This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TG.2021.3049539, IEEE Transactions on Games

Creating Pro-Level AI for a Real-Time Fighting Game Using Deep Reinforcement Learning

Inseok Oh*, Seungeun Rho*, Sangbin Moon, Seongho Son, Hyoil Lee and Jinyun Chung†
Game AI Lab, AI Center, NCSOFT, South Korea
{ohinsuk, gloomymonday, sangbin, hingdoong, onetop21, jchung2050}@ncsoft.com

***Abstract*—Reinforcement learning combined with deep neural networks has performed remarkably well in many genres of games recently. It has surpassed human-level performance in fixed game environments and turn-based two player board games. However, to the best of our knowledge, current research has yet to produce a result that has surpassed human-level performance in modern complex fighting games. This is due to the inherent difficulties with real-time fighting games, including: vast action spaces, action dependencies, and imperfect information. We overcame these challenges and made 1v1 battle AI agents for the commercial game "Blade & Soul". The trained agents competed against five professional gamers and achieved a winning rate of 62%. This paper presents a practical reinforcement learning method that includes a novel self-play curriculum and data skipping techniques. Through the curriculum, three different styles of agents were created by reward shaping and were trained against each other. Additionally, this paper suggests data skipping techniques that could increase data efficiency and facilitate explorations in vast spaces. Since our method can be generally applied to all two-player competitive games with vast action spaces, we anticipate its application to game development including level design and automated balancing.**

***Index Terms*—Deep learning, fighting game, imperfect information, reinforcement learning, self-play curriculum learning**

I. Introduction

Reinforcement learning (RL) is extending its boundaries to a variety of game genres. In PVE (player versus environment) settings, such as those found in Atari 2600 games, RL agents have exceeded human level performance using various methods [15], [16], [19], [5]. Likewise, in PVP (player versus player) settings, neural networks combined with search-based methods beat the best human players in turn-based player games with 2 or more players —such as Go, Chess [20], and Mahjong [28]. Recently, RL research in games has shifted focus to the PVP settings found in more complex video games such as StarCraft2 [24], Quake3 [10], and Dota2 [18]. Even grand-master level RL agents have been developed for StarCraft2 [29], which is a highly complex imperfect information game where an agent has to control multiple units at a time.

Fighting games—as one of the most representative types of complex PVP games—have been the focus of multiple studies that have made progress in this area. For instance, Monte-Carlo tree search (MCTS) based methods [27], [11], [9] have been applied to "FightingICE(FICE)", a game platform made for the Fighting Game AI Competition [13]. However, it is hard to fulfill real-time conditions when applied to heavier game engines with longer query times. Additionally, a deep RL based agent [12] was trained against a rule-based fixed opponent in "Little Fighter 2(LF2)". However, since the opponent's decision is unknown at a player's decision time, agents trained against rule-based AIs cannot be generalized for unseen opponents. Our approach is largely similar to that of [3] in which a self-play deep RL method was applied to "Super Smash Bros. Melee (SSBM)".

	Year	Commercial	Dimension	Pro-scene
FICE	2013	No	2D	No
LF2	1999	Yes	2.5D	No
SSBM	2001	Yes	2D	Yes
BAB	2013	Yes	3D	Yes

Table 1: Fighting games from other works

Figure 1. A scene from the B&S Arena Battle

* Equal contribution. Alphabetical ordering.
† Corresponding Author

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TG.2021.3049539, IEEE Transactions on Games

However, the complexity of state and action space is significantly limited compared to our 3D environment with complex game rules. We created pro-level AI agents for the real-time fighting game "Blade & Soul (B&S) Arena Battle" via novel self-play based reinforcement learning.

B&S is a commercial massively multiplayer online role-playing game. It supports duels between two players called "B&S Arena Battles (BABs)". As presented in Table 1, BAB is a more modern fighting game compared to the games considered in other works; hence, it has much more complex game dynamics and heavier game engines. Additionally, a large number of people play BAB and it has more active professional scenes [1] than other fighting games. BAB's larger number of active professional scenes stands out more significantly when compared to FightingICE, which was designed solely for research purposes.

Figure 1 displays a scene from BAB. In BAB, two players fight against each other to reduce their opponent's HP (health point) to zero within three minutes. To master BAB, an agent must be able to deal with multiple challenges.

First, an agent must manage vast action and state spaces compared to other fighting games [3][11][12]. An agent must make skill, move, and targeting decisions simultaneously, which yields many possible combinations. As a rough estimate, there are 144 potential actions for each time step: 8 (avg. # of avail. skills) * 9 (8 directional + no move) * 2 (facing opponent or moving direction). Since the average game length is 1,200 time steps (120 s), numerous scenarios are possible—not considering the opponent's actions.

Moreover, an agent must consider the dependencies between skills: e.g., a skill may become available only for a short period of time following the use of another skill. As a result, out of the 45 skills in total (including "no-op"), the set of skills available at a given time constantly changes. The agent must also consider the properties of each skill because they have different cooldown times (required interval for re-using a skill) and SP (skill point) consumptions, and serve one or more of five different functions: damage dealing, crowd control (a set of skills that reduces the number of possible actions the opponent can utilize; abbreviated CC) [32], resistance (which functions to make the player immune or resistant to CC skills), escape, and dash. In BAB, crowd control refers to a set of skills that reduces the number of possible actions the opponent can utilize. For example, when you "stun" a Destroyer (one of the classes in BAB), it can only use the skill "escape". When a Destroyer is "groggy", it becomes limited to the skills "escape" and "retreat". When it is "down", the Destroyer is limited to 5-6 possible skills.

Lastly, an agent must deal with imperfect information settings. Because BAB is a real-time game, two players make their decisions simultaneously. This indicates that an agent is required to make decisions without knowing the opponent's decision or strategy. Hence, BAB can be considered an imperfect information game [30][33]. For example, when a player uses a resistance skill and the opponent uses a crowd control skill at the same time, the player gains advantage over the opponent. As a result, the essence of the problem is to approximate a Nash equilibrium strategy so that the agent can respond appropriately to any opposing strategy.

