. Copyright Blizzard Entertainment.

game is set in a science fiction universe in which three races (human Terran, insectoid Zerg, and technologically advanced Protoss) interact to decide the fate of a sector of the Milky Way galaxy. Over the decades, StarCraft has become the most popular real-time strategy game of all time, with a thriving competitive esports scene and tens of millions of dollars of prize money. An image of StarCraft II in progress, playing as Protoss, can be seen in Figure 2.1.

In the standard competitive setting, two players select either one of the three races, or a random race, in which case the opponent is not told the other player's starting race. Each player begins with an initial home base and a dozen worker units seen in the upper-left of Figure 2.1. The race decides the set of units and buildings available to each player, each of which has its own abilities, advantages, and disadvantages. For example, the Zerg Infestor can take control of enemy units or slow and deal damage to a group of enemy units, but cannot attack. Over the course of the game, players mine resources (top-left corner), create

new units and buildings (center), research new technologies, develop new bases to expand their territory (visible in the minimap in the bottom-left), all while competing against an opponent who is trying to do the same. A player loses if they no longer have any buildings. In order to win, players must balance high level economic and research strategy, called macro, with low-level control of hundreds of individual units, called micro, in a real-time setting that necessitates issuing hundreds of distinct actions per minute.

2.3.1 Challenges of StarCraft

StarCraft is such a difficult and appealing domain for AI research because it contains many challenges not found in other games. Its raw complexity is extremely high (in our representation there are 10^{26} possible actions at each step), there can be tens of thousands of timesteps within a single game, and the observations consist of many streams of data including visual, entity, and scalar information [93]. The game itself is partially observable; the only information visible to a player is information within a certain distance of a unit or building the player controls, and the player only receives full information on a section of the map that they focus on [93]. In Figure 2.1, the entire high-level map is in the bottom left, and the opponent is too far away to be observed. StarCraft is also real-time and actions are issued continuously, so unlike turn-based games there is no defined time for planning. Players have to balance their attention between planning, gathering information (by positioning units and by moving the camera), responding to unforeseen events, and actually executing their plans.

Finally, StarCraft has a complex game-theoretic strategic landscape with cycles and intransitive strategies. These strategies rely on planning across thousands of timesteps often based on a single initial observation and theory-of-mind expectation of the opponents future interactions. This means that naive self-play exploration across the possible action-observation histories will almost never develop useful strategies, and furthermore any strategies developed by an agent that does not interact with humans may not be relevant when deployed with humans. For example, to take advantage of someone who does not build adequate defenses against the harassing air units called Banshees, a player would need to scout an opponent to see what researches and defenses they have available, build a series of buildings necessary to produce a Banshee and eventually produce the Banshee itself, research Cloak so that the Banshee can turn invisible, and finally position, retreat, cloak, and uncloak the Banshee itself at the correct times. Executing this strategy with any component missing is worse than avoiding the strategy entirely, so it must either be discovered at once or not at all. This high-level description could span the entire game in which many other strategies and tactics are used.

The challenges present in StarCraft are also present for an agent deployed in the real-world. Such an agent will also have to reason and plan across long time-horizons and complex actions, integrate information from different data streams, observe a limited part of the true environment, and issue actions continuously, all while acting in a human-aligned way.

2.4 Prior Work

2.4.1 AlphaStar's Implementation

AlphaStar depends on a massive body of prior work of deep, reinforcement learning. The network architecture consists of deep learning techniques including transformers [90], convolutions [53], LSTMs [39], pointer networks [94], autoregressive actions [58], and residual layers [36], albeit combined in novel ways different from other applications. Beyond reinforcement learning techniques mentioned in prior sections [27, 82], the network parameters are updated in other ways common to deep learning: a supervised cross-entropy loss to imitate labelled data [33], a regularizing entropy loss [34], and concurrent work on privileged baselines [3, 16]. Similar to AlphaStar's conditioning on human data, other work has used human preferences [63, 42] or human intervention [18] to create reinforcement learning reward functions. The idea of training a supervised policy to imitate humans, then using that policy as a prior was used by AlphaGo [77].

AlphaStar also arises from existing work in multi-agent research, in particular training a population of agents that compete against each other as in Quake III Arena [44]. More theoretically, league training is an extension of the Policy Space Response Oracle framework [50].

2.4.2 Reinforcement Learning in Games

Creating policies for games has long been a focus of reinforcement learning and artificial intelligence research. They define natural MDPs and POMDPs (where the game has its own logic that defines the state, possible actions, transitions, rewards, and observations seen by the player), and therefore can often be approached in transferable ways. Computers have achieved superhuman performance in games like chess [14], Go [77], Atari [61], and Capture the Flag [44].

Real-time strategy (RTS) games are a particular sub-genre of games which present a grand challenge for AI [13], and many games in the genre share some of the complexities of StarCraft. There has been research on particular sub-tasks of RTS games (e.g. controlling units), or RTS games in simplified environments [12, 19, 99, 89, 97, 73], and StarCraft has become the focus of the overall RTS AI research in a complete game [93, 1]. Prior work has looked at both StarCraft: Brood War and StarCraft II, examining strategy selection [41], simulation [83], reinforcement learning for micro [89, 76, 7, 85], imitation learning for strategies [47], search [88], game state prediction [84], and high-level hand-crafted actions [80]. StarCraft: Brood War and StarCraft II have active AI development communities [96, 1, 79, 78]. However, no prior work has overcome even strong casual players [29], even though almost all of them ignore considerations of fairness or human-like constraints (with some bots issuing tens of thousands of distinct actions per minute), or have components specific to StarCraft II [20, 78].

OpenAI 5 recently beat top professional players and 99.4% of professional players in Dota 2 [8], a recent video-game that shares several challenges with StarCraft II. In Dota 2, two teams of five players who each control a single hero unit face off against one another in a setting with complex strategies, long time horizons, and imperfect information. The agents within OpenAI 5 were trained together using PPO [75] and, in contrast with AlphaStar, used hand-crafted intermediate rewards, scripted high-level actions, a limited rule-set, and shared observations between all agents. They also used a technique they call surgery to train a single agent over 10 months by resuming training from a previous checkpoint as the domain shifted and new features were added [8].

2.4.3 Game Interfaces for Deep Learning

A game interface is the protocol through which an AI agent receives observations of a game and sends actions. For example, in Atari the agent could receive at each frame a 2D 210x160 grid observation, where each element is one of the 128 possible color values, and send an integer corresponding to the 18 possible actions with no delay or limit on processing time.

There has been minimal work focusing on developing effective interfaces for deep learning research; most work has been within a paper and tried to demonstrate that a particular agent or approach is fair. They tend to focus on reaction times, or present an interface with minimal explanation. OpenAI 5 discussed hiding information that would not be visible to humans, the delays present within their agents, and how often their agents could act in comparison to humans [8]. AlphaGo set time limits for the agents to be comparable

to humans [77]. Work in Quake III Arena Capture the Flag compared agent and human reaction times and reaction accuracies [44]. As far as we know, there is no standalone work discussing interfaces and human-like constraints for deep learning agents.

Significant work has also focused on developing safe interfaces for deep learning, where there are safety or performance constraints that limit how much an agent can explore [31].

Chapter 3

Overview of AlphaStar

AlphaStar consists of different systems and stages that work together to create a unified whole. An overview can be seen in Figure 3.1. The following sections will go into more detail on each component, but at a high level an AlphaStar agent is a policy $\pi_\theta(a_t|h_t, z) = \Pr(a_t|h_t, z)$, defined as in 2.2 with the addition of a statistic z which encodes a general strategy extracted from the record of a human game called a replay. This statistic could be seen as an additional component of the initial observations o_1 in the formal definition, but is separated here for clarity. Each timestep t, the agent uses the previous action a_{t-1}, the current observation o, the statistic z, and an LSTM state from the agent processing all previous observations and actions $o_1, a_1, ..., a_{t-2}, o_{t-1}$ (thereby extending the memory of the agent through the entire history) to decide which action to take next. The interaction of the agent with the environment, modulated by action limits and delays, can be seen in Figure 3.1 **a**.

To do the processing, AlphaStar uses a transformer [90] to embed unit information, a series of convolutions [53] to process the map, fully-connected layers [33] to embed scalar information, novel scatter connections to combine spatial and non-spatial data, a deep LSTM [39] to track memory, and an autoregressive policy with a pointer net [58, 94] to select the final action. The architecture, though designed for StarCraft, does not contain components that are specific to it, and it processes the multiple input streams and decomposes the action in a way that makes the entire problem amenable to learning. An opposing agent, or human, will simultaneously act.

The parameters of the policy were first trained in supervised learning to imitate a human policy (Figure 3.1 **b**). We used a large dataset of human replays, generated the actions the humans and observations the human saw each timestep, and trained the network to

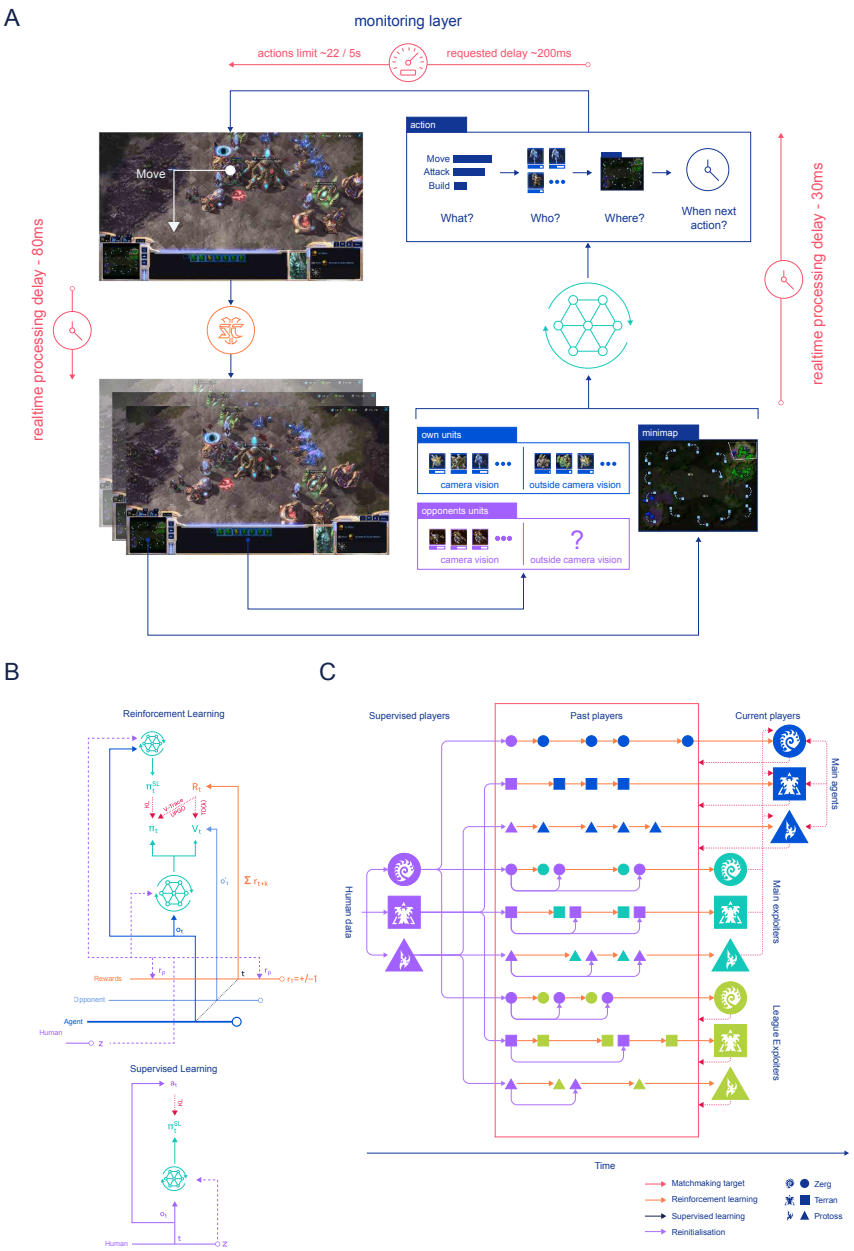

Figure 3.1: Overview of the training and evaluation of AlphaStar. **a** shows the agent interacting with its environment, **b** shows supervised learning (bottom) and reinforcement learning (top) training process, and **c** shows multi-agent league training. [92]

maximize the probability of taking the same actions given the previous history (conditioned on the statistic z a fixed probability of the time) and some additional regularization. The purpose of supervised learning is to establish a strong prior for taking meaningful actions, alleviating the exploration problem, while capturing diverse, human aligned strategies. The statistic z ensures that the network does not converge too early during reinforcement learning and that diverse strategies are sufficiently represented through the rest of the training process.

After supervised learning, the network is trained through multi-agent reinforcement learning (MARL), shown in Figure 3.1 **c**. Agents are paired against opponents in league training, which manages a continually expanding and learning set of players which balances finding weaknesses for top strategies, finding systematic weaknesses of all strategies, creating new strategies that overcome those weaknesses, and making existing strategies more robust. League training attempts to address the game-theoretic strategic landscape inherent in StarCraft.

Trajectories of experience from the games assigned by league training are replayed [95] to asynchronously update the agents themselves in reinforcement learning to maximize the win rate in the assigned match-up (i.e. compute a best response to the distribution of strategies it is matched up against). This is depicted in Figure 3.1 **b**, The network loss function is given by a combination of IMPALA [27] losses to update the policy, TD(λ) [81] losses to update the value network, a novel loss called UPGO to update the policy, and a distillation loss towards the initial supervised network [71, 67] to ensure the policy stays diverse and does not collapse during training. When z is given as input to the network, agents are given an additional reward for following the strategy encoded in z. The combination of distillation loss and z form human exploration that explores a wide variety of human aligned strategies.

3.1 Interface

The interfaces and motivations behind how it is designed will be explored in greater detail in Chapter 6, and are shown at a high level in Figure 3.1 **a**. The observations are described in Table 3.1, and consist of all visible entities in the game and their attributes, the map and its layers of information, various scalar player data, and game statistics. Data is hidden from the observations if it would also be hidden to humans, except some data which would need to be remembered or inferred. An example of hidden data is an underground enemy unit that would not appear in the entity list unless there is a way to detect it, or it starts moving in which case humans would see a moving underground unit. An example of information

Category	Field	Description
Entities: up to 512	Unit type	E.g. Drone or Forcefield
	Owner	Agent, opponent, or neutral
	Status	Current health, shields, energy
	Display type	E.g. Snapshot, for opponent buildings in the fog of war
	Position	Entity position
	Number of workers	For resource collecting base buildings
	Cooldowns	Attack cooldown
	Attributes	Invisible, powered, hallucination, active, in cargo, and/or on the screen
	Unit attributes	E.g. Biological or Armored
	Cargo status	Current and maximum amount of cargo space
	Building status	Build progress, build queue, and add-on type
	Resource status	Remaining resource contents
	Order status	Order queue and order progress
	Buff status	Buffs and buff durations
Map: 256x256 grid	Height	Heights of map locations
	Visibility	Whether map locations are currently visible
	Creep	Whether there is creep at a specific location
	Entity owners	Which player owns entities
	Alerts	Whether units are under attack
	Pathable	Which areas can be navigated over
	Buildable	Which areas can be built on
Player data	Race	Agent and opponent requested race, and agent actual race
	Upgrades	Agent upgrades and opponent upgrades, if they would be known to humans
	Agent statistics	Agent current resources, supply, army supply, worker supply, maximum supply, number of idle workers, number of Warp Gates, and number of Larva
Game statistics	Camera	Current camera position. The camera is a 32×20 game-unit sized rectangle
	Time	Current time in game

Table 3.1: The observations the agent receives through the raw interface. [92]

Field	Description
Action type	Which action to execute. Some examples of actions are moving a unit, training a unit from a building, moving the camera, or no-op. See PySC2 for a full list [93]
Selected units	Entities that will execute the action
Target	An entity or location in the map discretized to 256x256 targeted by the action
Queued	Whether to queue this action or execute it immediately
Repeat	Whether or not to issue this action multiple times
Delay	The number of game time-steps to wait until receiving the next observation

Table 3.2: The action arguments the agents can submit through the raw interface as part of an action. [92]

that needs to be inferred or remembered is the map, because humans see a single RGB image while an agent views the layers of information about the map including information humans would not be explicitly told, like which areas of the map can be traversed by foot.

