# Development of an Hierarchical Task Network AI in a Fighting Game

Bradley Ramsay

***Abstract:* This paper proposes an artificial intelligence agent, *KeepAwayFighter,* which implements a *Hierarchical Task Network* (HTN) planner with the use of *Upper Confidence Bounds* (UCBs) as the decision policy. The plan focuses on dealing damage from a distance with the use of long range attacks, and maintaining a health points (HP) superiority over the opponent. Experiments show that *KeepAwayFighter* can adhere to this strategy, despite failing to outperform the sample *MCTS* controller and the top controller submitted to the *2017 Fighting Game AI Competition.***

**Keywords:** artificial intelligence, fighting games, hierarchical task networks, upper confidence bounds, video games.

1. INTRODUCTION

AI in fighting games is either too predictable to offer a challenge to experienced players, or relies too much on cheating for the experience of fighting them to be fun. An AI opponent that can plan a path to victory, and change steps of that plan when they fail, is ideal as a challenging opponent. Thus, a *Hierarchical Task Network* is suggested as a way to allow the AI to plan and follow a series of steps to victory, with the flexibility to switch tactics if one of those steps fails.

HTNs are an approach to automated planning, where each individual action of a strategy and the dependency among actions can be represented as hierarchically structured networks. Solutions to problems are specified in the *Hierarchical Task Network* approach by providing a set of tasks, which can be *primitive tasks*, which are simple actions that would have specific effects in the real world, and *compound tasks*, which can be regarded as being composed of a set of *primitive tasks.*

The goal of an HTN is to provide a plan formed from an executable sequence of primitive tasks, which are usually obtained by decomposing compound tasks into their set of simpler tasks, which could then be sorted depending on ordering constraints. A plan could be one that, in the case of *KeepAwayFighter*, fulfils the goal of dealing damage to the opponent from a distance.

1. BACKGROUND

For the purpose of testing the AI agent, recording results and inputs used, the *FightingICE* framework will be used, seen in Figure 1. *FightingICE* provides an environment representative of many similar commercial fighting games: the setting is a two-dimensional stage with spatial limits, where two players can move around and perform several attack and defense actions. The game is played asynchronously in real-time, with a 15 frame delay on any information sent to the AIs. This is done to simulate a more human response time for AIs using the engine.

There are three unique characters for *FightingICE*: ZEN, GARNET and LUD. Each character has their own special skills (such as projectile attacks, throws, special attacks), which all have different requirements to activate and produce different effects. Information on each character’s skills is stored in their character data. *FightingICE* also provides a sample AI implemented with *Monte Carlo Tree Search*, aptly called *MctsAi*, as a way to test user created AI agents.

*FightingICE* has been used as a testing platform for AI research competitions (Feiyu Lu et al., 2013), namely the CIG conferences. It is developed and maintained by Intelligent Computer Entertainment Lab. (ICE Lab.), Ritsumeikan University, which is also the organizer for the AI competitions. It also provides the frameworks of winning AIs from previous competitions as resources for anyone to use, such as *GigaThunder*, the winner of the 2017 competition which was also implemented with a variation of *Monte Carlo Tree Search*.

