Creating Human-like Fighting Game AI through Planning

Roger Liu

December 2017

School of Computer Science
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee
........, Chair
Maxim Likhachev

Submitted in partial fulfillment of the requirements
for the Degree of Master of Science

Copyright © 2017 Roger Liu

CARNEGIE MELLON UNIVERSITY

Abstract

Faculty Name

Department or School Name

Doctor of Philosophy

Creating Human-like Fighting Game AI through Planning

by John SMITH

Games are a major testing ground for Artificial Intelligence. Though AI has become proficient at playing games such as Space Invaders, it behaves in a way that is distinctly artificial, lacking the human-like qualities of a real player. This human element is important in competitive multiplayer games, as a large part of the enjoyment comes from outwitting other human strategies. To address this issue, we investigate a novel AI technique that leverages planning and human demonstrations to create an opponent that exhibits desirable qualities of human play in the context of a fighting game. We introduce the idea of action-δs, which relate the action performed with the change in the game state. These action-δs are learned from human demonstrations and are used to help the AI plan out strategies to hit the opponent. We implement a simple fighting game called *FG* for the AI to compete in and provide it a human demonstration to learn from. The AI utilizes action-δs with other search techniques to emulate human behavior. Lastly, we evaluate the effectiveness of our AI by comparing its similarity score against other algorithms and other demonstrations by the same human player.

Acknowledgements

The acknowledgments and the people to thank go here, don't forget to include your project advisor…

Contents

Abstract **iii**

Acknowledgements **v**

1 Introduction **1**

2 Related Work **3**

2.1 Racing and Imitation 3

2.2 Super Mario Bros. 4

2.3 Fighting Games and Neural Nets 5

2.4 Ghost AI 5

2.5 Data-driven Finite State Machines 6

2.6 High Level Overview 6

3 Motivation for Planning-based Approach **7**

3.1 Why Use Search? 8

4 Algorithm Specifics: Representation and Implementation **11**

4.1 action-δs 11

4.2 Demonstration δ-Search 11

4.3 Reasoning 13

4.4 Full Implementation Details 14

4.4.1 Environment Description 14

4.4.2 Data Extracted from Demonstration 14

4.4.3 Generating Successors 16

4.4.4 Costs 18

4.4.5 The Goal and Heuristics 18

4.5 Dealing with long search times: Anytime Search 19

4.5.1 Dealing with a changing game state: replanning 19

4.6 PENDING SECTIONS (MAY NOT GET TO THEM IN THE NEXT 2 WEEKS) 20

4.7 Dealing with bad predictions, updating the predictor 20

4.8 Interacting with the opponent, how to respond on defense 20

4.9 Results 20

4.10 Discussion 20

4.11 Additional Details 20

A Frequently Asked Questions **23**

A.1 How do I change the colors of links? 23

List of Figures

3.2 The target player's strategy 8
3.3 An example of a planning AI's approach 9

4.1 Example of an action-δ 12
4.2 Overview of δ-search 13
4.3 A Time lapse of the demonstration 14
4.5 Transitions generated by the demonstration 16
4.6 Goal states generated by the demonstration 16
4.7 The AI follows the player's demonstration 17

List of Tables

4.1 How action-δ is calculated 15
4.2 The qualities extracted from state s 19
4.3 AI Situation description 20
4.4 Player Status descriptions 21
4.5 Player Action descriptions 22

For/Dedicated to/To my…

Chapter 1

Introduction

Fighting games are unique among competitive games in that they are real-time, 1-on-1 games where small mistakes lead to huge consequences. The best way to get better at these kinds of games is to practice against other humans, but that is not always an option. While online play exists, is not ideal because of network latency. In addition, the AI in these games are generally considered a poor substitute. They often exploit natural advantages such as perfect reaction time and perfect game state information, but even disregarding that they still only have fixed spectrum of behavior patterns which players can learn to exploit and consistently defeat. Worse still is that these behavior patterns might not even be representative of the human opponents that players encounter in competition.