The components of each action an agent can take are described in Table 3.2. Note that this is a highly structured action space, and that though the network submits all arguments for each action, some arguments are ignored depending on the action type. One argument of an action is the delay, which defines the number of game steps until the agent will next receive an observation and issue an action. Agents can only issue 22 non-duplicate actions per 5 second window (which also therefore limits how often they can observe), which loosely matches the action-rate of top professional players.

AlphaStar's observations and actions were modulated through a camera view, similar to how humans only focus on a section of the larger map at a time. This camera can be moved as an action to focus on different sections of the map. A minimap displays high level information about entities that are outside the current camera (e.g. their position) and allows certain actions to be issued outside the camera (e.g. moving a group of units), so we limited the observations and possible actions outside the camera accordingly.

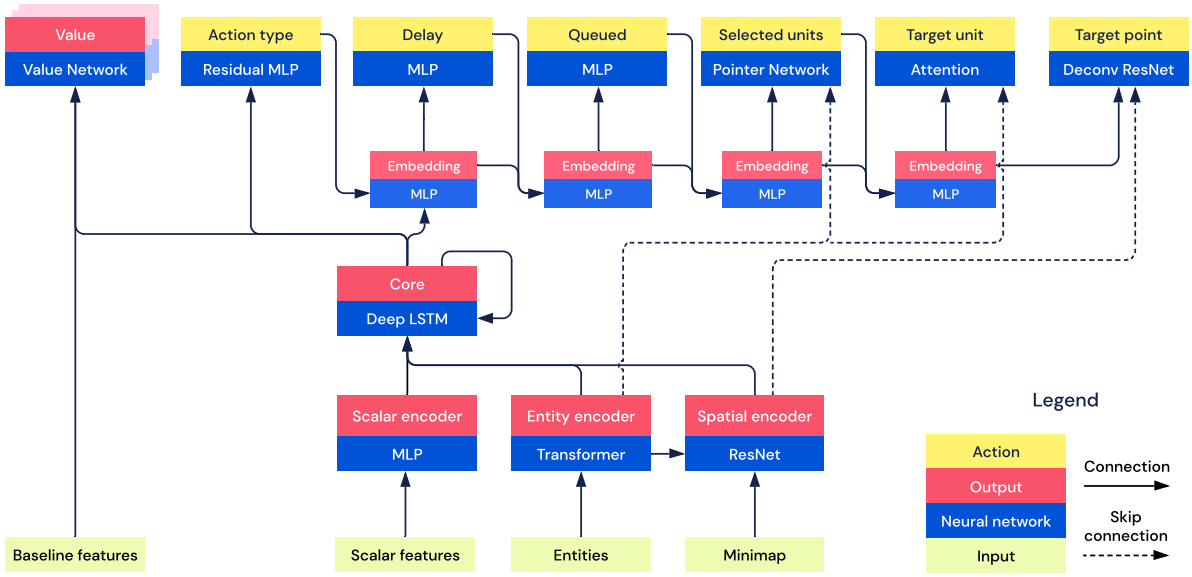

Figure 3.2: Overview of the architecture of AlphaStar. [92]

3.2 Architecture

AlphaStar's policy network embeds three different types (entity, map, and scalar) of information, concatenates them together, passes that concatenation and previous LSTM states through a Deep LSTM [39], and uses the LSTM output to autoregressively sample each action argument one at a time [58]. The value network instead uses the LSTM output and additional baseline features to generate a baseline. An overview of the architecture can be seen in Figure 3.2.

AlphaStar has one of the most complex network architectures in a deep learning project. However, because of the scale AlphaStar needs to run at, it was infeasible to iterate on the architecture purely through performance in multi-agent reinforcement learning. This was compounded by the inherent variability present in RL; the performance improvement from an architecture change could easily be overshadowed by variance in performance between runs. Instead, the architectural choices (and in fact, any changes that could be evaluated this way including interface and dataset ones) were made through supervised learning, which is a more stable objective that could be evaluated much faster. We trained a model with different adjustments to convergence in imitation learning, and made decisions based on the accuracies, losses, and performance against built-in StarCraft bots and AlphaStar

players from previous experiments. Note that performance in actual play is the most important metric here because, for example, giving the network access to the entire previous action improves accuracies (because humans tend to repeat the same action multiple times) but can lead to worse play (because repeating an action is often not a useful strategy). Real performance acts as a validation metric that the network does not directly optimize. If the choice managed to better capture and reproduce human play, we would integrate it into larger MARL experiments.

We will now go into more detail on the architecture, though a full description of layer sizes and other details can be found in the supplementary information of the original publication.

3.2.1 Policy Network

The entity list is a list of every discrete visible entity, including units, buildings, resources, and interactable terrain, in the game. The list has a maximum size of 512, and any entities after 512 are ignored. We found through analysis of supervised games that about 1% of recorded games had over 512 entities at any point in the game. Each entity has a 1D vector of information corresponding to that entity (for example, its position, unit type, and whether or not it is invisible) and data in those vectors is preprocessed to either fall within a uniform range around $[0, 1]$ or encoded as one-hot vectors. The preprocessed entities are then fed through a 3-layer transformer [90] with 2-headed self-attention. The mean of the transformer output across all entities is processed through a linear layer and rectified linear unit (ReLU) [64] to yield an embedding of all the entities that will be concatenated with data from the map and scalars, and the output for each entity is processed through a ReLU, 1D convolution, and another ReLU to yield embeddings of each entity that will be used both to enrich the embedding of the map and as a skip connection to help select entity action arguments.

The spatial encoder embeds the 2D visual information from the map enriched through scatter connections by entity embeddings. The map consists of several layers preprocessed to either fall within a uniform range or to be one-hot. Entity embeddings are converted into size 32 vectors by a 1D convolution and ReLU and transformed into a spatial representation with 32 layers where the entry at a specific (x, y) coordinate corresponds to the embedding of the entity that would be found at that coordinate. We call this a scatter connection because it scatters information about entities in space to where they would be found, allowing us to combine spatial and entity information. The map layers and entity layers are concatenated and down-sampled through a series of convolutions and ResBlocks [36],

and finally collapsed and embedded into a 1D tensor that will be concatenated with data from the entities and scalars. The spatial information at each step of down-sampling is saved and used when selecting spatial action arguments.

All data that is not an entity or spatial is embedded by the scalar encoder. This includes the list of available actions, the amount of resources available to players, and the statistic z. Each scalar feature is preprocessed, similar to entities and spatial features, and individually embedded by linear layers or small transformers, then concatenated together as the overall scalar embedding. Certain features that are relevant for action type selection (e.g. the statistic z and the available actions) are also concatenated into a separate vector that will be used for action type selection.

The concatenated entity, spatial, and scalar embeddings are combined into a single 1D tensor, and passed through a 3-layer LSTM which adds memory between timesteps.

Subsequently, each action argument is selected autoregressively [58]. That is, by the chain rule, the probability of selecting an action a with arguments $a_1, a_2, ..., a_n$ is $\Pr(a|h_t) = \Pr(a_1|h_t) * \Pr(a_2, ..., a_n|h_t, a_1) = ... = \Pr(a_1|h_t) * \Pr(a_2|h_t, a_1) * ... * \Pr(a_n|h_t, a_1, ..., a_{n-1})$. This selection decomposes the action space from 10^{26} to something tractable and easier to learn on while letting the selection of each argument be aware of all previous argument selections, to avoid events like building with a unit that cannot build.

Argument selection then consists of several heads, one for each argument type, where each head is conditioned on the action type argument (except the action type head and the delay head) and a 1D autoregressive embedding that encodes all previous argument selections (except the action type head, which is first and uses the LSTM output). Each head outputs its corresponding argument, along with a new autoregressive embedding that is updated through a fully connected network to include the new argument. The action type head processes the LSTM output through several ResBlocks [36] and a gated linear unit [22] gated by the previously mentioned scalar information to select the action type. The delay head processes the autoregressive embedding through a fully-connected network to select the delay, and the queued head processes the embedding through a fully connected network to select whether or not to queue the action. The selected units head computes a key for each entity corresponding to the previously mentioned entity embedding skip connection, masked by whether or not the entity is a valid selection, and uses those keys in a recurrent pointer network [94] to decide which of up to 64 selected entities to apply the action to. The target unit head is similar, except selecting a single unit. Finally, the location head reshapes the autoregressive embedding to the same shape as the final previously mentioned map skip-connections, concatenates the embedding and skip connections, and processes the concatenation through a series of up-scaling convolutions, Gated ResBlocks [22] (gated by

the autoregressive embedding), and FiLM [69] layers to select the location arguments. Through this process, the network can reason about sampling an action.

3.2.2 Value Network

The last part of the network architecture are the split baselines which constitute the value network. The agent rewards in reinforcement learning consist of the win-loss reward and the reward for following the statistic z, both of which will be described in the following sections. There is a separate baseline for each distinct reward component, which improves learning by disentangling the various rewards. Each baseline gathers and preprocesses various input information, including some of the scalar features, the selected action type, and the LSTM output. Notably, each baseline also has access to privileged information that is not available to the policy network, like the opponent observations (a concept which was developed concurrently with several other publications [3, 16]), and does not bias the converged objective. The various input features are concatenated and passed through a series of ResBlocks, and finally re-scaled to generate the baseline value for each baseline.

3.3 Supervised Learning

Supervised learning, where the AlphaStar agent learns to imitate human actions and strategies, is the first step of training an AlphaStar agent. Because learning to imitate an existing expert policy is significantly easier and more reliable than exploring to discover that policy, this step helps address the representational, exploration, and strategic difficulties inherent in the problem. Our objective was to create a policy that would be useful for reinforcement learning and so develop not necessarily the strongest policy possible, but instead a policy that meaningfully captures a wide variety of human strategies. This step proceeds similarly to most supervised learning applications: the observations the human would have seen and inputted into the policy, and the outputs of the policy are adjusted by gradient descent to improve the probability of selecting the actions the human actually selected. This can be seen in Figure 3.1 **b** (bottom).

3.3.1 Dataset

The dataset consisted of 971000 replay recordings of human games, gathered by Blizzard Entertainment, played on StarCraft II versions 4.8.2 to 4.8.6, which we then filtered to

only include players with matchmaking rating (MMR) greater than 3500, approximately the top 22%. MMR is a metric similar to Elo [26] or TrueSkill [38] which records both a player's estimated skill level and the uncertainty of that skill window. Different versions of StarCraft II have slightly different rulesets, but strategies and representations are generally transferable. The replays were executed in StarCraft to generate a stream of observations and actions, and all observations where the human did not act were filtered out (with the delay argument for the action set to when the human would next act). This both improved performance, because the policy network did not have to act on every game step and much less data needed to be sent between processes, but also imposed a human prior for how often an agent would act.

3.3.2 Strategy Statistic z

The statistic z was extracted from the replays. It encodes an overall view of the strategy into a summary vector, and is used to encourage the policy to follow a specific strategy by conditioning agents during supervised learning, and both conditioning and explicitly rewarding agents during reinforcement learning. z consists of each player's cumulative statistics and build order. The cumulative statistics are a 1-hot vector of whether or not each unit, building, effect (e.g. a force-field, which is produced by a special ability), or research upgrade was present at any point in the game, which gives a high-level overview of the strategy the player executed. The player's build order is the first 20 units and buildings the player constructed during the game, as well as the positions of the buildings. Build orders are a common way of summarizing a player's opening strategy in StarCraft. Agents are conditioned only on z extracted from a randomly sampled replay corresponding to the top 2% of players, and only from replays where the match-up and map matches the one the agent is currently playing (e.g. playing as Zerg against Protoss on the map Kairos Junction). One limitation of this formulation of z is that strategies in StarCraft are reactive, and depend on observation of the opponents strategy, so while agents may be diverse they may not be sufficiently reactive. Therefore, during the final evaluation described in 3.7, agents do not use z.

Though the specification of z is specific to StarCraft, the idea of conditioning based on high-level human strategies is transferable to other domains. During supervised learning, the cumulative statistics portion of z is used 50% of the time, and the build order portion is used 80% of the time.

3.3.3 Supervised Learning Training

A separate agent is trained for each of the three StarCraft races (Terran, Protoss, and Zerg), though all agents can play versus any of the three races. Each step of a replay, we compute the logits corresponding to sampling each action argument of a_t for the policy $\pi_\theta(a_t|h_t, z)$, and minimize the cross-entropy between these logits and the true human actions. The cross-entropies are weighted differently for each argument (e.g. the action type is more important than whether or not the action is queued), there is an additional L2 regularization loss [33], and the combined losses are applied with an Adam optimizer [48] with a learning rate that decays by 0.2 every $5 * 10^9$ observations.

After the policy has converged, there is an additional fine-tuning step in which the agent is trained on replays only from the perspective of the winning player, and only from players with MMR $\geq$ 6200, which yields about 16000 games. In fine tuning, the learning rate starts lower and decays more often, every $1.5 * 10^9$ observations. Note that the policy is given as an input the MMR of the player it is imitating, and during evaluation this MMR is set at 6200.

Fine-tuned agents perform surprisingly well even though they have never received a single reward and are only trained to imitate humans. They can win 96% of the time against the built-in AI in Protoss versus Protoss matches, and were evaluated as performing within the top 16% of players. However, more importantly for the next stage, they are capable of executing and responding to a wide variety of strategies across games.

3.4 Reinforcement Learning

Agents are matched against distributions of other agents and historical players by the AlphaStar League, the matches generate trajectories of experience, and that experience is used to asynchronously refine the policy and value networks, with the policy initialized from supervised learning, through reinforcement learning. This can be seen in Figure 3.1 **b** (top).

3.4.1 Rewards

The reinforcement learning rewards are the win-loss final reward (+1 on a victory, 0 on a draw, and -1 on a loss), and the rewards from following a statistic z. The win-loss reward is not discounted so that agents take a longer term strategic view and do not

converge to "rush" strategies that can be executed quickly. The rewards from z are the build order rewards, the change from step to step in the negative Levenshtein distance between z and the build order executed in the game with an additional penalty term for building buildings in a different location from z, and the cumulative statistics rewards, the change from step to step of the Hamming distance between the z and an indicator for whether each unit, building, research upgrade, and effect is present within the game. The cumulative statistic rewards decrease to half after 8 minutes, a quarter after 16 minutes, and 0 after 24 minutes (the average game is around 20 minutes) so that agents do not delay victory in order to gather as much cumulative statistic reward as possible. In contrast to supervised learning, the policy is conditioned on the build order 25% of the time, and on the cumulative statistics 25% of the time. As described in 3.2.2, there is a separate value function for each reward component, which is an easier learning objective than learning all parts of a multi-faceted reward jointly.

3.4.2 Exploration

There is no explicit ϵ-greedy style exploration present in AlphaStar's training, and in fact such exploration would be intractable due to the sizes of the environment space and action space. Instead, we use human exploration to explore the action and strategic space of StarCraft. As mentioned in 3.4.1, the agent is rewarded for following a human strategy, and we can therefore force an agent to consider a strategy it would have otherwise avoided. The agent is also initialized from a supervised checkpoint, and through a distillation loss forced to not diverge too far from the initial checkpoint [71, 67]. That is, the supervised policy and the current reinforcement learning policy are both run on the same observation, and the optimizer minimizes the KL divergence between each supervised and RL action argument. This distillation loss ensures that the policy does not collapse prematurely to a single strategy (it can be seen as a regularization entropy loss except with a supervised prior instead of a zero prior), and also ensures that the agent continues to be able to explore a diverse array of strategies throughout training. Without this exploration, policies quickly collapse to local minima.