Fig. 1: ZEN fighting ZEN on *FightingICE.*

1. KEEP AWAY FIGHTER

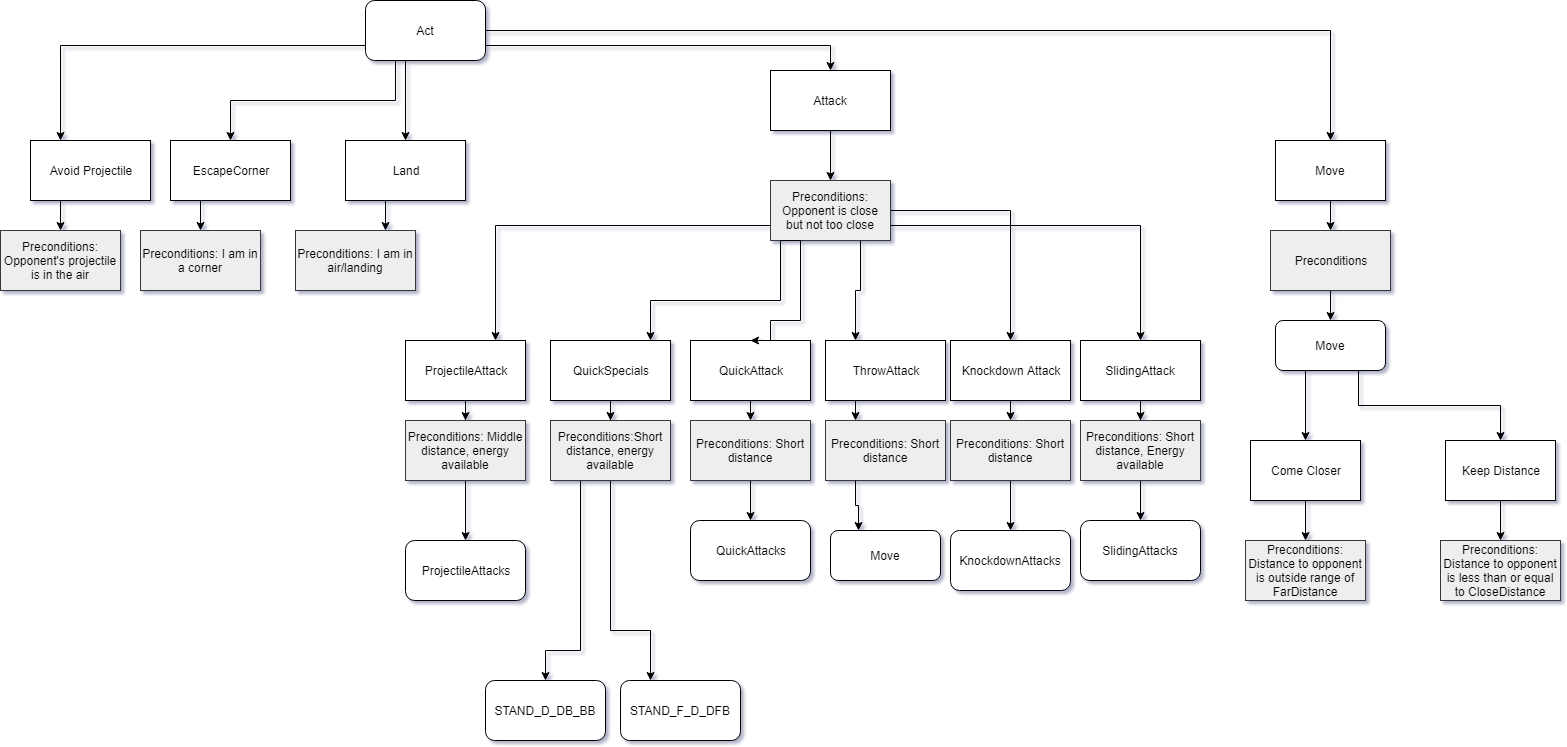
The artefact, *KeepAwayFighter*, is an artificial intelligence program created in Java that was made to run on *FightingICE.* *KeepAwayFighter* was created with the use of *Hierarchical Task Networks* (HTNs) and *Upper Confidence Bounds* (UCBs).

An *Agile* development process was utilised in creating *KeepAwayFighter*, with prototypes of the AI created at different stages of research. The first stage began with learning how to use *FightingICE*. A sample AI was created as a proof of concept for the engine. Its function was simple: every action it chose would be random based on Java’s *Random* library, meaning that it followed no uniform strategy. Once the AI was proven to work on *FightingICE*, time was then taken to research HTNs in depth. It was here that the HTN for *KeepAwayFighter* was developed in diagram form, as seen in Figure 2.

A prior implementation of HTNs was published in 2017, which was used as inspiration for *KeepAwayFighter*. That implementation, called *HTNFighter*, prioritised utilizing strings of attacks, also called combo attacks, that would leave the opponent stunned as a way to maximise damage (Neufeld, Mostaghim, & Perez-Liebana, 2017). *HTNFighter* was only compatible with version 3.20 of *FightingICE* however, being unusable for the current version 4.00. This was due to several factors, one of which being that the mechanisms for accessing frame data for combo attacks were stripped away, crippling the main feature of *HTNFighter*.

