BACHELOR OF SCIENCE IN SOFTWARE ENGINEERING

Creating a novel adaptive AI for a Real-Time Fighting game.

Abdullah Al Muti
190042117
Iftikhar Imrul Khan
190042149
Taszid Izaz
190042138

Department of Computer Science and Engineering
Islamic University of Technology
June, 2024

Declaration of Candidate

This is to certify that the work presented in this thesis is the outcome of the analysis and experiments carried out by **Abdullah Al Muti** , **Iftikhar Imrul Khan** , and **Taszid Izaz** under the supervision of **Md. Nazmul Haque** , Assistant Professor, Department of Computer Science and Engineering, Islamic University of Technology, Dhaka, Bangladesh. It is also declared that neither this thesis nor any part of it has been submitted anywhere else for any degree or diploma. Information derived from the published and unpublished work of others have been acknowledged in the text and a list of references is given.

Md. Nazmul Haque
Assistant Professor
Department of Computer Science and Engineering
Islamic University of Technology (IUT)
Date: June 04, 2024

Abdullah Al Muti
Student ID: 190042117
Date: June 04, 2024

Iftikhar Imrul Khan
Student ID: 190042149
Date: June 04, 2024

Taszid Izaz
Student ID: 190042138
Date: June 04, 2024

Dedicated to all the cats in our campus

Contents

List of Figures

List of Tables

List of Abbreviations

GVGAI	General Video Game Artificial Intelligence
FTGAIC	Fighting Game AI Competition
RHEA	Rolling Horizon Evolutionary Algorithm
MCTS	Monte-Carlo Tree Search
UCB1	Upper Confidence Bound 1
RL	Reinforcement Learning
PG	Policy Gradient
QL	Q-Learning
AI	Artificial Intelligence
DareFightingICE	Dare Fighting ICE

Acknowledgement

I am profoundly grateful to my supervisor, Md. Nazmul Haque, for his infinite patience, insightful critiques, and for the support. His expertise and encouragement were crucial in the completion of this thesis. There's been multiple time we have been in the pit of despair and our supervisor saved us each time.

Thanks to my parents, whose love and support have been a constant source of strength. Thank you for always being there, for pretending to understand my research when I rambled on, and for not asking too many questions about why I was still not employed after all these years.

Abstract

AI has transformed how we play, bringing both online and offline gaming to exciting new levels.While progress has been made in many areas, research in game AI hasn't kept up like other fields. In this research, we want to create adaptable AI for video games that learns in real time from both the environment and opponents. Our pipeline consists of two major modules: a Monte Carlo Tree Search (MCTS) module for decision-making and a machine learning module to handle the action list. We use reinforcement learning, specifically Q-learning, to dynamically change the action list, which boosts the AI's performance. Our results show that this hybrid method not only improves AI speed and efficacy, but also helps it to react in real time, mimicking human strategic behavior. Experiments on limited-resources show that our approach works, with high win rates against a variety of AI opponents. Our approach provides an insight into combining MCTS and reinforcement learning for real-time adaptive AI in gaming situations.

Chapter 1

Introduction

Playing video games has always been a popular part of human life. Particularly since the turn of the twenty-first century, the evolution of online and offline video games has been accelerated by advances in artificial and computational intelligence. Over the course of the last 15 years or more, the subject of artificial intelligence research in games, or artificial intelligence in games, has seen a number of significant advances [43].

AI has been included into video games since the 1950s, and as a result, games have frequently been used as a helpful gauge of advancements in AI. In this category, several essential games, apps, software, and events are identified as game AI milestones[40]. For instance, the world's top Go player, Ke Jie, was recently beaten by Google DeepMind's AlphaGo[31].

DareFightingICE is a one-on-one fighting game platform organized by the Intelligent Computer Entertainment lab of Ritsumeikan University[27]. A one-on-one fighting game has unique opportunities for evaluating various AI systems. This game falls in between board and real-time strategy simulation genres. AIs in board games like Deep Blue[1]and AlphaGo[32] are given adequate time to anticipate player movements during gameplay. In one-on-one fighting games, AI must respond quickly due to limited time constraints. This makes fighting games more challenging than board games. Meanwhile, fighting games have shorter playtimes and lower action complexity than real-time simulation games like Starcraft or DOTA2. It offers the benefit of building and assessing AI algorithms with minimal computing resources.

Engineering algorithms that require minimal computational power opens up new horizons by enabling the inclusion of AI in devices with limited resources. This advancement allows for broader accessibility, ensuring that even less powerful computers can benefit from sophisticated AI technologies[41]. Such improvements enhance

the quality of generalized video game AI and have significant implications for the entertainment sector, particularly in adaptive video game AIs and Procedural Content Generation[26]. By optimizing resource usage, we can create more efficient, accessible, and sustainable AI solutions that can be widely applied across various domains. The main problem this research addresses is the creation of a more efficient and effective MCTS-based AI solution to enhance the General Video Game AI (GVGAI) framework.

1.1 Video Games and AI

Video Game: A video game is any game played on a video device that requires interacting with a user interface to produce visual feedback. A reward system, like a score, may be present in video games and is dependent on the player completing objectives inside the game[4].

Artificial Intelligence: The creation of computer systems that are capable of carrying out activities that normally require human intellect is referred to as artificial intelligence. Learning, thinking, problem-solving, speech recognition, comprehension of natural language, and visual perception are some of these skills.

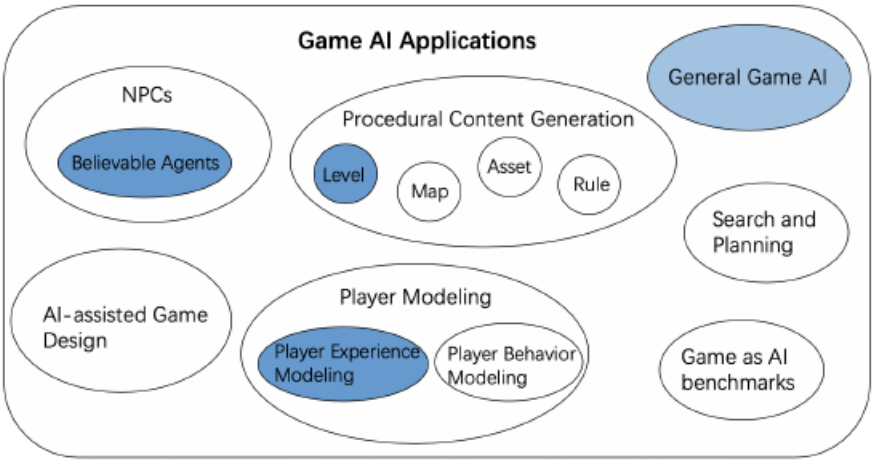

Figure 1.1: Game AI Applications

AI in Video Games: AI has made great advances in numerous elements of gaming, particularly in real-time strategy games. In this context, AI is used to manage NPC (non-player character) troops and make strategic decisions, increasing the complexity and difficulty of gaming [3], [11]. Furthermore, by mimicking variables such as weather patterns and animal behavior, AI approaches create more immersive and

realistic gaming worlds. Another important use of AI in gaming is adaptability to player behavior, which allows for a more customized and engaging experience[38]. For example, in a combat game, an AI-controlled opponent may evaluate the player's tactics and adjust its plans appropriately, resulting in more difficult and dynamic gaming[35]. By embracing AI's broad capabilities, game makers may build more intelligent, dynamic, and fascinating experiences for gamers, boosting the gaming industry[41]. Figure Figure 1.1 shows different applications of video game AI.

Game Architecture

Several essential elements of a game architecture come together to produce a seamless gaming experience. The main component is the "Game State," which is a representation of the virtual world as it is right now, complete with player positions, item placements, and other pertinent data. To ensure that motions and interactions seem genuine, the Simulator for Physics and Animation analyzes the game state to mimic realistic physics and animations.

In this virtual world, agents—who might be either players or non-player characters (NPCs)—interact[29]. AI algorithms or programming inform the decisions that agents make, affecting how they behave and react to external stimuli. These choices have an impact on the current state of the game and, in turn, the gameplay as a whole.

The task of converting the game state into visual components that are shown on the

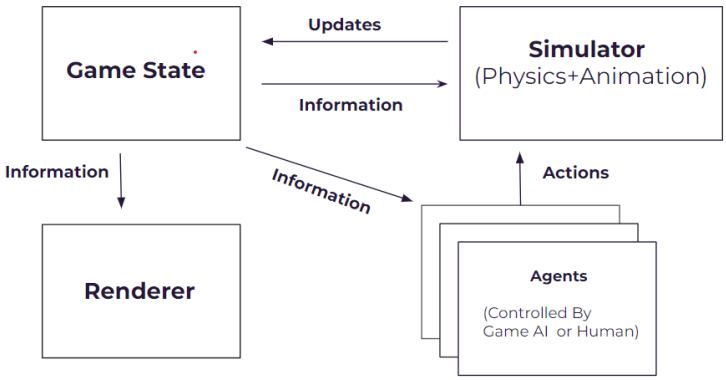

Figure 1.2: Game Architecture

screen falls to the renderer. It takes care of graphics, producing 2D or 3D visuals and making sure the user sees a true picture of the gaming environment.

The agents making decisions, the simulator updating the game state, and the renderer showing the player the updated state are all part of the continuous communication that occurs within the interaction loop. In order to provide the impression of dynamic, real-time gaming, this loop repeats quickly.

1.2 Adaptive AI

Adaptive AI: The term "adaptive artificial intelligence" describes intelligent systems or algorithms that may modify and improve their performance, behavior, or strategy in response to shifting situations, user interactions, or external factors [42].