To tackle these challenges, we have made improvements to vanilla self-play algorithm by diversifying opponent pools and skipping data to facilitate exploration. The main contributions of this work are as follows:

- We devised a novel self-play curriculum [35] with agents of different styles. The curriculum made these agents compete against each other and reinforced the agents simultaneously, rendering the agents capable of handling a variety of opponents. We empirically demonstrate that our curriculum outperforms vanilla self-play method.
- We diversified the fighting style of the game-playing AIs by reward shaping [17]. We created three types of agents with different fighting styles: aggressive, defensive, and balanced. We anticipate its application to game development including level design and automated balancing.
- We introduced data skipping techniques to enhance exploration in vast space. These can be generally applied to any two-player real-time fighting games.
- We evaluate our agents by pitting them against professional players in the 2018 B&S World Championship Blind Match. Our AI agents won three out seven matches, while the aggressive one beating all professional players both in the live event and pre-test.

II. Background

A. Reinforcement Learning

In reinforcement learning [23], agent and environment can be formalized as a Markov decision process (MDP) [8]. For every discrete time step t, an agent receives a state $s_t \in S$ and sends an action $a_t \in A$ to the environment. Then, the environment makes a state transition from s_t to s_{t+1} with the state transition probability $P^a_{ss'} = P[s'|s,a]$ and gives a reward function $R: S \times A \rightarrow \mathbb{R}$, with the reward signal $r_t = R(s_t, a_t) \in \mathbb{R}$ given to the agent. Therefore, this process can be expressed with $\{S, A, P, R, \gamma\}$, where $\gamma \in [0,1]$ is a discount factor, which represents the preference for immediate reward over long-term reward. Here, the agent samples an action from a policy $\pi: S \rightarrow P(A)$, where $P(A)$ represents the set of probability distributions. The learning process modifies the policy to encourage good actions and suppress

[1] 9 regional league winners from all over the world (including KOR, NA, EU, RUS, and CHN) participated in the 2018 B&S world championship (fourth annual event). The winning prize was approx. $50k (compared to Tekken7: $30k)

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TG.2021.3049539, IEEE Transactions on Games

bad actions. The objective of the learning is to find the optimal policy π^* that maximizes the expected discounted cumulative reward.

$$\pi^* = argmax_\pi E_\pi[\Sigma_t r_t * \gamma^t]$$

B. Real-Time Two Player Game

In a real-time two player game, there are two players, namely, the agent and the opponent. Both of them send an action to the environment at the same time. Let us denote the policy of the agent as π^{ag}, and the policy of the opponent as π^{op}. Each samples an action from its own policy for every time step.

$$a_t^{ag} \sim \pi^{ag}(a_t^{ag}|s_t),\ a_t^{op} \sim \pi^{op}(a_t^{op}|s_t)$$

Then, the environment makes a state transition by considering those two actions jointly.

$$s_{t+1} \sim P(s_{t+1}|s_t, a_t^{ag},\ a_t^{op}),\ r_{t+1} = R(s_t, a_t^{ag},\ a_t^{op})$$

Here, the MDP can be expressed as $\{S, A^{ag}, A^{op}, P, R, \gamma\}$. If π^{op} is fixed, then we can regard the opponent as a part of the environment by marginalizing the policy of the opponent. This way, we can obtain P' and R':

$$P'(s_{t+1}|s_t, a_t^{ag})$$

$$= \sum_{a_t^{op}} \pi^{op}(a_t^{op}|s_t) * P(s_{t+1}|s_t, a_t^{ag},\ a_t^{op})$$

$$R'(s_t, a_t^{ag}) = \sum_{a_t^{op}} \pi^{op}(a_t^{op}|s_t) * R(s_t, a_t^{ag},\ a_t^{op})$$

Then, the MDP expression turns into a simpler form with P' and R': $\{S, A^{ag}, P', R', \gamma\}$. This expression is coherent with the one player MDP. Therefore, any methods for the original MDP work in this form as well.

C. BAB as MDP

If we assume π^{op} or the pool of π^{op} is fixed, BAB can be expressed as an MDP [34]. Figure 2 illustrates the agent-environment framework in BAB. LSTM [6] based agents interact with the BAB simulator, which acts as the environment. For every time step with 0.1 sec intervals, state s_t is constructed from the history of observations $H_t = \{o_1, o_2, \ldots, o_t\}$. To be specific, s_t is composed of any information that a human can access during a game, such as HP, SP, distance from opponent, distance from the arena wall, current position, remaining game time, remaining cooldown times for all 44 skills, an agent's status info (midair, stun, down, kneel, etc.), and so on. Then, the agent decides on an action $a_t = (a_t^{skill}, a_t^{move,target})$ for every time step. Note that the targeting action (i.e., orientation) space was originally continuous. We discretized the space into two actions – facing the opponent and facing away from the opponent –

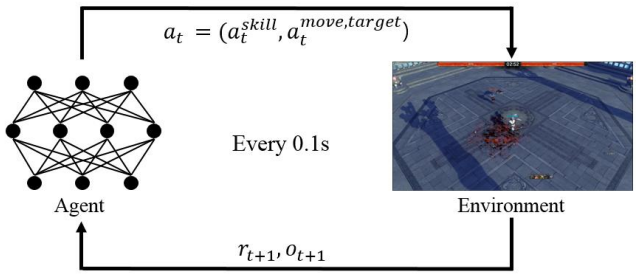


Figure 2. Agent-environment plot in BAB

and jointly considered it along with the quantized move decision. Following this, the action is then sent to the environment and a state transition occurs accordingly.

