Evaluating Hierarchical Task Networks in a Video Game Environment

Frank-Edward Nemeth,
University of Toronto,
1002158899

Abstract

Hierarchical Task Network (HTN) Planners have several applications in artificial intelligence and automated planning. Commonly used in videogames, HTNs often must include robust presentations of world states, as well as have mechanisms to deal with plan failures[1]. This project will assess the application of a HTN Planner in a fighting game environment, FightingICE. This project builds upon HTNFighter, a prior HTN Planner used in the FightingICE domain[1]. A modified HTN representation of the world state was constructed to account for some of the limitations of HTNFighter. Different augmentations to HTNs inspired by a plan Reuse algorithm[2], and Adversarial HTN structures[3], were implemented to test their effectiveness in this domain. These different implementations, including a proposed modified HTN Planner named Rapid Replanning were tested against prior Fighting Game AI Competition winners. While some of the implementations underperformed, the developed Rapid Replanning performed better than most the opposing AIs, and was able to take advantage of the fighting game environment combo system to maximize damage. Limitations of the other planners can be primarily attributed to the world state lacking the degree of complexity required to compete with several top level AIs.

1 Introduction

1.1 Hierarchical Task Network Planners

Hierarchical Task Network (HTN) planners are common to many fields, such as robotics and video games. A HTN planner attempts to solve a planning or artificial intelligence problem by structuring an environment as a decomposition of tasks, eventually leading to primitives which an agent can act upon[1]. HTN planners typically require a clear and accurate representation of the planning environment, referred to a world state. However, when using a HTN in an automated planning problem, there are several instances where plans can fail in execution. This occurs when some state or information changes, causing the plan to reach an unreachable situation. This is especially prevalent in dynamic environments executed in 'real-time', such as those in video games[2]. Different systems of either replanning (discard old plan and start a new one) or repairing (take the existing plan, and adjust it at the critical point) are used to deal with the failure[4]. Particularly in video games, replanning is especially prevalent.

1.2 Fighting Games

Fighting games are a genre of games where typically two players compete head-to-head, trading blows within a time limit to become victorious[5]. Players dynamically move and attack in order to whittle away at each other's life meters until one player wins the round. These type of games have a strong following in esports and other competitive settings, and often break down into deep and complex strategizing[6].

1.3 FightingICE

FightingICE is a platform that allows for the implementation of artificial intelligence (AI) in a fighting game environment. FightingICE simulates a completive fighting game, with two characters on a spatially limited arena performing different defensive and attacking maneuvers[7]. Agents controlled by developed AIs must choose between different actions in order knock out their opponent by depleting all their health. The simulation displays a 'real' fighting game, and encapsulates the same nuance motions, timings, and dynamics of a real game. The environment is described in more detail in the Methods section of this report.

1.4 Project Objective

The primary goal of this project is to create an AI that can compete in the FightingICE environment. Specifically, an agent controlled by a HTN planner will be used, in order to evaluate the effectiveness of a HTN planner in a dynamic 'real-time' environment. This serves as an extension of the HTNFighter, a prior implementation of a HTN planner in the FightingICE environment[1].

Furthermore, different HTN enhancements in literature will be implemented and tested alongside the HTN planner. This includes a plan Reuse algorithm, which takes a failed plan as a starting point when replanning, and Adversarial HTN structures, which use information from an opponent agent in order to guide the search of the HTN.

This project attempts to explore different modifications to HTN planning to account for more complex, dynamic problem spaces. The planners are evaluated in terms of efficiency and optimality, measured primarily by how well each agent performs against opponent AIs. Furthermore, this paper aims to form a proof a concept design for an AI that could potentially enter an upcoming Fighting Game AI Competition.

2 Related works

2.1 HTNFighter[1]

In a work by Neufeld et al., a HTN Planner was constructed for the FightingICE environment. The approach focuses on performing close range attacks in order to maximise combos in the game. Experiments indicated that the agent was able to outperform the top controllers in the 2016 Fighting Game AI Competition.

This paper will serve as the main inspiration and baseline implementation for this project. While the HTN was successful, in order to be relevant, the architecture must be tested and adapted to work with modern submissions to the Fighting Game AI Competition. There is room for improvement, as the implemented HTN neglects many fighting game elements and potential actions performed by agents.

2.2 Plan Reuse[2]

The basis of the Plan Reuse is to keep track of failed plans and give priority to plans that were optimal. This can aid in finding higher quality solutions quicker by prioritizing branches that led to the ‘old plan’. This implementation was tested on the SimpleFPS domain, and was found to improve the planning capabilities of their agent.

This method serves as a potential improvement to the HTN planner and gives a possible decomposition algorithm that can yield increased performance in video game environments.

2.3 Adversarial HTN[3]

In order to increase the efficiency of HTN Planning an Adversarial tree structure that makes use of minimax searching with Alpha-beta pruning was implemented. By optimizing state searching based on agent and opponent actions, the search space for the problem can be drastically reduced. This was tested in the μRTS domain, a simulation for Real-Time Strategy games which are known to have extremely large and complex search spaces.

Since the FightingICE domain similarly involves two agents with several possible sequences of actions, it may be beneficial to consider such a design to ensure plans are efficiently created

3 Methods

3.1 FightingICE

This section will introduce several of the game mechanics for the FightingICE domain. Additional game mechanics and character specific traits are available on the FightingICE website[8].