That said, there are avenues to improve the AI in fighting games to make them useful for players. One approach is to make an optimal AI which is able to adapt its strategy based on its performance. This would provide players a challenge by removing the ability to exploit the AI, but it still doesn't necessarily capture the strategies and techniques used by other human players. Another approach is to make a *human-like* AI, one that plays like another specific human opponent. This task seems feasible, as long-time fighting game players can identify the differences between the playstyles of different players, meaning that there is some quality that differentiates one behavior from another.

In this research, we investigate planning-based approaches to creating human-like AI. To do this, we first explore previous approaches taken to create human-like AI and discuss their merits and limitations. We also describe other attempts at creating AI for fighting games to contextualize our efforts compared to theirs. We then introduce the environment we created to test our approach and define concepts and terminology used by our algorithm. Lastly, we describe our algorithm, where we plan on the actions provided by human demonstrations to reach a desired outcome.

Chapter 2

Related Work

2.1 Racing and Imitation

One of the few documented instances of Imitation AI came from the racing game *Forza Motorsport*. In this game, players could train "drivatars" to race just like themselves. This was implemented by having the AI learn how the player behaves on different segments of track and try to replicate that behavior when it encounters those sections. However, this imposed a restriction on the types of tracks that could be present in the game, as they had to be formed from the basic segment building blocks.

Researchers then expanded on this approach through the use of genetic algorithms [DrivingPlayerModeler]. In these kinds of algorithms, candidate solutions are continually evolved toward better solutions. These candidates have a set of parameters which are altered between each iteration of the process, and are then evaluated according to a fitness function. For the particular case of creating robust human-imitating racers, a fitness function made up of 3 different components was used. 1 focused on matching the players progress on the track. Another focused on it matching the player′s steering and a final one had it match the player′s speed. The resulting AI did an alright job of mimicking some aspects of the repsective players, as the AI imitating a slow, careful driver behaved in a significantly different way from the faster, reckless driver. However, closer inspection of the driving AI showed that the resulting behavior was not conceivably human. Later attempts which had the fitness function that incoporated a focus on having the AI drive optimally also did not obtain convincing results, but it showed a clear trade-off between driving optimally and improving driver similarity. [MultiObjectiveMimic]

2.2 Super Mario Bros.

Researchers also developed several methods to mimic human behavior is in the space of 2D platformers, specifically a modified version of Super Mario Bros. [MarioImitation]. Novel methods tested in this space were Inverse Reinforcement Learning and Neuroevolution.

The results of the Inverse Reinforcement Learning approach were discouraging, as the agent wasn't able to consistently handle situations that weren't often seen in the demonstration and was unable to match a human's ability to predict things not in the immediate detection area. In addition, the optimal policy obtained by IRL is deterministic, further reducing the human-like appearance of the AI.[MarioImitiation2]

Neuroevolution produced much better results. In this method, a neural network was first trained to play Super Mario Bros. The state of the game was encoded into various genre specific variables that denoted the state of Mario and the distance of Mario to various obstacles. This was handed as input to the neural network, which was then expected to output the buttons that should be pressed in that situation. The resulting weights were then evolved and evaluated using a fitness function. The fitness function in this case was the distance between the AI and player's traces through the level. A key improvement made to suit this genre was to reset the AI's position once the distance exceeded some threshold and apply a flat penalty. This is because an AI can easily get stuck in a 2D platformer, leading to a very bad final fitness score. The result was that the AI did the best job of mimicking human playstyles compared to many other algorithms. However, the agent achieved a lower score in the game compared to human players, showing that the agent had not really achieved a truly human-level of performance in the game.