Here, exact rewards should also be determined. Rewards are closely related to high performance in BAB. We provided r_t^{WIN}, which is the reward for winning a game, and r_t^{HP}, the reward for the changes in HP margin. These rewards are designed based on the assumption that the more a player wins, and with more remaining HP, the better that player's performance is. r_t^{WIN} is given at the terminal step of each episode with +10 for a win and -10 for a loss. r_t^{HP} may occur at every time step when the agent deals damage to the opponent and vice versa. Since HP is normalized to [0, 10], r_t^{WIN} and r_t^{HP} have the same scale.

$$r_t = r_t^{WIN} + r_t^{HP} + r_t^{EXTRA}$$

$$r_t^{HP} = (HP_t^{ag} - HP_{t-1}^{ag}) - (HP_t^{op} - HP_{t-1}^{op})$$

$$r_t^{EXTRA} = -(r_t^{time} + r_t^{distance})$$

r_t^{EXTRA} is an additional reward for guiding battle styles. It is the sum of the time penalty and the distance penalty. r_t^{time} is a reward based on the game length, and $r_t^{distance}$ is a reward based on the distance between agents. These additional rewards are described in further detail in the next section. The value of γ is set to 0.995, which is close to 1.0, since all episodes in BAB are forced to terminate after 1,800 time steps (= 3 min).

III. Self-Play Curriculum with Diverse Styles

π^{op} needs to be fixed to formalize BAB as an MDP. However, π^{op} is not fixed in general and our agent does not know which π^{op} it is going to face. We propose a self-play curriculum with a diversified pool of π^{op} to solve this issue. Existing self-play methods ([21], [22]) generally use opponent pools for training. Parameters of a network are stored at regular intervals during training to create a pool of past selves. Opponents are then sampled from this pool.

Although the self-play method of RL offers a way to learn the Nash equilibrium strategy [4], high coverage of strategy space is essential to efficiently find one. Vanilla self-play alone does not guarantee enough coverage for games with large problem spaces. To tackle this problem, AlphaStar [25]

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TG.2021.3049539, IEEE Transactions on Games

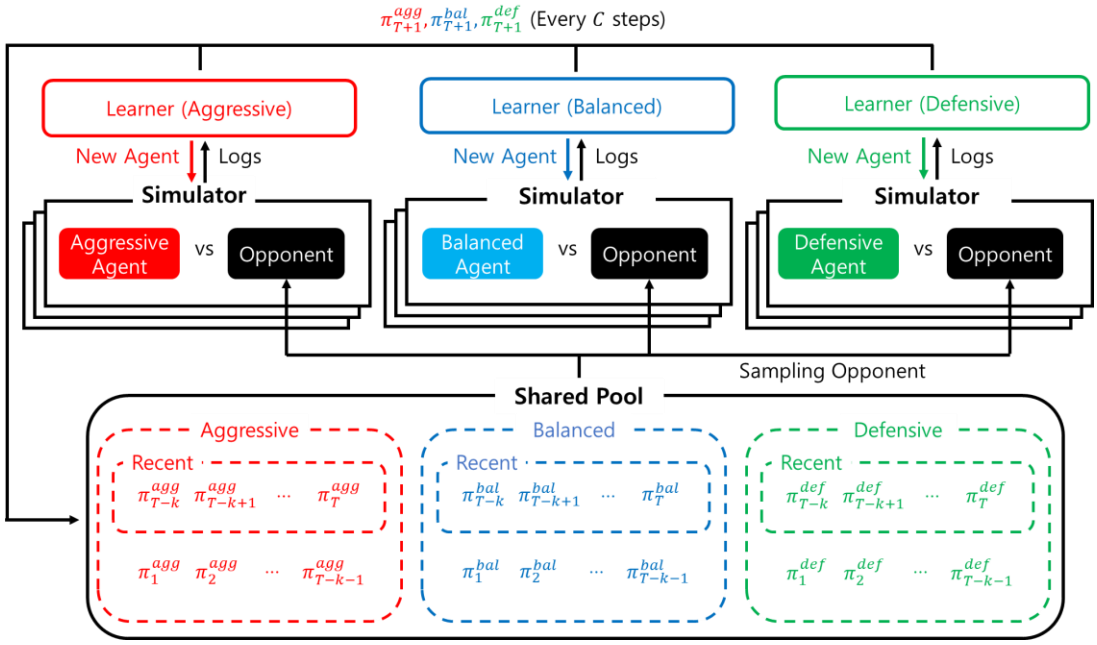


Figure 3. Overview of self-play curriculum with three different styles

diversified the opponent pool by imitating different human strategy and introducing three types of agents with different match making scheme. The Poker AI, Pluribus [1], hand-tuned three different strategies on top of basic blueprint strategy. The three strategies are biased toward raising, calling, and folding respectively.

Concurrently, we devised a novel self-play curriculum. We enforced diversity of agents' strategies by introducing a range of different battle styles, and agents of different styles were made to compete against each other.

A. *Guiding Battle Styles through Reward Shaping*

One of the most noticeable fighting styles to invest with is the degree of aggressiveness. We used three dimensions of rewards to control the degree of aggressiveness. The first dimension is the "time penalty". The aggressive agent receives larger penalties per time step, and this motivates it to finish the match in a shorter period of time. The second dimension is the relative importance of the agent's HP to the opponent's HP. Aggressive players will try to reduce the opponent's HP rather than preserving their own HP, while defensive players tend to act the opposite way. The final dimension is the "distance penalty". Defensive players tend to ensure a certain distance from their opponents to respond appropriately against attacks, while aggressive players tend to approach their opponents and attack relentlessly. To realize these properties, the aggressive agent received larger penalties in proportion compared to the distance between itself and its opponent. The specific reward weights used for each style are shown in Table 2. Note that each of these three dimensions can take continuous values. This means that it is possible to create a spectrum of different fighting styles with varying degrees of aggressiveness. However, to effectively demonstrate the viability of this method, we limited the number of fighting styles to three. By using any type of additional reward signals along with r_t^{WIN} and r_t^{HP}, this method could be applied to other fighting games in general to create agents with various fighting styles.