3.1.1 Game Flow

In the game, there is a selection of 3 characters. Each character has unique traits and actions they can perform. For this project only one will be used named "ZEN". Two characters begin a round on opposite sides of the screen, with the left player being "Player 1" and the right being "Player 2". A timer counts down from 1 minute, signifying to total time per round. Three rounds are played per each game. Player 1 and 2 engage in combat, exchanging blows until one player runs out of health, or the timer runs out. The last player standing, or the player with the most amount of health when the timer ends wins the round.

The game itself runs at 60 frames per second (fps) and has delay of 15 frames[8]. This means that an artificial intelligence planning the characters actions must take into account this delay, as one would not have full knowledge about the agent's and world state[1]. There is a deal of uncertainty about whether actions were executed and the outcome due to this delay.

3.1.2 Game Features

Figures 1 and 2 display the game screen and a close up of "ZEN" respectively. Referring to these figures we can identify the following features in the game:

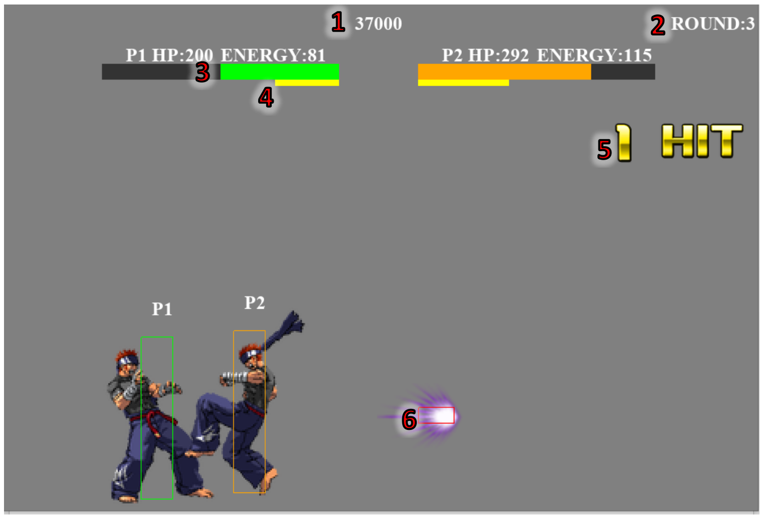


Figure 1: Screenshot of FightingICE with Numbered features

1. Timer: Counts down from 60s (60000 milliseconds displayed on the screen). When the timer reaches 0 the round will end
2. Round: A number that indicates what round is being played. After 3 rounds a game is over
3. Hp bar and value: The green bar in Figure 1. Hit points (Hp) represents the total amount of health each player has. Above the bar is a numerical value showing this more accurately. When a player successfully attacks an opponent, their HP lowers. HP starts at 400 for each round.
4. Energy Meter and value: The yellow bar in Figure 1. Energy is a resource required to perform certain powerful actions. Energy is gained when landing successful attacks, and is depleted to perform certain actions.

5. Combo Counter: The yellow text in Figure 1. Consecutive successful attacks on an opponent create a combo. In FightingICE combos of length 4 or higher grants bonus damage per attack. A combo of length at least 4 is considered a “Full Combo”
6. Projectile: Certain attacks can cause small projectiles which deal damage upon contact.
7. Hitbox: The red box shown in Figure 2 represents the area of an attack that causes damage. Intersecting with a hurtbox will cause the opponent to take damage
8. Hurtbox: The green box shown in Figure 2. the hurtbox represents the area where if an attack hit, will cause the player to take damage.

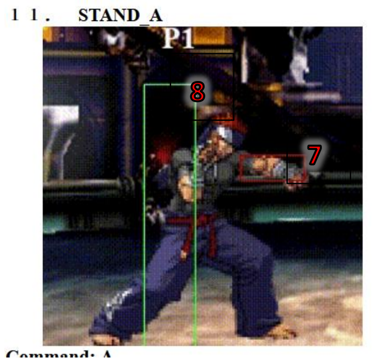


Figure 2: Image of “ZEN” performing a standing punch, with numbered features

3.1.3 Basics of skills

In this environment, all actions performed by a player are referred to as skills. These can include basic motions, such as jumping and walking, as well as attacks.
Each skill contains three parts, (shown in Figure 3) :

1. Startup: For a certain amount of frames after starting to execute a skill, the player will be stuck in an animation
2. Active: the active frame (particularly for attacks) represent the frames which a hitbox will be active for a move
3. Recovery: for a certain amount of frames after executing a skill, the player will be stuck in an animation

In the combo system, there are also a certain amount of cancellable frames, as shown in Figure 3. A player can act out of the recovery animation frames sooner by timing another attack during the cancellable frames. Only certain attacks can cancel other, making it easier to perform combos if specific sequences are used[8]. This is discussed further in the construction of the HTN planner in the subsequent sections