2.3 Fighting Games and Neural Nets

Neuroevolutionary techniques have also been applied to fighting games. On a simple fighting game with 1 axis of movement, researchers found that evolutionary neural networks were able to quickly converge to an optimal playstyle [FightingAIComparison]. Additionally, Deep Reinforcement Learning has been able to create AI agents that can compete with top players in the popular commercial game Super Smash Bros. [SuperSmashBros]. However, optimal AI are not ideal substitutes for playing against human opponents. In the case of the Deep RL AI, it was specifically trained against only one kind of opponent, meaning that it was limited in the range of matchups it could perform well in. In addition, it exhibits obviously artificial traits such as impossible to perform back and forth movements.

2.4 Ghost AI

With regards to creating AI that was specifically human-like, the most notable attempt was something called Ghost AI, which was implemented on a version of the commercial game Street Fighter. This AI essentially initializes a table with the frequencies that the target player did a move and then used those moves at the same frequency[Ghost AI]. In the adaptive version, the AI would update the frequencies based on the reward gained from performing those moves [Ghost AI2].

To evaluate this AI, they recorded player sessions and sessions of the corresponding mimicked AI. The players were then asked to watch these recored sessions and perform inputs as if they were in the same situation. The recordings were then scored by the similarity of the recordings inputs to the "fake" test subject inputs. By this evaluation metric, this method showed promising results, as it was able match around 75% of the real recordings accuracy. Players also expressed high qualitative satisfaction with their recordings, and the adaptive component allowed the AI to adjust itself to the strategies of the opponent. This approach has some notable pitfalls, as it does not account for specific player strategies that include varying timing. It also fails to account for contextual information, such as position on the screen, which factors into the decision making process for a real human player.

2.5 Data-driven Finite State Machines

One final approach that to this problem was to create Data-Driven Finite State Machines. In this method, a multi-layered finite state machine is formed from the a log of a human demonstration. Specifically, the moves performed during the demonstration are annotated and used as designations for the AI states that it transitions between.

This approach has some clear limitations. For one, the annotation of moves is cumbersome and not well suited for a general purpose algorithm. Furthermore, the strategy that a player uses could be determined by an arbitrary number of in-game and out of game variables, which makes reducing player behavior to an FSM an daunting task. Lastly, this method was implemented on a 1D fighting game, which puts a huge limitation on the types of techniques that can be expressed by players.

2.6 High Level Overview

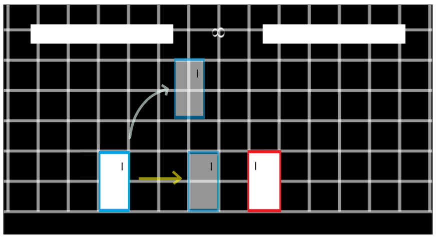
In this section we discussed several different existing methods for creating AI that mimic human-behavior. In domains where players progress on a path to an objective, such as racing games and 2D platformers, neuroevolution proves to be a strong strategy. However, there is a clear tradeoff between improving similarity and improving performance in these games, and even then these AI's have a hard time recovering from getting stuck.

When looking specifically at fighting games, there is currently a lack of new developments. Though neural methods have proven effective at creating optimal agents in certain environments, they exhibit traits that prevent them from being suitable substitutes for human players. Other techniques such as Ghost AI have demonstrated an ability to express traits of human play, but lack the capability to account for the in-game situation in a human-like manner.

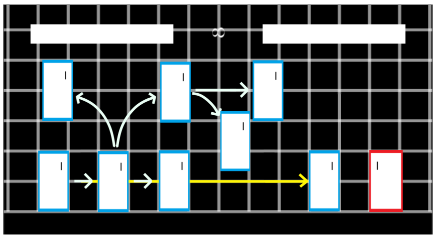
Chapter 3

Motivation for Planning-based Approach

A commonality of all of the previous algorithms is that they all use some degree of learning to try to determine the best actions to take at any given situation. However, this approach has some issues. For one, learning requires a large amount of training in order to generalize across the large state space. This is problematic, as the player data gathered over multiple sessions is relatively small. Additionally, because the agent only learns the best actions to take at the current time step, it lacks the ability to plan for a sequence of actions that form a cohesive strategy. This part is important, as individual playstyles are categorized by the strategies that a player tends to use. Lastly, there is a natural trade off between optimality and employing a degree of randomness when these agents decide the next action to take. If an AI always does the same action in response to a situation, it loses the unpredictability of human play. However, once an AI does a random action that is unreasonable for the current situation, players will instantly recognize it as an AI.