B. *Our Self-Play Curriculum*

Figure 3 shows an overview of the proposed self-play curriculum with three different types of agents. Agents of each style have their own learning process, and all three agent types were trained in a concurrent manner.

Each learning process consists of a learner and multiple simulators. The learner and the simulators work asynchronously. In the simulators, an agent constantly plays matches against randomly sampled opponents from the shared pool. The most recent k models of each style are uniformly selected with total probability mass of p, while other models are chosen uniformly with probability $1-p$. As training goes on, p is linearly annealed from 0.8 to 0.1. A higher p assists in swift adaptation to the latest opponents, while a lower p stabilizes the learning process by alleviating catastrophic forgetting. Each simulator sends a match log to the learner

	Aggressive	Balanced	Defensive
Time penalty	0.008	0.004	0.0
HP ratio	5:5	5:5	6:4
Distance penalty	0.002	0.0002	0.0

Table 2: Reward details of each style

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TG.2021.3049539, IEEE Transactions on Games

at the end of every match and updates its agent with the latest parameters received from the learner. The same procedure continues to be used through subsequent games.

The learner trains its agents in an off-policy manner using logs gathered from multiple simulators and sends the latest network parameters to the simulators on request. In addition, the learner sends its network parameters to the shared pool every C steps (e.g. C=10,000) of update. Thus, the pool has varying policies that come from the different learning processes of the different styles. These sets of model parameters are provided as opponents to each learning process. By sharing a pool, every learning agent encounters opponents of every style during training and learns how to deal with them. Therefore, agents trained via our self-play curriculum can ultimately learn how to face opponents with varying fighting styles while maintaining their own battle styles.

IV. Data Skipping Techniques

In this section, we detail two data skipping techniques: "no-op" and "maintain move decision". Data skipping techniques refer to the process of dropping certain data during training and evaluation procedures.

A. *Discarding Passive "No-op"*

In fighting games, using skills generally consumes resources, such as SP and cooldown time. Therefore, if a player overuses a certain skill, it will not be available for use during actual times of need. Thus, players should strategically use and retain their skills to ensure their availability when needed. To take this aspect into account, we concatenated a "no-op" action to the output of the policy network, allowing the agent to choose "no-op" and do nothing for a certain period if necessary. This means that our action space has 44 skills, plus an additional "no-op" action. This is significant because human play logs of BAB show that "no-op" actions take up the largest portion of skill usage among human players.

"No-op" decisions can be categorized as passive and active use cases. The passive use of "no-op" implies that an agent chooses "no-op" because there is no skill available for use. For example, when an agent is out of resources or is hit by an opponent's CC skill, an agent has no option but to choose "no-op". The active use of "no-op" means that an agent selects "no-op" strategically, even though other skills are available for use.

We discarded passive "no-op" data from both the training and evaluation phases because passive "no-ops" are not used deliberately by an agent. In addition, the method enables LSTM to reflect representations of longer time horizons because the data is not provided to the network. We show in the experiment section that skipping passive "no-ops" greatly improves learning efficiency. Note that this methodology is generally applicable to other domains where a set of available skills changes constantly and the "no-op" action is a valid option to choose.

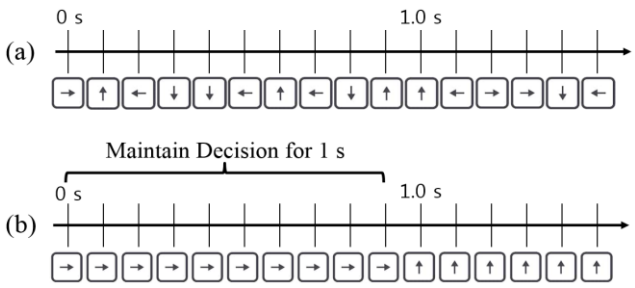


Figure 4. Examples of (a) regular move decisions and (b) maintaining decisions for 1 second

B. *"Maintain Move Decision"*

Although a single skill decision can have a substantial influence on the subsequent states, the effect of a single move decision is relatively limited. The reason is that the distance a character moves in a single time step (0.1 s) is very short considering its speed. In order for any moving decision to have a meaningful effect, the agent should make the same moving decision consecutively for several ticks in a row. This allows the agent to literally "move" and leads to changes in subsequent states and rewards. Therefore, it is difficult to train a move policy from the initial policy with random move decisions. Since the chance of a random policy making the same decision consecutively is very low, exploration is extremely limited. We therefore propose maintaining the move decision for a fixed number of time steps.

Figure 4 shows how "maintain move decision" works with an example. If the agent selects a move action, it skips the move decision for the following *n-1* time steps. This means that the agent maintains the same move decision for *n* steps in total. Note that our method has different purpose from frame skip technique [15] in Atari domain. Frame skip technique was introduced for simulator's efficiency. However, we cannot just skip the frames because skill decisions must still be made. Although we could not enjoy advantage in the simulator's efficiency, "maintain move decision" still facilitates training and this is solely because maintaining the move decision increases the influence of a single move decision, as we will confirm with experiments. In this sense, "maintain move decision" rather can be viewed as 'amplifying advantage' from [14].

V. Experiments

A. *Implementation Details*

1) *Network*

The network is composed of LSTM-based architecture which has four heads with a shared state representation layer. Each head consists of π_{skill}, Q_{skill}, $\pi_{move,target}$ and $Q_{move,target}$. Q_{skill} and $Q_{move,target}$ are used for the gradient update of π_{skill} and $\pi_{move,target}$, respectively. Before

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TG.2021.3049539, IEEE Transactions on Games

the network output goes into the softmax layer, a Boolean vector indicating the availability of each skill operates to make the output of unavailable skill to negative infinity.