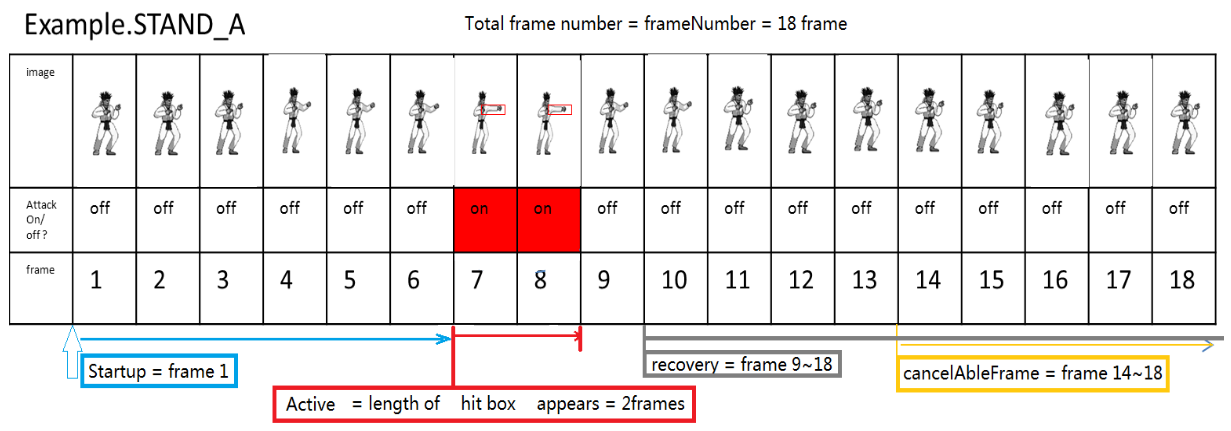

Example.STAND_A
Total frame number = frameNumber = 18 frame
Startup = frame 1
Active = length of hit box appears = 2frames
recovery = frame 9~18
cancelAbleFrame = frame 14~18

image																		
Attack On/ off ?	off	off	off	off	off	off	on	on	off	off	off	off	off	off	off	off	off	off
frame	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Figure 3: Skill execution with frame counting

3.2 HTN planner

Figure 4 shows an overview of the designed HTN planner as a diagram. A higher resolution complete version of the planner appears in Appendix B, for ease of reading. Images for each compound task will appear in Table 1.

3.2.1 HTN structure

The overall structure is based heavily on the implementation of HTNFighter[1]. Many of the states and primitives had to be interpreted and adjusted. The planner always starts with the first task, "Act". This task decomposes into 7 compounds. These compounds are ordered in terms of their priority when decomposing the action. Figure 3 refers to these as "ordered compounds". For instance, when creating a new plan, after Act is decomposed, the planner always starts with "avoid_projectile" to see if the agent and game state match its preconditions. If not, then it proceeds to "escape_corner", followed by the rest (top to down order in Figure 3). This was to ensure that important game conditions were prioritised, as per the original HTNFighter[1]. The subsequent compounds after decomposition do not need to be ordered.

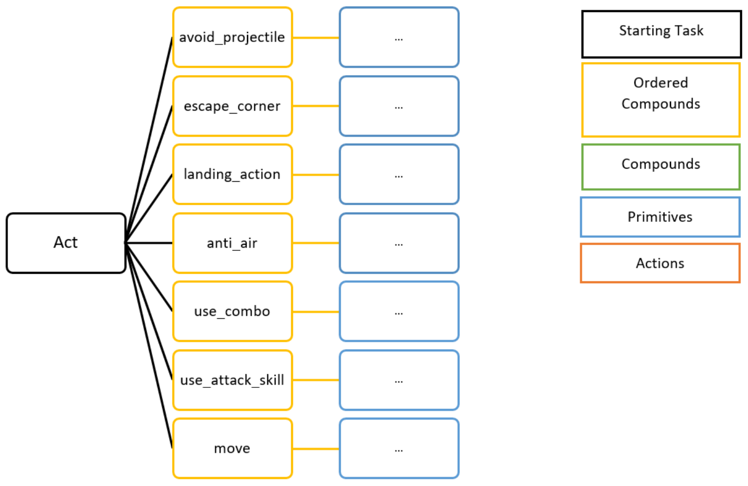
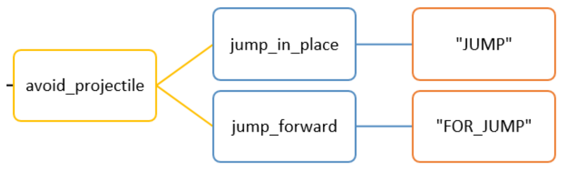


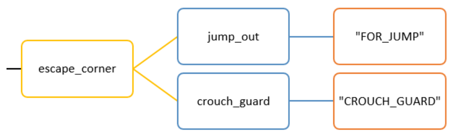
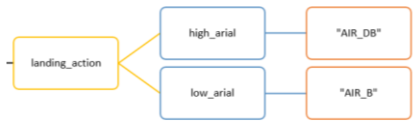
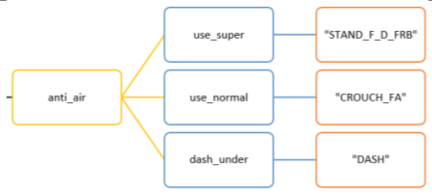
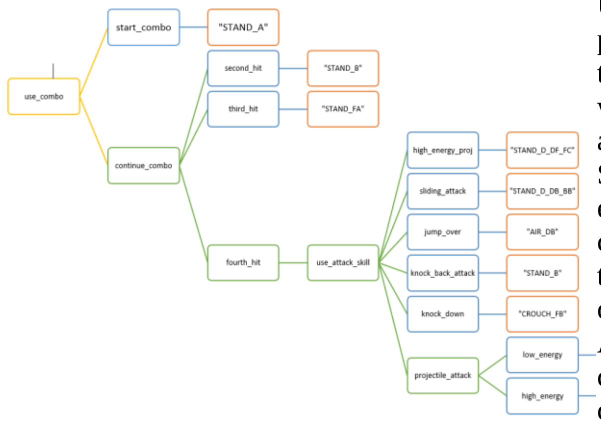
Figure 3: Compound States for proposed HTN Planner