Learning in Games

- **Online learning,** also known as incremental learning or learning from streaming data, involves updating a machine learning model continuously as new data becomes available.
- **Offline learning,** also referred to as batch learning or static learning, involves training a machine learning model on a fixed dataset without the ability to update the model as new data arrives.
- **Direct learning,** often referred to as model-free learning, involves the agent directly learning a policy or value function without explicitly modeling the underlying dynamics of the environment.
- **Indirect learning,** also known as model-based learning, involves the agent building an internal model of the environment to understand its dynamics and then using this model to make decisions.

Goals of Adaptive AI

1. Customize the game to fit each player's preferences and skill level.
2. Change the game's difficulty on-the-fly to keep it challenging but not too hard.
3. Make non-player characters (NPCs) act more like real people, responding intelligently to players.
4. Teach the AI to learn from experience and get better at the game over time.
5. Make sure the AI doesn't slow down the game and uses computer resources wisely.
6. Keep players interested and having fun by making the game react to their choices.
7. Enable the AI to handle updates or changes to the game without causing problems.

1.3 FightingIce and Real-Time Fighting Games

The FightingICE game platform was developed by the ICE Lab at Ritsumeikan University[16]. It has been the platform for the Fighting Game AI Competition (FTGAIC) series since 2015 and has undergone numerous improvements to provide a faster, more robust forward model and to address various exploits, enhancing the challenge. It's the official platform for Fighting Game AI Contest organized by the IEEE Conference on Games yearly. Players select a character and perform actions such as walking, jumping, crouching, punching, kicking, and guarding to combat their opponent. The objective is to defeat the opponent while avoiding their attacks. The main challenges in fighting games are the need for real-time responses, dealing with incomplete information (simultaneous moves without knowing the opponent's immediate response), and managing a complex and variable state-action space (each character has its own set of actions). This study uses FightingICE as the test platform, which features a simulator for evolution planning. The simulator allows for the use of search-based algorithms in the game. However, each character can perform up to 56 actions per frame, and the search must be completed within 16.67ms.

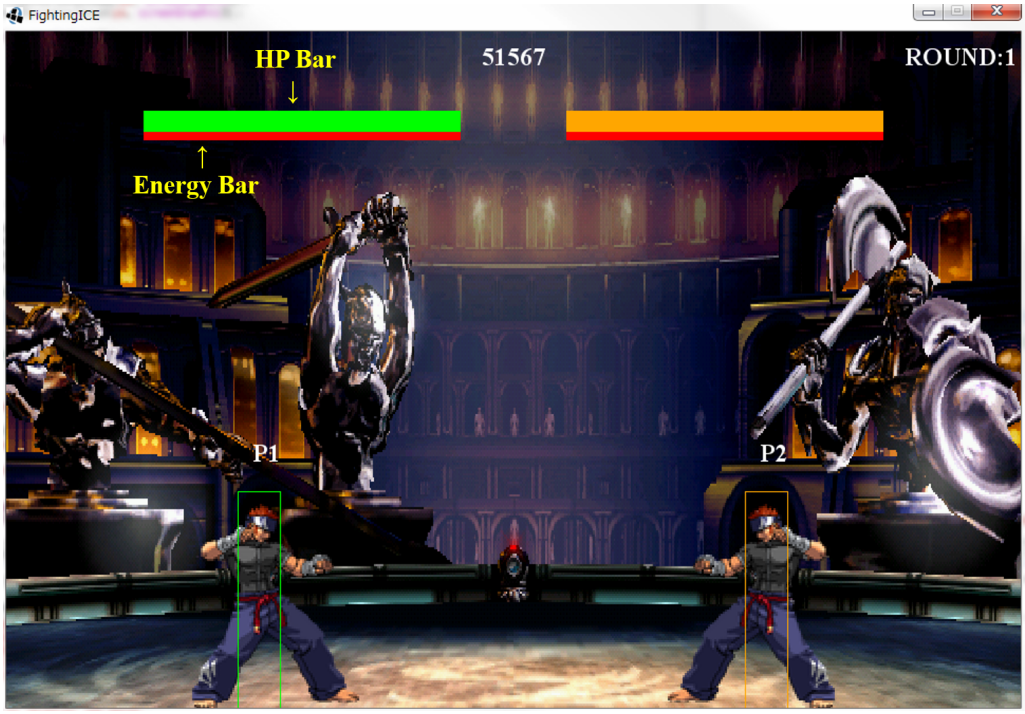

Figure 1.3: FightingICE Game Screenshot

1.4 Adaptive AI and Fighting Games

Fighting games test adaptive AI because of the inherent difficulties caused by the unpredictable nature of states, the requirement for extremely fast response times, and the requirement for real-time interaction. Testing makes sure the adaptive AI can manage fast changes, respond appropriately and quickly, and keep players' experiences responsive and interesting in the fast-paced, dynamic world of fighting games, where player actions continually impact the game state.
Because of its special features and architecture, DareFightingIce is an excellent platform for testing and developing adaptive AI. With its three-round format and unique knockout criteria, this one-on-one fighting game offers players a difficult setting in which to demonstrate their adaptive AI abilities. Contestants must model intricate moves since the 56 actions are intricate combinations. In addition, the imposed lag in the game's processing simulates human response time, making the experience more realistic and difficult[36].
All of these characteristics come together to make DareFightingIce a perfect testing ground for the creation of adaptive AIs in video games. It provides a dynamic and varied environment where competitors can showcase their skills in building intelligent systems that can quickly adjust to changes in real time and the strategic complexities of a competitive gaming scenario.

1.4.1 Motivation and Scope

As games increasingly reflect complicated real-world circumstances, implementing adaptable AI for gaming becomes critical. The aim is to develop AI systems that can dynamically adapt to various user actions and changing situations, not only improving gaming experiences but also serving as a testing ground for larger AI breakthroughs. Furthermore, the flexibility displayed in gaming settings may be applied to real-world applications such as autonomous systems, tailored learning environments, and responsive decision-making in a variety of sectors, emphasizing the social importance of adaptive game AI research.

AI development for the Fighting Game AI Competition (FTGAIC) and other gaming platforms has progressed significantly thanks to a variety of unique techniques. Yoshida et al[45] developed an AI for FTGAIC using the Monte-Carlo Tree Search (MCTS) algorithm, which has since become the Fighting Ice platform's standard. Lam et al[15] improved on this by including heuristics into the MCTS algorithm, lowering

computing costs while retaining performance. Oh et al[22] created a reinforcement learning (RL)-based AI with a self-play mechanism for the game *Blade & Soul*, which allows the model to constantly improve by training against itself. Perez et al[25] introduced the Rolling Horizon Evolution Algorithm (RHEA), which has successfully handled a variety of real-time control challenges and video games using evolutionary algorithms. Kim et al[13] created a deep learning-based AI that incorporates self-learning processes and bespoke heuristics with reward structuring to enable autonomous adaptability
. Creating a adaptive self learning AI in video games is hard as it can produce unpredictable behaviour which can ruin the fun of a game. Fun is what, most important for a game to have. Existing research in AI for gaming, particularly the use of the Monte-Carlo Tree Search (MCTS) algorithm, is quite popular. Although MCTS-based AI remains dominant in competitions, it becomes highly computationally intensive as the depth of the tree search increases, leading to performance bottlenecks. Pure reinforcement learning algorithms, while effective, also suffer from significant computational demands and slow processing times, further limiting their practicality. Additionally, these methods face significant challenges in real-time adaptive AI applications, where quick and efficient decision-making is crucial. The computational intensity of MCTS and reinforcement learning algorithms makes them less suitable for environments requiring rapid adaptation to dynamic changes. Furthermore, while integrating heuristics into MCTS has shown promise, there is still substantial room for optimizing these algorithms to balance computational efficiency with performance.

1.5 Problem Statement

MCTS algorithm with a big list of actions takes a lot of time and resource to complete a search. We can overcome this shortcoming of MCTS by creating an Adaptive AI that could actively shorten the list of available actions on demand learning from the environment online. For that the AI has to learn for which move it is getting punished and which moves provides the best reward. A model free learning algorithm in game environment to use it as the base learning model to detect punishing actions online and temporarily remove it from the action list so that MCTS AI agent can sample only the useful actions and find the best result in the given time. More precisely, our goal can be expressed as: **"To build an adaptive AI using which can adapt to environment and perform effective action for general video games."**

Objectives we want to achieve:

- Develop a simulation-based adaptive AI for general video games.
- Integrate heuristics to improve the efficiency and performance of MCTS AI.

1.6 Research Challenges

Developing adaptive game AI presents several research challenges:

- **Complex Decision Spaces:** Traditional search-based AI techniques struggle with the vast decision spaces found in modern computer games, particularly in genres like real-time strategy games or interactive dramas. To address this, learning techniques or higher-level representations are required to manage the complexity inherent in these games[28].
- **Knowledge engineering:** even if strategies or behaviors are created, creating these behavior sets in a game needs significant human engineering effort. Game creators must encode all of their information about a topic (either to create strategic behavior or convincing human behavior) in some form of behavior language[28].
- **Runtime Behavior Adaptation:** Removing the responsibility from game authors to anticipate every possible situation that can occur during gameplay is crucial for creating believable characters and engaging narratives. A framework for runtime behavior adaptation can help in this regard**chao2023multimagCNN**.
- **Adaptive AI Constraints:** Ensuring that adaptive AI does not disrupt the player's original conception of the game rules and elements is essential. Changes to game mechanics, such as difficulty levels, must be subtle enough not to break the player's immersion. Additionally, explaining the logic behind adaptive systems to players can be challenging, especially if the system involves complex rules or predicates[19].
- **Exploration vs. Exploitation:** Developing strategies that effectively balance exploration of new actions and exploitation of known successful strategies is crucial for adaptive performance[45].
- **Computational Efficiency:** Adaptive AI algorithms often require substantial computational resources, which can limit their deployment on devices with limited processing power. Optimizing these algorithms for efficiency is critical.