(A) A predictable AI is not very human-like



(B) One unreasonable action can break immersion

3.1 Why Use Search?

Because human-play is heavily predicated on the usage of different kinds of strategies, dynamically creating and executing the same strategies as a target player should mimic their playstyle quite well. Since strategies are essentially a sequence of actions that a player takes in order to arrive at a desired goal, the formulation of strategies can naturally be represented as a search problem over the state space. The target player's demonstrations can be used to inform the AI of the goals to target and the actions to use, and custom heuristics can bias the search towards actions that closely mimic the player. In addition, the demonstrations can be used to create a model of the game state's dynamics, which would allow the AI to form cohesive plans even in unfamiliar starting states.

To understand how an AI might effectively use search, consider the following situation:

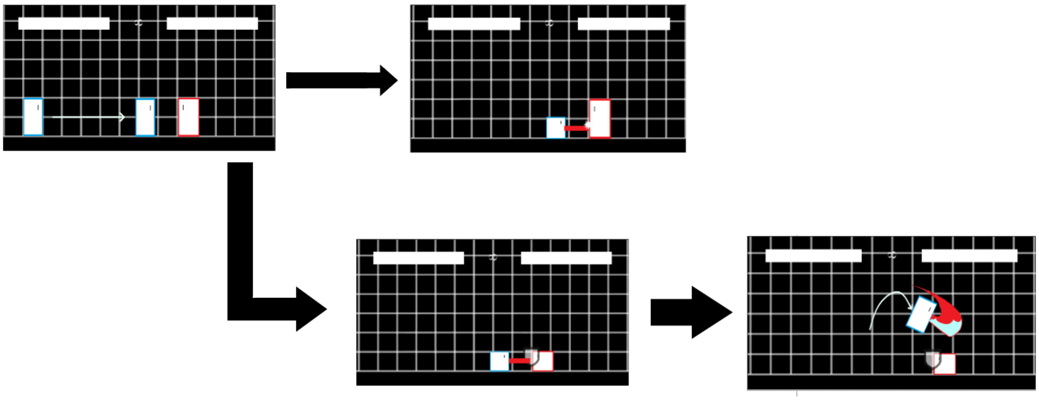


FIGURE 3.2: The target player's strategy

In the demonstration, player 1 tends to try to stay within a certain range of the opponent and try to hit them with a low attack. If the opponent blocks the low attack, it sometimes then tries to jump in while the opponent can't move out of the way and hit them with a air attack. Essentially, player 1's behavior is composed of 2 strategies, one where they try to hit the opponent with a low attack and another where they force the opponent to try to block the air attack. We can easily identify the strategy that the player is currently executing by looking at the ultimate result that they are aiming for.

When the AI plans, it first determines what goal state to target. It then uses the actions pulled from the target player's demonstration and uses them to search the state space to reach the goal. For example, to reach the goal where the opponent is hit with a low attack, the search would return a plan where it walks forward to get close to the opponent, crouches, and then uses the low attack. If the opponent moves during that time, the AI replans picks a walking action that would put it in the correct range for hitting the opponent. With a single demonstration, search is able to formulate a plan that approximately resembles the strategy executed by the target player.