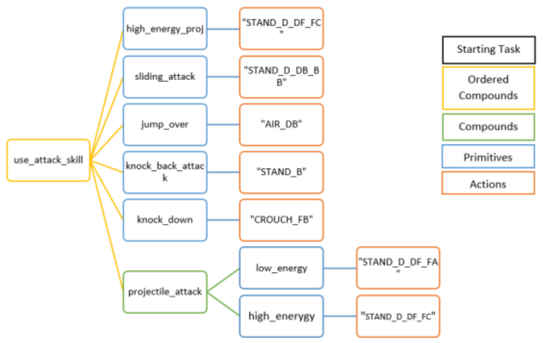
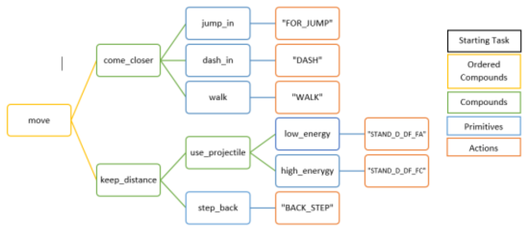
3.2.2 HTN Decomposition

Table 1 summarizes the decomposition of each task. Each task has associated preconditions, that the planner checks before decomposing the task. Appendix C also contains a table of preconditions for each task in the network. The planner can move or decompose to a new task by decomposing a higher task, given all the preconditions for that task are met.

Table 1: Decomposition of Compounds in HTN

#	**Compound Decomposition Image**	**Description**
1	 avoid_projectile jump_in_place jump_forward "JUMP" "FOR_JUMP"	Projectiles can be the highest damage move in the game[8]. As a result, avoiding an opponent's projectile is a necessity, as getting hit by one can often end a round. Avoid projectile decomposes into two jumping tasks, either jumping forward if the opponent is far enough away, or jumping in place if the opponent is close enough to punish the jump. Some projectiles cannot be blocked, so for simplicity, all projectiles are jumped over. However, caution is needed. Being in the air is often a vulnerability as you have less available commands, and jumping into an opponent can often lead to them knocking you back into the projectile. Hence the need for different jumps, to avoid danger from the opponent.

2	 escape_corner jump_out "FOR_JUMP" crouch_guard "CROUCH_GUARD"	The corner is often a limiting place in fighting games. Being stuck in the corner can limit retreat options, and makes it easier for opponents to maintain an optimal distance to extend combos. There is a high importance, even if it bears a risk, to escape the corner as soon as possible by jumping out. However, if the opponent is too close, guarding will help to avoid them starting a long combo.
3	 landing_action high_arial "AIR_DB" low_arial "AIR_B"	Landing action refers to imputing an action while in the air (after a jump command). It decomposes into two states, chosen based on the proximity of the opponent.
4	 anti_air use_super "STAND_F_D_FRB" use_normal "CROUCH_FA" dash_under "DASH"	An anti air refers to an attack meant to hit an airborne opponent. These moves often have hitboxes placed further above the player, in order to hit the opponent before their air attack can hit you. This state was added to more rigorously deal with mobile opponents. Using a super will create a stronger and larger attack if there is sufficient energy, otherwise a normal anti air is used. If the opponent is close, then the player can dash away, dodging the opponents air attack and preparing to punish.
5	 use_combo start_combo "STAND_A" continue_combo second_hit "STAND_B" third_hit "STAND_FA" fourth_hit use_attack_skill high_energy_proj "STAND_D_DF_FC" sliding_attack "STAND_D_DB_BB" jump_over "AIR_DB" knock_back_attack "STAND_B" knock_down "CROUCH_FB" projectile_attack low_energy high_energy	Using a combo allows a plan to form where consecutive attacks are planned against the opponent. The intention is to hit 4 consecutive hits, to take advantage of the bonus damage on the last hit. The combo tree was updated from the HTNFighter. In HTN fighter, states existed to start a combo, or continue it. The combo consisted of four actions, STAND_A, STAND_B, STAND_FA, STAND_FA. However, when experimenting with the game engine, it is possible to interrupt this combo between STAND_B and STAND_FA if the opponent has a well-timed move with a short startup animation. The most devastating is if the opponent has a large amount of energy and can use a powerful option. As a result, the third combo state has additional preconditions to stop the combo early if the opponent has sufficient energy. Furthermore, the combo in HTNFighter is very difficult to execute, as the recovery between the 3rd and 4th skills only works in very situational spacing. An easier combo to execute was replacing the last STAND_FA with a sliding_attack. The sliding attack requires a certain amount of energy, thus the last hit in the combo ends up branching into several possible moves. However, the sliding attack has increased priority, meaning it can easily cancel the animation of any move used before it. This makes it easier to combo, as it has a fast startup and can reduce the recovery of a prior action. If there is sufficient energy, the main 4 hit combo will be STAND_A, STAND_B, STAND_FA, STAND_D_DB_BB (the last move being the sliding attack). If possible, a high energy projectile can be used instead, which in most cases can win the round even without being a part of the combo. In most cases 2 or 3 hit combos (consisting of STAND_A, STAND_B and STAND_D_DB_BB) are performed instead to be safe, as many of the more recently developed AIs were designed to escape from the combos.