- **Logical Consistency and Coherent 'Magic Circle'**: Maintaining a coherent and logically consistent game world is critical for player engagement. Players expect a self-consistent logic within the game, and sudden deviations can lead to a loss of immersion. Furthermore, adaptive AI must respect the boundaries of the 'Magic Circle,' avoiding changes that force players to reconsider their understanding of the game world[2].

1.7 Contributions

We can encapsulate our significant contributions of our work in the following points:

- Developed a novel approach using MCTS [45] that optimizes action selection, reducing computational overhead and enhancing decision-making efficiency in adaptive AI.
- Integrated heuristics to streamline the search process within MCTS, making the algorithm more suitable for real-time applications in video games.

1.8 Organization

The rest of the paper is organized as follows: in Chapter 2, we describe some of the relevant works related to our domain, as part of our literature review. In Chapter 3, the overall pipeline of our work including a detailed explanation of each module of the framework is described. In Chapter 4, we analyze the results obtained by our pipeline under different hyperparameters and show necessary comparisons. Finally, we draw some conclusions and highlight potential future research directions in Chapter 5.

Chapter 2

Related Works

Scripted methods are common in game AI design, heavily relying on human expertise [19], [44]. In FightingICE, [30] uses the UCB algorithm to choose rule-based controllers dynamically. [18] created a real-time dynamic scripting AI that won the 2015 competition. However, these scripted agents are limited by predefined scenarios and are easily exploited by opponents. Since 2016, MCTS-based agents have dominated the FTGAIC. The FightingICE platform includes a built-in simulator, enabling the use of MCTS in real-time fighting games [45]. [14] attempts to model opponent strategies with a manual action table, but this method cannot defeat the 2016 FTGAIC winner due to its limited scope and complexity.

Deep reinforcement learning (DRL) has achieved impressive results in many real-time video games [12], [20], [23], [32], [39]. [8] showcases the potential of deep Q-learning in two-player real-time fighting games. [37] uses Hybrid Reward Architecture (HRA)-based DRL, which breaks down the reward function into parts and learns the value functions separately. Despite its promise, HRA-based DRL was still beaten by the MCTS-based AI GigaThunder, the 2017 FTGAIC champion.

Opponent modeling approaches are split into implicit and explicit methods. Implicit modeling maximizes the agent's expected reward without estimating the opponent's actions [9]. Explicit modeling, on the other hand, directly predicts the opponent's strategy [7]. Explicit modeling is more efficient in training, easier to explain, and simpler to combine with other algorithms compared to implicit methods.

2.1 MCTS based AI

Yoshida et al [45] explained in their paper how they used Monte-Carlo Tree Search algorithm for the DareFightingICE platform and gained success. The Four Basic Steps of the Monte Carlo Tree Search Algorithm (MCTS): Selection, Expansion, Simulation, and Backpropagation are outlined in the paper. As a whole, these procedures allow the AI to construct and improve a game tree iteratively, which directs decision-making via repeated stochastic simulations as shown at Figure Figure 2.1. One aspect of the

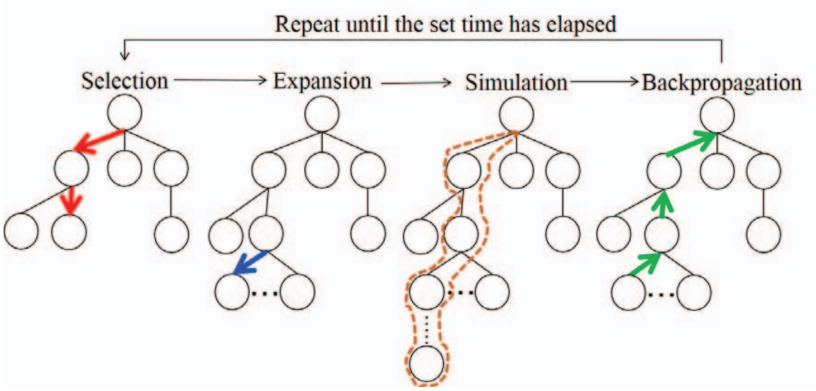

Figure 2.1: Outline of a Monte-Carlo Tree Search.

implementation is the tree policy, which is Upper Confidence Bounds 1 (UCB1). The UCB1 formula (3.1) finds a happy medium between exploration and exploitation by taking into account metrics such as the average playout and the number of node visits. To successfully traverse the ever-changing FightingICE environment, MCTS-based AI relies on this adaptive selection process of a node.

$$UCB1_i = \bar{X}_i + C\sqrt{\frac{2\ln N_i^p}{N_i}} \tag{2.1}$$

here the the exploitation part the first term of the equation is the average reward for the node for the current number of times it had been visited which is given by the equation (3.2)

$$\bar{X}_i = \frac{1}{N_i}\sum_{j=1}^{N_i} eval_j \tag{2.2}$$

where $eval_j$ is the value of the reward at the j-th simulation given by (3.3)

$$eval_j = (orginalHP^{my} - remainingHP^{my}) - (originalHP^{opp} - remainingHP^{opp}) \tag{2.3}$$

The experimental setup involves testing the MCTS-based AI against rule-based AIs, the predominant competitors in the Fighting Game AI Competition. The parameters chosen for the proposed AI, including the balance parameter (C), maximum number of visits ($N_{max} = 10$), maximum depth of the game tree ($D_{max} = 2$), simulation time ($T_{sim} = 60f$), and total time ($T_{max} = 16.5s$), are empirically determined to suit the unique characteristics of the gaming environment.When pitted against Machete, a finely tuned rule-based AI, the suggested AI fails miserably because Machete executes brief actions repeatedly, needing a low amount of frames. The MCTS simulations do not take this type of opponent behaviour into account. This means the planned AI would fail miserably when pitted against Machete.

Lam et al[15] pointed out a basic issue: the limitations of conventional methods when it comes to responsiveness and flexibility. While AI programs excel at turn-based games like chess, this study shows how these methods fail miserably when faced with real-time situations requiring quick decisions.
The article focuses on the Fighting Game AI challenge, a real-time tournament where AI combatants face severe time limits and restricted computing resources. Traditional AI approaches face new challenges in real-time gaming due to the increased emphasis on rapid and adaptive decision-making compared to turn-based games. The goal of the authors' algorithmic method is to combine the power of MCTS with the flexibility of case-analysis to handle a wide variety of game conditions.

Potential-Based Action Selection: To implement the idea in the fighting game AI challenge, author defined a potential for every action according to its startup time and its likelihood to send the opponent to a blocked state. Algorithm 1 outlines this heuristic.

Algorithm 1 Potential-Based Action Selection

1: **Input:** Current game state S
2: **Output:** Selected action a
3: Calculate potential for each action based on startup time and likelihood to block opponent
4: Choose action with highest potential

Defense-Attack Switch: This heuristic defines an upper HP threshold. When the HP difference compared to the opponent exceeds this threshold, AI switch to defense mode. Otherwise, the AI stay in attack mode. Algorithm 2 outlines this approach.

Algorithm 2 Defense-Attack Switch

```
Input: Our HP, Opponent HP
Output: Mode (attack/defense)
if (Our HP - Opponent HP) > threshold then
    Enter defense mode
else
    Enter attack mode
end if
```

Monte-Carlo Tree Search (MCTS): MCTS is used for gathering information from simulated futures via random simulations. The basic structure of MCTS involves selection, expansion, simulation, and backpropagation as shown in Figure 2.1.

Algorithm 3 Monte Carlo Tree Search

```
Input: game state S
Output: action a
while within computational budget do
    leaf ← selectNode(S)
    reward ← simulate(leaf)
    backpropagate(leaf, reward)
end while
return the action with the highest visit count
```

Hybrid AI Design: This hybrid AI design effectively blends simple, generic case-analysis with the more computationally intensive MCTS to handle real-time constraints in fighting games. This approach reduces dependency on training data and ensures robust performance across various game scenarios

Algorithm 4 Hybrid AI Design

```
function GETBESTACTION(S)
    if S ∈ 𝒦 then
        return SMARTSEARCH(S)
    else
        return MCTS(S)
    end if
end function
```

This work contributes significantly by presenting an algorithmic architecture that is scalable and opti- mised for real-time AI, with a focus on how to succeed in the difficult category. The suggested combination of basic case-analysis and Monte Carlo tree

search (MCTS) is significant since it has been successful in real-time combat games, an area where the majority of approaches relying on training are not feasible.

2.2 Reinforcement Learning based AI

Inseok Oh [22] introduced figting game AI using DL, which was published in IEEE transactions on games in 2022. The paper introduces a practical reinforcement learning [34] approach for 1v1 battles in the game "Blade & Soul," overcoming challenges like vast action spaces and imperfect information. They employ a novel self-play curriculum [33], shaping three distinct agent styles through reward mechanisms. Additionally, the paper proposes data skipping techniques to enhance efficiency and exploration in expansive action spaces. The trained agents achieved a 62% win rate against professional players, showcasing the method's potential applicability to various competitive games and game development tasks like level design and automated balancing.