-Figure Needed here-

FIGURE 3.3: An example of a planning AI's approach

Chapter 4

Algorithm Specifics: Representation and Implementation

4.1 action-δs

First, we explain the concept of action-δs, which our work relies on. When a player takes action a from state s and arrives at state s', the game state changes because of a. For example, the action of walking to the right causes the player's x position to increase. We refer to this change in the game state as an action-δ, which represents how the game state changes as a result of taking action a.

These action-δs are used to understand and build a model of the game's dynamics. If we know how an action affects the game state in one situation, we can predict how the action will affect the game state in similar situation.

4.2 Demonstration δ-Search

The task of emulating a player's behavior is represented as a graph search problem. Specifically, the objective is to form a plan to hit the opponent using the actions demonstrated by the target player. The plan should be feasible and resemble a plan executed by the target player as much possible.

-Figure Needed here-

FIGURE 4.1: Example of an action-δ

The search space is represented as graph $G = (V, E)$. The vertices V of the graph are the various states of the game. The edges E represent the possible transitions between game states.

The transitions that we search on are generated from the training data and fall into two classes. The first class, known-transitions, are tuples (s, a, s') which are identical to ones captured from the demonstration. The other class of transitions are referred to as δ-transitions. The result s' of these transitions are generated by a predictor function $\phi(s, a)$, where a is an action taken by the target player. This predictor function generates the predictions by learning from the action-δs of action a obtained from the training data.

Since we want to form a plan that hits the opponent, a valid goal state is one where the opponent has the "FirstHit" status. During a run of the search, the goal is defined to be a state that has the same characteristics as a random goal state selected from the demonstration. This ensures that the planner's ultimate objective matches that of the target player.

When searching for a feasible plan to get to the goal, we use a modified version of heuristic graph search. We maintain two priority queues throughout the search, one called KNOWN and another called UNKNOWN. When deciding to expand a state,

-Figure Needed here-

FIGURE 4.2: Overview of δ-search

we prioritize expanding states in KNOWN. These states are states which have been seen in the demonstration, which allows us to use the known-transitions to generate the successor states. If there are no states in KNOWN, then we expand states from UNKNOWN using the δ-transitions. After expanding a state, all successors states which have not been expanded by δ-transitions are added to the UNKNOWN priority queue. If a state has been seen in the demonstration and it has not yet been expanded by known-transitions, it is added to the KNOWN priority queue.

4.3 Reasoning

The reason we keep two separate priority queues is to encourage the AI to follow the demonstration before trying to explore situations that it hasn't seen before. This is both because of the reduced branching factor of following the demonstration and because following the demonstration of a human-player is inherently more human-like and less-likely to have unexpected outcomes.

4.4 Full Implementation Details

4.4.1 Environment Description

The environment used to test this approach is a fighting game we created called *FG*. This gave us complete control over the dynamics of the game. It also gave us access to internal game data which would have been considerably more difficult to access had we instead opted to modify an existing fighting game.

The game is structured as a traditional fighting game. Players move back and forth on in a 2D space, trying to land blows on one another to reduce the opponents health to zero. There are a total of 21 types of actions that the player can perform, and each of these actions can be done for a duration that corresponds to some number of frames. The specific types of actions that players can take are described in Table 4.5.

The state of the game is represented by a combination of the states of the player and opponent. A player's state includes its world position in discretized space, an indicator of its velocity, its current status. Details are described in Table 4.3 and Table 4.4

4.4.2 Data Extracted from Demonstration

In order for the AI to generate plans, we need a human demonstration to build a model of the game dynamics. Throughout this section, we will refer to a simple human demonstration where the player moves forward, hits the opponent with a low attack, and then jumps to hit the player with a jumping attack.