3.4.3 Reinforcement Learning Training

Distributed actors play matches to generate trajectories (each trajectory consists of 128 timesteps of StarCraft) of experience, those trajectories are sent to a replay buffer [95] that repeats each trajectory twice, and a central learner samples from the replay buffer

to generate experience for updates. Because computation is distributed, the policy that generated trajectories may be different from the current policy on the learner (because of delays in receiving updated parameters and sending trajectories) so on-policy reinforcement learning methods like A3C need to have their updates adjusted [82].

Therefore, there is a modified IMPALA [27] actor-critic loss for each reward and corresponding baseline. The actor policy network loss needs to be corrected by V-trace to avoid instability, but the critic value network loss can instead be updated using just TD(λ) [81] which, though biased because it learns the value for a slightly different policy, decreases variance and improves learning efficiency. This bias for the critic is necessary because the action space is large and structured, so many different actions can have similar or even identical effects. For similar reasons, even though we applied V-trace to the policy, we assumed independence between action arguments and applied corrections to them separately. This is also biased, but necessary because the large action spaces means that V-trace would otherwise down-weight even slightly off-policy trajectories.

AlphaStar also introduces a novel, general policy update called the upgoing policy update (UPGO) which delays bootstrapping and instead looks at n-step returns [68] as long the actions in the trajectory are better than average. This objective is similar to self-imitation learning [65], and uses:

$$G_t = \begin{cases} r_t + G_{t+1} & \text{if } r + V_\theta(h_{t+2}, z) \geq V_\theta(h_{t+1}, z) \\ r_t + V_\theta(h_{t+1}, z) & \text{otherwise} \end{cases} \tag{3.1}$$

G_t is used instead of the directly bootstrapped $r_t + V_\theta(h_{t+1}, z)$ to compute advantages for the UPGO loss, which is otherwise similar to an IMPALA actor loss. UPGO propagates information for partial trajectories that perform better than average faster than normal actor critic losses.

The TD(λ) losses for each baseline, the V-trace policy losses for each baseline, an UPGO loss for the win-loss baseline, the KL divergence loss from the supervised policy, and a standard entropy regularization loss are all combined and optimized using Adam [48] with a fixed, non-decaying learning rate. The learning rate does not decay because the objective changes throughout training as the opponents learn and adapt.

3.5 Multi-agent Learning

The novel league training procedure introduced in AlphaStar manages the complex strategic landscape present within StarCraft by maintaining a pool of historical players and

constantly training live agents. It assigns match-ups to distributed actors to generate reinforcement learning trajectories and to estimate the Elo of everything within the population, stores snapshots of live agents into a pool of historical players, and replaces live agents with new ones (initialized from supervised learning) when necessary. In StarCraft, as in many multi-agent partially observable domains, there is no single dominant strategy, and single purpose strategies that are weak against most others (called "cheeses" in StarCraft) can exploit more generally robust ones. Gathering information and responding appropriately to a wide variety of unexpected tactics is core to this partially observable environment, and league training attempts to instill this responsiveness onto its agents. A visualization of league training can be seen in Figure 3.1 **c**.

3.5.1 Matchmaking Algorithm

League training uses three matchmaking strategies: self-play, prioritized fictitious self-play (pFSP) focusing on hard opponents, and pFSP focusing on a smooth curriculum. Self-play, the most familiar setting, means that a set of agents are matched against one another and learn in parallel.

Naive self-play matchmaking between two agents quickly adapts but can devolve into cycles even in simple games [4]. For example, in rock-paper-scissors if one agent always plays rock, the other will learn to always play paper, so the first will learn to always play scissors, and so-on. To avoid this, fictitious self-play (FSP) plays against all previous players in a pool which avoids cycles entirely and converges to a Nash equilibrium in zero-sum two-player games [10, 55, 37]. However, because FSP plays uniformly across all previous players, it both wastes compute by playing an agent against many strategies it can already beat, but also learns to maximize performance against the average strategy in the pool. Humans can take advantage of this by shifting the distribution of strategies they sample from to strategies that an agent is weak against. Even if the agent has learned to maximize the average win-rate, if there is a single strategy the agent is weak against, that weakness can be exploited many times. Instead, the agent should learn to maximize performance against strategies it is weakest against, which has the side effect of providing more learning signal by not playing games against already beaten opponents.

Therefore, AlphaStar introduced prioritized fictitious self-play (pFSP) which plays an agent A against an opponent B from a pool of opponents C according to:

$$\frac{f(\Pr(A \text{ wins against } B))}{\sum_{c \in C} f(\Pr(A \text{ wins against } C))} \tag{3.2}$$

where $f(x) = (1-x)^2$ or $f(x) = \min(0.5, 1-x)$ if the goal is to focus on opponents who cannot yet be beaten (with the former version weighting such opponents more heavily), or $f(x) = x(1-x)$ if the goal is to focus on opponents of similar skill levels. The latter goal is useful when there are much stronger opponents.

pFSP relies heavily on accurate win-rate estimations, so a payoff matrix is continually estimated and updated during league training.

3.5.2 League Agents

There are three main types of agents within the league training, each of which differs in their matchmaking algorithm, when they are saved to the global pool of past players, and when they are replaced with a new agent initialized to the supervised learning policy. They are main agents, which are intended to be the strongest and least exploitable agents, league exploiters, which find systematic weaknesses in the league, and main exploiters, which find weaknesses specifically in the main agents.

There are three main agents trained simultaneously, one for each of the three StarCraft races. Main agents are trained against each other in self-play 35% of the time (although if the expected win-rate against another main agent is below 0.3 they will instead play curriculum pFSP against that agent's previously saved checkpoints). Main agents play hard opponents pFSP (using $f(x) = (1-x)^2$) against all previous players saved in the league pool 50% of the time. The remaining matches, 15% of the time, main agents play hard opponents pFSP (using $f(x) = (1-x)^2$) against saved main exploiter players that it has a win-rate below 0.3 against, and saved main agent players against which the agent used to have, but no longer has, a win-rate above 0.7. Checkpoints of main agents are saved to the global pool of players every $2 * 10^9$ steps, and main agents are never reset. The main agents are meant to be robust against the entire league pool and its various exploiters, and stay robust throughout training while continually improving.

There are six league exploiters trained at any point, two for each StarCraft race. League exploiters match with hard opponents pFSP (using $f(x) = \min(0.5, 1-x)$ because unlike main agents they are reinitialized through training and the normal hard opponents objective against a mature league would be too difficult), and are checkpointed into the player pool either every $2 * 10^9$ steps if they can defeat all players in the pool more than 70% of the time, or every $4 * 10^9$ steps otherwise. Each time the agent is checkpointed, there is a 25% chance it will be reinitialized to supervised parameters. League exploiters find weaknesses within the entire league, though note that no other agents will see the league exploiter's strategy until it has been checkpointed (at which point there is a chance it will

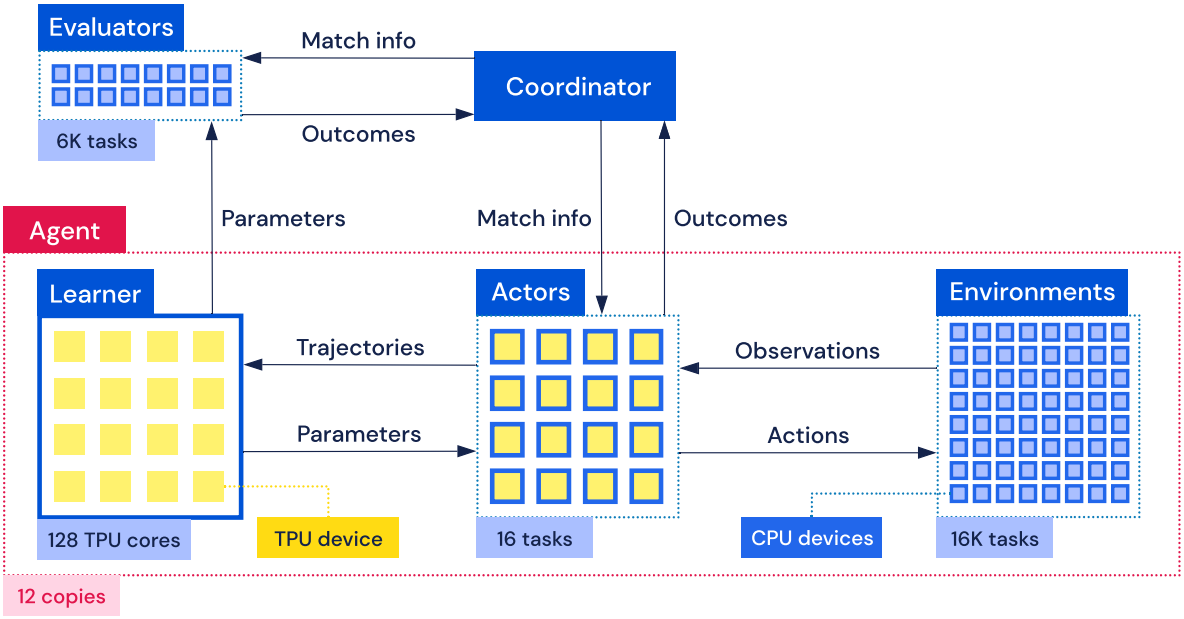

Figure 3.3: Training infrastructure. Diagram of the training setup for the entire league. [92]

be reinitialized) and added to the player pool so unlike main agents the league exploiters do not always have to be robust.

Finally, there are three main exploiters. If the main exploiters can win more than 20% of the time against main agents, they will play against main agents 50% of the time and spend the rest of the time playing curriculum pFSP against all previous checkpoints of the main agents. Otherwise, main exploiters will only play curriculum pFSP against previous main agents. The curriculum is necessary here because, like league exploiters, main exploiters can reinitialize and therefore may be much younger than main agents. Main exploiters are checkpointed and reset either when they can beat all three main agents 70% of the time or after $4 * 10^9$ steps. Main exploiters find weaknesses in main agents and, like league agents, no other agents play against main exploiters until they checkpoint and so main exploiters are not expected to be robust.

3.6 Infrastructure

AlphaStar trained in a distributed setting with actors, which run the agent network to generate trajectories of experience; learners, which use those trajectories to update parameters; environments, which run the StarCraft II game instances; evaluators, which update the payoff matrix for league training; a coordinator, which decides matchups for the actors to play; coordinator cachers, which cache requests to and from the coordinator, and learner cachers, which cache requests for parameters from the learner server. A simplified diagram of this can be seen in Figure 3.3.

Each agent in the league runs 16000 parallel StarCraft II game instances (on the equivalent of 150 modern 28-core processors) and 16 actors (on TPUv3 instances with 8 TPU cores [46]). Actors request parameters and matches from the learner cachers and coordinator cachers respectively, and execute matches by asynchronously executing StarCraft II game instances and batching together neural network executions to maximize use of the TPU. To perform this batching, the interface and network preprocessing were implemented with highly performant C++ or in-network TensorFlow code. The StarCraft II instances were run with rendering and other unnecessary features disabled.

Each agent also used a 128-core TPU learner, which updated the parameters of the network using trajectories of experience from the actors with a total batch size of 512 trajectories at once. Each trajectory is placed into a replay buffer and replayed twice for efficiency. Actors and evaluators request parameters from the learner through the learner cachers. There are 5 learner cachers per learner which cache the parameters of the learners, requesting new parameters up to once per second. Each learner can process approximately 50000 steps per second.

There are 6000 additional evaluator StarCraft II instances run on CPU that help update the payoff matrix. These instances receive matches from coordinator cachers, receive parameters from learner cachers, and send results of matches to coordinator cachers. A central coordinator performs the duties described in 3.5 and maintains the payoff matrix for all players. Two coordinator cachers batch calls to and from the coordinator.

In total, AlphaStar was trained for 44 days of league training with this setup.

3.7 Evaluation and Results

AlphaStar was evaluated three times: in an initial series of demonstration matches, in a more rigorous Battle.net evaluation, and in a more casual public exposition.

3.7.1 Battle.net Evaluation

The most rigorous and formal evaluation of AlphaStar was on Battle.net, Blizzard's online multiplayer system. Humans play on Battle.net, starting with an initial MMR that is updated based on the results of subsequent matches, and the goal was to see what performance level AlphaStar would reach in this anonymized human setting. AlphaStar reached a Grandmaster rating in all three StarCraft races, the highest rating tier possible, with a rating higher than 99.8% of all players who played enough games in the months before to be assigned to a rating tier (about 90000 players).

Games were played using Battle.net's matchmaking system, on StarCraft II version 4.9.3 in the Summer of 2019, against players who had opt-ed in to play against AlphaStar [24] (most players opted in), using a single machine with a consumer GPU. AlphaStar played on the maps Kairos Junction, New Repugnancy, Cyber Forest, and King's Cove. Though there were seven maps matches could take place on, humans can also decide to play on a subset of four of those seven. The maps were selected because they were the only maps common from when training AlphaStar began to when AlphaStar was evaluated. AlphaStar also played under an anonymous account name, which humans also frequently do, for blind conditions where neither AlphaStar nor its opponent were aware of the other's identity. Note that this focuses on evaluating AlphaStar against approximately stationary conditions, not exploitation under repeated trials. To avoid identification and conform to community standards, AlphaStar would communicate opening greetings ("gl hf", which stands for "good luck, have fun", or some variation of that message) and closing end-of game messages ("gg", which stands for "good game), and we would manually resign if there appeared to be no way for AlphaStar to win a given game. Note that premature resignation would only decrease AlphaStar's performance.

We evaluated AlphaStar at three points in time: AlphaStar Supervised, the supervised policy, AlphaStar Mid, the league main agents after 27 days of league training, and AlphaStar Final, the league main agents after 44 days of league training. Each evaluation consisted of one agent for each of the three StarCraft races. Though they were trained with the statistic z, they were not conditioned on z during evaluation because of responsiveness issues mentioned in 3.3.2.

AlphaStar Supervised and AlphaStar Mid agents were evaluated by playing 30 and 60 matches respectively starting from a newly created account. AlphaStar Supervised's rating was found to have approximately converged after those 30 matches. Evaluation of AlphaStar Mid was halted after 60 games because its anonymity assumptions were violated after 50 games. Players had identified the accounts AlphaStar Mid used by analyzing irregularities in AlphaStar's play, and were taking advantage of the matchmaking system

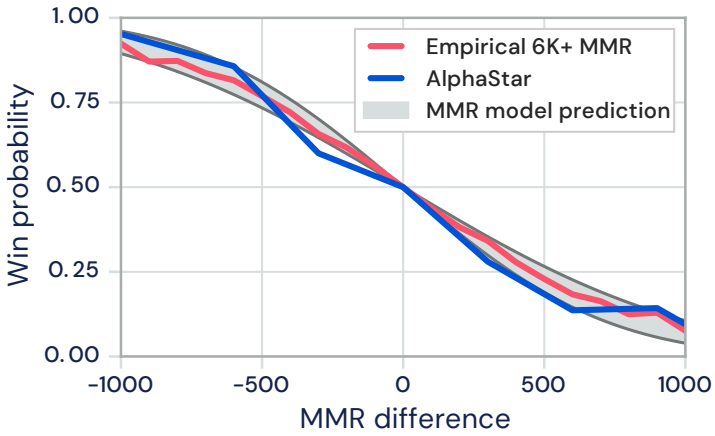

Figure 3.4: Win probability versus gap in MMR. The shaded grey region shows MMR model predictions when players' uncertainty is varied. The red and blue lines are empirical win rates for players above 6,000 MMR and AlphaStar Final, respectively. Both human and AlphaStar win rates closely follow the MMR model. [92]

to target it. This meant both that players began the game knowing they were playing against AlphaStar, and therefore would play differently from normal, and players would deliberately play against AlphaStar multiple times, moving the evaluation into a different repeated trial domain.