6	 use_attack_skill high_energy_proj "STAND_D_DF_FC" sliding_attack "STAND_D_DB_BB" jump_over "AIR_DB" knock_back_attack "STAND_B" knock_down "CROUCH_FB" projectile_attack low_energy "STAND_D_DF_FA" high_enerygy "STAND_D_DF_FC" Starting Task Ordered Compounds Compounds Primitives Actions	If a combo cannot be performed, or if it was interrupted, then a series of alternative attacks can be used. This is the same logic as when choosing a 4th hit in the prior combo chain. Combos usually require the player to be close to the opponent. These actions help to narrow the distance in most cases. If there is sufficient energy, projectiles can be used to maintain a safe distance, as well as provide and active hitbox to help approach (ie, walking behind your own projectile to shield you from your opponent)
7	 move come_closer jump_in "FOR_JUMP" dash_in "DASH" walk "WALK" keep_distance use_projectile low_energy "STAND_D_DF_FA" high_enerygy "STAND_D_DF_FC" step_back "BACK_STEP" Starting Task Ordered Compounds Compounds Primitives Actions	If an attack cannot be performed and there are no immediate threats (such as being in the corner or avoiding a projectile) then the character can reposition to aggress. Different approach options are based on how close the opponent is. If there is the potential for a threat, then the player can plan to keep or make distance by using a projectile or stepping back.

3.2.3 Planning Implementation

To follow with the original HTNFighter, the network was implemented in order to map the agent's actions. In the original paper, plans consisted primarily of one action, except during combos where the entire sequence was planned. To make this behave more like an automated planner, plans were decided to always consist of 4 actions. A set of predicted resulting conditions was maintained and updated in order to plan the sequence of actions. This allows for several tasks to be planned before receiving any updates on plans. If replanning needed to occur, then sequence is flushed.

3.2.4 Replanning

Before the next action of the plan is executed two conditions are checked. First, if the previous action was executed. Second, if the preconditions for the upcoming action still hold. If either of these checks do not return True, then the plan is discarded and replanning occurs.

3.3 HTN Variations

3.3.1 Implementation with Plan Reuse

Originally, a different domain was going to be used for this project, which made the Plan Reuse augmentation to HTN an interesting endeavor. For this particular HTN, the highest compounds have a priority order to ensure important actions are considered first. Reusing a failed plan may jeopardise the agent if the plan failed because one of these dangers became imminent. However, there is an interesting application for this project during combo routes. When replanning, if a combo route fails due to non-threatening conditions (for instance, a punch missed because the character is too far away, but not in danger), then it would be very advantageous to search the HTN to find the next best skill to continue the combo (for instance, substituting a kick for the punch). As a result, this method of augmentation was still tested, but implemented in a much simpler way. Instead of a similarity functions from the original paper, the compound decomposition was stored, then upon replan, that particular compound was prioritised, followed by adjacent compounds.

3.3.2 Implementation with AHTN

The AHTN approach is a small extension in the scope of the project. There was not a lot of time and the efforts in this section were minimal. For this project, the world state representation did not become very large. Creating the architecture for the means of improving efficiency likely will not make an impact in performance. Instead, the idea of considering the opponent's actions in addition to the player's was an interesting topic. Using the built in simulator, first a plan is created using the HTN for the opponent rather than the player. The outputted action is then passed into the simulator to create a simulated world state where the opponent chose the best possible action according to the HTN planner. Then using this updated world state, a plan is found for the agent. This creates a crude implementation meant to mimic the idea of agent interaction in minimax searching.

This implementation can also test the robustness of our HTN planner. Predicting the opponent's actions helps show what our implementation considers the "worst that the opponent can do". If our planner is a strong representation of world state and finds the optimal choice, then the actions of the opponent could be considered what they would do best improve their position. Further, if we choose options best deal with the opponent choosing an optimal option, then we are prepared for the worst, and any other situation would be equivalently less of a risk. Likewise if our world state and planner is flawed, then this strategy would exasperate it.

3.3.3 Implementation with Rapid Replanning

When testing the HTN planner against other AIs during development, a common situation of unrecognized plan failures would occur. Often the planner would return long committable actions in sequence, only to get greatly punished by the opponent. This may be due primarily to the 15 frame delay, computational delays of connecting the HTN planner to the FightingICE domain, or the world state not being descriptive enough to cause replanning upon further inspection.

A proposed change was tested. We assume that after each action a new plan should be formed, in essence assuming a plan failure at every stage of planning. This turns the HTN planning problem into more of a search problem, looking for an action each time the planner is called. The executed action is the one that ends up being discovered on the frame the game checks for an input. This allowed a much greater flexibility in choosing actions, as many combo paths can be interrupted to reposition to a safer spot. This implementation was named Rapid Replanning for the sake of comparisons and references in this project. Since the environment is relatively small, this was possible without affecting computational performance (actions were still executed and the game was not stalled).

4 Results

4.1 Initial Testing Procedure

Three AIs were selected to test against this project's implementation. Each AI placed within the top 3 from a different year of the competition. One AI from each year was used to highlight that the HTN Planner, while in its first iteration could realistically place within the top 3 each year.