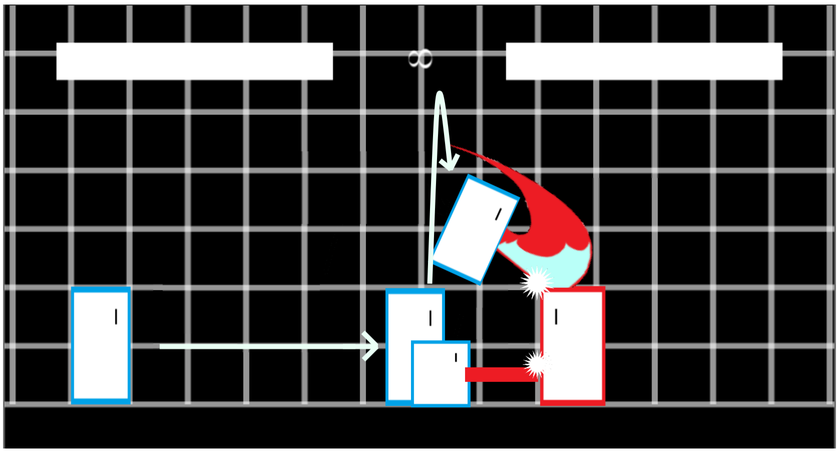


FIGURE 4.3: A Time lapse of the demonstration

As the demonstration plays out, the target player performs actions to transition between different game states. A transition (s, a, s') is recorded in each of the following cases

1. When the player starts performing a new action
2. When the player is hit during the current action, ending it early.
3. When the game state changes during the current action.

-Figure Needed here-

(A) case 1

-Figure Needed here-

(B) case 2

-Figure Needed here-

(C) case 3

The last case is particularly important for the algorithm, as it breaks down the single player's action of walking forward into multiple smaller component actions that the AI can use.

These transitions are saved as both known-transitions and δ-transitions. All known transitions are stored in a table K where $K[s]$ contains a list of all outgoing transitions (a, s'). All δ-transitions are also saved into a table D where $D[a]$ contains all action-δs encountered.

An action-δ is calculated as follows given an observed transition (s, a, s')

TABLE 4.1: How action-δ is calculated

	s	s'	action-δ
x Position	p_x	${p_x}'$	$p'_x - p_x$
y Position	p_y	${p_y}'$	$p'_y - p_y$
x Velocity	p_{xVel}	${p_{xVel}}'$	$p'_{xVel} - p_{xVel}$
y Velocity	p_{yVel}	${p_{yVel}}'$	$p'_{yVel} - p_{yVel}$
opponents x Position	q_x	${q_x}'$	$q'_x - q_x$
opponents y Position	q_y	${q_y}'$	$q'_y - q_y$
opponents x Velocity	q_{xVel}	${q_{xVel}}'$	$q'_{xVel} - q_{xVel}$
opponents y Velocity	q_{yVel}	${q_{yVel}}'$	$q'_{yVel} - q_{yVel}$
grounded	$p_{grounded}$	${p_{grounded}}'$	$p'_{grounded}$
opponent grounded	$q_{grounded}$	${q_{grounded}}'$	$q'_{grounded}$
status	p_{status}	${p_{status}}'$	p'_{status}
opponents status	q_{status}	${q_{status}}'$	q'_{status}

For the simple demonstration, the extracted set of known and δ-transitions can be seen in figure 4.5.

-Figure Needed here-

FIGURE 4.5: Transitions generated by the demonstration

Lastly, we extract goal-states from the demonstration. These are simply states s' found from the transitions where the opponent's status is *FirstHit*.

The set of goal states obtained form the demonstration are seen in figure 4.6

-Figure Needed here-

FIGURE 4.6: Goal states generated by the demonstration

4.4.3 Generating Successors

When generating a successor using a known-transition, the successor is the same s' as the one observed during the demonstration. By traveling along known successors, the plan generated by the search closely follows the exact actions taken by the player during demonstration

When generating a successor using δ-transitions, we rely on a predictor function $\phi(s, a)$. The predictor works as follows.