2) Algorithm

We used actor-critic off-policy learning algorithm [26]. It enables us to deal with policy lag between the simulators and learner through truncated importance sampling. Moreover, we could also use the advantages of stochastic policy, which responds more stably to changes in the environment due to smooth policy updates and works well in the domain of games like rock-paper-scissors where deterministic policy is vulnerable to exploitation. For this specific algorithm, both π_{skill} and $\pi_{move,target}$ are updated in an alternating manner with following gradient:

$$g_t^{acer} = \bar{\rho}_t \nabla_\theta log\pi_\theta(a_t|x_t)\left[Q^{ret}(x_t,a_t) - V_{\theta_v}(x_t)\right] + \mathbb{E}_{a\sim\pi}\left(\left[\frac{\rho_t(a)-c}{\rho_t(a)}\right]_+ \nabla_\theta log\pi_\theta(a|x_t)\left[Q_{\theta_v}(x_t,a) - V_{\theta_v}(x_t)\right]\right),$$

where $\bar{\rho}_t = \min\{c, \rho_t\}$ with behavior policy μ and importance sampling ratio $\rho_t = \frac{\pi(a_t|x_t)}{\mu(a_t|x_t)}$. $[x]_+ = x\ if\ x > 0$ and zero otherwise.

3) Learning System

In total, there are three learning processes with each learning process consisting of a learner and 100 simulators. Each learning process is largely similar to that proposed by [7]. The final agent is trained for two weeks, which is equivalent to four years of game play.

B. Effect of Self-Play Curriculum with Three Styles

To demonstrate the effects of the proposed self-play curriculum, we trained agents with and without the proposed curriculum. A baseline agent was trained with the vanilla self-play curriculum without any style-related rewards (only win reward and HP reward were included) and a pool of past selves was used. Meanwhile, three agents with different styles were trained with the self-play curriculum using the shared pool that we proposed. Our aggressive, balanced and defensive agents[2] then played 1,000 matches each against the baseline agent to measure the performance. As shown in Table 3, the agents that followed the learnings from our curriculum outperformed the baseline agent.

Next, we conducted an ablation study to observe how the shared pool helps generalization. We wanted to confirm whether an agent would be able to deal with opponents of unseen style, when it experienced only a limited range of opponents during training. Thus, we created three styles of agents trained in exactly the same manner, except that they had their own independent opponent pools. We denote the three types of agents using shared pools as π_{sh}^{agg}, π_{sh}^{bal}, and π_{sh}^{def}, and three type of agents using independent pools as π_{ind}^{agg}, π_{ind}^{bal}, and π_{ind}^{def}. All of six agents were trained for 5M steps (equivalent to 6 days) each.

	Aggressive	Balanced	Defensive	Average
Vs. Baseline	59.5%	63.8%	63.2%	62.2%

Table 3: Winning rate of three style of agents against baseline (1,000 games each)

	Aggressive	Balanced	Defensive	Average
Shared	64.8%	79.6%	75.3%	73.6%
Ind.	64.7%	72.1%	56.5%	64.4%

Table 4: Generalization performance of three styles of agents for both with and without shared pool (7,000 games each)

Our assumption is that the agent trained with the shared pool is more robust when it faces opponents it has never encountered. Thus, we compared the winning rate of π_{sh}^{agg} vs. $\{\pi_{ind}^{bal}, \pi_{ind}^{def}\}$ and π_{ind}^{agg} vs. $\{\pi_{ind}^{bal}, \pi_{ind}^{def}\}$. This experimental setting is based on three key ideas. First, π_{sh}^{agg} and π_{ind}^{agg} have the same training settings except for sharing the pool. Second, π_{sh}^{agg} and π_{ind}^{agg} are evaluated against the same opponents. Finally, although π_{sh}^{agg} has encountered other styles from its pool, it has not confronted $\{\pi_{ind}^{bal}, \pi_{ind}^{def}\}$, for they were trained using independent opponent pools. If our assumption is correct, π_{sh}^{agg} should have a higher winning rate. It is to be noted that π_{ind}^{bal} and π_{ind}^{def} are not a single model, but 10 models each sampled at the same fixed intervals from their pools. We then conducted the same experiments for the remaining two styles; the results are presented in Table 4. As shown in the table, agents trained with shared pool outperform their counterparts.

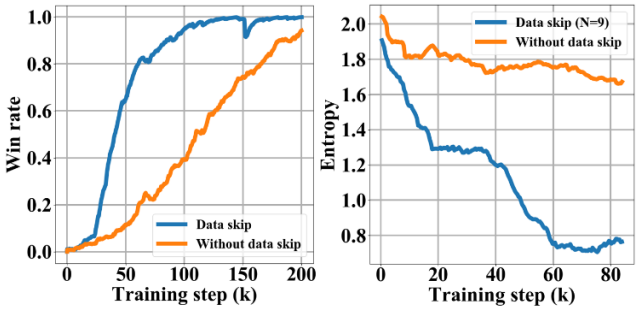
Based on the data in Table 4, the effect of using a shared pool is marginal in the case of aggressive agents. It indicates that the strategy spaces in which trainings take place are similar whether or not various opponents are provided. This is related to the nature of fighting games in which one side should fight back if the other side approaches and initiates a brawl. Thus, in the case of an aggressive agent that attacks consistently, there is a little difference in the experience regardless of the diversity of the opponent's fighting style.