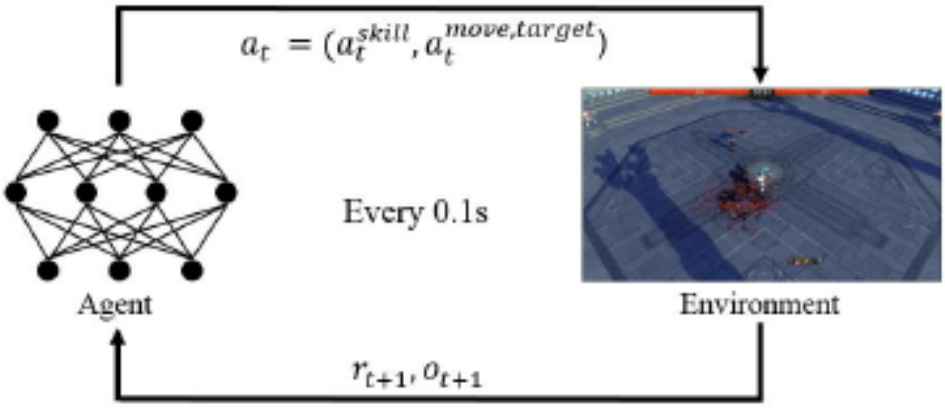

Figure 2.2: Agent-environment plot in BAB

To express BAB as a Markov Decision Process (MDP)[10], the author assumes a fixed policy pool, π_{op}. Figure Figure 2.2 illustrates the agent-environment interaction in BAB, where LSTM-based agents interact with a BAB simulator environment. At each time step (0.1 sec intervals), the state s_t is derived from the observation history $H_t = \{o_1, o_2, \dots, o_t\}$. s_t includes information accessible to a human player, such as HP, SP, opponent distance, arena wall distance, position, remaining game time, skill cooldowns, and status effects (e.g., midair, stun).

The agent's action $a_t = (a_{t,\text{skill}}, a_{t,\text{move}}, \text{target})$ is determined at each time step. The targeting action space, originally continuous, is discretized into facing towards or away

from the opponent. The chosen action is then executed in the environment, leading to a state transition.

Rewards in BAB are tied to performance. The overall reward r_t is defined as:

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

where r_t^{WIN} is a terminal reward of +10 for a win and -10 for a loss, and r_t^{HP} reflects HP margin changes:

$$r_t^{\text{HP}} = (\text{HP}_t^{\text{agent}} - \text{HP}_{t-1}^{\text{agent}}) - (\text{HP}_t^{\text{opponent}} - \text{HP}_{t-1}^{\text{opponent}})$$

Additionally,

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

where r_t^{time} penalizes longer game durations, and r_t^{distance} penalizes distance between agents. The discount factor γ is set to 0.995, as episodes terminate after 1,800 steps (3 minutes).

To control the degree of aggressiveness in fighting styles, the author used three reward dimensions. The first dimension is the "time penalty." An aggressive agent receives larger penalties per time step, motivating it to finish the match quickly. The second dimension is the relative importance of the agent's HP compared to the opponent's HP. Aggressive players focus on reducing the opponent's HP, while defensive players prioritize preserving their own HP. The final dimension is the "distance penalty." Defensive players maintain a certain distance to respond to attacks effectively, whereas aggressive players close the distance and attack relentlessly. The aggressive agent receives larger penalties proportional to the distance from the opponent.

Specific reward weights for each style are shown in Table 2.1. Each dimension can take continuous values, allowing for a spectrum of fighting styles with varying aggressiveness. However, for demonstration purposes, the author limited the fighting styles to three. This method, using additional reward signals along with r_t^{WIN} and r_t^{HP}, can be applied to other fighting games to create agents with various fighting styles.

Table 2.1: Reward details of each style

Style	Time Penalty	HP Ratio	Distance Penalty
Aggressive	0.008	5:5	0.002
Balanced	0.004	5:5	0.0002
Defensive	0.0	6:4	0.0

The proposed self-play curriculum, illustrated in Figure Figure 2.3, involves three types of agents, each with its own learning process, trained concurrently. Each learning process consists of a learner and multiple simulators working asynchronously. In the simulators, an agent plays matches against randomly sampled opponents from a shared pool. The most recent k models of each style are selected with total probability mass p, while other models are chosen with probability $1 - p$. The value of p is linearly annealed from 0.8 to 0.1 during training. A higher p enables rapid adaptation to recent opponents, and a lower p stabilizes learning by reducing catastrophic forgetting.

After each match, simulators send logs to the learner and update the agent with the latest parameters. The learner trains agents off-policy using logs from multiple simulators and updates simulators with the latest parameters on request. Additionally, the learner updates the shared pool with its network parameters every C steps (e.g., $C = 10{,}000$). The shared pool contains diverse policies from different learning processes, providing varied opponents for training. This ensures that agents encounter and learn to handle opponents with different fighting styles while maintaining their own.

Data skipping techniques involve dropping certain data during training and evaluation.

- **Discarding Passive "No-op":** In fighting games, skill usage consumes resources like SP and cooldown time. Overusing a skill can render it unavailable when needed. To address this, the author added a "no-op" action to the policy network's output, allowing the agent to do nothing when necessary. Thus, the action space includes 44 skills plus the "no-op" action. Human play logs of BAB indicate that "no-op" actions constitute the largest portion of skill usage.
  "No-op" decisions are categorized as passive and active. Passive "no-op" occurs when the agent has no available skills, such as when out of resources or hit by an opponent's CC skill. Active "no-op" is a strategic choice, made even when other skills are available.
  The author discarded passive "no-op" data from both training and evaluation be-

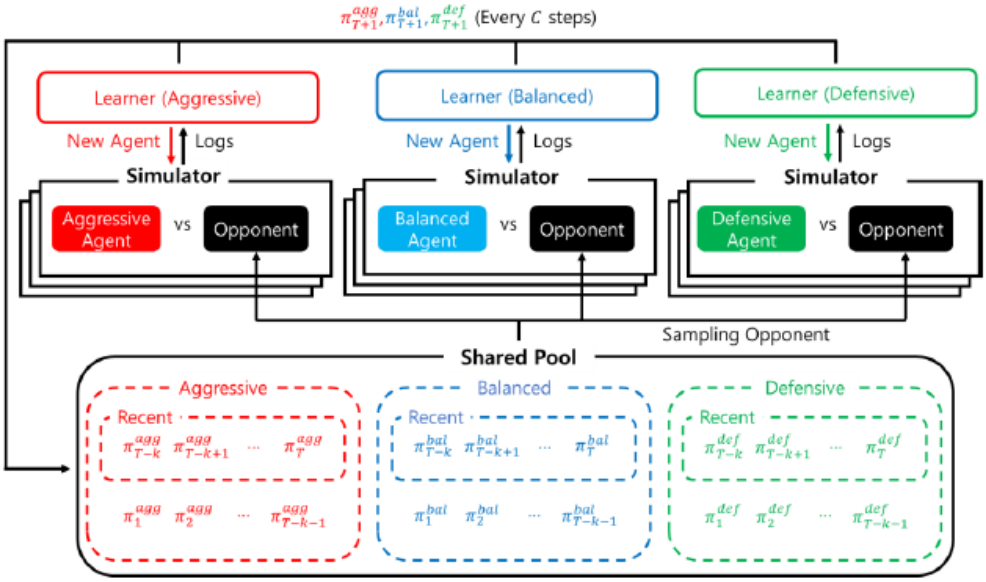

Figure 2.3: Overview of self self-play curriculum with three different styles

cause these actions are not deliberate. This method also helps the LSTM reflect longer time horizons since the data is not fed into the network. Experimental results show that skipping passive "no-ops" significantly improves learning efficiency. This methodology can be applied to other domains where the available skill set changes constantly, and the "no-op" action is a valid choice.

- **Maintain Move Decision:** While a single skill decision can significantly influence subsequent states, a single move decision has a limited effect due to the short distance a character moves in 0.1 seconds. For a move decision to be impactful, it must be made consecutively over several time steps. Training a move policy from an initial random policy is challenging because random decisions rarely repeat, limiting exploration. To address this, the author proposes maintaining the move decision for a fixed number of time steps.
  Figure 4 illustrates the "maintain move decision" method. When the agent selects a move action, it skips the move decision for the next $n - 1$ time steps, maintaining the same move decision for n steps total. This method differs from the frame skip technique used in the Atari domain [21], which improves simulator efficiency. In contrast, "maintain move decision" aims to enhance training by increasing the impact of a single move decision, which the author confirms through experiments. This approach is akin to the concept of 'amplifying advantage' from [21].

Kim et al [13] implemented a reinforcement learning (RL) agent using Proximal Policy Optimization (PPO) and self-play to achieve high-level performance in the Fighting-

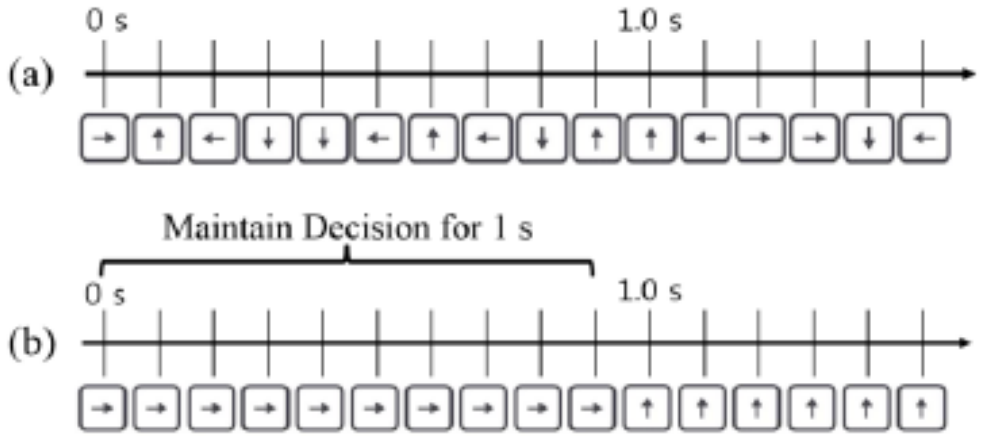

Figure 2.4: Figure 4. Examples of (a) regular move decisions and (b) main- taining decisions for 1 second

ICE game. They employed a two-stage learning schedule and a prioritized self-play algorithm, with a multi-layer perceptron (MLP) network architecture.