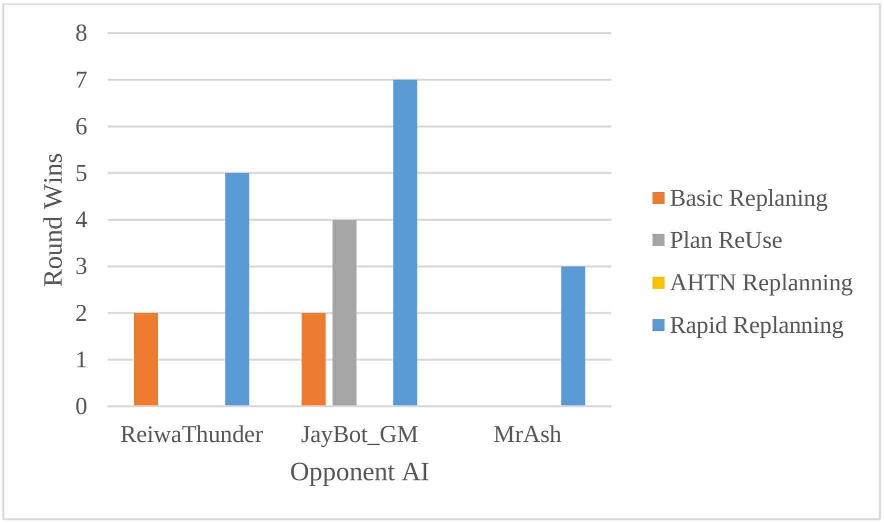


Figure 5: Round wins for each implementation against opponent AIs

Specifically our HTN planner was tested against ReiwaThunder (1st place 2019), JayBot_GM (3rd place 2018), and MrAsh (3rd place 2016). Several other AIs were tested during development, but these 3 were chosen to highlight potential strengths and weaknesses found.

Each AI was played for 3 games (totaling 9 rounds each). Four versions of the HTN planner was tested based on Methods 3.2 and 3.3 ; Basic Replanning, Plan Reuse, AHTN Planning, and Rapid Replanning. The games were configured to run as per the actual Fighting Game AI Competition Requirements described in the Methods. Results of these tests are shown in Figure 5.

4.2 Additional Iterations

Since the simulation take a long time and require a decent computational workload, several iterations during testing is difficult. With the small number of games played, it is hard to statistically justify the performance of our planners. As an extension, the best performing planner from the previous Figure (Rapid Replanning) was run for 20 addition games (60 rounds) against ReiwaThunder and JayBot_GM. This was to ensure the Rapid Replanning Implementation was still performing well, and the results were not skewed due to a small sample size. Figure 6 displays the totals wins and losses from this experiment.

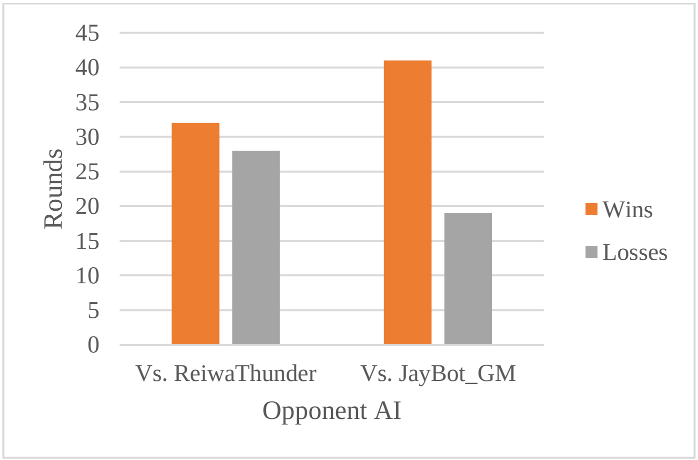


Figure 6: Round wins and losses against AIs after 20 games (60 rounds). The HTN achieved a win rate of 53.3% against ReiwaThunder, and 70% against JayBot_GM

4.3 Combo Potential

There was not an easy way to extract the combo information from game engine without some additional development. In order to study combos as the original HTNFighter paper, an approximation experiment was conducted. From the additional iterations, one game where the HTN planner performed successfully (won all rounds) and one where the planner performed unsuccessfully (lost all rounds) were selected. For each of the rounds, a replay was watched and the amount of combos were counted for both AIs. This was done to estimate the combo potential of each AI, as well as to see if there was a correlation between the HTN planner performing well, and if long combos being executed. Figures 7 and 8 display the results of these tests.