C. Effect of Discarding Passive "No-op"

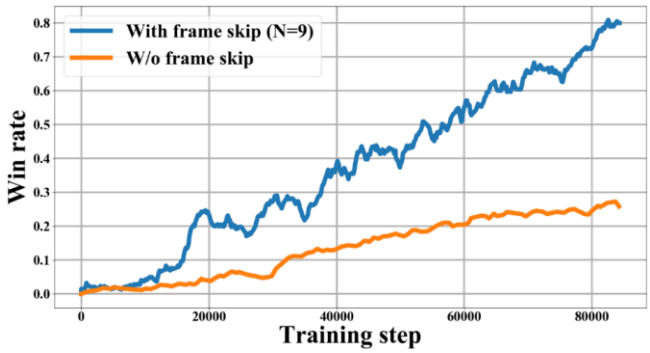
As discussed in the previous section, the "no-op" decision may be either active or passive. We conducted an experiment to investigate the effect of discarding such passive "no-op" data from learning. The sparring partner for the experiment was the built-in BAB AI, with a performance comparable to the top 20% of the players. We measured how fast

[2] We measured how the average game length differs for each style because game length is a good proxy for assessing the degree of defensiveness of an agent's game play. The results were as follows: 66.6 sec for the aggressive, 91.7 sec for the balanced, and 179.9 sec for the defensive agent.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TG.2021.3049539, IEEE Transactions on Games



(a) Skipping passive "no-op" (b) "Maintain move decision"



(c) Winning rates of built-in BAB AIs with and without "maintain move decision"

Figure 5. Results of data skipping experiments

agents learned to defeat it, and the results are shown in Figure 5 (a). If "no-op" ticks are discarded from the learning data, the winning rate reaches 80% after 70k steps, whereas 170k steps are required when "no-op" ticks are included. The amount of time steps required to reach 90% winning rate was reduced to half when passive "no-op" data was skipped. This experiment confirms that the training performance is improved by discarding passive "no-op" from the learning data.

D. *Effects of the "Maintain Move Decision"*

To examine the effects of the "maintain move decision", we developed two learning processes, with both processes involving learning on a self-play basis. One process makes a moving decision at every time step, while the other makes a moving decision and sends the same decision for 9 more times in a row. We measured the entropy of the move policy to observe the effects. Entropy of the move policy for a given state s_t is as follows.

$$\mathrm{H}(s_t) = -\sum \pi_{move}(s_t) * \log \pi_{move}(s_t)$$

Generally, entropy gradually decreases as learning progresses. Figure 5 (b) shows that the entropy declines faster if the technique is applied. A noticeable difference was also observed in the quality of movement which the agent learned. Before the technique was applied, the agent did not make any improvement from random motion, but it learned to approach and retreat with data skip. In addition, Figure 5 (c) shows that the relative winning rate of the built-in BAB AI is made higher by applying "maintain move decision".

The longer the decision was repeated, the agent's reaction became less immediate, but the agent moved more consistently. In this case, we tested 1, 3, 5, 10 ticks for maintaining time. 10 ticks (equivalent to 1 s) yielded the best performance.

	Aggressive	Balanced	Defensive
Pro-Gamer 1	5-1	2-1	1-2
Pro-Gamer 2	4-0	2-4	4-1
Blind Match	2-0	1-2	0-2
Total	11-1 (92%)	5-7 (42%)	5-5 (50%)

Table 5: Final score of AI vs. Human

VI. Pro-Gamer Evaluation

This section will address the results of both the pre-test and the Blind Match, and conditions to ensure fairness for human players.

A. *Conditions for Fairness*

1) *Reaction Time*

When humans confront an AI in a real-time fighting game, the most important factor that affects the result is the reaction time. Humans require some time to recognize the skill used by the opponent and to press a button by moving his/her hand. Therefore, we applied 230 ms of delay time on average to the AI's decision-making process to add fairness against human players. The average response time of a human is approximately 270ms [31]. However, a skilled pro-gamer's response time will be shorter than the average. This amount of delay corresponds to the average reaction time of professional players in BAB.

2) *Classes and Skill Set*

There are 11 classes in B&S, and each class has unique characteristics. Since there exists relative superiority among classes, we fixed the class of both AI and pro-gamer as "Destroyer" for all matches, training, and experiments conducted in section V. Destroyer is a class that has an in-fighting style and steadily appears in the B&S world championship. Additionally, AI's and pro-player's skill trees were set as identical to ensure a fair match. The skill tree was chosen to match what the majority of users selected, based on the BAB user statistics.

3) *Evaluation Results*

We invited two prominent pro-gamers, Yuntae Son (GC Busan, Winner of 2017 B&S World Championship), and Shingyeom Kim (GC Busan, Winner of 2015 and 2016 B&S World Championship), to test our agents before the Blind

Match. Note that the total number of games played is different for each style because the testers can play as many games as they want for each style. After the pre-test, we went for the Blind Match of 2018 World Championship. Our agents had matches against three pro-gamers: Nicholas Parkinson (EU), Shen Haoran (CHN), and Sungjin Choi (KOR). The video recording of the game highlights can be found at https://goo.gl/7VUTzV.

The results of both the pre-test and the Blind Match are provided in Table 5. As can be seen from the table, the aggressive agent dominated the game, while the other two types of agents had rather intense games. Based on our interviews of pro-gamers, we concluded that the reason why the aggressive agent showed the best performance against a human player was because the aggressive agent delivered continuous attacks preventing the human player from taking short breaks in between moves required for decision-making; the AI, on the other hand, does not require these breaks.

VII. CONCLUSION

Using deep reinforcement learning, we created AI agents that competed evenly with professional players in a 3D real-time fighting game. To accomplish this, we proposed a method to guide the fighting style with reward shaping. With three styles of agents, we introduced a novel self-play curriculum to enhance generalization performance. Our self-play curriculum with three styles is effective against general or unknown opponents or when self-play training converges into an equilibrium that is not optimal. Agents will likely end up exploring only a small portion of the vast BAB space when they are limited to self-play. Thus, training with different styles obtained through reward shaping can be an effective way to find the optimal policy. We also proposed data-skipping techniques to improve data efficiency and enable efficient exploration. Consequently, our agents were able to compete with the best BAB pro-gamers in the world. The proposed training methods are generally applicable to other fighting games.

