

---

# Creating Human-like Fighting Game AI through Planning

---

*Roger Liu*

CMU-CS-17-128

December 2017

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

**Thesis Committee**

Maxim Likhachev, Chair

Jessica Hodgins

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

Copyright © 2017 Roger Liu

**Keywords:** Artificial Intelligence, Human-Computer Interaction, Game AI, Planning, Human-Imitation, Fighting Games

CARNEGIE MELLON UNIVERSITY

*Abstract*

Faculty Name

Department or School Name

Master of Science

**Creating Human-like Fighting Game AI through Planning**

by Roger LIU

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- $\delta$ s, which relate the action performed with the change in the game state. These action- $\delta$ s 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- $\delta$ s 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*

Thanks to Maxim Likhachev for being my advisor and for his assistance in throughout this endeavor. This project spiraled in a lot of different directions at first, and he was a tremendous help in bringing it all to this state. His patience and understanding are remarkable.

Thanks to Jessica Hodgins for being my committee member. I would also like to acknowledge all of the faculty whose classes I've had the pleasure of taking for driving me to pursue doing research. Additional thanks to the everyone who has made this program possible in the first place.

Lastly, I'd like to thank my friends and particularly my roommates for making me who I am today. This research would not exist if not for the interests I have cultivated with you all.



# Contents

<b>Abstract</b>	<b>iii</b>
<b>Acknowledgements</b>	<b>v</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Related Work</b>	<b>3</b>
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
<b>3 Planning-based Approach Overview</b>	<b>9</b>
3.1 Why Use Search? . . . . .	10
3.2 Agent Framework Overview . . . . .	11
<b>4 Planning-based Approach in Details</b>	<b>13</b>
4.1 Action- $\delta$ s . . . . .	13
4.2 Demonstration $\delta$ -Search . . . . .	14
4.3 Environment Description . . . . .	15
4.4 Extracting Data from Demonstrations . . . . .	16
4.5 Generating Successors . . . . .	18
4.6 Costs . . . . .	20
4.7 The Goal and Heuristics . . . . .	20
4.8 Pseudo-Code . . . . .	22
4.9 Additional Considerations . . . . .	22

4.9.1	Dealing with Long Search Times: Real-time Search . . . . .	22
4.9.2	Dealing with a Changing Game State: Replanning . . . . .	23
4.9.3	Dealing with Bad Predictions, Updating the Predictor . . . . .	23
<b>5</b>	<b>Additional Details</b>	<b>25</b>
<b>6</b>	<b>Results and Discussion</b>	<b>27</b>
6.1	Results . . . . .	27
6.1.1	Similarity . . . . .	28
6.1.2	Effectiveness . . . . .	29
6.1.3	Qualitative Analysis . . . . .	31
<b>7</b>	<b>Conclusion</b>	<b>33</b>
	<b>Bibliography</b>	<b>35</b>



# List of Figures

3.2	Player Strategy Example	10
3.3	AI Framework	11
4.1	Example Action- $\delta$	13
4.2	Overview of $\delta$ -search	15
4.3	Example of a Human Demonstration	16



# List of Tables

4.1	How Action- $\delta$ is Calculated	17
4.2	Demonstration Known-Transitions	18
4.3	Demonstration Action- $\delta$ s	18
4.4	Demonstration Goal States	18
4.5	Qualities Extracted from State $s$	20
5.1	AI Situation Description	25
5.2	Player Status Descriptions	26
5.3	Player Action Descriptions	26
6.1	Similarity Measurements: One Demonstration	28
6.2	Similarity Measurements: Half of the Demonstrations	28
6.3	Similarity Measurements: Human Play	28
6.4	Hits Landed Per Session: : One Demonstration	29
6.5	Actions required per Hit: One Demonstration	30
6.6	Hits Landed Per Session: Half of the Demonstrations	30
6.7	Actions required per Hit: Half of the Demonstrations	30
6.8	Hits Landed Per Session: : Human Player	31
6.9	Actions required per Hit: Human Player	31



## Chapter 1

# Introduction

Fighting games are unique among competitive multiplayer games in that they are real-time, 1-on-1 contests 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 possible. While the option to play online exists, it is not ideal due to the lag introduced by network latency. In addition, the AI in these games are generally considered a poor substitute for real players. 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. Then, we describe our algorithm, where we plan on the actions provided by human demonstrations to reach a desired outcome. Lastly, we test our algorithm and compare its performance to other existing implementations.