Because of the exposure from AlphaStar Mid, AlphaStar Final was evaluated on another 30 games regularly switching accounts to avoid identification. Each account had an MMR seeded at the average MMRs of AlphaStar Mid evaluations. Battle.net's MMR system could not compute AlphaStar Final's ratings because the games were played over multiple accounts, so we computed the ratings based on a maximum-likelihood estimate of the underlying probabilistic model. The maximum likelihood estimate was validated on the last 200 matches of profession StarCraft II player Dario "TLO" Wünsch, yielding an estimate of 6334 compared to the value reported by Battle.net 6336, and validated on the matches AlphaStar Final played in Figure 3.4.

Results of this evaluation can be seen in Figure 3.5. AlphaStar Supervised reached an average MMR rating of 3699, within the top 16% of human players. AlphaStar Mid's rating did not fully converge, so its MMR is unreliable.

AlphaStar Final achieved a rating of 6048 for Terran, 5835 for Zerg, and 6275 for Protoss, higher than 99.8% of active human players and within Grandmaster level in all three races. Results of AlphaStar Final in individual StarCraft race match-ups can be seen

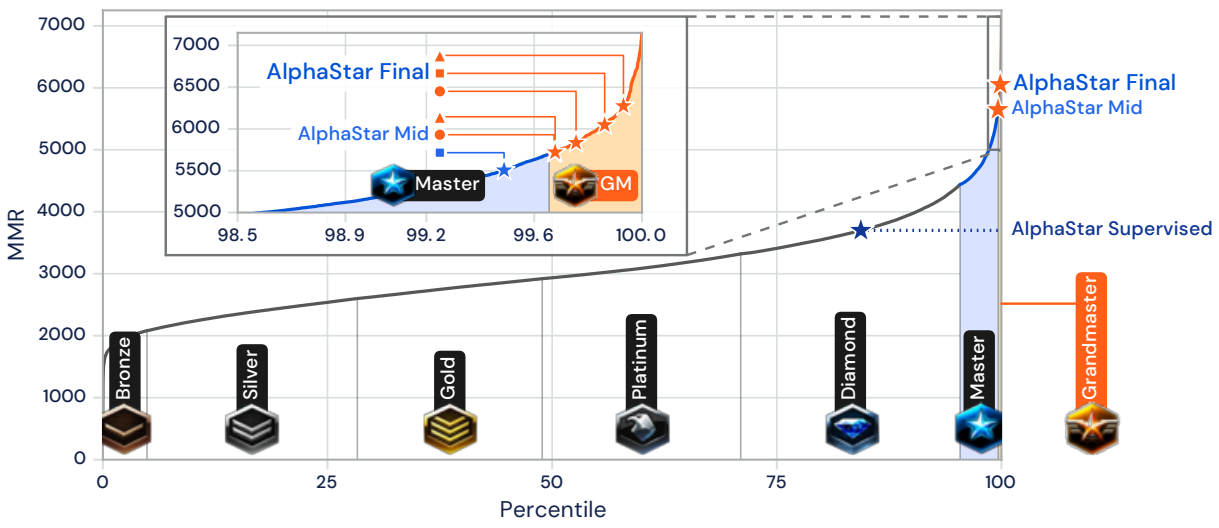

Figure 3.5: On Battle.net, StarCraft II players are divided into seven leagues, from Bronze to Grandmaster, according to their ratings (MMR). We played three variants of AlphaStar on Battle.net: AlphaStar Supervised, AlphaStar Mid, and AlphaStar Final. Protoss, Terran, and Zerg are denoted by a triangle, square, and circle respectively. The supervised agent was rated in the top 16% of human players, the midpoint agent within the top 0.5%, and the final agent, on average, within the top 0.15%, achieving a Grandmaster level rating (denoted in orange) for all three races. [92]

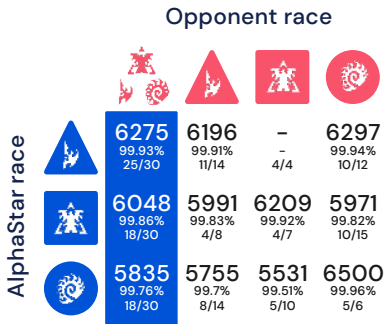

Figure 3.6: MMR ratings of AlphaStar Final per race (from top to bottom: Protoss, Terran, Zerg) versus opponents encountered on Battle.net (from left to right: all races combined, Protoss, Terran, Zerg). Note that per-race data are limited; AlphaStar won all Protoss versus Terran games. [92]

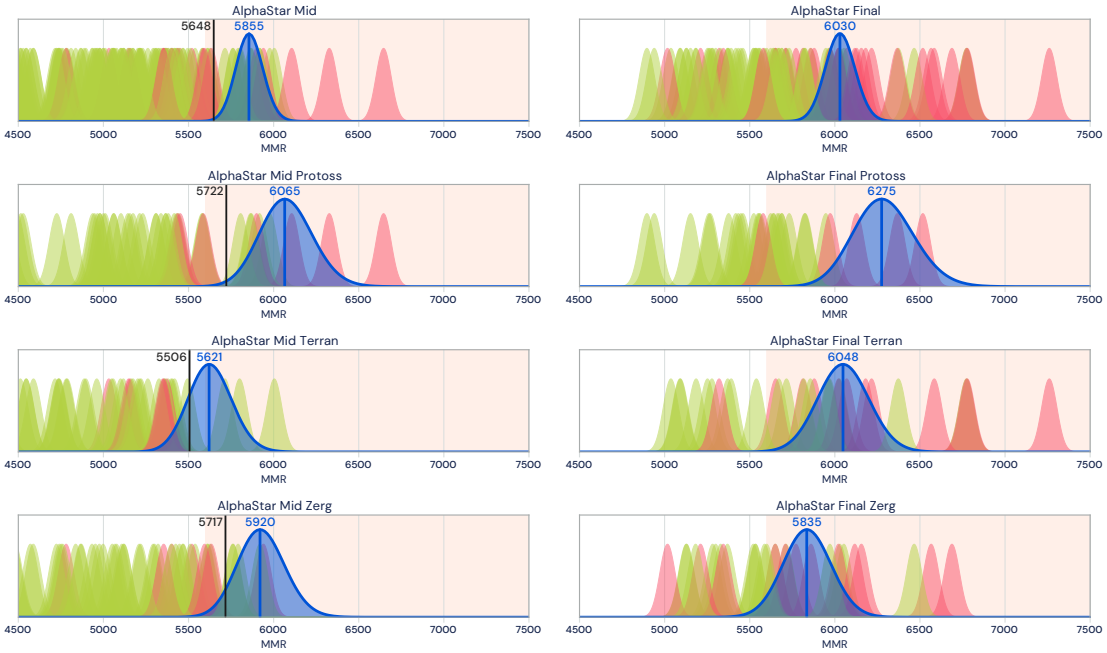

Figure 3.7: Battle.net performance details. Visualization of all the matches played by AlphaStar Final (right) and matches against opponents above 4,500 MMR of AlphaStar Mid (left). Each Gaussian represents an opponent MMR (with uncertainty): AlphaStar won against opponents shown in green and lost to those shown in red. Blue is our MMR estimate, and black is the MMR reported by StarCraft II. The orange background is the Grandmaster league range. [92]

in Figure 3.6. Account switching managed to keep AlphaStar Final anonymous for the duration of the remaining trials, though some of the games were identified in later weeks.

Additional details of the performance of each instance of AlphaStar Mid and AlphaStar Final can be seen in Figure 3.7. Note that because of how MMR is calculated, AlphaStar Final Protoss was placed at 6275 MMR despite not beating any players above 6000 MMR because of consistently high win-rates against other players.

3.7.2 BlizzCon Matches

AlphaStar was evaluated in a more casual and open setting in November 2019 at BlizzCon, Blizzard's yearly convention for their games. At the convention, we set up a demo sta-

tion with up to 20 machines where attendees could play against AlphaStar in non-blind, repeated trial conditions (and therefore different conditions from the Battle.net evaluation). Two versions of AlphaStar were available to play: AlphaStar Basic, a distribution of agents including AlphaStar supervised and weak main and league exploiters, and AlphaStar Advanced, a distribution of agents including AlphaStar Final, AlphaStar Final with the statistic z, and strong league exploiters.

These matches were under less scientific conditions because they took place in the convention hall, and though players could customize some settings (e.g. hotkeys) they were using different keyboard, mice, and headphones from those they were normally used to playing with. Matches also took place on the same balance patch and maps as those used in the Battle.net evaluation, which at that point were several months out of date. Many players played against AlphaStar repeatedly, testing strategies and trying to find systemic weaknesses.

AlphaStar Basic won approximately 90% of its 500 games at BlizzCon, and AlphaStar Advanced won approximately 95% of its 500 games. Notably, professional player Joona "Serral" Sotala, ranked number one in the 2018 StarCraft II World Championship Series, played 5 games against AlphaStar Final (not AlphaStar Advanced) at BlizzCon under these conditions and AlphaStar won four out of the five games.

3.7.3 Demonstration Matches

An earlier version of AlphaStar played five matches against Dario "TLO" Wünsch and Grzegorz "MaNa" Komincz in December 2018, although TLO played Protoss instead of the race he plays professionally. AlphaStar won all ten games in these demonstration matches, although it lost a follow-up game in January that added an early camera prototype. This earlier version differed from the final one in that it only played with and against one of the StarCraft races, it had a less limited and less human-like interface (for example, it could act inhumanely often), league training was manually managed and agents were manually selected, and the architecture and losses were iterated on several times after the matches [91]. The goal of those matches was to measure what progress the project had made.

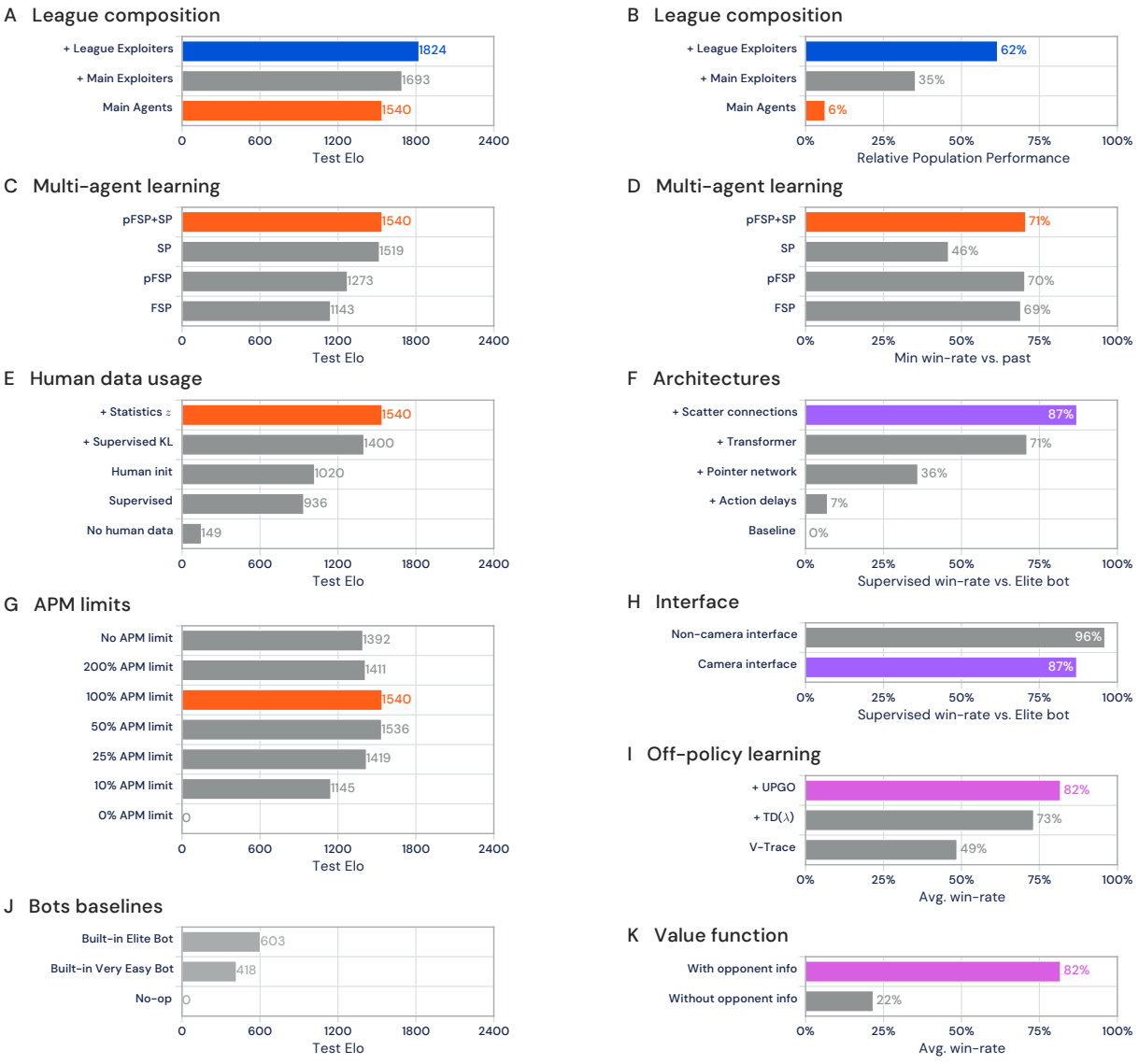

Figure 3.8: Ablations for key components of AlphaStar. These experiments use a simplified setup: one map (Kairos Junction), one race match-up (Protoss versus Protoss), reinforcement learning and league experiments limited to 10^{10} steps, only main agents, and a 50%–50% mix of self-play and pFSP, unless stated otherwise. The first column shows Elo ratings [26] against ablation test agents (each rating was estimated with 11,000 full games of StarCraft II). [92]

3.8 Analysis

We ran several additional internal ablations and analyses of AlphaStar, the results of which can be seen in Figure 3.8. Analysis of that figure follows.

(a) Figure 3.8 **a** shows the Elo improvement of main agents when adding exploiters into league training.

(b) Figure 3.8 **b** shows the relative population performance of all league agents, a measure of exploitability, when adding exploiters.

(c) Figure 3.8 **c** shows the Elo of main agents trained alone according to different matchmaking algorithms.

(d) Figure 3.8 **d** shows the minimum win-rate of main agents trained alone according to different matchmaking algorithms against past versions of themselves, averaged over training. Note that self-play agents can learn rapidly, but forgets how to defeat its past versions. More detailed analysis can be seen in Figure 3.9. pFSP decreases exploitability against rare counter strategies that a uniform mixture could more easily avoid considering.

(e) Figure 3.8 **e** shows the impact of different human exploration strategies on Elo. Supervised is supervised learning by itself, human init is initializing from the supervised checkpoint followed by RL, and supervised KL is both initializing and then continuing to distill from the supervised checkpoint in RL.

(f) Figure 3.8 **f** shows the change in win-rate against the hardest non-cheating built-in AI when training a Protoss supervised network using different neural network architecture components. Action delays refers to the action argument requesting when next to act, seen in Table 3.2.

(g) Figure 3.8 **g** shows the Elo performance when trained with different action-per-minute (APM) limits relative to the one used in AlphaStar. Lower APM limits decrease performance because agents are not able to execute enough actions. High APM limits also decrease performance because agents learn to control individual units. This shifts the strategy distribution away from human ones (making evaluation against other agents more difficult) and is harder to learn because of competition against the supervised KL and because there are more timesteps per game (the more often an agent acts the more often it observes) which makes the learning problem harder.