REFERENCES

[1] Brown, N., and Sandholm, T. 2019. Superhuman AI for multi-player poker. Science, 365(6456), 885-890.

[2] Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., Legg, S., and Kavukcuoglu, K. 2018. IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures. In Proceedings of the 35th International Conference on Machine Learning, 80:1407-1416.

[3] Firoiu, V., Whitney, W. F., and Tenenbaum, J. B. 2017. Beating the world's best at Super Smash Bros. with deep reinforcement learning. arXiv preprint arXiv:1702.06230.

[4] Heinrich, J., and Silver, D. 2016. Deep reinforcement learning from self-play in imperfect-information games. arXiv preprint arXiv:1603.01121.

[5] Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, et al. 2018. Rainbow: Combining improvements in deep reinforcement learning. In Thirty-Second AAAI Conference on Artificial Intelligence.

[6] Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. Neural computation, 9(8):1735-1780.

[7] Horgan, D., Quan, J., Budden, D., Barth-Maron, G., Hessel, M., Van Hasselt, H., and Silver, D. 2018. Distributed prioritized experience replay. arXiv preprint arXiv:1803.00933.

[8] Howard, R.A. 1960. Dynamic programming and markov processes. MIT Press.

[9] Ishihara, M., Ito, S., Ishii, R., Harada, T., and Thawonmas, R. 2018. Monte-Carlo Tree Search for Implementation of Dynamic Difficulty Adjustment Fighting Game AIs Having Believable Behaviors. In 2018 IEEE Conference on Computational Intelligence and Games: 1-8.

[10] Jaderberg, M., Czarnecki, W. M., Dunning, I., Marris, L., Lever, G., Castaneda, A. G., et al. 2018. Human-level performance in first-person multiplayer games with population-based deep rein-forcement learning. arXiv preprint arXiv:1807.01281.

[11] Kim, M. J., and Kim, K. J. 2017. Opponent modeling based on action table for MCTS-based fighting game AI. In 2017 IEEE Conference on Computational Intelligence and Games (CIG):178-180.

[12] Li, Y. J., Chang, H. Y., Lin, Y. J., Wu, P. W., and FrankWang, Y. C. 2018. Deep Reinforcement Learning for Playing 2.5 D Fighting Games. In 2018 25th IEEE International Conference on Image Processing (ICIP):3778-3782.

[13] Lu, F., Yamamoto, K., Nomura, L. H., Mizuno, S., Lee, Y., and Thawonmas, R. 2013. Fighting game artificial intelligence competition platform. In IEEE 2nd Global Conference on Consumer Electronics: 320-323.

[14] Mladenov, M., Meshi, O., Ooi, J., Schuurmans, D., and Boutilier, C. 2019. Advantage amplification in slowly evolving latent-state environments. arXiv preprint arXiv:1905.13559.

[15] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., et al. 2015. Human-level control through deep reinforcement learning. Nature, 518(7540), 529.

[16] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Harley, T., Lillicrap, T. P., et al. 2016. Asynchronous methods for deep reinforcement learning. In Proceedings of the 33rd International Conference on International Conference on Machine Learning-Volume 48:1928-1937.

[17] Ng, A. Y., Harada, D., and Russell, S. 1999. Policy invariance under reward transformations: Theory and application to reward shaping. In ICML Vol. 99: 278-287.

[18] OpenAI. 2018. OpenAI five, https://blog.openai.com/openai-five.

[19] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. 2017. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347.

[20] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., et al. 2016. Mastering the game of Go with deep neural networks and tree search. nature, 529(7587), 484.

[21] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., et al. 2017. Mastering the game of go without human knowledge. Nature, 550(7676), 354.

[22] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., et al. 2017. Mastering chess and shogi by self-play

with a general reinforcement learning algorithm. arXiv preprint arXiv:1712.01815.

[23] Sutton, R. S., and Barto, A. G. 2018. Reinforcement learning: An introduction. MIT press.

[24] Vinyals, O., Ewalds, T., Bartunov, S., Georgiev, P., Vezhnevets, A. S., Yeo, M., et al. 2017. Starcraft II: A new challenge for reinforcement learning. arXiv preprint arXiv:1708.04782.

[25] Vinyals, O., Babuschkin, I., Chung, J., Mathieu, M., Jaderberg, M., Czarnecki, W. M., et al. 2019. Alphastar: Mastering the real-time strategy game starcraft ii. DeepMind blog, 2.

[26] Wang, Z., Bapst, V., Heess, N., Mnih, V., Munos, R., Kavukcuoglu, K., and Freitas, N. 2017. Sample efficient actor-critic with experience replay. In International Conference on Learning Representations (ICLR).

[27] Yoshida, S., Ishihara, M., Miyazaki, T., Nakagawa, Y., Harada, T., and Thawonmas, R. 2016. Application of Monte-Carlo tree search in a fighting game AI. In IEEE 5th Global Conference on Consumer Electronics:1-2.

[28] Li, J., Koyamada, S., Ye, Q., Liu, G., Wang, C., Yang, R., ... & Hon, H. W. (2020). Suphx: Mastering Mahjong with Deep Reinforcement Learning. arXiv preprint arXiv:2003.13590.

[29] Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., ... & Oh, J. et al. (2019). Grandmaster level in StarCraft II using multi-agent reinforcement learning. Nature, 575(7782), 350-354.

[30] Gibbons, R. (1992). A primer in game theory, 56.

[31] Reaction Time Statistics. Available online: https://humanbenchmark.com/tests/reactiontime/statistics (accessed on 07 December 2020).

[32] https://en.wikipedia.org/wiki/Crowd_control_(video_games)

[33] Harrington, J. 2009. Games, strategies and decision making. Macmillan. 28.