## 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 (Julian Togelius and Lucas, 2007). 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 three different components was used. One focused on matching the player's 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 respective players, as the AI imitating a slow, careful driver behaved in a significantly different way compared to the one that imitated a faster, reckless driver. However, closer inspection of the driving AI showed that the resulting behavior was not conceivably human. A later attempt which incorporated a focus on driving optimally into the fitness function also did not obtain convincing results. However,

the study showed a clear trade-off between driving optimally and improving driver similarity. (Van Hoorn et al., 2009)

## 2.2 Super Mario Bros.

Researchers also developed several methods to mimic human behavior in the space of 2D platformers, specifically a modified version of Super Mario Bros. (Ortega et al., 2013). 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 that were 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 (Lee et al., 2014).

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 if the distance between traces 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 (Cho, Park, and Yang, 2007). 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. (Firoiu, Whitney, and Tenenbaum, 2017). However, the optimal AI were not ideal substitutes for playing against human opponents. For example, the Super Smash Bros. AI was specifically trained against only one kind of opponent, meaning that it was limited in the kinds of matchups it could perform well in. In addition, it exhibits obviously artificial traits such as impossible to perform rapid back and forth movements.

## 2.4 Ghost AI

With regards to creating AI that was specifically human-like, the most notable and widespread technique is Ghost AI. Researchers implemented a version of this algorithm on the commercial game Street Fighter. This AI initializes a histogram with the frequencies that target player performs actions in different situations uses those actions at the same frequency (Thunputtarakul and Kotrajaras, 2007). In the adaptive version, the actions also have an associated weight that updates based on the reward gained from performing them (Lueangrueangroj and Kotrajaras, 2009). These weights are then used to adjust the frequency that actions are selected.

To evaluate this AI, they recorded sessions of the player and their corresponding Ghost AI. The players were then asked to watch these both their own and the AI's recorded sessions and perform "phantom" inputs as if they were in the same situation. The recordings were then scored by the similarity of the recording inputs to the subject's "phantom" inputs. This method showed promising results, as the Ghost AI's similarity was able match around 75% of the real recordings similarity. 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 doesn't account for the opponent's state, such as how they are blocking, which factors into the decision making process for a real human player.

## 2.5 Data-driven Finite State Machines

One final approach utilizes 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 to designate the states of Finite State Machine. The transitions between these states are learned from the demonstrations. The state machine is then used to govern the AI's behavior during gameplay.

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 are unable to capture things like a player's sense of timing.

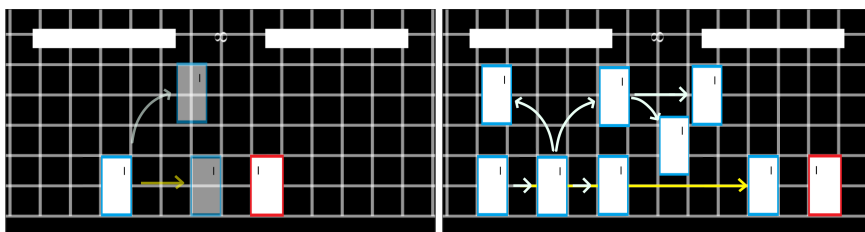


## Chapter 3

# Planning-based Approach

## Overview

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 tradeoff between optimality and employing a degree of randomness when these agents decide on 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 non-human.



(A) Always doing the same action is too predictable  
 (B) Some actions don't make sense in the given context

### 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 the 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 cost functions and 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 kind of player behavior:

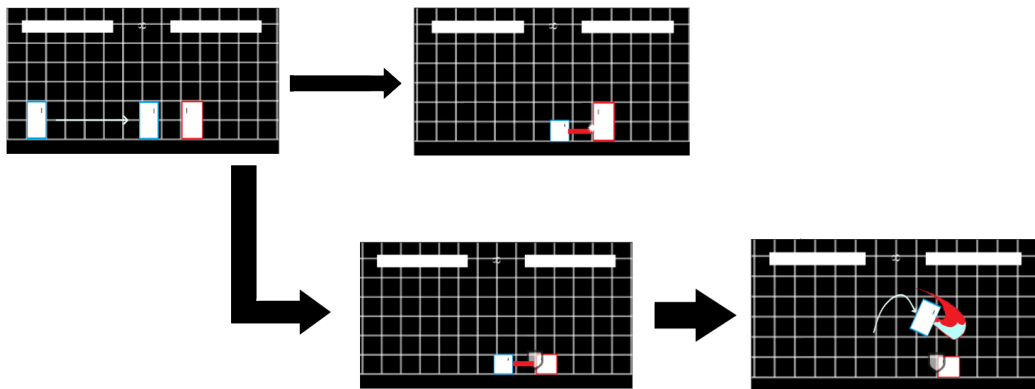


FIGURE 3.2: Player Strategy Example

During the game, 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 two 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.