However, the overall code structure of *HTNFighter* was sound, with the use of the *Singleton* and *Factory* design patterns (JournalDev, 2018) for creating the various elements of an HTN. Thus when creating the HTN of *KeepAwayFighter*, *HTNFighter’s* code base was used as a guide. One such way to do this was to re-configure *HTNFighter* so that it could be run on the current version of *FightingICE*.

Fig. 2: A high level HTN for *KeepAwayFighter*



This became the second prototype for KeepAwayFighter.

With a working version of the stripped down *HTNFighter*, it was then possible to implement the HTN developed for *KeepAwayFighter*. This became the third prototype and first proper iteration of *KeepAwayFighter*, as it was here when the AI truly employed the strategy that was its namesake.

After doing some research on UCBs, the final build of *KeepAwayFighter* was developed. This build relied on using UCBs to determine the order of methods that *KeepAwayFighter* would access during gameplay.

1. UPPER CONFIDENCE BOUNDS

An algorithm that balances the concepts of exploitation and exploration of a given list of actions. Confidence bounds show with a high degree of certainty, the range of values that the true average reward of an action can be found in. The more times that action is chosen, the more data gained on the expected value of the reward. Thus with enough observations on each possible action, a good idea of which action has the highest upper confidence bound, and the greatest likelihood of giving the highest reward. By finding the action with the highest UCB, we can exploit that action to get the best reward.

In regards to *KeepAwayFighter,* UCBs will determine which primitive tasks would receive more priority in fulfilling the “keep away” strategy, with more successful actions being used more often. To reflect this implementation of *KeepAwayFighter*, a *UCB Planner* was created, to distinguish from the *Ordered Planner* which relied instead on the order in which tasks were defined to determine what actions *KeepAwayFighter* would use.

1. EVALUATING KEEP AWAY FIGHTER

In order to evaluate the AI, a series of tests were performed. The AI was made to fight against four other AIs: *MctsAi, GigaThunder, JayBotAI*, and *FooAI*. They fought for 100 matches, each 3 rounds of 60 seconds. Half of these rounds would have *KeepAwayFighter* start from the left side of the stage, with the other half starting from the right. Both contestants started with 400 HP, with the winner of a round determined by who had more HP than the other when time was up, or by who could reduce the other’s HP to 0 first. While these matches were in progress, *KeepAwayFighter’s* plans were recorded, as well as the number of wins/losses and the remaining health of both participants at the end of the round. By evaluating the win/loss ratio of *KeepAwayFighter*, it was possible to determine if its strategy was effective against the chosen opponents, whereas evaluating the plans made would determine how closely it was able to follow the proposed strategy. Both the *Ordered* and *UCB Planner* were tested to gauge if UCBs improved on *KeepAwayFighter’s* decision making process.

1. RESULTS

After pitting *KeepAwayFighter* against the four AIs, it was found that *KeepAwayFighter* was no match for either of them, losing nearly every match. This was the case with both the *Ordered Planner* and *UCB Planner*. For discussion purposes, we will focus on *MctsAi* and *GigaThunder* (1st place winner of CIG 2017).

The *Ordered Planner* was only successful in beating the *MctsAi* once in 300 rounds. It was also able to reduce *MctsAi’s* health to 50% or below 27% of the time, whilst keeping a health level above 0 36% of the time in 300 rounds. The *UCB Planner* was not as successful, failing to beat *MctsAi* in any rounds. It was also less likely to reduce *MctsAi’s* health to 50% or below, only accomplishing it once, while also only keeping an HP level above 0 23.7% of the time, as is seen in Figures 3 and 4.

|  |  |
| --- | --- |
| **When Using Ordered Planner** | |
| **Health** | **Number of Rounds** |
| MctsAi HP <= 50% | 82 |
| MctsAi HP > 50% | 218 |
| KeepAwayFighter HP == 0 | 191 |
| KeepAwayFighter HP > 0 | 109 |

Fig. 3: table showing health levels at the end of each round.

|  |  |
| --- | --- |
| **When using UCB Planner** | |
| **Health** | **Number of Rounds** |
| MctsAi HP <= 50% | 1 |
| MctsAi HP > 50% | 299 |
| KeepAwayFighter HP == 0 | 229 |
| KeepAwayFighter HP > 0 | 71 |

Fig. 4: table showing health levels at the end of each round.