[34] Heinrich, J., Lanctot, M. and Silver, D., 2015, June. Fictitious self-play in extensive-form games. In International Conference on Machine Learning. 805-813.

[35] Sukhbaatar, S., Lin, Z., Kostrikov, I., Synnaeve, G., Szlam, A., & Fergus, R., "Intrinsic motivation and automatic curricula via asymmetric self-play," in International Conference on Learning Representations, 2018. [Online]. Available: https://openreview.net/forum?id=SkT5Yg-RZ

APPENDIX

A. State Space

A state input is a feature vector of length 606, rather than an image. Features include the following agent information: health point, skill point, distance from the opponent, distance from the arena wall in 16 directions, position and orientation, relative position from the opponent, remaining game time, remaining cooldown time and availability of each skill, abnormal condition (such as midair, stun, down, or kneel), etc. Abnormal condition refers to a state where the character's actions are restricted. All such information is also available to human players. And all inputs are normalized into a value between 0 and 1.

B. Feature Discretization

In the BAB environment, the current distance between oneself and the opponent determines whether some non-targeting skills (i.e., skills that can be implemented regardless of the distance between oneself and the opponent) can successfully hit the opponent. For example, the Destroyer's "axe sweep″ skill can only hit the opponent if one's character is less than or equal to 5 m from the opponent. Because the distance between oneself and the opponent is a feature that consists of a continuous value, the network has difficulty learning whether these non-targeting skills will be successful. Thus, in order to speed up the process, we used prior knowledge to discretize the following 4 crucial features:

1) Distance between oneself and the opponent: 5 levels (0-3 m, 3-5 m, 5-8 m, 8-16 m, over 16 m)

2) Skill state: 3 levels (skill has been recently used, skill has been used but not recently, skill has not been used)

3) Final six remaining ticks before crowd control loses effect: 6 levels (1 tick, 2 ticks, 3 ticks, 4 ticks, 5 ticks, 6 ticks)

4) Elapsed time upon using "retreat" skill: 6 levels (1 tick, 2 ticks, 3 ticks, 4 ticks, 5 ticks, 6 ticks)

To test the effectiveness of the feature discretization method, we designed an experiment comparing the success rates of counter attacks made against the "retreat" skill depending on the use of this technique. The "retreat" skill renders the player invincible for a certain amount of time at the cost of being unable to implement any kind of action for a slightly longer period which places them at a vulnerable position for a short time space. To be successful against BAB pro-gamers, an agent must learn how to seize this moment to apply crowd control against the opponent.

The agents were trained for 1M steps and tested for 20 games. Success rates were 75.0% and 13.9% with and without feature discretization respectively. These results show that attacks against retreat skills became 5.4 times more successful by employing feature discretization.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TG.2021.3049539, IEEE Transactions on Games

C. Action Space

1) Skill

"Skills" refer to a number of unique actions defined in the game of BAB. They can be used separately or in a series to develop a myriad of strategies. The Destroyer class has 44 skills; every skill has its own function such as dealing damage, applying crowd control, escaping, dashing, resisting, etc. Our agents make a skill decision out of 45 options including "no-op".

2) Move and Target

"Move and target" refer to changing a player's position and orientation. Although human players can move in 8 cardinal directions and have 360-degree vision, not all combinations are necessary to achieve pro-level play. We reduced the action space into the following 6 options by combining move and target decisions: 1) hold position and orientation, 2) move forward facing the opponent, 3) move to the right facing the opponent, 4) move to the left facing the opponent, 5) move backward facing the opponent (results in slower movement but the player can react to the opponent's attack), 6) move forward facing away from the opponent (faster retreat but unable to react to the opponent's attack).

D. Hyperparameters

Name	Value
start learning rate	1e-4
decay steps	33,000
decay rates	0.96
end learning rate	2e-5
optimizer	Adam
batch size	8 (240 transitions)
LSTM sequence length	30
send model to shared pool every C step	15,000
recent k models	10
replay size	800,000
entropy bonus coefficient	0.01
gradient clipping	20
Total HP reward weight	10
Win reward weight	10
Time reward weight per tick	-0.008~0.0
Distance reward weight per tick	-0.002~0.0

Table 6: Hyperparameter settings

E. Network Architecture

The network architecture employed in the BAB training process is described in Figure 6. Rectified linear unit (ReLU) is used as the activation function in all Multilayer perceptrons (MLPs) except the final MLP. The agent selects a skill by sampling from the pointwise multiplication result of (the Destroyer's) skill probability vector and skill availability vector. The discretized features are constructed as shown in Appendix B and are concatenated with the LSTM output.

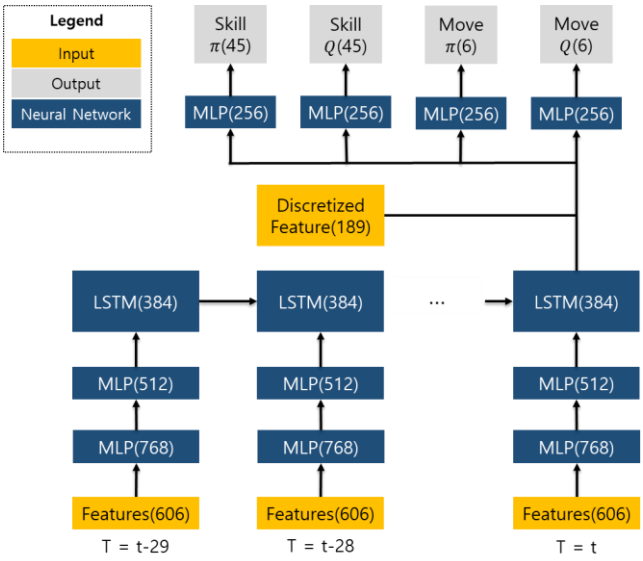


Figure 6. Network architecture