A. State and Action

The state s is derived from the game environment and used as input to the neural network. It includes features like HP, energy, position, and more, as shown in Table 2.2. The actions a are represented by 56 possible moves, with action masking to filter invalid actions.

Table 2.2: Features for State

Feature Name	**Size**
HP	1
Energy	1
EnergyT5	1
...Additional Features...	
Total Features	165

B. Reward

The reward r is composed of three components: HP difference, win/lose result, and a time penalty. The reward for differences in HP is normalized between -10 and 10. The overall reward G_t at time t is:

$$G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} r_k$$

C. Reinforcement Learning Algorithm

The Proximal Policy Optimization (PPO) algorithm is used. The policy gradient loss L_{PG} is:

$$L_{PG} = \hat{E}[\log \pi_\theta(a_t|s_t)G_t]$$

The advantage function $\hat{A}_t$ is:

$$\hat{A}_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$

The final clipped loss L_{CLIP} is:

$$L_{CLIP} = \hat{E}\left[\min\left(r_\theta \hat{A}_t, \text{clip}(r_\theta, 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right]$$

D. Self-play Learning

The agent is first trained against an MCTS AI and then through self-play. This method involves a two-stage training schedule, with initial training against a heuristic opponent followed by self-play to enhance robustness and strategy diversity. Figure 2.2 illustrates the two-stage training schedule.

E. Network Architecture

The network is a multi-layer perceptron (MLP) with layers defined in Table 2.3. The architecture is designed to process the state vector and produce action probabilities and state value estimates.

Table 2.3: Network Architecture

Layer	**Size**
Input	165
Hidden Layer 1	256
Hidden Layer 2	128
Output	56

Initially, training against a Monte Carlo Tree Search (MCTS) AI provided a strong foundational strategy, while subsequent self-play refined and diversified the agent's tactics. This approach, combined with a well-structured reward system and an effective neural network architecture, resulted in an AI that outperformed other competi-

tion entries, demonstrating high win rates and strong overall performance .

The paper's limitations include the reinforcement learning algorithm's specific design for the FightingICE platform, limiting its generalizability to other games. The training process demands substantial computational resources, making it less accessible. The two-stage learning process involving self-play and MCTS AI adds complexity, complicating result reproduction. Evaluation metrics like win rate and HP difference may not fully capture the agent's adaptability and learning efficiency. Additionally, the reward structure, although optimized, might still lead to suboptimal training outcomes if not balanced correctly. These limitations suggest areas for future research, such as improving generalization, optimizing resources, simplifying strategies, and refining metrics.

2.3 Evolutionary algorithm based AI

2.3.1 Rolling Horizon Evolutionary Algorithms

Perez et al. (2013) [6], [25] explored the potential of Rolling Horizon Evolutionary Algorithms (RHEA) in the context of general video game playing. Their paper provided an analysis of several enhancements designed to improve the performance of RHEA on games from the General Video Game AI (GVGAI) framework. The key enhancements explored in their study include Bandit-Based Mutation, Statistical Tree for Action Selection, Shift Buffer for Population Management, and Additional Monte Carlo Simulations.

Bandit-Based Mutation: The approach selects mutation operators based on their previous performance. The Upper Confidence Bounce (UCB) model is used to balance exploration and exploitation.In the equation Equation 2.4 the first term (Q(s, a)) tries maximizing the value of the play. The second term favors levers which were pulled the least number of times, N(s, a) indicating the number of times lever a was pulled and N(s) the total number of plays. The constant C is that which balances between the two terms.This strategy enhances the adaptive selection process in the game environment. similar to the approach used by Yoshida et al. (2019) with MCTS [45].

$$U = \arg\max_{a \in A(s)} \left\{ Q(s,a) + C\sqrt{\frac{\ln N(s)}{N(s,a)}} \right\} \tag{2.4}$$

Statistical Tree for Action Selection: This method involves maintaining a statistical tree alongside the evolutionary process and monitoring the effects of various activities shown in figure Figure 2.5. It chooses actions based on the highest value averaged over evolution. This data enables the algorithm to make informed decisions based on historical performance, similar to the tree policy in MCTS, which guides decision-making through repeated stochastic simulations.

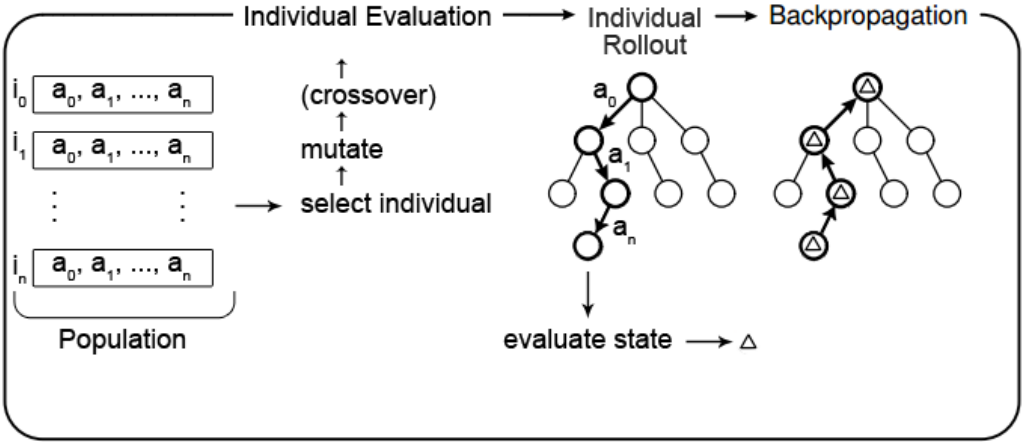

Figure 2.5: RHEA statistical tree steps

Shift Buffer for Population Management: This approach maintains population by shifting a buffer, which keeps decent people for generations.This helps to preserve useful genetic material from past generations. This is conceptually similar to maintaining a diverse set of simulations in MCTS to ensure robust performance [17].

Additional Monte Carlo Simulations: Running extra simulations to provide a more robust assessment of each candidate solution's potential performance, reducing the likelihood of selecting suboptimal actions. This method enhances the reliability of the evaluations, similar to the role of extensive simulations in MCTS [24].

Experimental Setup: Their enhancements were tested against baseline RHEA and MCTS. The parameters for RHEA were determined to suit the characteristics of the gaming environment. Similar to the parameters chosen for MCTS by Yoshida et al. (2019) [45], these include balance parameters, maximum visits, tree depth, and simulation time [25].

2.3.2 Enhanced Rolling Horizon Evolution Algorithm with Opponent Model Learning

Tang et al. (2020) [36] suggested an updated Rolling Horizon Evolution Algorithm (RHEA) coupled with opponent model learning for the Fighting Game AI Competi-

tion (FTGAIC), building on the previous paper. Tang et al. introduced a novel technique that combines RHEA with opponent model learning, making it adaptable to any two-player video game.

Opponent Model Learning: Unlike conventional RHEA, the proposed approach includes an opponent model that is optimized through supervised learning with cross-entropy and reinforcement learning using policy gradient and Q-learning. This model shown in Figure 2.6 learns during live gameplay based on historical observations of the opponent's actions. The opponent model allows the algorithm to make more realistic plans by predicting the opponent's likely actions.

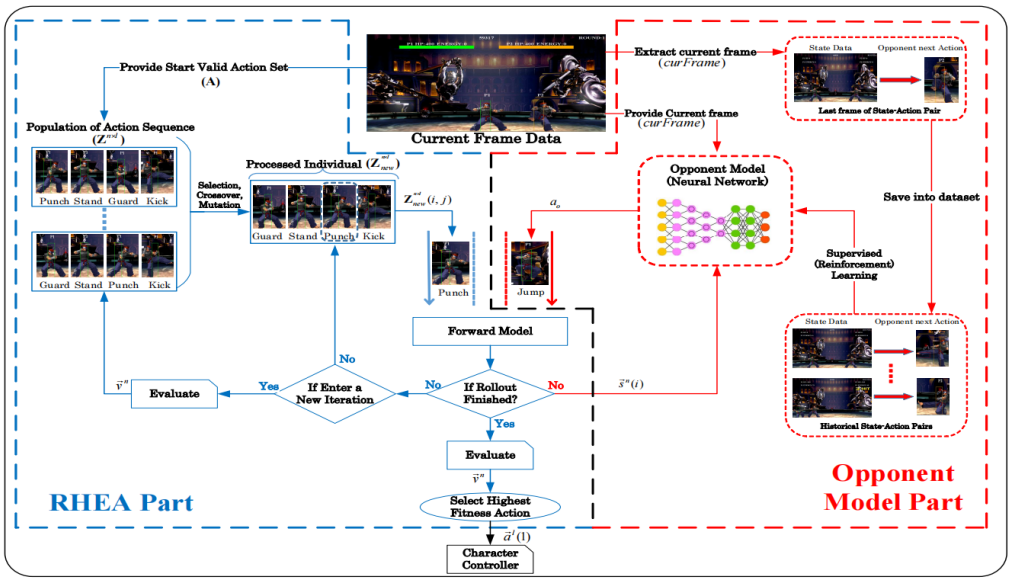

Figure 2.6: Flow diagram of RHEA Opponent Modeling

Rolling Horizon Evolution Algorithm (RHEA): RHEA is used to optimize the player's actions over a rolling horizon. By incorporating the learned opponent model, the algorithm can anticipate the opponent's moves and make more informed decisions. The flow model of RHEA is shown in the Figure 2.7

Supervised Learning and Reinforcement Learning: The opponent model is trained using a combination of supervised learning and reinforcement learning techniques. Supervised learning with cross-entropy helps in predicting the opponent's actions based on previous observations, while policy gradient and Q-learning refine the model to improve its predictive accuracy.