### 3.2 Agent Framework Overview

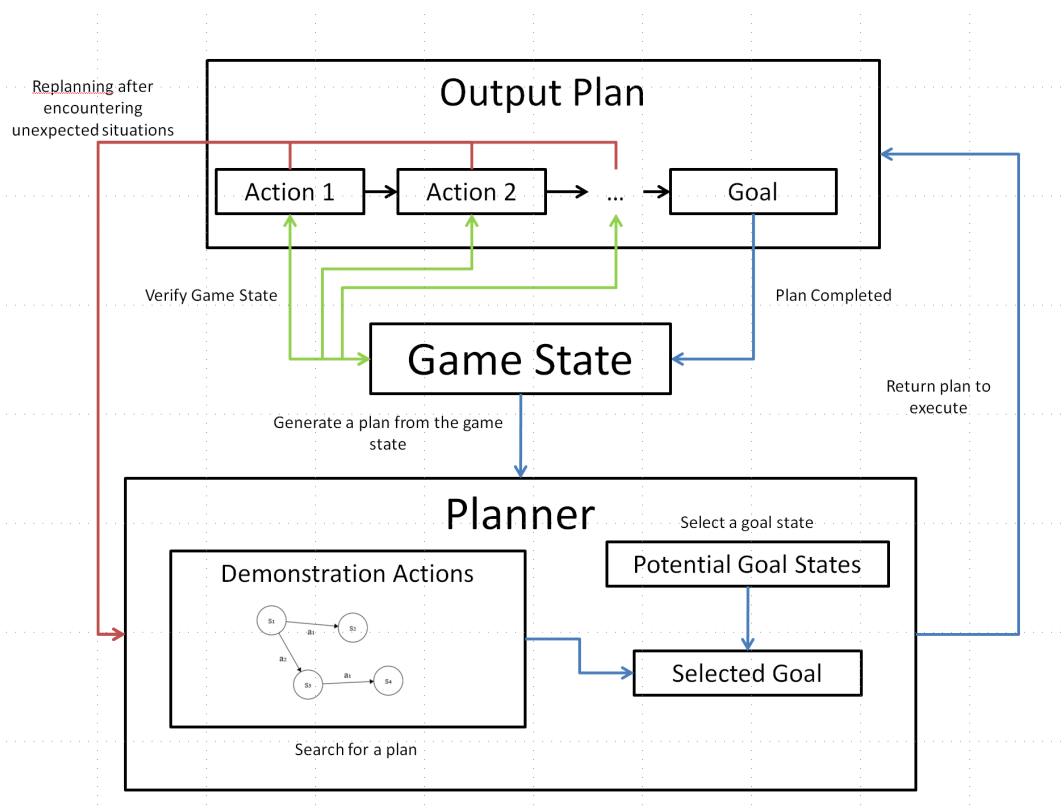


FIGURE 3.3: AI Framework

The agent will work as follows. After taking in the game state, the agent forms a plan to reach a predefined goal state. The goal states are generated from the demonstration data, and the plan that the agent forms uses actions found in from the demonstration data. After producing the best available plan in the allotted amount

of time, the agent then tries to execute that plan. After executing an action, it evaluates the game state. If the action brought the agent to the expected game state, it then will try to execute the next action in the plan. However, if the agent finds itself in an unexpected game state, it forms another plan to try to get to the goal. Once a plan is successfully executed or after enough time has passed, the agent picks a new goal state to target and repeats the process.