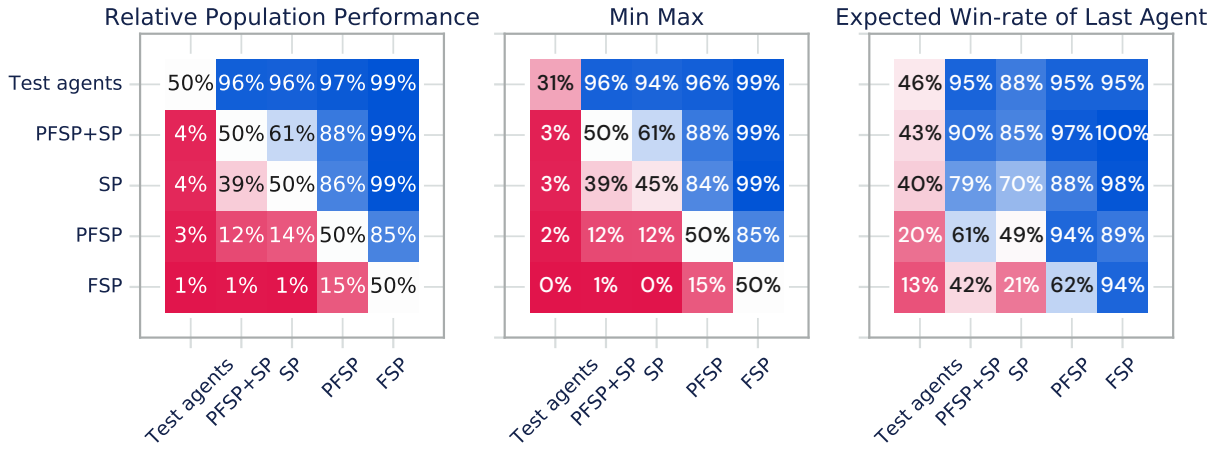

Figure 3.9: More detailed analysis of Figure 3.8 **c, d**, comparing different populations of agents in terms of relative population performance, minimum win-rate across the opposing distribution (which measures exploitability), and win-rates of final agents. [92]

(h) Figure 3.8 **h** shows the change in win-rate against the hardest non-cheating built-in AI when training a Protoss supervised network using different camera interfaces.

(i) Figure 3.8 **i** shows the average win-rate improvements of different RL off-policy corrections against a fixed set of opponents.

(j) Figure 3.8 **j** shows the Elo of the hardest and easiest non-cheating built-in AIs, and an agent that never acts which anchors the scale.

(k) Figure 3.8 **k** shows the average win rate improvements of using privileged information in the RL value function.

The distribution of units constructed by Protoss AlphaStar Supervised and Protoss AlphaStar Final are shown in Figure 3.10. AlphaStar Supervised is capable of building every unit in the game, which indicates it is effectively capturing a diverse range of human strategies.

Multi-agent learning dynamics were evaluated through both a round-robin tournament of all players produced throughout league training, shown in Figure 3.11, and through play against a held-out set of validation agents trained to exhibit specific human strategies, shown in Figure 3.12. Main agents continued to improve in both internal Elo and robustness against the held-out strategies as training progressed. Main exploiters in Figure 3.11 grew

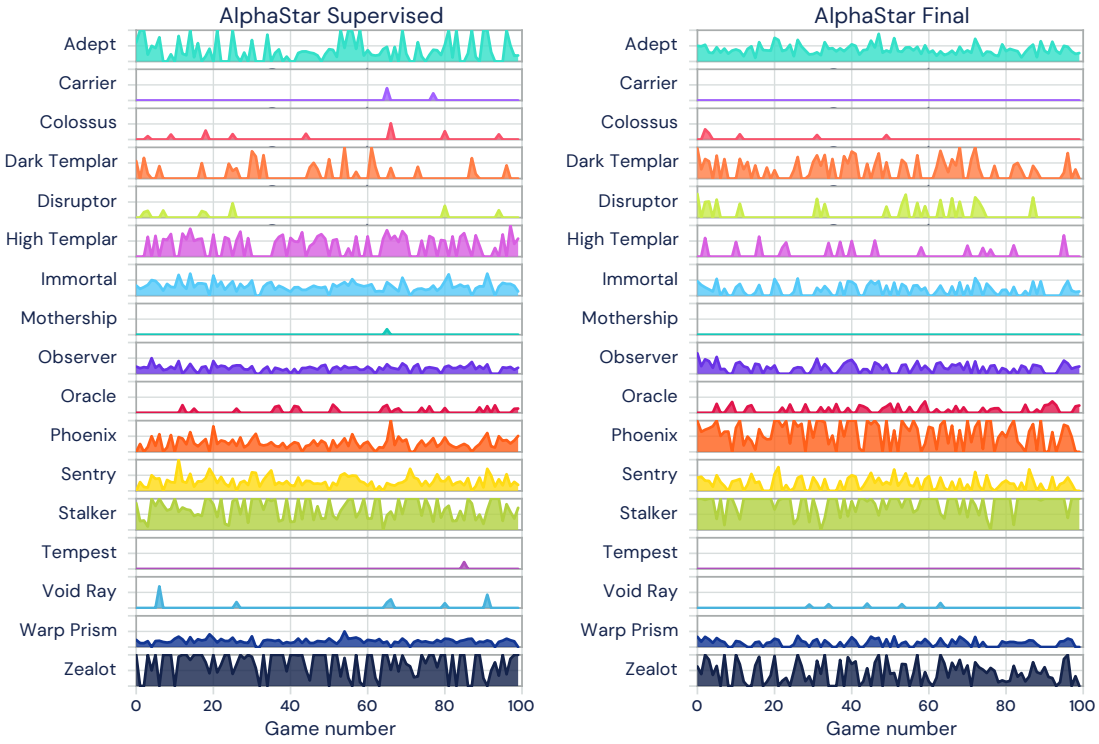

Figure 3.10: Distribution of units built in a game. Units built by Protoss AlphaStar Supervised (left) and AlphaStar Final (right) over multiple self-play games. AlphaStar Supervised can build every unit. [92]

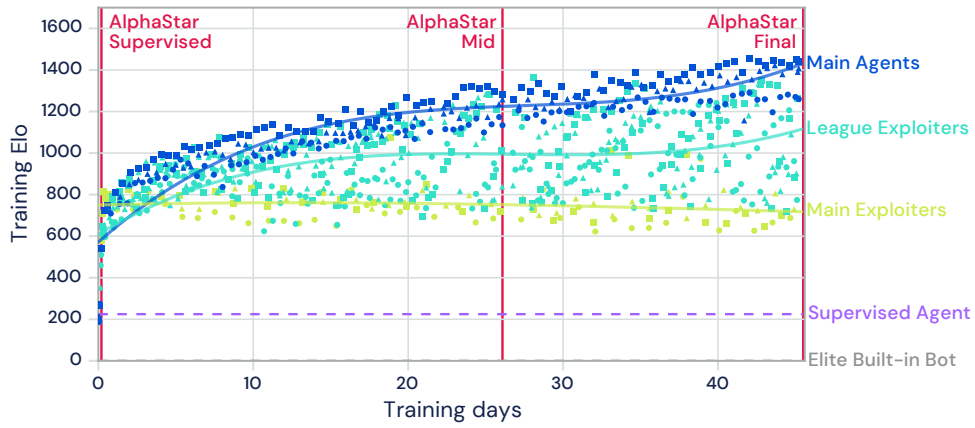

Figure 3.11: Training Elo scores of agents in the league during the 44 days of training. Each point represents a past player, evaluated against the entire league and the elite built-in bot (whose Elo is set to 0). [92]

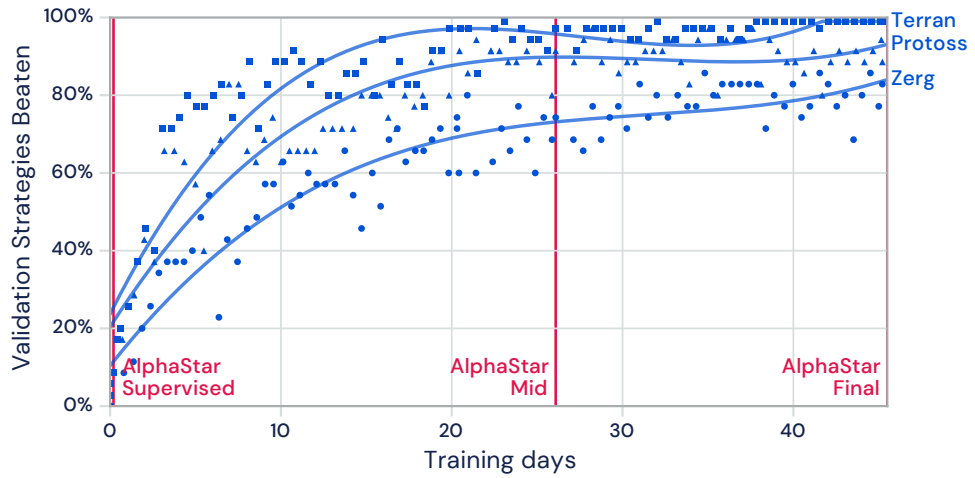

Figure 3.12: Proportion of validation agents that beat the main agents in more than 80 out of 160 games. [92]

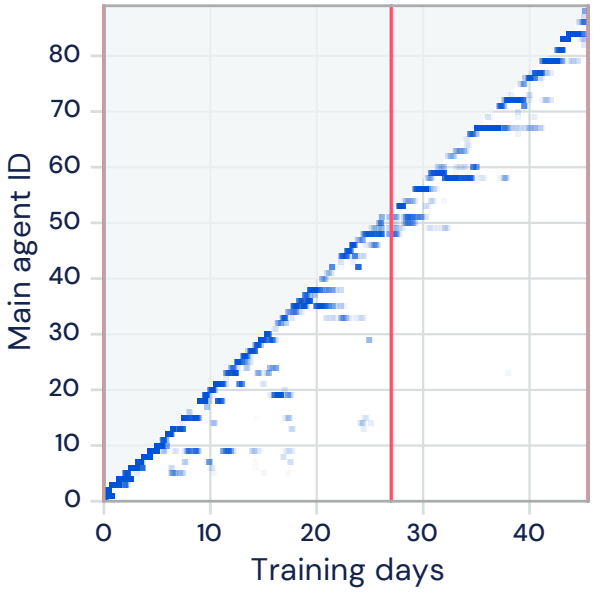

Figure 3.13: The Nash distribution (mixture of the least exploitable players) of the players in the league, as training progressed. It puts the most weight on recent players, suggesting that the latest strategies largely dominate earlier ones, without much forgetting or cycling. For example, player 40 was part of the Nash distribution from its creation at day 20 until 5 days later, when it was completely dominated by newer agents. [92]

weaker as training progressed, indicating that it became harder to find new strategies that could exploit the main agents.

Figure 3.13 shows the Nash equilibrium of main agents through training. As expected, new agents continuously entered the Nash equilibrium and old agents left as they became dominated. Old agents rarely re-enter the Nash after leaving it, which indicates that league training successfully avoids degenerate cycles.

Figure 3.14 shows a portion of the payoff matrix used during league training. Each row is a different player in the league, with age within an agent type increasing further down the matrix. As we would expect, main agents trained to be the strongest agents in the league have transitive payoffs, continuing to win against older agents, while exploiters who have different objectives do not share this transitivity.

Figure 3.15 shows the units constructed throughout training, which gives insight into strategic diversity as each unit has a specific purpose. The unit compositions change

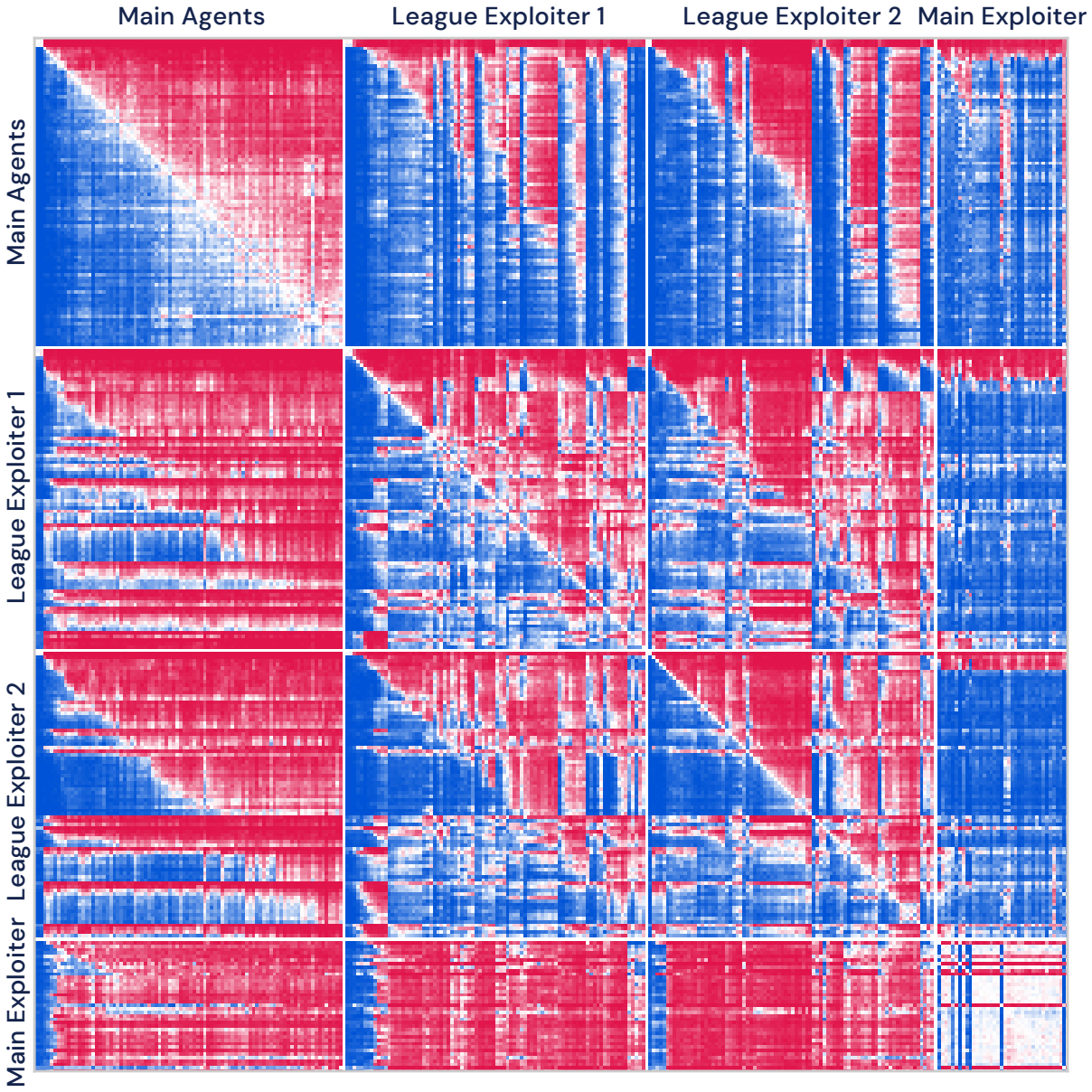

Figure 3.14: Payoff matrix (limited to only Protoss versus Protoss games for simplicity) split into agent types of the league. Blue means a row agent wins, red loses, and white draws. There are around 3,000,000 rock–paper–scissor cycles (with requirement of at least 70% win rates to form a cycle) that involve at least one exploiter, and around 200 that involve only main agent. [92]

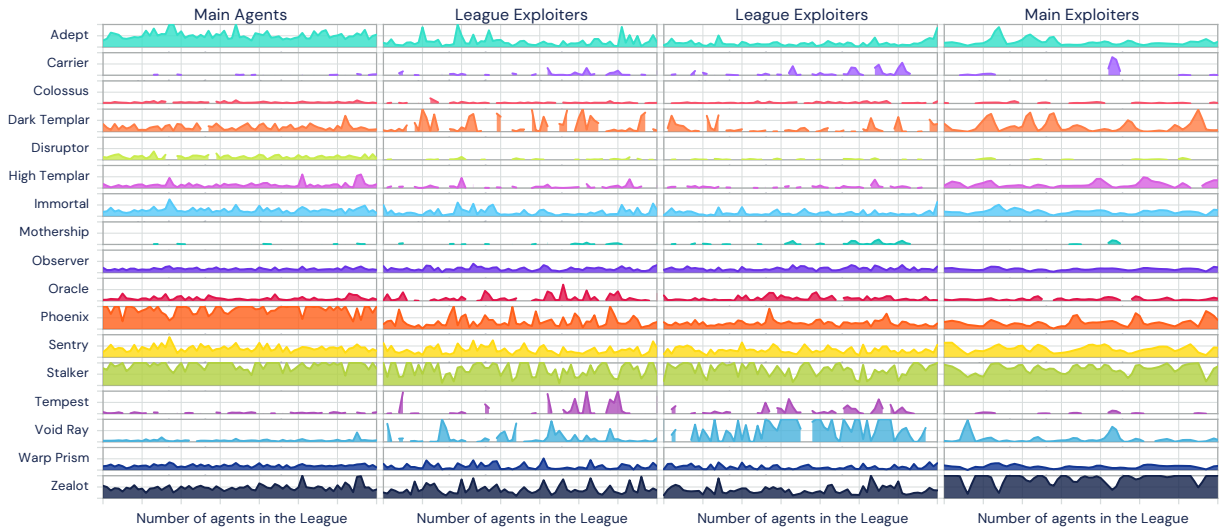

Figure 3.15: Average number of each unit built by the Protoss agents over the course of league training, normalized by the most common unit. Unlike the main agents, the exploiters rapidly explore different unit compositions. Worker units have been removed for clarity. [92]

throughout training, even late in the training of the main agents which never reset back to initial checkpoints, indicating that meaningfully different strategies continue to be explored and agents have not converged by the end.