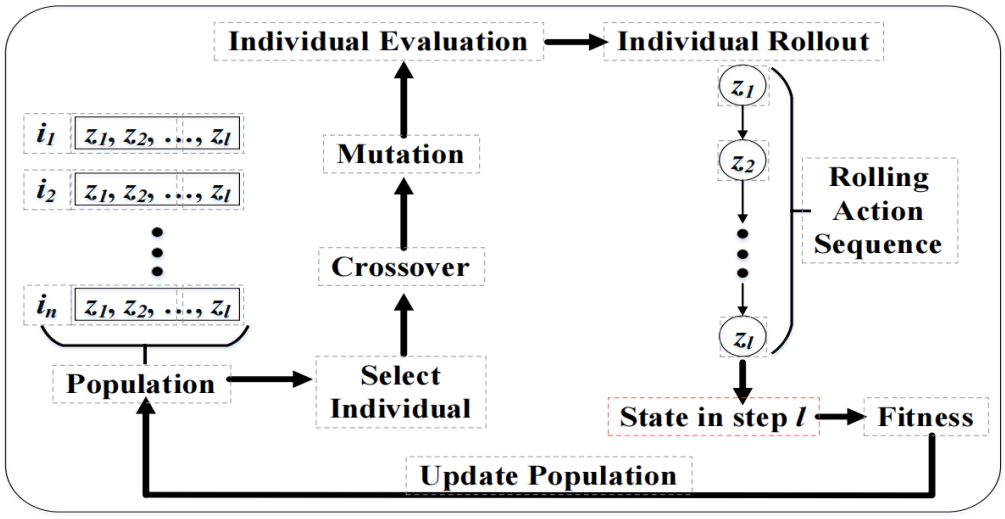

Figure 2.7: Flow diagram of RHEA

Chapter 3

Proposed Methodology

The objective of our research is to create adaptive AI for video games that will in runtime learn from environment and opponent. Our general pipeline is made of two modules. Firstly a MCTS module which will decide the action for the AI, and secondly a machine learning module which maintains the action list of the AI. In the background study, MCTS based AI [45] dominating over other machine learning based AI. But pure MCTS AI performs very badly against AIs with repeated moves[15]. We observed if we, change the action set of our AI based on what opponent AI is performing, MCTS wins by a big margin. Also because we are shortening the action list, the AI becomes fast and less unsuccessful on attack and defence. To mimic human behaviour where, where human learn and figure out a strategy in real time, we adopted Reinforcement learning approach for AI, so it can adapt to opponent in real time[13]. We also have to be conservative of the resources we have as RL based approach can be quite resource consuming. We are employing Q learning and Deep Q learning for our machine learning module which will prune action list for the MCTS module. This section begins with the description of general pipeline, followed by overview of every modules.

3.1 Overview of Pipeline

Action list is actions the agent can perform. There are currently 56 actions for each character. We start the agent with a small action list to shorten the search tree for MCTS. When the first round starts, agent will start fighting using its MCTS module. We set this action list based on our observation during various tests. With it's action list and a low search depth, AI will select the best action from the list against opponent AI. We chose MCTS as it is best fit because of it's speed and accuracy. During the

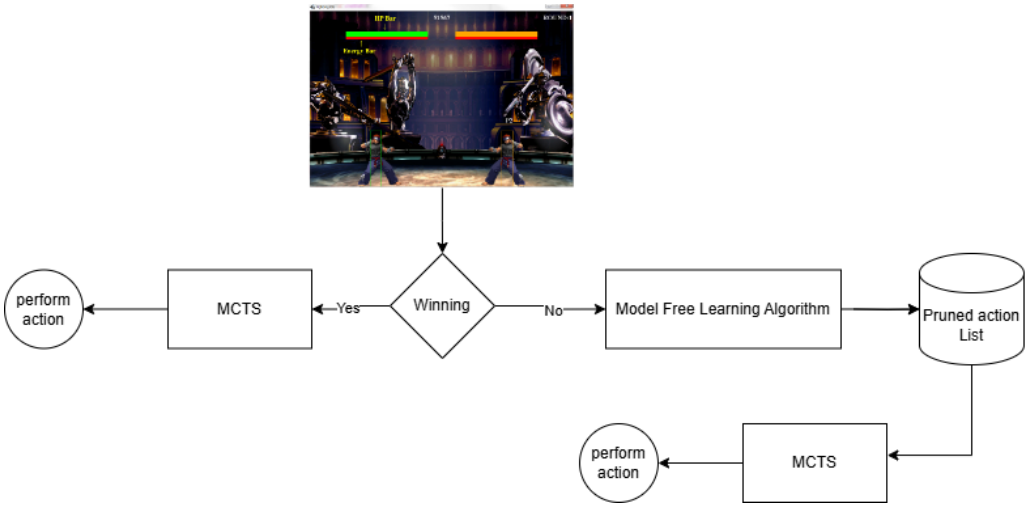

Figure 3.1: Overall Pipeline

fight agent will keep calculating if it's winning or not. If agent is winning then there will be no need to change the action list. Current action list is good enough for the opponent. If agent start to lose against opponent, it indicates the current action list is disadvantageous. Then we use our model free learning algorithm to create a new list of actions that can be advantageous against the current opponent. Once the list is complete the, MCTS again start picking actions based on the new list as per shown in Figure Figure 3.1.

3.2 Action Searching Module

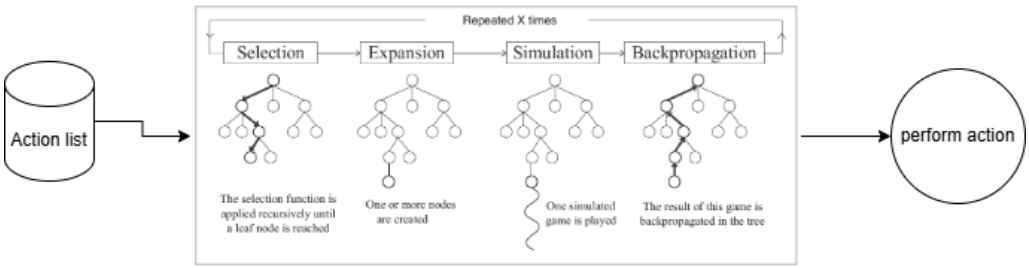

Figure 3.2: Outline of Action Searching Module

For action searching from the list of action we used MCTS. We are employing UCB1 formula for balancing exploration and exploitation. The process involves constructing a tree where nodes represent actions. The UCB1 is used to select the most promising node to explore further.

$$UCB1_i = \bar{X}_i + C\sqrt{\frac{2\ln N_i^p}{N_i}} \tag{3.1}$$

In formula Equation 3.1 $\bar{X}_i$ is the average reward of the node. C is the exploration

constant. N_i^p is the number of times the parent node has been visited. $\bar{X}_i$ represents exploitation, ensuring that the nodes with high average rewards are prioritized. The second term $C\sqrt{\frac{2\ln N_i^p}{N_i}}$ represents exploration, encouraging the algorithm to explore less visited nodes.
Average reward for a node is calculated using formula Equation 3.2

$$\bar{X}_i = \frac{1}{N_i}\sum_{j=1}^{N_i} eval_j \tag{3.2}$$

where $eval_j$ is the value of the reward at the j-th simulation given by (3.3)

$$eval_j = (orginalHP^{my} - remainingHP^{my}) - (originalHP^{opp} - remainingHP^{opp}) \tag{3.3}$$

For exploration constant C, the value is chosen $C = 3$. C controls the balance between exploration and exploitation. A higher value will make MCTS explore more nodes. In the context of our platform, the environment is dynamic so $C = 3$ is a balanced choice to explore and priortize promising nodes.The parameters, such as the balance parameter (C), maximum number of visits ($N_{max} = 10$), maximum depth of the game tree ($D_{max} = 2$), simulation time ($T_{sim} = 23f$), and total time ($T_{max} = 16.5s$), are empirically determined to fit the gaming environment.

3.3 Action Pruning Module

We can express Dare Fighting ICE as an Markov Decision Process [5] to formulate a model free algorithm to learn from the game environment.

- **State:** The state is a vector containing the player's position relative to the enemy, player energy, enemy energy, and the current state of both player and enemy (Standing, Air, Crouch, Down).
- **Action Space:** All 56 named actions supported by the environment.

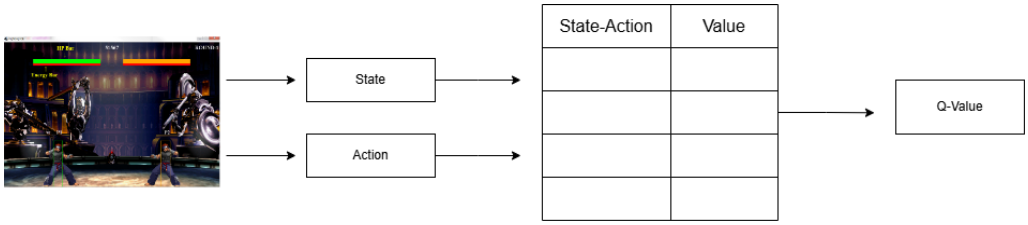

Figure 3.3: Outline of Q-Learning

The agent maintains a Q-table, which stores the expected reward (Q-value) for taking a specific action (a) in a given state (s). The Q-table is initialized with zeros and updated throughout training using the following equation:

$$Q(s,a) \leftarrow (1-\alpha) \cdot Q(s,a) + \alpha \cdot \left[R(s,a) + \gamma \cdot \max_{a'} Q(s',a') \right] \tag{3.4}$$

where,

- $Q(s,a)$: The maximum Q-value for any action in state s'.
- α: This parameter controls how much the agent prioritizes new experiences $R(s,a)$ vs. past experiences $Q(s,a)$.
- $R(s,a)$: The reward received by the agent for taking action a in state s.
- γ:This parameter determines how much the agent considers future rewards when evaluating an action.
- s': The state the agent transitions to after taking action a in state s.
- a': The action the agent would take in state s'.