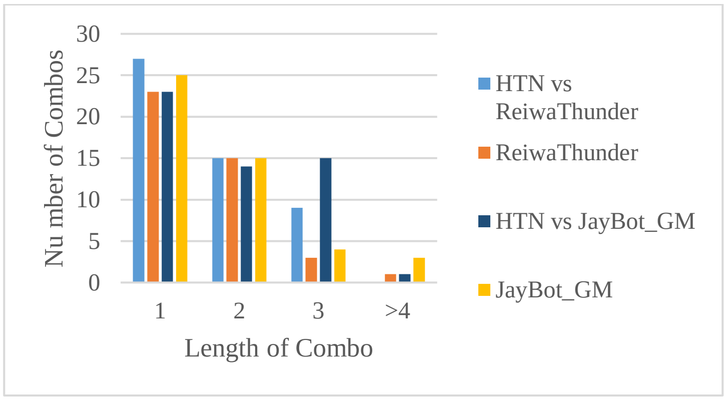


Figure 7: Number of Combos performed in Games where HTN Rapid Replanning Won

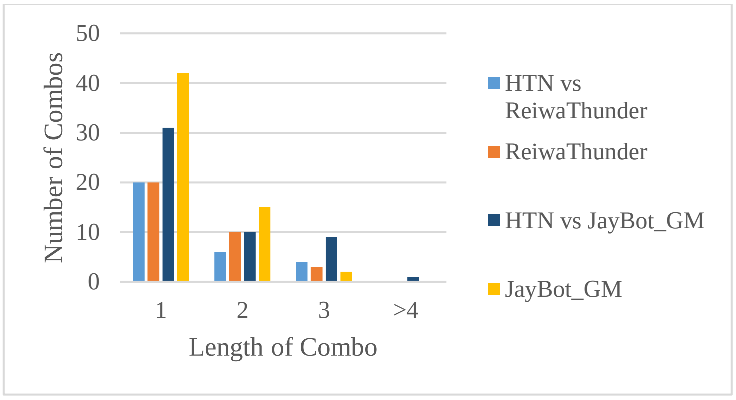


Figure 8: Number of Combos performed in Games where HTN Rapid Replanning Lost

5 Discussion

The implementation of the Rapid Replanning HTN planner was able to stay competitive with the other tested agents. The main goal of this paper was to test if a HTN architecture could hold up against other AI algorithms in this particular game setting. This project works as a proof of concept extending upon the original HTNFighter to test its effectiveness against more recent competition submissions. A specific point of interest was its performance relative to the 2019 1st place AI, ReiwaThunder. It also highlights that an HTN structure can be effective in a ‘real-time’ dynamic environment.

Furthermore the additive states and modified combo structure worked well against the different opponents. The Rapid Replanner implantation was able to perform more three hit combos than both ReiwaThunder and JayBot_GM in both winning and losing games. Particularly JayBot_GM did not have as aggressive a combo breaking algorithm, so often several 3 hit combos were performed in succession, easily securing round wins. There are also some full combos (4 hits) that were performed when the situations permitted which was interesting. Opponents also performed 4 hit combos, but these were mostly repeated actions in corners, which was an area of weakness in our implementation.

Appendix D contains a brief video commentary of some sample games, highlighting the performance of the HTN planner.

5.1 Failure points

The other implementations besides Rapid Replanning all performed quite poorly against the opponent AIs. Upon watching replays of the trials (as well as from observations during the development process) a series of failure points can be identified. To help visualise these issues, Videos can be found in Appendix D, with commentary discussing them.

5.1.1 HTN world model not descriptive enough

There are several overarching cases that occur when the HTN planner loses to opponent AIs. There were several exploits that opponents were able to ‘find’ against the player. In some cases plans would be stuck repeating actions without making forward progress in the game. In other cases, planned actions were severely punished, causing close rounds to be decided by one or two misplays. The following cases summarize these scenarios:

Case 1; opponent in corner, agent unable to approach; This commonly occurred against MrAsh, which actively would disengage from the player. The current preconditions for approach are based on three effective ranges (whether the opponent is a close, medium or long distance from the player). Based on the HTN, usually when an opponent is moving away from the opponent, or is far away, projectiles are used to bridge the gap with a hurtbox, as well as help approach. In order to time approaches through the 15 frame delay, the player approaches slowly and expect the player to either approach or jump over the projectiles. However, some agents when achieving a lead will continually back step to the opposing corner and stop engaging. This was not heavily considered in the design of the world state interpretation, as the corner is often intuitive an area a player would not want to be, as it is a limiting and potentially compromising position. After energy is spent, the HTN gets stuck repeating a ranged kick and no longer approaches the opponent. Time eventually runs out and the round is lost.

Case 2; close range projectile; when trying to execute a skill, sometimes the opponent will use their most powerful projectile as a mean to punish the player during their animation frames. This occurs several times when dashing into the opponent, or after using a move that has a very long recovery animation. Some more sophisticated AIs use this time to spend all their energy on a powerful projectile. The projectile usually hits, as the planner cannot do the avoid projectile tasks since the player is unable to perform an action. These projectiles can take away approximately a third of the Hp of the player, and often end up costing the round.

Case 3; Well-timed moves in the corner; The tested AIs all had corner pressure options that cause the player to be unable to escape. Using moves that cause the player to get knocked down force a certain animation from the player. If timed well, these actions can be repeated, causing the player to be stuck in animation frames, unable to escape from damage.

Case 4; Opponent use of anti airs; The HTN planner has a simple approach technique to jump toward the opponent if the distance is long. However, certain AIs are designed to time an anti air with a long reaching move. As a result, the player tries to approach, gets hit out of the air, and returns to their start point. The player then tries to approach the same way upon recovery since it meets its preconditions once again. For agents such as ReiwaThunder, this was apparent.

All of these cases have the potential to be solved with a more defined world state representations. Currently the world state has limited preconditions. Having a larger state space that includes more precise spacing and approach options could greatly help. Furthermore, including more detailed information about frame data, and how committable actions are could greatly improve which actions are chosen at which times. Additionally, specific corner guarding techniques could be implemented. The character can try to avoid getting into the corner at an earlier point, or also have specific guard plans to avoid getting into a cycle of damage.

Additionally, through the delay, often combo routes that would seem safe to use end up being dangerous, as during execution the opponent would gain adequate energy to punish the player. Having a stronger world state representation to accurately predict opponent and player energies after interaction can greatly aid in this issue.