Chapter 4

Complexities in Deep Learning Game Interfaces

4.1 Complexities in Game Interfaces

Artificial intelligence agents cannot feasibly interact with computers and games in the same way humans can. Even a robot that has sensors to view a computer screen and fingers to press keys will perceive the screen display differently from humans, and the gulf between humans and agents grows much wider when interacting through a non-physical interface. Direct interfaces are necessary because humanoid manipulation is a difficult unsolved problem that can be disentangled from the other research directions that games present.

However, this means that the interfaces that humans use must be adapted in some way to APIs that can be used by computers. For board games, this can be relatively straightforward. The interface can expose the state of the board and all pieces on it, though this can already have complexities of whether or not the agent can see the clock (for games like chess or Go) or the face and emotional state of opponents (particularly for social games like Poker or Diplomacy). Interface design for many games does not have a clear solution, though there are few Poker agents that look at their opponents faces, and there are two areas of consideration that go into developing an interface: what should be exposed in an interface and how should it be exposed.

4.1.1 What should be exposed

In simpler domains like chess or Poker, it is apparent what data a human has access to and what actions they can take when playing the game, and the essential question is instead what is and is not relevant for research purposes. However, while questions about relevance are certainly present in more complex games like StarCraft II, determining what humans see and do is much more complicated and requires a significant amount of domain knowledge. In modern video games:

1. Information may be conveyed by appearances. For example, in StarCraft II, there is a building called a bunker that units can be placed within. If units are within the bunker, internal lights present in the bunker sprite will be turned on, and otherwise the lights will be turned off. There are dozens of examples of this in StarCraft II, from research upgrades changing the sprites of units in subtle ways (e.g. adding wings to Zerglings if a movement upgrade for them is researched) to showing a distorted image of the unit a Stargate building is currently constructing.

2. Information may be transmitted through multiple streams. For example, Nydus worms trigger a sound alert whenever an enemy constructs them, even if they are not within the section of the map a player can see. Similarly, first-person shooter games generate footstep sounds when players move, and because the sound is directional it is possible to figure out the relative orientation of those players even when they are behind walls or floors.

3. Information may be from a side channel that does not affect the game-theoretic solution. These include chat messages and voice communication present in many games. These also include external resources, like wikis or videos of other players, that provide information about the game.

4. Interfaces may be player-specific. For example, the screen aspect ratio usually affects how much information is exposed at once, because using a constant aspect ratio with empty borders would look bad to most players. This affects both observations, but also the set of available actions. Also, many games allow game modifications that expose extra additional information (e.g. many World of Warcraft UI mods [5, 23]), and players may use different input mechanisms (e.g. controllers vs. keyboards).

5. Information may be hidden in the interface. For example, by selecting an enemy unit in StarCraft, it is possible to see some of the research upgrades the unit has active.

6. Information may be intentionally obscured. For example, invisible units in StarCraft cannot be targeted and many of their features are hidden, but it is possible to see their position and unit type because they distort what is behind them in ways that are different for each unit type.

7. Actions may be technically possible, but infeasible for humans. Many games have vulnerabilities where inputting an exact sequence of actions can reprogram the game itself which has been used to great effect in the speedrunning community (which aims to complete games as fast as possible) [54]. In StarCraft II, while it is possible to issue commands through the side minimap, humans find it difficult to click precisely on the small area and tend to avoid it for many actions.

8. Human skill and attention affects the interface. More experienced, enervated players will tend to react more precisely and faster to perceived stimuli, while lower skill players may not even notice subtle appearance changes. Experienced players will also tend to act more often.

Each of these factors can require careful thought and significant domain experience, and there are many equally valid choices for what should be exposed.

4.1.2 How it should be exposed

However, even when what should be exposed is decided, how observations and actions should be exposed to an agent requires additional thought. These are beyond the, admittedly complex, difficulties inherent in implementing a programmatic interface for real systems. Some additional decisions that need to made are:

1. Some observations and actions may take more effort to use than others. Clicking on a precise region of the screen, in a first-person shooter, is usually harder than pressing a key. In StarCraft, even if players can identify an invisible unit, detecting the characteristic shimmer is harder than seeing a normal unit (one reason why professional players often play on the lowest possible graphics settings).

2. Interfaces can be complex with one-off or ad-hoc data. In modern games, information can appear in hard-to-generalize ways, like knowing whether a bunker is occupied in StarCraft II, or knowing what unit at Stargate is producing.

3. Some observations are inferred or remembered by humans. In a first-person shooter, as in many genres, it is possible to be attacked by someone not currently visible or outside your field of view. The existence of those entities is inferred by the lost health and graphical screen effects. Similarly, observations may be remembered by humans in partially observable domains, such as enemy research upgrades in StarCraft which are only accessible when the corresponding enemy entity is within the visible area.

4. Interfaces change over time. Obviously, updates to games can change the interface itself. However, even on the same game version, the interface can constantly shift due to factors like latency between the player's machine and a central server.

4.2 Deep Learning Interfaces

At this point a clear solution is to have an interface as close to humans as possible. A deep learning agent could view the same pixels (or rather a 3D tensor of the color values for each location), hear raw directional audio waveforms, and input keystrokes, mouse movements, and mouse clicks. Many issues are naturally addressed by such an interface, though some issues still remain, like whether or not an agent is allowed to watch online video tutorials while playing the game, differences between interfaces different players use, and the ability for computers to detect minute differences in appearance more easily (and generally process information faster) than humans.

The main reasons to avoid this are that using such an interface alters the problem by removing much of the focus that makes the problem appealing for research, by decreasing the performance of final systems, and by dramatically increasing the amount of resources necessary to address the problem.

First, humans are wonderfully complex systems capable of creating and integrating representations across multiple modalities. Unfortunately, artificial intelligence systems are not yet at that level and, while deep learning has made tremendous strides in representation learning, an agent that needs to interact with its environment in the same ways as humans means tackling many more difficult problems simultaneously. Games and simplified environments are useful for research precisely because they allow researchers to focus on specific problems, like hierarchical planning or partial observability, without simultaneously tackling other domains, like humanoid locomotion or image processing. Even though it can make creating an interface more complex, using a structured, clearly defined observation space and action space trivializes some aspects of the representation learning

challenge, the same way that giving an image classifier direct class information about the objects in the images it classifies is an almost meaningless problem.

Second, adding additional challenges to a neural network will make it perform worse than a well designed agent without those challenges. This is not simply a factor for producing exciting sounding results but also, similar to the first point, a factor relevant to making any progress at all. A certain amount of performance is necessary for determining failure modes of an integrated learning system. If that system has trouble identifying the entities within the game and cannot beat the weakest of opponents, the fact that a network architecture cannot position itself in an environment will never become apparent. This was very common in AlphaStar. It was only when the agent reached a certain baseline level of performance and we investigated its play did we notice certain deficiencies, such as missing observations in the interface or inability to combine spatial and entity information. Lower performance can obscure flaws in a research direction.

Third and most simply, using interfaces more similar to humans tends to increase the amount of resources needed to run experiments. These can come from a variety of factors. For example, information needs to be rendered which increases the amount of simulation compute needed, a harder learning problem increases the amount of training needed to address it, and robots are expensive and cannot generate much experience, in the extreme case of acting in the real world. All of these are barriers to fast and flexible research iteration.

4.2.1 Low-level Interface Design

The low-level designing of interfaces for deep learning, that represent a problem in a way that is straightforward for a neural network to learn, is a subject not typically covered in research. Instead, learning tends to happen through osmosis in a combination of examples, tutorials, and experience that generates a toolbox of tricks and ways of preprocessing data. Low-level design is relevant for designing a new interface and so will be briefly covered here.

Fundamentally, an interface should be uniform, separated, and not misleading. Uniformity means that inputs to a neural network should tend to have the same ranges and variances so as to not take precedence over one another. The exact ranges depend on the network architecture (some networks can accommodate less uniform inputs [43]) and initialization (e.g. Glorot initialization [32]), but normally fall within $[-1, 1]$.

Separation means that the interface should try to have as few axes of variation within an input as possible in order to simplify the learning problem. For example, an image has

several more axes of variation than that image decomposed into several 2D layers, where each layer has information about particular types of objects present within that image.

Not misleading means that structure within network inputs should be meaningful. The most common example of this is one-hot encoding [33]. Instead of representing categorical variables as single real values that vary from $[-1, 1]$, categorical values are instead represented as 1D vectors where a 0 entry means that the object does not fall in the category and 1 means that it does. Using a real value for, for example, the city a person is born in could be misleading because it imposes the inductive bias that two cities that are close together in the real number scale would be similar.

These design principles for interfaces often take precedence over other desirable factors, such as conciseness; separating an input into several layers or one-hot encoding an input is preferable yet much more verbose than the alternatives.

Chapter 5

Fairness in Games

Interface design is complex, but why does research in games not use as permissive an interface as possible that exposes all observations and actions (within the confines of the rules of the game) and manipulate that interface to be as easy as possible for research. Agents could act hundreds of times per second with inhuman precision, perceiving the game through a resolution that exposes more information than humans would typically see. There is the informal notion that some methods may be unfair for comparisons to humans, which is certainly a concern for fair evaluation, but another concern is that research in these distorted domains is less meaningful. To explore fairness, we will first examine why real games are used at all and why fairness matters from a research perspective, followed by various notions of what it means for an agent to play games fairly.

5.1 Real Domains for Research

As a brief aside, we will first discuss the advantages of real domains over synthetic domains created solely for research. Such synthetic domains often focus on a single research direction (for example, trying to create a system that can find patterns in data [21] or understand basic language [6]), and avoid most interface and fairness issues entirely. These synthetic domains are extremely useful for iterating on specific problems or testing ideas without using many resources.

Nevertheless, such domains may represent problems that are not relevant to the real world and thus solutions found in such domains may not scale when applied to more complex ones. Even if designed to test a single competency like hierarchical planning,

without careful human testing the domain could require a different solution entirely. Well-designed domains can also produce solutions that do not apply to similar larger instances, common when trying to apply a technique developed on MNIST [52] to a larger domain like ImageNet [70]. Many problems are also created by the same researchers who use them and some research areas also have a multitude of such synthetic problems, both of which can make reproducing research and comparing results difficult. Real-world domains are recognizable and evaluate factors that are important for human intelligence.

5.2 Fairness for Research

We can now address why concerns about fairness are important from a research perspective, beyond comparisons with humans. Changes to game interfaces, but especially changes that make the game unfair, fundamentally change the learning objective and in extreme cases can shift the research objectives away from what made the real domain attractive for research in the first place.

As mentioned earlier in 1, one reason chess is interesting for research is that it involves planning multiple moves into the future. Chess no longer becomes a useful domain for planning, however, if time limits are significantly changed from human ones (because no time limits turns it into brute-force search and strict time limits decrease time available for planning). While it is still possible to approach this modified game with decision rules like Minimax [98], it no longer becomes necessary. Even smaller changes can change the problem; giving the relevant upgrades a unit has instead of the sprite of the image in StarCraft II means that while long-term planning and strategic complexity is still present, an agent no longer has to learn to detect some minute changes in its environment.

Domains should be evaluated in the regime they are designed for, which often means using as fair and human-like an interface as possible, so they exhibit the research complexities they are intended for. Of course, there is a trade-off between having as fair interfaces as possible described in 4.1, and interfaces that make research feasible described in 4.2. To design an interface that balances these competing factors, it is important to understand exactly which complexities a domain are and aren't focused on, as well as the different ways in which an interface can be fair.

5.3 Notions of Fairness

There are many different ways in which an agent's capabilities within a game can be seen as unfair, and each can be important for different purposes. While these notions are related to complexities involved in game interfaces described in 4.1, deciding what notions of fairness are and are not important often guides interface design choices considered there.

- An agent can have access to hidden information or actions a human would never be able to access. For example, "map hacks" allow players to see everything in the game, including entities that would normally be hidden (behind obstacles or outside a player's field of view). Alternatively, by directly modifying the game logic it is often possible to change the game state in a way that is not allowed by normal play. Almost all game interfaces disallow this kind of interaction.

- An agent could gather information faster than humans. For example, the OpenAI Five, similar to a pre-publication versions of AlphaStar, could simultaneously view every unit on the map, their items, and when they could next use their cooldowns, subject to game visibility rules mentioned in the first point [8]. This example would be akin to players manually clicking to view every region of the game and every unit each observation. Less extreme examples would be how agents can typically process an entire observation simultaneously, instead of viewing one part of a screen at a time with foveal vision.

- An agent can receive information that humans would need to remember or learn. For example, an agent could refer to game tutorial information while it executes, or continue to receive observations of the last positions of a unit even when that unit is not visible.

- An agent can act or react faster than humans. Agents might immediately locate and respond to stimuli, as in aimbots, or simply execute tens of thousands of actions per minute, as in many bots designed for real-time strategy games.

- An agent can use more energy, FLOPS, or other resources than humans. Humans thinking and learning is remarkably efficient. Deep learning agents often require many orders of magnitude more resources to learn than humans, and slightly fewer orders of magnitude more resources to execute.

Whether or not each of these notions of fairness are relevant depend on the application, and adhering to one set of notions is not necessarily better than adhering to any other.

Chapter 6

Interface in AlphaStar

We will now look at AlphaStar as a case study of applying the considerations and discussions in the previous chapters to develop a game interface.

When developing this deep learning game interface, we examined our research objectives, used those research objectives to shape which notions of fairness from 5.3 were important, and applied those notions to the many inherent interface complexities described in 4.1.

As mentioned in 2.3.1 some interesting challenges in StarCraft, and the challenges we focused research on, were the raw complexity (the large action space, length of games, and different modalities of data), partial observability, acting in a real-time setting, complex strategic possibilities, exploration, and human-alignment. The goal was also to produce research insights generally applicable to other problems, not necessarily producing the best StarCraft agent possible.