The agent uses the exploration-exploitation principle to balance between trying new actions (exploration) and taking the action with the highest expected reward (exploitation). Exploration Rate, ϵ parameter determines the probability of the agent taking a random action instead of the one suggested by the Q-table.ϵ-greedy policy: With probability $(1-\epsilon)$, the agent takes the action with the highest Q-value in the current state. With probability ϵ, it takes a random action.
The parameters were chosen based on adjustments made through experimentation. α reduced from 0.7 to 0.1 due to longer episode lengths in the fighting game environment. This allows the agent to learn slower and retain knowledge over time. γ left at 0.95 as suggested. This value considers future rewards when making decisions.

Chapter 4

Results and Discussion

4.1 Experiment Setup

All the Experiments were done at our local machine. We used virtual machine to limit resources and keep consistency among other machines. Each VM were allocated with 3 cores and 8GB RAM. GPU was not used as our goal was to create AI with limited resource access. The DareFightingIce version was 6.3 and JDK version 17.

4.2 Evaluation Criterias and Score

To judge a how a AI is performing most researchers used previous year AIs to benchmark their AI. The criteria is simple. If the AI win against Most AI, it can be considerate a good AI. As well as since the MCTS AI is the current benchmark AI of Dare Fighting Ice, researchners would pit their AI against their AI MCTS to see how better it does or can it win.
Yoshida et al [45] proposed a scoring formula Equation 4.1 for his AI. A game consists of three rounds and each round runs for 60s.

$$Score_my = \frac{HP_opp}{HP_my + HP_opp} * 1000 \tag{4.1}$$

Where, HP_x is the amount of Health Points the or the opponent at the end of a round. We calculated win rate of the AI we created by using the formula Equation 4.2.

$$W_my = \frac{R_w}{R_t} * 100 \tag{4.2}$$

Where, W_my is the win rate, R_w is the rounds won and R_t is the total rounds played.

4.3 Evaluating Different AI Models

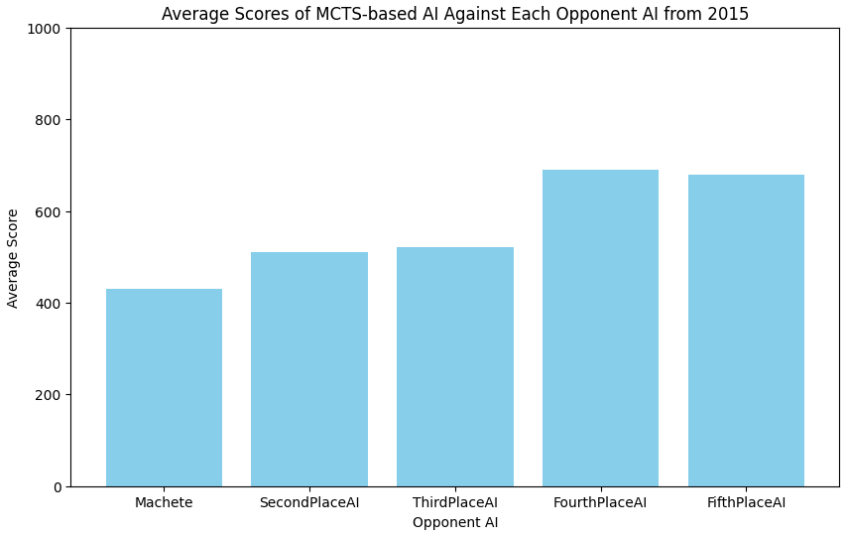

Figure 4.1: Average scores against each AI in 2015

The suggested MCTS-based AI achieved average scores versus each opponent AI throughout 100 games as shown in the graph Figure 4.1. In this graph, the vertical axis indicates the average score, while the horizontal axis reflects the opponent AI in descending order of the 2015 competition rating. The suggested AI outperformed all opponent AIs, with scores above 500.

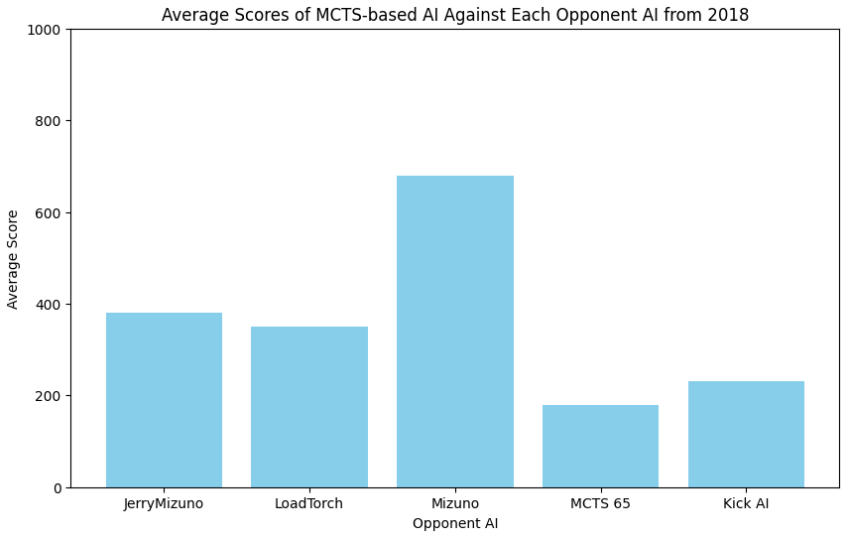

Figure 4.2: Average scores against each AI in 2018

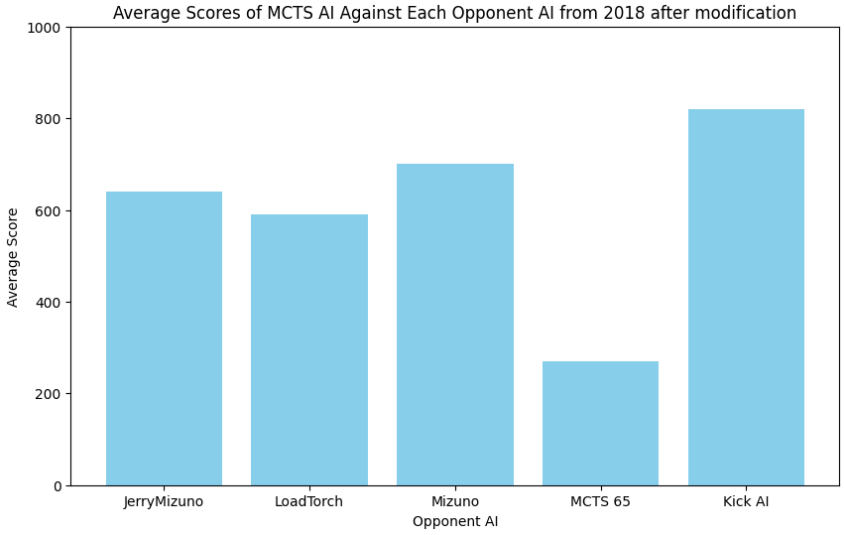

Figure 4.3: Average scores against each AI in 2018 after modification

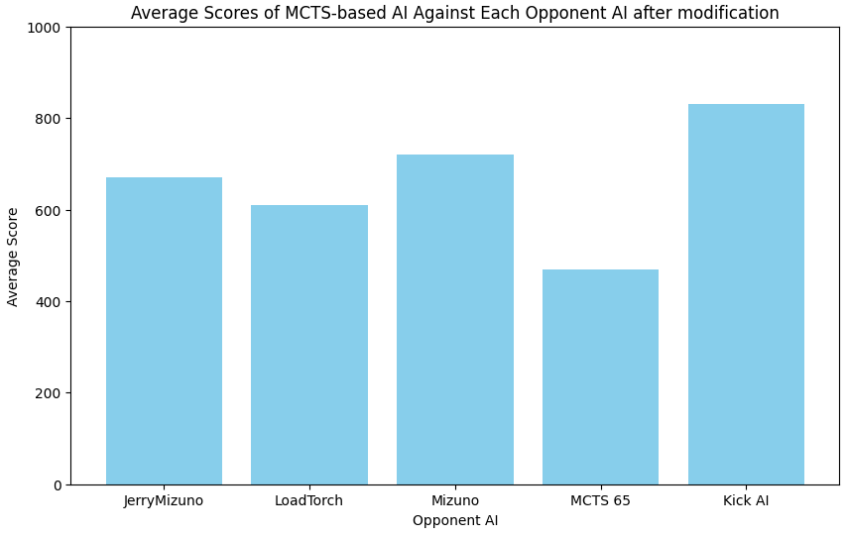

Figure 4.4: Average scores against each AI after modification

We tested the same AI against AIs after 2018. Notable inclusion is three RL based AI. We recorded score throughout 30 games against the AIs and the results are shown in figire Figure 4.2. *JerryMizuno* and *LoadTorch* is a deep learning based AI. MCTS AI lost most of the matches. But It performed poorly against *KickAI*. KickAI is the simplest AI which does nothing but performs kicks only. We then noticed against *KickAI*, MCTS was getting punished to the same move over over again. We removed that action from the list and the MCTS gained overwhelming victory over *KickAI*.

For each AI then observed and handpicked actions for the action list for every round. If for each AI, tailor the action list then we get a 100% win streak. But the action list we picked for *KickAI* does not work well against *LoadTorch* or vice versa.

The MCTS AI we were using had iteration rate of 23. Dare Fighting Ice provides us

with simulator. The MCTS AI uses this simulator to find the best action possible using random opponent action. Since the time difference is very low (.16mS) we simulated using the previous frame. The results were not changed much but we noticed something signaficant. With this change MCTS-23 was able to dodge every long distance attack.