## Chapter 4

# Planning-based Approach in Details

### 4.1 Action- $\delta$ s

First, we explain the concept of action- $\delta$ s, 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- $\delta$ , which represents how the game state changes as a result of taking action  $a$ . Figure 4.1 shows that jumping to the right has an action- $\delta$  that puts the player up and to the right

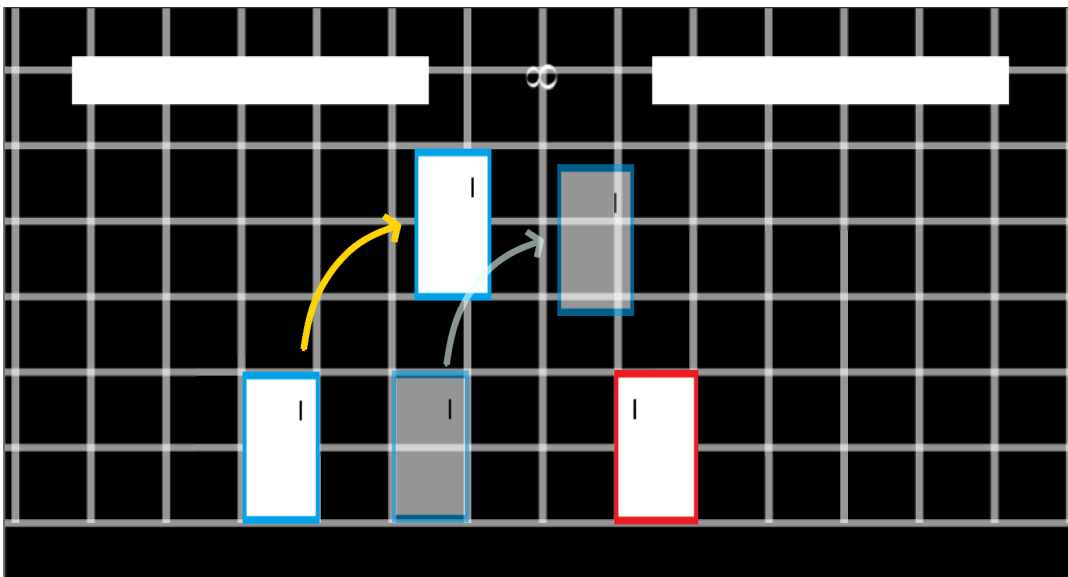


FIGURE 4.1: Example Action- $\delta$

These action- $\delta$ s 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 $\delta$ -Search

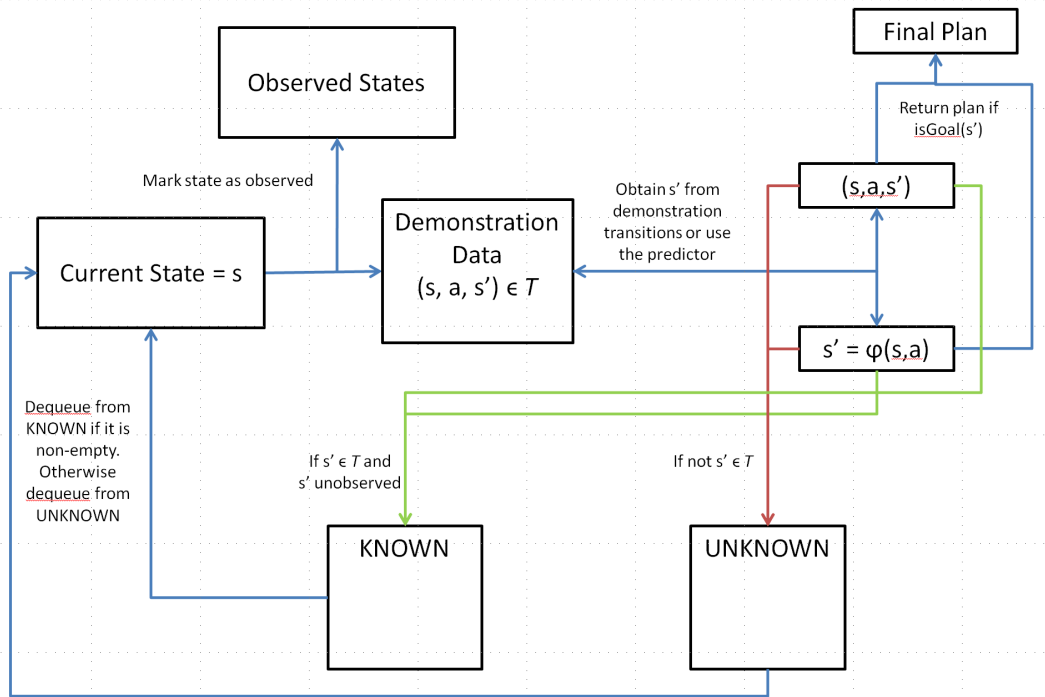
The task of emulating a player's behavior is represented as a graph search problem. Specifically, the objective is to form a plan to hit the opponent using the actions demonstrated by the target player. The plan should be feasible and resemble a plan executed by the target player as much possible.