With these in mind, we approached the agent's capability with respect to each notion of fairness as follows:

- Hidden information or impossible actions: These were never allowed because they would, for example, modify the strategic landscape.
- Faster information gathering: Partial observability and acting in real-time are important challenges within StarCraft, and gathering information faster would subvert that. Mild violations of these were allowed when they would simplify the interface (for example, avoiding the need to select enemy units in order to determine their basic research upgrades because the interface has no notion of selection) and extreme

violations were not allowed (for example, a camera view that encompasses the whole map instead of a section). Because of the complexities of the interface, there were many gray areas.

- Remembering or access to information in external resources. Though important in StarCraft, memory was not a major research focus so we often allowed agents to receive at each step information they had gathered in the past. Information not present in the game was used on occasion, though game-specific hand-coded information would lead to less generalizable insights and was therefore avoided when possible.

- Superhuman action rate and reaction time. The action and reaction times present in StarCraft are a key part of its nature as a real-time game, and changing this aspect dramatically alters which strategies are viable. For example, units that take a fraction of a second to target another unit are no longer relevant when the targeting can be responded to by immediately moving the target and all units around it. For this reason, we attempted to limit our action and reaction times to those of top professional players.

- Energy and resource usage. Though we were limited in the scale of experiments we could run and we made many optimizations to improve efficiency, direct comparisons to humans were not a concern.

These fairness principles and research directions played a key role in shaping the exact interface which will be examined in more detail throughout the rest of this chapter. The interface was developed in consultation with and using the expert domain knowledge of Blizzard employees and professional StarCraft II players, and importantly (because there are trade-offs inherent in the interface that are hard to evaluate quantitatively) AlphaStar's play under this interface was professional-player approved.

6.1 Camera

Humans interact with StarCraft through a camera interface, where a single region of the map is viewed with at any point in time and only high-level information is given about entities outside that region through a small minimap. This can be seen in Figure 2.1 where only a small segment of the entire map is focused on, denoted on the minimap in the bottom-left by a white-bordered polygon. Actions targeting a unit can generally only be

issued within the camera view, and actions that target a location can be issued through the minimap or the camera view, though humans lack the precision to issue most commands through the minimap.

AlphaStar used a similar camera-like interface. Agents only receive complete information about entities within their current camera view and can move around their camera as an action, and some actions can only be issued if their target is within the camera. Thus AlphaStar too has an attention-economy, trading off issuing commands for gathering information and allowing future actions.

The camera view is a 32x20 game unit sized rectangle. Because the camera view varies based on human monitor resolution, AlphaStar's dimensions were chosen by analyzing where players tend to issue commands relative to the center of the camera, creating a bounding box that captures the majority of human actions. Similarly, which actions could be feasibly issued when the target was outside the current camera view (because they require more precision than actions within the camera) was decided by sorting the possible action types based on how often humans issued that action to somewhere outside their camera, and allowing all actions above a cutoff. Related, because AlphaStar uses a single spatial argument for the entire map, it can target locations outside the camera more precisely than humans and locations inside the camera less precisely.

Because of the protocol we selected, there is also no notion of selecting units, or assigning units to control groups. Humans typically select units before they issue commands to them, and save groups of units to control groups (seen just above the bottom-middle bar in Figure 2.1) for more convenient selection in the future. Fully-flexible unit selection is only possible within the camera view. AlphaStar, however, is capable of issuing commands to arbitrary subsets of units, no matter where they are on the map, although it does not seem to exploit these capabilities.

6.2 APM Limits

AlphaStar is limited to acting at a rate roughly equivalent to weak professional players by a monitoring layer. Like the attention-economy for the camera, this introduces an action-economy that requires certain actions be prioritized.

Counting the number of actions-per-minute (APM) a player uses is a complicated subject in StarCraft. Some actions are repeated actions, may take more effort to execute than others, may require precision, or may be no-ops with no effect. StarCraft II therefore

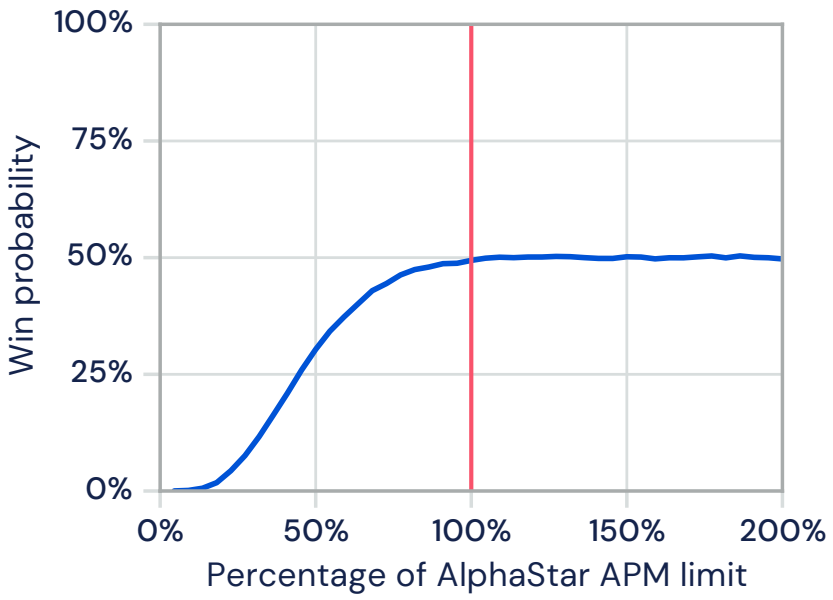

Figure 6.1: Win probability of AlphaStar Supervised against itself, when applying various agent action rate limits. [92]

tracks a metric called effective actions-per-minute (eAPM) that removes many repeated and no-op actions, and serves as a better metric.

Even ignoring the difficulty of executing different actions, the interface we use has no simple conversion to either eAPM or APM; although the game reports both statistics for our agents, those metrics (used throughout the paper) are overestimates. Agent actions are decomposed into 2-3 game actions. For example, casting a force field would require selecting a unit that can create force fields, activating force field targeting, and selecting the location to place the force field. Notice that if another action was issued that used the same units, the selection would happen an additional time even though it is not necessary to re-select already selected units.

Because of these complexities, and because computers can precisely execute different actions in sequence, we conservatively limited agents to executing at most 22 non-duplicate actions in every 5 second window. This action limit was selected to not decrease the performance of the supervised agent trained to follow a human policy (seen in Figure 6.1), and the in-game APM distribution was lower in all respects when compared to humans. Comparison of reported APM and eAPM can be seen in Figure 6.2. Figure 3.8 **g** shows

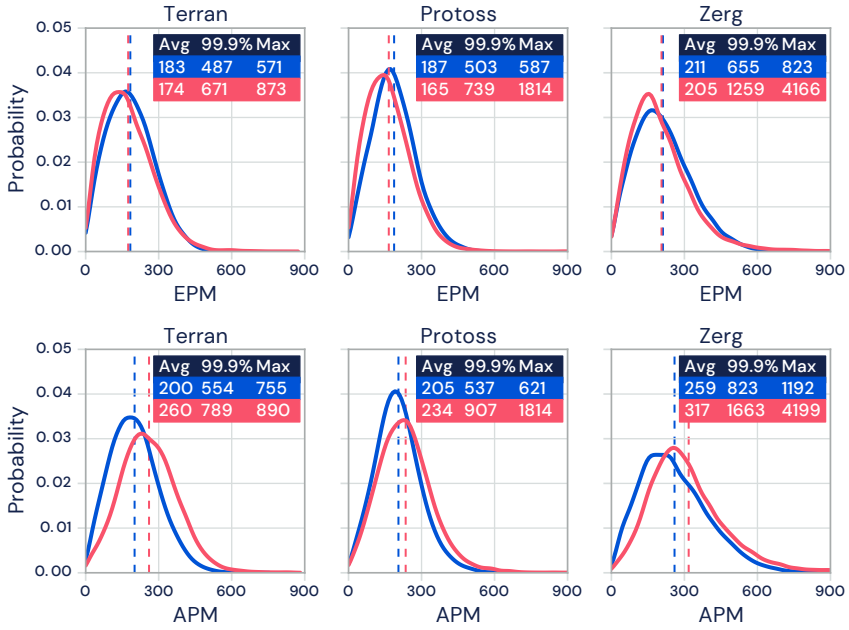

Figure 6.2: Distribution of effective actions-per-minute (top) and actions-per-minute (bottom) as reported by StarCraft II for both AlphaStar Final (blue) and human players (red). Dashed lines show mean values. [92]

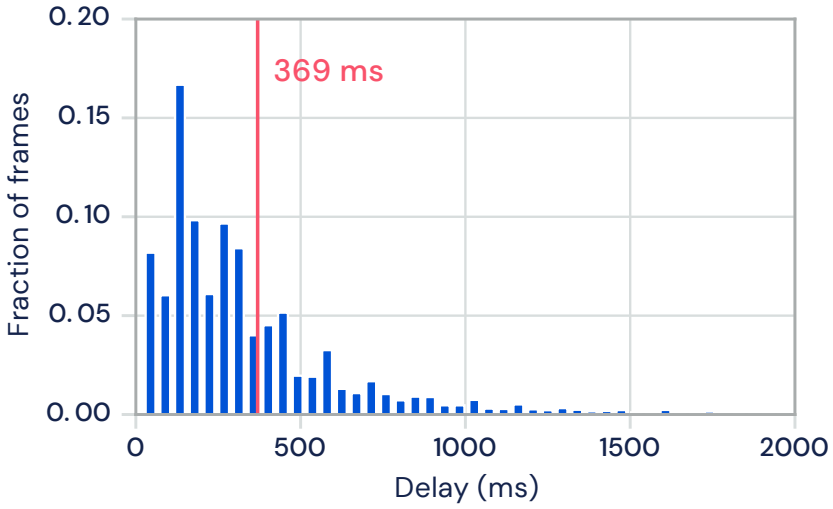

Figure 6.3: Distribution of how long agents request to wait without observing between observations. [92]

the impact of different APM limits on final performance.

6.3 Delays

Humans are limited in how fast they react to stimuli, with various experiments pointing to numbers including 250ms [45], 180ms [87], or 400ms [59] depending on the methodology and the task. AlphaStar also has two sources of delays. First, AlphaStar decides ahead of time when to next observe and therefore act, so unexpected situations can lead to a delayed response. This delay, seen in Figure 6.3 is on average 369ms but especially at the beginning of the game when there is less activity can be several seconds. Second, due to observation creation and processing, network evaluation, action processing, and network latency there is a delay between when a frame is first observed and when an action is executed. This delay, seen in Figure 6.4 is on average 113ms.

The combination of these delays leads to human-like behavior and AlphaStar does not respond to stimuli instantly, especially when the camera is positioned elsewhere. Nonetheless, the delay due to deciding when to act next is suboptimal because humans make such decisions continuously. That delay is unfortunately necessary for efficiency reasons, as otherwise every frame would need to be rendered and executed upon, and for network training,

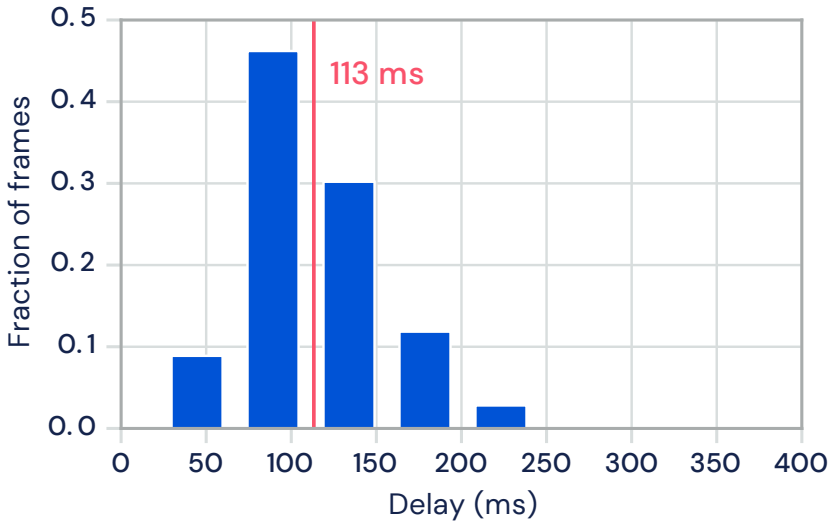

Figure 6.4: Distribution of delays between when the game generates an observation and when the game executes the corresponding agent action. [92]

as otherwise there would be many steps where an agent issues no action, complicating the optimization objective and memory capacity (seen in Figure 3.8 **f**).

6.4 Observation Details

There are many additional observation details that were considered in the interface, with aspects particularly relevant to complexities in game interfaces or fairness discussed below. The list of observations can be found in Table 3.1.

- There are many observations that human players can use which are not exposed. These are typically edge-cases which do not significantly alter strategies, like an indication of whether or not units are present within a bunker, the radius sensor towers can detect within, queued research, which units are targeting others with abilities such as Neural Parasite, and other assorted interface features and bugs.
- Entities normally appear as sprites on the screen. We disentangle them, passing entities through a separate observation category, because learning to identify entity concepts was not a research focus and the separation provided a more suitable deep

learning objective (see 4.2.1). In particular edge cases, this led to a trade-off of faster information gathering by exposing information such as entities that are fully occluded by other entities.

- As previously mentioned, agents could also gather information marginally faster by seeing unit research upgrades without needing to either select them or have them inside the camera view, because selecting a unit was not possible within the final interface. Some other examples of this include units that are stored in cargo, which are only visible when the storage is selected, and how close a building is to completing its current tasks, which is only visible when the building is selected.

- The map was enriched by additional static map information about what areas can be built on, and which areas can be navigated over. Which areas can be navigated over is something that is normally inferred by humans. This information could largely be inferred by comparing relative elevations of tiles, and is a minor instance of using external information.

- The time until entities can next attack is another example of information which humans would have to remember based on when they last attacked, since it is not displayed in the game.

Chapter 7

Conclusion

AlphaStar is the first agent to reach the highest tier of human performance in a widely played professional esport. This thesis presented AlphaStar, and the combination of novel and existing general-purpose techniques that it was built with. Some novel techniques include league training, which used various matchmaking strategies and introduced prioritized fictitious self-play to maintain a continually improving multi-agent league of players; scatter connections, which integrate spatial and entity information; UPGO, a self-imitation based update rule that improves off-policy learning; and the statistic z, which encodes high-level human strategic information and uses it as a basis for human-aligned exploration.

This thesis also presented fairness, game complexities, and deep learning concerns that are necessary when designing interfaces for deep learning, specifically focused on games and game-like domains, with a deeper look into how those principles applied to the design of AlphaStar's StarCraft II interface. Interface design is a hard problem with different trade-offs depending on what research problems need to be addressed, and being aware of the constraints and possibilities is necessary to make the right trade-offs.

Though unconstrained agents are useful for many applications (e.g. nuclear fusion plasma confinement), human-like behavior and interfaces can serve many purposes beyond facilitating progress on research problems. In the future, AI systems could teach humans how to perform tasks more effectively or act to minimize the spread of disease. Understanding how humans interact with the world and finding comprehensible improvements is important in both applications. In other domains, limiting an AI to act as a human would even after it undergoes some optimization is critical; when driving, people rely on theory of mind predictions of other drivers, so self-driving cars that violate these predictions can lead to accidents. Human-like constraints could serve as a prior to develop more

general intelligence, both to encourage learning of transferable representations and skills we know humans have, but also to ensure a final agent's actions and limitations are better understood.

However, such agents also have extra associated ethical concerns. Besides privacy concerns about using human data in deep learning and the risks of exposing personally identifiable information, systems might also learn human biases. Large language models trained on human natural language exhibit biases and stereotypes with respect to race, religion, and gender because they are trained to model an internet that has such biases [11]. These biases can cause real harm to people by perpetuating stereotypes and creating negative portrayals, for example in inverse reinforcement learning where a self-driving car value function could learn to value different people differently. While biases may be less of a concern in StarCraft II, future research is critical to ensure deployed systems are not prejudiced.