-Figure Needed here-

FIGURE 4.7: The AI follows the player's demonstration

To determine the effect of taking action a, we look at all action-δs associated with action a. We will refer to these action-δ's as δ Each δ has a *prior* called s_δ, which indicates the starting state of that particular recorded transition. We can assign a similarity score between the s and s_δ, which we use as a rough approximation of our confidence in the truth of that action.

$$sim(s, s_\delta) = 1 - \frac{\sum_i dist(s[i], s_\delta[i])}{\sum_i max_i}$$

$$dist(s[i], s_\delta[i]) = \begin{cases} s_\delta[i] - s[i] & \text{if } i \text{ represents the x position of either player} \\ s_\delta[i] == s[i] & \text{otherwise} \end{cases}$$

$s[i]$ represents the value of field i in state s and $max_[i]$ represents the maximum value of $dist(s[i], s_\delta[i])$.

We then create a predicted action-δ by taking a weighted average over the action-δs and the similarity score and then rounding the result.

$$\delta^*[i] \approx \begin{cases} argmax_\delta \frac{sim(s,s_\delta)}{\sum_\delta sim(s,s_\delta)}[i] & \text{if } s[i] \text{ is a categorical variable} \\ \sum_\delta \frac{sim(s,s_\delta)\delta[i]}{\sum_\delta sim(s,s_\delta)} & \text{otherwise} \end{cases}$$

To get the final prediction, we apply δ^* to the current state s to get s'.

The confidence value c that is returned with this prediction is calculated as follows.

$$sim(s, s_\delta)\text{where}\delta = argmax_\delta \frac{sim(s, s_\delta)}{\sum_\delta sim(s, s_\delta)})$$

This represents our belief in the predicted result and it also gives an indication of likelihood that the player would take this action.

4.4.4 Costs

In order to differentiate the qualities of plans, we need a suitable cost function. The cost of taking a known-transition is 1, as there is no qualitative way to evaluate one demonstrated action as being more "human-like" than another. For a δ-transitions, we apply an additional penalty that is inversely proportional to the confidence returned by the predictor.

$$(s', c) = \phi(s, a)$$

$$Cost(s, s') = \lambda / c$$

Where λ is a hypervariable. This makes it so that shorter plans which use higher confidence transitions are preferred.

===Section about Theoretical guarantees? Not sure if we have any strong ones======

4.4.5 The Goal and Heuristics

Before beginning the search, we select a random goal state from the demonstration to target. This goal state has certain qualities that are important to target. Namely, we care about the distance between the player and the opponent and the status of the player and opponent. The search tries form a plan that results in a state which matches these qualities.

In order to efficiently guide the search towards such a state, we reduce the current state to these qualities. The heuristic we use is then a measure the total distance between the current state's qualities and the goal state's qualities. The quality of a state is shown in the below table.

TABLE 4.2: The qualities extracted from state s

Field Name	Value
x Distance	$\lvert p_x - q_x \rvert$
y Distance	$\lvert p_y - q_y \rvert$
grounded	$p_{grounded}$
opponent grounded	$q_{grounded}$
status	p_{status}
opponents status	q_{status}

4.5 Dealing with long search times: Anytime Search

Because of fast-paced nature of fighting games, players need to be able to reliably make split second decisions. This constraint then extends to our AI, as it can't afford to plan seconds at a time, as the game state might change drastically within that time period. In our implementation, the AI is required to come up with a plan within 50 milliseconds. If it cannot reach a goal state, it instead formulates a plan to get to the state with the highest projected value. The idea is that by reaching this intermediate state, it can then resume plannning from the position that is closer to the goal, giving the impression of one seemless plan, when it in fact generated that plan during execution.

4.5.1 Dealing with a changing game state: replanning

Because the opponent is allowed to move during plan execution, the plan we formulate is likely to encounter a state that we didn't expect. Because of the short time to plan we enforced, we can seemlessly replan whenever we hit an unexpected state and have the AI adjust accordingly. An example of this is when the opponent moves back while we're approaching them. Due to replanning, the AI will then know to continue to move towards the opponent, rather than stopping at it's original location and attacking like initially planned.