*KeepAwayFighter* had a harder time against *GigaThunder*, with neither *Planner* successful in beating it in any of the rounds. As is seen in Figures 5 and 6, the *Ordered Planner* performed better than the *UCB Planner*, being able to reduce *GigaThunder’s* HP to 50% or below in 4.7% of the rounds, whereas the *UCB Planner* was not able to do so at all. A similar case exists for keeping *KeepAwayFighter’s* HP above 0 at the end of each round, with the *Ordered Planner* doing so in 0.7% of the rounds whilst the *UCB Planner* had a 0% success rate.

|  |  |
| --- | --- |
| **When using Ordered Planner** | |
| **Health** | **Number of Rounds** |
| GigaThunder HP <= 50% | 14 |
| GigaThunder HP > 50% | 286 |
| KeepAwayFighter HP == 0 | 298 |
| KeepAwayFighter HP > 0 | 2 |

Fig. 5: table showing health levels at the end of each round.

|  |  |
| --- | --- |
| **When using UCB Planner** | |
| **Health** | **Number of Rounds** |
| GigaThunder HP <= 50% | 0 |
| GigaThunder HP > 50% | 300 |
| KeepAwayFighter HP == 0 | 300 |
| KeepAwayFighter HP > 0 | 0 |

Fig. 6: table showing health levels at the end of each round.

In order to tell if *KeepAwayFighter* was attempting to follow its strategy, the plans made against *MctsAi* and *GigaThunder* were recorded, with note taken of the plans conceived by *Ordered Planner* versus the ones made by *UCB Planner*.

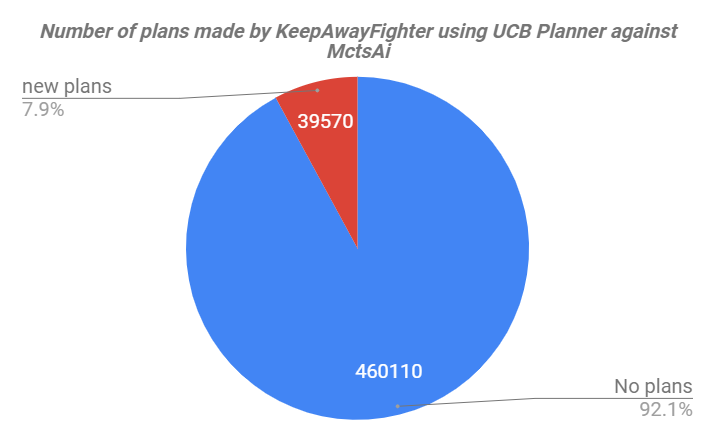


Fig. 7: Pie chart showing *UCB Planner’s* ability to make plans for *KeepAwayFighter.*