Increasing the iteration rate from 23 to 65 made a big difference. MCTS 65 was able to defeat every AI with 100% win rate of every AI mentioned. We notied how the MCTS 23 was punished by MCTS 65, then we removed the actions that were being punished for. From 0% of winning rate against MCTS 65, it became 18%. After more action chosing and shortening the list we increased the winning rate to 71%. Observing these behaviour we proposed our methodology. Figure Figure 4.4 shows how shortening the action list and choosing the correct actions significantly increases win rate.

Chapter 5

Conclusion

The main research objectives were to create adaptive AI for video games by combining Monte Carlo Tree Search (MCTS) with machine learning approaches. We utilize a better fighting model where we shorten the action list in runtime and perform search action. This approach showed better results than past AIs we tested against. Major understandings of the performance of various AI models in gaming contexts have been generated through experimentation and analysis.

These findings add to the present body of literature in game AI by demonstrating the effectiveness of adaptive techniques in addressing the constraints of standard AI systems. In addition, the study highlights the need for more research and development in this area, as gaming AI research has not evolved at the same rate as other sectors.

Understanding the study's limitations, such as resource constraints and the complexity of gaming settings, provides useful insights into future research directions. Future research should focus on developing a general AI not only for this platform, but also also games to test adaptability. There are some other novel techniques and algorithms for adaptive AIs which need to be incorporated and tested against mainstream video games.

References

[1] M. Campbell, A. J. Hoane Jr, and F.-h. Hsu, "Deep blue," *Artificial intelligence*, vol. 134, no. 1-2, pp. 57–83, 2002.

[2] B. U. Cowley and D. Charles, "Adaptive artificial intelligence in games: Issues, requirements, and a solution through behavlets-based general player modelling," *arXiv preprint arXiv:1607.05028*, 2016.

[3] A. Dachowicz, K. Mall, P. Balasubramani, *et al.*, "Mission engineering and design using real-time strategy games: An explainable ai approach," *Journal of Mechanical Design*, vol. 144, no. 2, p. 021 710, 2022.

[4] N. Esposito, "A short and simple definition of what a videogame is," in *Proceedings of DiGRA 2005 Conference: Changing Views: Worlds in Play*, 2005.

[5] E. A. Feinberg and A. Shwartz, *Handbook of Markov decision processes: methods and applications*. Springer Science & Business Media, 2012, vol. 40.

[6] R. D. Gaina, S. M. Lucas, and D. Perez-Liebana, "Rolling horizon evolution enhancements in general video game playing," in *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, IEEE, 2017, pp. 88–95.

[7] S. Ganzfried and T. Sandholm, "Game theory-based opponent modeling in large imperfect-information games," in *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, vol. 2, 2011, pp. 533–540.

[8] M. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable mdps," in *Proceedings of the AAAI Fall Symposium Series*, 2014, pp. 29–37.

[9] H. He, J. Boyd-Graber, K. Kwok, and H. D. III, "Opponent modeling in deep reinforcement learning," in *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, 2016, pp. 1804–1813.

[10] R. A. Howard, "Dynamic programming," *Management Science*, vol. 12, no. 5, pp. 317–348, 1966.

[11] S. Huang, S. Ontañón, C. Bamford, and L. Grela, "Gym-μrts: Toward affordable full game real-time strategy games research with deep reinforcement learning," in *2021 IEEE Conference on Games (CoG)*, IEEE, 2021, pp. 1–8.

[12] M. Kempka, M. Wydmuch, J. Runc, T. Toczek, and W. Jaśkowski, "Vizdoom: A doom-based ai research platform for visual reinforcement learning," in *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, 2016, pp. 1–8.

[13] D.-W. Kim, S. Park, and S.-i. Yang, "Mastering fighting game using deep reinforcement learning with self-play," in *2020 IEEE Conference on Games (CoG)*, IEEE, 2020, pp. 576–583.

[14] M. Kim and K. Kim, "Opponent modeling based on action table for mcts-based fighting game ai," in *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, 2017, pp. 178–180.

[15] G. T. Lam, D. Logofătu, and C. Bădică, "A novel real-time design for fighting game ai," *Evolving Systems*, vol. 12, pp. 169–176, 2021.

[16] F. Lu, K. Yamamoto, L. H. Nomura, S. Mizuno, Y. Lee, and R. Thawonmas, "Fighting game artificial intelligence competition platform," in *2013 IEEE 2nd Global Conference on Consumer Electronics (GCCE)*, IEEE, 2013, pp. 320–323.

[17] S. M. Lucas, S. Samothrakis, and D. Perez, "Fast evolutionary adaptation for monte carlo tree search," in *EvoGames*, 2014.

[18] K. Majchrzak, J. Quadflieg, and G. Rudolph, "Advanced dynamic scripting for fighting game ai," in *Lecture Notes in Computer Science*, vol. 9353, 2015, pp. 86–99.

[19] I. Millington, *AI for Games*. CRC Press, 2019.

[20] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[21] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[22] I. Oh, S. Rho, S. Moon, S. Son, H. Lee, and J. Chung, "Creating pro-level ai for a real-time fighting game using deep reinforcement learning," *IEEE Transactions on Games*, vol. 14, no. 2, pp. 212–220, 2021.

[23] OpenAI, "Dota 2 with large scale deep reinforcement learning," *arXiv preprint arXiv:1912.06680*, 2019.

[24] D. Perez, S. Samothrakis, and S. M. Lucas, "Knowledge-based fast evolutionary mcts for general video game playing," in *IEEE Conference on Computational Intelligence and Games*, 2014, pp. 1–8.

[25] D. Perez, S. Samothrakis, S. M. Lucas, and P. Rolfshagen, "Rolling horizon evolution versus tree search for navigation in single-player real-time games," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, 2013, pp. 351–358.

[26] D. Perez-Liebana, J. Liu, A. Khalifa, R. D. Gaina, J. Togelius, and S. M. Lucas, "General video game ai: A multitrack framework for evaluating agents, games, and content generation algorithms," *IEEE Transactions on Games*, vol. 11, no. 3, pp. 195–214, 2019.

[27] D. Perez-Liebana, S. Samothrakis, J. Togelius, S. M. Lucas, and T. Schaul, "General video game ai: Competition, challenges and opportunities," in *30th AAAI Conference on Artificial Intelligence*, 2016.

[28] A. Ram, S. Ontañón, and M. Mehta, "Artificial intelligence for adaptive computer games.," in *FLAIRS*, 2007, pp. 22–29.

[29] F. Safadi, R. Fonteneau, and D. Ernst, "Artificial intelligence in video games: Towards a unified framework," *International Journal of Computer Games Technology*, vol. 2015, pp. 5–5, 2015.

[30] N. Sato, S. Temsirirkkul, S. Sone, and K. Ikeda, "Adaptive fighting game computer player by switching multiple rule-based controllers," in *3rd International Conference on Applied Computing and Information Technology/2nd International Conference on Computational Science and Intelligence*, 2015, pp. 52–59.

[31] D. Silver, A. Huang, C. J. Maddison, *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[32] D. Silver, A. Huang, C. J. Maddison, *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[33] S. Sukhbaatar, Z. Lin, I. Kostrikov, G. Synnaeve, A. Szlam, and R. Fergus, "Intrinsic motivation and automatic curricula via asymmetric self-play," *arXiv preprint arXiv:1703.05407*, 2017.

[34] R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction mit press," *Cambridge, MA*, vol. 22447, 1998.

[35] M. Świechowski, D. Lewiński, and R. Tyl, "Combining utility ai and mcts towards creating intelligent agents in video games, with the use case of tactical

troops: Anthracite shift," in *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*, IEEE, 2021, pp. 1–8.

[36] Z. Tang, Y. Zhu, D. Zhao, and S. M. Lucas, "Enhanced rolling horizon evolution algorithm with opponent model learning: Results for the fighting game ai competition," *IEEE Transactions on Games*, vol. 15, no. 1, pp. 5–15, 2020.

[37] C. Tessler, S. Givony, T. Zahavy, D. J. Mankowitz, and S. Mannor, "A deep hierarchical approach to lifelong learning in minecraft," in *Proceedings of the 31st Conference on Neural Information Processing Systems*, 2017, pp. 4905–4914.

[38] C. Tian, Z. Xu, L. Wang, and Y. Liu, "Arc fault detection using artificial intelligence: Challenges and benefits," *Mathematical Biosciences and Engineering*, vol. 20, no. 7, pp. 12 404–12 432, 2023.

[39] O. Vinyals, I. Babuschkin, W. M. Czarnecki, *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[40] Wikipedia, *Artificial intelligence in video games*, Accessed: 2024-06-03, 2024. [Online]. Available: `https://en.wikipedia.org/wiki/Artificial_intelligence_in_video_games`.

[41] Y. Wu, A. Yi, C. Ma, and L. Chen, "Artificial intelligence for video game visualization, advancements, benefits and challenges," *Mathematical Biosciences and Engineering*, vol. 20, no. 8, pp. 15 345–15 373, 2023.

[42] B. Xia, X. Ye, and A. O. Abuassba, "Recent research on ai in games," *2020 International Wireless Communications and Mobile Computing (IWCMC)*, pp. 505–510, 2020.

[43] G. N. Yannakakis and J. Togelius, "A panorama of artificial and computational intelligence in games," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 7, no. 4, pp. 317–335, 2014.

[44] G. N. Yannakakis and J. Togelius, *Artificial Intelligence and Games*. Springer, 2018, vol. 2.

[45] S. Yoshida, M. Ishihara, T. Miyazaki, Y. Nakagawa, T. Harada, and R. Thawonmas, "Application of monte-carlo tree search in a fighting game ai," in *2016 IEEE 5th Global Conference on Consumer Electronics (GCCE)*, 2016, pp. 1–2.