It is very motivating that several of the round losses occur from similar scenarios and cases. Given the Rapid Replanning still was able to perform well despite losing rounds to what appear to be clearly identified problems, there is great validity in the performance of the HTN planner in this particular domain.

5.1.2 Specific failures with planners

The base HTN planner, HTN planner augmented with Plan Reuse and HTN planner augmented with AHTN all perfumed very poorly. Most the round loses occurred from either the aforementioned Case 1 or Case 2 occurring. All these versions of the planner would continue performing a committable action, which usually led to the opponent punishing, often costing the round. Creating a more accurate world state representation would likely remedy this, as more information in choosing the action could allow for less committable combo routes to be chosen.

In terms of the Augmentations, Plan Reuse had a slight improvement when fighting against JayBot_GM. While the trials are a small sample size and could easily skew either way, in the replays the plan Reuse was able to forcefully execute many combos on JayBot_GM. Sometimes these would get punished causing the round to be lost, but other times, the specific Reuse would quickly solve a plan failure with an alternative action. JayBot_GM was noticed to not have as strong combo breaking tools as the other AIs. If the opponent is not taking advantage of committable actions, then damage can snowball in the player’s favor. This was interesting, and shows signs that against some opponents, prioritising alternative combo routes through a plan Reuse may be beneficial. However, with ReiwaThunder, the base planner performed better, likely due to the Plan Reuse overlooking evasive maneuvers against the agent.

The AHTN augmented planner performed the worst. This is likely due once again to the world state having issues. By using the flawed world state, often more committable actions and jumps were used, causing the agent to be punished aggressively by opponents.

5.1.3 Python and computer computational resources

The different planners were implemented using Python while the FightingICE is a Java based project. Often the simulations would run slow, and occasionally Python would timeout in its connection to the Java project. Furthermore, the FightingICE often caused high disk usage, and computer slowdown in general. While this is not considered to be a major issue, sometimes it cause some performance issues with the planners while developing, and some losses may end up attributed to these. If future development occurs, it may be worthwhile to convert the existing program to Java to limit the additional connection.

6 Conclusion

This project found that a HTN planner in a ‘real-time’ video game environment was able to compete, as well as perform well against other competition winning AIs. While several of the tested planners suffered from world state limitations, many of these issues are expected to be resolved with an updated world state representation.

6.1 Future work:

Since the planner showed promise of competing against the other AIs, some future steps are planned. Future development can also lead to a submission in next year’s Fighting Game AI Championship:

- Fixing the world state to be more descriptive is a starting point. For this project, the simple world state was found to be competitive with the Rapid Replanning. The idea is to grow the world state to be more descriptive. This should hopefully allow the other planners to perform well.
- The 2020 agents were not able to be tested due to some unresolved issues with the Java environment. Hopefully these AIs, as well as the eventually released 2021 AIs could be tested against the HTN planner in order to see what additional improvements are needed.
- With a larger world state, the Plan Reuse and AHTN implementations may be necessary to allow efficient computation. In this project, these two augmentations were not implemented to the fullest, and future work will try to implement them much more faithfully to their respective papers.
- Looking into combo routes at different areas of the stage would be interesting. Finding for each position in the stage and distance to the opponent the ‘optimal’ highest damage combo that does not allow them to break out would be greatly beneficial to maximise damage output

7 References

[1] X. Neufeld, S. Mostaghim and D. Perez-Liebana, "HTN Fighter: Planning in a Highly-Dynamic Game", *IEEE*, 2017. Available: https://ieeexplore.ieee.org/document/8101623/. [Accessed 8 May 2021].

[2] D. Soemers and M. Winands, "Hierarchical Task Network Plan Reuse for video games", *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, 2016. Available: 10.1109/cig.2016.7860395 [Accessed 2 March 2021].

[3]M. Buro and S. Ontañón, "Adversarial Hierarchical-Task Network Planning for Complex Real-Time Games.", *IJCAI*, 2015. Available: http://ijcai.org/Abstract/15/236. [Accessed 1 March 2021].

[4] I D. Holler et al., "HTN Plan Repair via Model Transformation", *KI 2020: Advances in Artificial Intelligence*, 2020. [Accessed 6 April 2021].

[5] "What is Fighting Game? - Definition from Techopedia", *Techopedia.com*, 2021. [Online]. Available: https://www.techopedia.com/definition/27154/fighting-game. [Accessed: 08 May 2021].

[6] T. Murray, "An Introduction to Watching Fighting Games as Esports – The Esports Observer", *The Esports Observer – The Home of Esports Business*, 2018. [Online]. Available: https://esportsobserver.com/introduction-fighting-games-esports/. [Accessed: 08 May 2021].

[7] "Welcome to Fighting Game AI Competition", *Ice.ci.ritsumei.ac.jp*, 2021. [Online]. Available: http://www.ice.ci.ritsumei.ac.jp/~ftgaic/index.htm. [Accessed: 08- May- 2021].

[8] "Get Started ----- Advanced". *Ice.ci.ritsumei.ac.jp*, 2021. [Online]. Available: http://www.ice.ci.ritsumei.ac.jp/~ftgaic/index-2a.html. . [Accessed: 08- May- 2021].

8 Appendix

Appendix available at:

https://drive.google.com/drive/folders/1ddXSDKNFV1HDeWhdQsO_GrtfZaN981Ek?usp=sharing

Appendix A: Coding project and implementations
Appendix B: Hierarchical Task Network Structure
Appendix C: HTN Preconditions
Appendix D: Discussion Videos