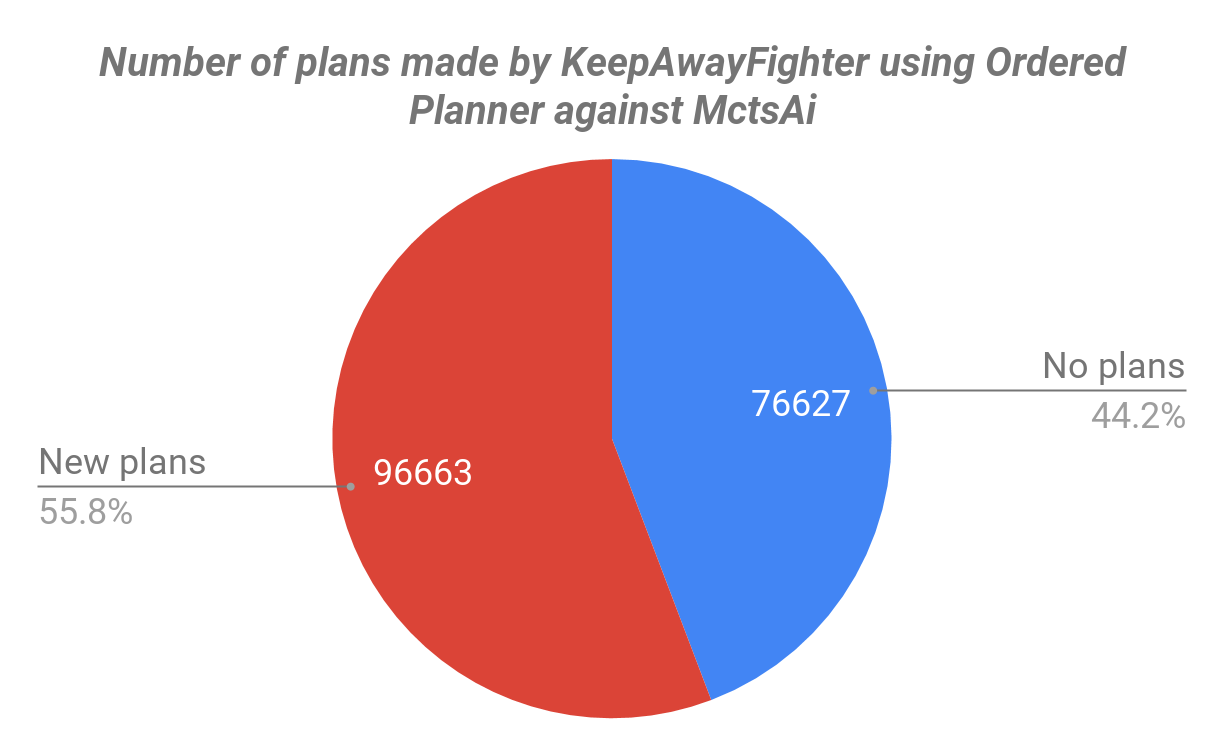


Fig. 8: Pie chart showing *Ordered Planner’s* ability to make plans for *KeepAwayFighter.*

Against *MctsAi*, *KeepAwayFighter* conceived and executed plans successfully 55.8% of the time when using an *Ordered Planner*, as opposed to the 7.9% when using *UCB Planner*, as seen in Figures 7 and 8.

A further analysis of the actions used in each plan provided interesting results, as seen in the bar charts shown in Figures 9 and 10. With the *Ordered Planner, KeepAwayFighter* employed a greater variety of offensive moves that supported the main strategy, such as *STAND\_D\_DF\_FA and STAND\_D\_DF\_FB*, which are projectile atacks, and knockback attacks such as *THROW\_A* and *STAND\_F\_D\_DFB*. The *UCB Planner* relied more on movement based actions like *FOR\_JUM*P and *DASH* and used attacks more sparingly, limiting common usage of ranged attacks to *STAND\_D\_DB\_BB.* This could be seen as the effect of UCBs, as the *Planner* determined through training that evasive actions provided less risk than offensive actions.

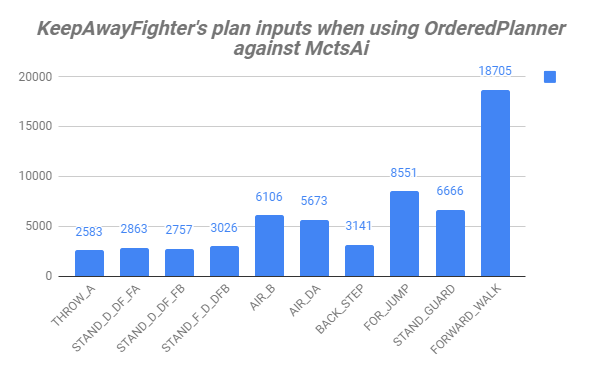


Fig. 9: The top ten moves used in an *Ordered Planner’s* plan.