4.6 PENDING SECTIONS (MAY NOT GET TO THEM IN THE NEXT 2 WEEKS)

4.7 Dealing with bad predictions, updating the predictor

IN PROGRESS

4.8 Interacting with the opponent, how to respond on defense

IN PROGRESS This might need its own subsection depending on how everything goes. Right now, results are promising for an offensive AI whose training data is a basic demonstration of high and low attacks, but we have yet to see if the AI can really respond to dealing with an actively moving opponent who will also attack it. Basically, will it learn to block? Do we have to do something to force it to learn to block?

4.9 Results

4.10 Discussion

4.11 Additional Details

TABLE 4.3: AI Situation description

Field Name	Description
x Position	x position of the target player. Descrtized by 0.5 unit increments
y Position	y position of the target player. Descrtized by 1.0 unit increments
x Velocity	The sign of the x velocity of the target player.
y Velocity	The sign of the y velocity of the target player.
opponents x Position	x position of the opponent. Descrtized by 0.5 unit increments
opponents y Position	y position of the opponent. Descrtized by 1.0 unit increments
opponents x Velocity	The sign of the x velocity of the opponent player
opponents y Velocity	The sign of the x velocity of the opponent player
grounded	Whether or not the target player is on the ground
opponent grounded	Whether or not the opponent is on the ground
status	The target player's current status
opponents status	The opponent's current status

TABLE 4.4: Player Status descriptions

Status	Description
Stand	When the player stands still
Crouch	When the player is crouching
Air	When the player is airborne
Highblock	When the player is blocking high
Lowblock	When the player is blocking low
FirstHit	When the player was initially hit by an attack
Hit	When the player is in hitstun after being hit
KnockdownHit	When the player has been knocked down after being hit multiple times
Tech	When the player is getting up after being knocked down
Moving	When the player is walking on the ground
Dashing	When the player is performing a dash on the ground
AirDashing	When the player is performing a dash in the air
StandAttack	When the player has the stand hitbox out
LowAttack	When the player has the low hitbox out
OverheadAttack	When the player has the overhead hitbox out
AirAttack	When the player has the AirAttack hitbox out
DP	When the player has the Dp hitbox out
Recovery	The recovery period after an attack

TABLE 4.5: Player Action descriptions

Action	Description
Stand	The player is standing still
Crouch	The player is crouching
WalkLeft	The player is walking left
WalkRight	The player is walking right
JumpNeutral	The player jumped in place
JumpLeft	The player jumped to the left
JumpRight	The player jumped to the right
Attack	The player did a standing attack. Can be blocked high or low
Overhead	The player does a standing overhead attack. Can only be blocked high
LowAttack	The player does a crouching low attack. Can only be blocked low
AirAttack	The player does an attack in the air. Can only be blocked high
StandBlock	The player is actively blocking high.
CrouchBlock	The player is actively blocking high.
DashLeft	The player does a single quick dash to the left
DashRight	The player does a single quick dash to the right
AirdashLeft	The player does a single quick dash to the left in the air
AirdashRight	The player does a single quick dash to the right
DP	The player does a quick invulnerable strike. Has long recovery
TechNeutral	The player gets up from being knocked down
TechLeft	The player rolls to the left and gets up from being knocked down.
TechRight	The player rolls to the right and gets up from being knocked down.

Appendix A

Frequently Asked Questions

A.1 How do I change the colors of links?

The color of links can be changed to your liking using:

`\hypersetup{urlcolor=red}`, or

`\hypersetup{citecolor=green}`, or

`\hypersetup{allcolor=blue}`.

If you want to completely hide the links, you can use:

`\hypersetup{allcolors=.}`, or even better:

`\hypersetup{hidelinks}`.

If you want to have obvious links in the PDF but not the printed text, use:

`\hypersetup{colorlinks=false}`.