StarCraft has properties in common with real-world applications. Agents interact with other agents in partially-observable environments, with complex action spaces in domains like self-driving cars, energy management, and personalized healthcare. While AlphaStar is far from an artificial general intelligence, techniques and algorithms used by it will hopefully lead to progress in many other domains.

7.1 Future Work

There is a significant amount of future work possible. Applying or refining the interface design principles in other domains is an interesting future step. Of particular note are applying them to non-game domains, and especially real-world domains where fairness is not a major concern, or creating quantifiable metrics for human-like interfaces.

Algorithms introduced in AlphaStar could be iterated upon to make them more general-purpose. League training in particular, though effective, consists of hand-crafted niches of agent populations.

StarCraft is also not a solved domain. Interesting future directions include reducing the amount of human demonstrations necessary for training or eliminating reliance on them entirely (using intrinsic motivation for exploration [2]), creating agents that are robust under non-blind repeated trial conditions, or using the same graphical interface available to humans.

References

[1] AIIDE StarCraft AI competition. https://www.cs.mun.ca/~dchurchill/starcraftaicomp/. (accessed July 20, 2020).

[2] Adrià Puigdomènech Badia, Pablo Sprechmann, Alex Vitvitskyi, Daniel Guo, Bilal Piot, Steven Kapturowski, Olivier Tieleman, Martin Arjovsky, Alexander Pritzel, Andrew Bolt, and Charles Blundell. Never give up: Learning directed exploration strategies. In *International Conference on Learning Representations*, 2020.

[3] Bowen Baker, Ingmar Kanitscheider, Todor Markov, Yi Wu, Glenn Powell, Bob McGrew, and Igor Mordatch. Emergent tool use from multi-agent autocurricula, 2019.

[4] David Balduzzi, Marta Garnelo, Yoram Bachrach, Wojciech Czarnecki, Julien Perolat, Max Jaderberg, and Thore Graepel. Open-ended learning in symmetric zero-sum games. In *International Conference on Machine Learning*, pages 434–443, 2019.

[5] Bartender4. https://www.curseforge.com/wow/addons/bartender4. (accessed July 20, 2020).

[6] Charles Beattie, Joel Z Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, et al. DeepMind Lab. *arXiv preprint arXiv:1612.03801*, 2016.

[7] Berkeley Overmind. https://http://overmind.cs.berkeley.edu/. (accessed July 20, 2020).

[8] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

[9] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[10] George W Brown. Iterative solution of games by fictitious play. *Activity Analysis of Production and Allocation*, 13(1):374–376, 1951.

[11] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

[12] Michael Buro. ORTS: A hack-free RTS game environment. In *International Conference on Computers and Games*, pages 280–291. Springer, 2002.

[13] Michael Buro. Real-time strategy games: a new AI research challenge. In *International Joint Conference on Artificial Intelligence*, pages 1534–1535, 2003.

[14] Murray Campbell, A Joseph Hoane Jr, and Feng-hsiung Hsu. Deep Blue. *Artificial intelligence*, 134(1-2):57–83, 2002.

[15] Ilias Chalkidis and Dimitrios Kampas. Deep learning in law: early adaptation and legal word embeddings trained on large corpora. *Artificial Intelligence and Law*, 27(2):171–198, Jun 2019.

[16] Dian Chen, Brady Zhou, Vladlen Koltun, and Philipp Krähenbühl. Learning by cheating. In Leslie Pack Kaelbling, Danica Kragic, and Komei Sugiura, editors, *Proceedings of the Conference on Robot Learning*, volume 100 of *Proceedings of Machine Learning Research*, pages 66–75. PMLR, 30 Oct–01 Nov 2020.

[17] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*, pages 7–10, 2016.

[18] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*, pages 4299–4307, 2017.

[19] David Churchill. SparCraft: open source StarCraft combat simulation, 2013.

[20] David Churchill, Zeming Lin, and Gabriel Synnaeve. An analysis of model-based heuristic search techniques for StarCraft combat scenarios. In *Artificial Intelligence and Interactive Digital Entertainment Conference*, 2017.

[21] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? Try ARC, the AI2 reasoning challenge. *ArXiv*, abs/1803.05457, 2018.

[22] Yann N. Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 933–941, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

[23] Deadly Boss Mods. https://www.curseforge.com/wow/addons/deadly-boss-mods. (accessed July 20, 2020).

[24] DeepMind research on ladder. https://starcraft2.com/en-us/news/22933138. (accessed July 20, 2020).

[25] Y. Deng, F. Bao, Y. Kong, Z. Ren, and Q. Dai. Deep direct reinforcement learning for financial signal representation and trading. *IEEE Transactions on Neural Networks and Learning Systems*, 28(3):653–664, 2017.

[26] Arpad E Elo. *The rating of chessplayers, past and present*. Arco Pub., 1978.

[27] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. IMPALA: Scalable distributed Deep-RL with importance weighted actor-learner architectures. *arXiv preprint arXiv:1802.01561*, 2018.

[28] Andre Esteva, Alexandre Robicquet, Bharath Ramsundar, Volodymyr Kuleshov, Mark DePristo, Katherine Chou, Claire Cui, Greg Corrado, Sebastian Thrun, and Jeff Dean. A guide to deep learning in healthcare. *Nature Medicine*, 25(1):24–29, Jan 2019.

[29] Sehar Shahzad Farooq, In-Suk Oh, Man-Jae Kim, and Kyung Joong Kim. StarCraft AI competition report. *AI Magazine*, 37(2):102–107, 2016.

[30] D. A. Ferrucci. Introduction to "This is Watson". *IBM Journal of Research and Development*, 56(3.4):1:1–1:15, 2012.

[31] Javier Garcıa and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015.

[32] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[33] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. Adaptive Computation and Machine Learning series. MIT Press, 2016.

[34] Yves Grandvalet and Yoshua Bengio. Entropy regularization., 2006.

[35] A. Graves, A. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649, 2013.

[36] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[37] Johannes Heinrich, Marc Lanctot, and David Silver. Fictitious self-play in extensive-form games. In *International Conference on Machine Learning*, pages 805–813, 2015.

[38] Ralf Herbrich, Tom Minka, and Thore Graepel. Trueskill$^{\text{TM}}$: a bayesian skill rating system. In *Advances in neural information processing systems*, pages 569–576, 2007.

[39] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[40] Kurt Hornik. Some new results on neural network approximation. *Neural networks*, 6(8):1069–1072, 1993.

[41] Ji-Lung Hsieh and Chuen-Tsai Sun. Building a player strategy model by analyzing replays of real-time strategy games. In *IEEE International Joint Conference on Neural Networks*, pages 3106–3111. IEEE, 2008.

[42] Borja Ibarz, Jan Leike, Tobias Pohlen, Geoffrey Irving, Shane Legg, and Dario Amodei. Reward learning from human preferences and demonstrations in Atari. In *Advances in Neural Information Processing Systems*, pages 8011–8023, 2018.

[43] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[44] Max Jaderberg, Wojciech M Czarnecki, Iain Dunning, Luke Marris, Guy Lever, Antonio Garcia Castaneda, Charles Beattie, Neil C Rabinowitz, Ari S Morcos, Avraham Ruderman, et al. Human-level performance in 3D multiplayer games with population-based reinforcement learning. *Science*, 364(6443):859–865, 2019.

[45] Aditya Jain, Ramta Bansal, Avnish Kumar, and KD Singh. A comparative study of visual and auditory reaction times on the basis of gender and physical activity levels of medical first year students. *International Journal of Applied and Basic Medical Research*, 5(2):124, 2015.

[46] Norman P. Jouppi, Cliff Young, Nishant Patil, et al. In-datacenter performance analysis of a Tensor Processing Unit. Technical report, Google, 2017.

[47] Niels Justesen and Sebastian Risi. Learning macromanagement in StarCraft from replays using deep learning. In *IEEE Conference on Computational Intelligence and Games (CIG)*, pages 162–169. IEEE, 2017.

[48] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[49] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[50] Marc Lanctot, Vinicius Zambaldi, Audrunas Gruslys, Angeliki Lazaridou, Karl Tuyls, Julien Pérolat, David Silver, and Thore Graepel. A unified game-theoretic approach to multiagent reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 4190–4203, 2017.

[51] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436, 2015.

[52] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

[53] Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. Object recognition with gradient-based learning. In *Shape, contour and grouping in computer vision*, pages 319–345. Springer, 1999.

[54] Patrick LeMieux. From NES-4021 to moSMB3.wmv: Speedrunning the serial interface. *Eludamos. Journal for Computer Game Culture*, 8(1):7–31, 2014.

[55] David S Leslie and Edmund J Collins. Generalised weakened fictitious play. *Games and Economic Behavior*, 56(2):285–298, 2006.

[56] Timothée Lesort, Vincenzo Lomonaco, Andrei Stoian, Davide Maltoni, David Filliat, and Natalia Díaz-Rodríguez. Continual learning for robotics: Definition, framework, learning strategies, opportunities and challenges. *Information Fusion*, 58:52–68, 2020.

[57] James Martens and Roger Grosse. Optimizing neural networks with Kronecker-factored approximate curvature. In *International conference on machine learning*, pages 2408–2417, 2015.

[58] Luke Metz, Julian Ibarz, Navdeep Jaitly, and James Davidson. Discrete sequential prediction of continuous actions for deep RL. *arXiv preprint arXiv:1705.05035*, 2017.

[59] Jeff O Miller and Kathy Low. Motor processes in simple, go/no-go, and choice reaction time tasks: a psychophysiological analysis. *Journal of experimental psychology: Human perception and performance*, 27(2):266, 2001.

[60] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.

[61] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, Feb 2015.

[62] Mario Molina and Filiz Garip. Machine learning for sociology. *Annual Review of Sociology*, 45(1):27–45, 2019.

[63] Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations. In *IEEE International Conference on Robotics and Automation*, pages 6292–6299, 2018.

[64] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, pages 807–814, 2010.

[65] Junhyuk Oh, Yijie Guo, Satinder Singh, and Honglak Lee. Self-imitation learning. In *International Conference on Machine Learning*, pages 3875–3884, 2018.

[66] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. WaveNet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[67] Emilio Parisotto, Jimmy Ba, and Ruslan Salakhutdinov. Actor-mimic: Deep multitask and transfer reinforcement learning. In *International Conference on Learning Representations*, 2016.

[68] Jing Peng and Ronald J Williams. Incremental multi-step q-learning. In *Machine Learning Proceedings 1994*, pages 226–232. Elsevier, 1994.

[69] Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *AAAI Conference on Artificial Intelligence*, 2018.

[70] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[71] Andrei A. Rusu, Sergio Gomez Colmenarejo, Çaglar Gülçehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. In *International Conference on Learning Representations*, 2016.

[72] A. L. Samuel. Some studies in machine learning using the game of Checkers. *IBM Journal of Research and Development*, 3(3):210–229, July 1959.

[73] Mikayel Samvelyan, Tabish Rashid, Christian Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim GJ Rudner, Chia-Man Hung, Philip HS Torr, Jakob Foerster, and Shimon Whiteson. The starcraft multi-agent challenge. In *International Conference on Autonomous Agents and MultiAgent Systems*, pages 2186–2188, 2019.

[74] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015.

[75] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[76] Kun Shao, Yuanheng Zhu, and Dongbin Zhao. StarCraft micromanagement with reinforcement learning and curriculum transfer learning. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 3(1):73–84, 2018.

[77] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, Jan 2016.

[78] Starcraft 2 AI ladder. https://sc2ai.net/. (accessed July 20, 2020).

[79] Student StarCraft AI tournament and ladder. https://sscaitournament.com/. (accessed July 20, 2020).

[80] Peng Sun, Xinghai Sun, Lei Han, Jiechao Xiong, Qing Wang, Bo Li, Yang Zheng, Ji Liu, Yongsheng Liu, Han Liu, et al. Tstarbots: Defeating the cheating level builtin AI in StarCraft II in the full game. *arXiv preprint arXiv:1809.07193*, 2018.

[81] R. Sutton. Learning to predict by the method of temporal differences. *Machine Learning*, 3(9):9–44, 1988.

[82] R.S. Sutton and A.G. Barto. *Reinforcement Learning, second edition: An Introduction.* Adaptive Computation and Machine Learning series. MIT Press, 2018.

[83] Gabriel Synnaeve and Pierre Bessiere. A bayesian model for plan recognition in RTS games applied to StarCraft. In *Artificial Intelligence and Interactive Digital Entertainment Conference*, 2011.

[84] Gabriel Synnaeve, Zeming Lin, Jonas Gehring, Dan Gant, Vegard Mella, Vasil Khalidov, Nicolas Carion, and Nicolas Usunier. Forward modeling for partial observation strategy games-a StarCraft defogger. In *Advances in Neural Information Processing Systems*, pages 10738–10748, 2018.

[85] Gabriel Synnaeve, Nantas Nardelli, Alex Auvolat, Soumith Chintala, Timothée Lacroix, Zeming Lin, Florian Richoux, and Nicolas Usunier. Torchcraft: a library for machine learning research on real-time strategy games. *arXiv preprint arXiv:1611.00625*, 2016.

[86] Gerald Tesauro. TD-Gammon, a self-teaching Backgammon program, achieves master-level play. *Neural Computation*, 6(2):215–219, 1994.

[87] PD Thompson, JG Colebatch, P Brown, JC Rothwell, BL Day, JA Obeso, and CD Marsden. Voluntary stimulus-sensitive jerks and jumps mimicking myoclonus or pathological startle syndromes. *Movement disorders: official journal of the Movement Disorder Society*, 7(3):257–262, 1992.

[88] Alberto Uriarte and Santiago Ontañón. Improving Monte Carlo tree search policies in StarCraft via probabilistic models learned from replay data. In *Artificial Intelligence and Interactive Digital Entertainment Conference*, 2016.

[89] Nicolas Usunier, Gabriel Synnaeve, Zeming Lin, and Soumith Chintala. Episodic exploration for deep deterministic policies: An application to StarCraft micromanagement tasks. In *International Conference on Learning Representations*, 2017.

[90] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc., 2017.

[91] Oriol Vinyals, Igor Babuschkin, Junyoung Chung, Michael Mathieu, Max Jaderberg, Wojtek Czarnecki, Andrew Dudzik, Aja Huang, Petko Georgiev, Richard Powell, Timo Ewalds, Dan Horgan, Manuel Kroiss, Ivo Danihelka, John Agapiou, Junhyuk Oh, Valentin Dalibard, David Choi, Laurent Sifre, Yury Sulsky, Sasha Vezhnevets, James Molloy, Trevor Cai, David Budden, Tom Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Toby Pohlen, Dani Yogatama, Julia Cohen, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Chris Apps, Koray Kavukcuoglu, Demis Hassabis, and David Silver. AlphaStar: Mastering the real-time strategy game StarCraft II, 2019.

[92] Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander S. Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom L. Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver.

Grandmaster level in StarCraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, Nov 2019.

[93] Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, et al. StarCraft II: A new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782*, 2017.

[94] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2692–2700. Curran Associates, Inc., 2015.

[95] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Rémi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay. In *International Conference on Learning Representations*, 2017.

[96] Ben George Weber. AIIDE 2010 StarCraft competition. In *Artificial Intelligence and Interactive Digital Entertainment Conference*, 2010.

[97] Ben George Weber and Michael Mateas. Case-based reasoning for build order in real-time strategy games. In *Artificial Intelligence and Interactive Digital Entertainment Conference*, 2009.

[98] Michel Willem. *Minimax theorems*, volume 24. Springer Science & Business Media, 1997.

[99] Vinicius Zambaldi, David Raposo, Adam Santoro, Victor Bapst, Yujia Li, Igor Babuschkin, Karl Tuyls, David Reichert, Timothy Lillicrap, Edward Lockhart, et al. Relational deep reinforcement learning. In *International Conference on Learning Representations*, 2018.