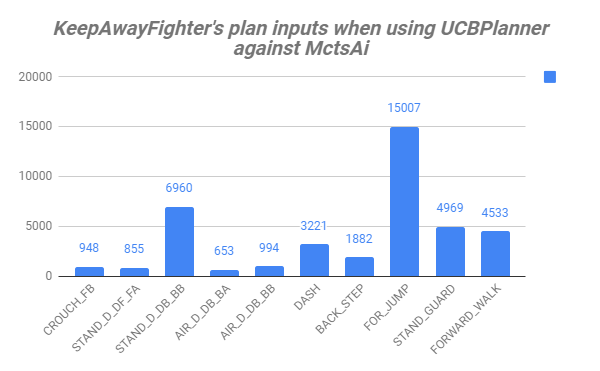


Fig. 10: The top ten moves used in a *UCB Planner’s* plan.

As for *GigaThunder*, an analysis of the actions used in an *Ordered Planner* showed a similar tendency for a large variety of attacks, as seen in Figure 11. However, there was a particular focus on *STAND\_D\_DB\_BA*, which was an attack that could also be used evasively. *STAND\_D\_DB\_BB* also stood out as the second most-used attack, mirroring the *UCB Planner’s* most used attack against *MctsAi.* This was also the case for the *UCB Planner* when coming up for plans against *GigaThunder*, as is seen in Figure 12, easily being the top attack used in most plans.

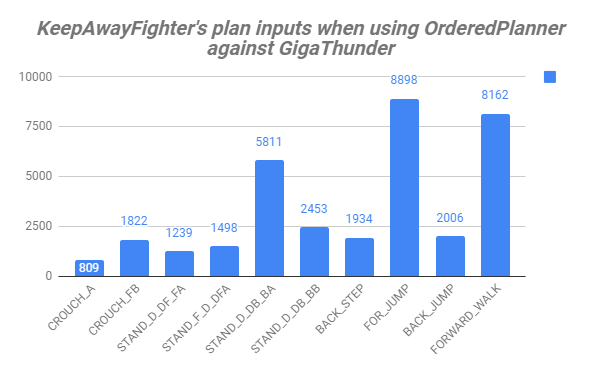


Fig. 11: The top ten moves used in an *Ordered Planner’s* plan.

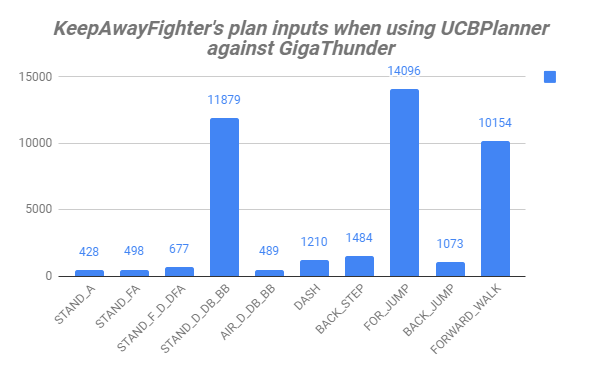


Fig.12: The top ten moves used in a *UCB Planner’s* plan.

1. DISCUSSION AND CONCLUSION

When enough energy was gathered, *KeepAwayFighter* adhered to its strategy quite well, utilizing actions that fell in line with dealing damage from afar. However, this reliance on energy is a key weakness of *KeepAwayFighter*, because several of the more effective actions required significant amounts of energy. This was the case with ZEN’s *STAND\_D\_DB\_BB* (one of *KeepAwayFighter’s* most popular moves), and *STAND\_F\_D\_DFB*, as both required 50 units of energy to perform. *KeepAwayFighter* would then spend a significant amount of the match either getting damaged or using less effective attacks in order to build the necessary energy for those actions. Most of ZEN’s projectile attacks required significantly less energy, but also had significant startup periods, making them easy to defend against if used at the wrong time.

This was likely taken into account in the plans made by *UCB Planner,* as they focused more on evasive actions instead of dealing damage. While training *KeepAwayFighter*, offensive actions that relied on projectiles or special attacks with high energy costs would end up being phased out of potential plans, due to their confidence bounds getting too low. Unfortunately, this sometimes led to situations with the *UCB Planner* where *KeepAwayFighter* stopped attacking entirely and only employed evasive actions, inevitably leading to its loss.

#### Future Work

Future implementations of *KeepAwayFighter* could focus on redesigning the HTN to further distinguish between evasive actions and attacks. Such a separation could be advantageous when it comes to UCBs, as different confidence bounds can be defined for evasive actions and attacks, so that *KeepAwayFighter* is less likely to be put in a position where it cannot attack.

Alternatively, *KeepAwayFighter’s* HTN could have tasks devoted towards gathering energy safely so that it would have more options in disadvantageous situations.

Different behaviours can also be encouraged depending on the difference in health between *KeepAwayFighter* and its opponent, with *KeepAwayFighter* being more evasive if it has significantly more health than its opponent, and more aggressive if the situation was reversed.

A significant amount of time was spent understanding UCBs, which impacted on the time spent implementing them. Future implementations of *KeepAwayFighter* should consider how UCBs will impact the decision making process of *KeepAwayFighter* around the same time as when the HTN is conceived.

Available character data should also be considered. A “keep away” strategy may simply be a bad fit for ZEN given the energy requirements for some of his more effective attacks, but may be more suitable for GARNET or LUD. A few tests were run on both characters, but not enough to provide an educated answer on whether this would be the case, due to the fact that these characters’ data was not as readily available as ZEN’s was at the time.

REFERENCES

Feiyu Lu, Yamamoto, K., Nomura, L. H., Mizuno, S., YoungMin Lee, & Thawonmas, R. (2013). Fighting game artificial intelligence competition platform. Paper presented at the *Consumer Electronics (GCCE), 2013 IEEE 2nd Global Conference* Retrieved from <https://ieeexplore.ieee.org/document/6664844>

JournalDev. (2018). Java Design Patterns - Example Tutorial - JournalDev. [online] Available at: https://www.journaldev.com/1827/java-design-patterns-example-tutorial [Accessed 5 Sep. 2018]

Kempka, M., Wydmuch, M., Runc, G., Toczek, J., & Jaskowski, W. (2016). ViZDoom: A doom-based AI research platform for visual reinforcement learning. Paper presented at the *Computational Intelligence and Games (CIG), 2016 IEEE 4th Global Conference,* Retrieved from <https://ieeexplore.ieee.org/document/7860433>

Kim, M., & Kim, K. (2017). Opponent modeling based on action table for MCTS-based fighting game AI. Paper presented at the *Computational Intelligence and Games (CIG), 2017 IEEE 4th Global Conference,* Retrieved from <https://ieeexplore.ieee.org/document/8080432>

Nakagawa, Y., Yamamoto, K., & Thawonmas, R. (2014). Online adjustment of the AI's strength in a fighting game using the k-nearest neighbor algorithm and a game simulator. Paper presented at the *Consumer Electronics (GCCE), 2014 IEEE 3rd Global Conference,* Retrieved from <https://ieeexplore.ieee.org/document/7031274>

Neufeld, X., Mostaghim, S., & Perez-Liebana, D. (2017). HTN fighter: Planning in a highly-dynamic game. Paper presented at the *2017 9th Computer Science and Electronic Engineering (CEEC)* Retrieved from<https://ieeexplore.ieee.org/document/8101623>

Tozour, P., & Austin, I. S. (1980). The Evolution of Game AI. In Rabin, S. (Ed.). *AI game programming wisdom* (pp 3-5). Cengage Learning.

Yoshida, S., Ishihara, M., Miyazaki, T., Nakagawa, Y., Harada, T., & Thawonmas, R. (2016). Application of monte-carlo tree search in a fighting game AI. Paper presented at the *Consumer Electronics, 2016 IEEE 5th Global Conference*, Retrieved from <https://ieeexplore.ieee.org/document/7800536>