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  $\delta$ -transitions. These transitions are of the form  $(s, a, \phi(s, a))$ , where  $\phi(s, a)$  is a predictor function that takes in the current state  $s$  and an action  $a$  performed by the target player. This predictor function generates the predictions by learning from the action- $\delta$ s of action  $a$  that were 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 goal state selected from the demonstration. The goal state is selected according to a weighted distribution. 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, 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  $\delta$ -transitions. After expanding a state, all successors states

FIGURE 4.2: Overview of  $\delta$ -search

which have not been expanded by  $\delta$ -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. Once we find a state that is a goal state, we return the plan to that state.

Prioritizing known-transitions makes it so that the plan we form tries to use actions shown in the demonstration as much as possible. This is a desirable quality as replicating the demonstrated actions in the proper situations precisely replicates that human's behavior in those instances. It also has the effect of reducing the number of successors we add to our priority queues, which is important as the number of  $\delta$ -transitions increases with the number of unique actions in our demonstration data set.

Pseudo-code for this algorithm can be found in Section 4.8.

### 4.3 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 opponent's 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 5.3.

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, and its current status. Details are described in Table 5.1 and Table 5.2

#### 4.4 Extracting Data from Demonstrations

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.

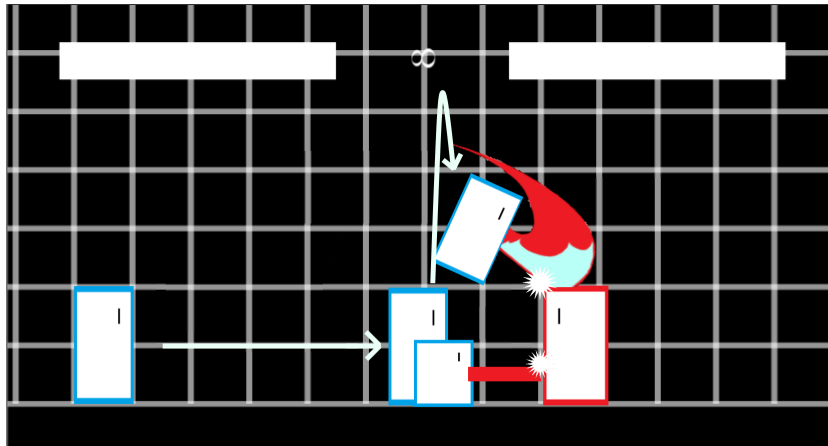
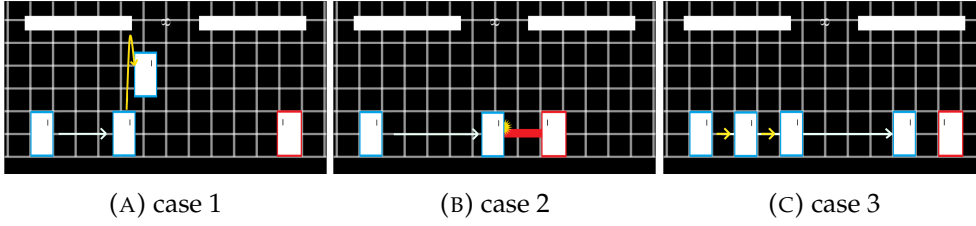


FIGURE 4.3: Example of a Human Demonstration

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.



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  $\delta$ -transitions. All known transitions are stored in a table  $K$  where  $K[s]$  contains a list of all outgoing transitions  $(a, s')$ . All  $\delta$ -transitions are also saved into a table  $D$  where  $D[a]$  contains all action- $\delta$ s encountered.

An action- $\delta$  is calculated as follows given an observed transition  $(s, a, s')$ .  $p$  represents the target player's state information and  $q$  represents the opponent's state information

TABLE 4.1: How Action- $\delta$  is Calculated

	$s$	$s'$	action- $\delta$
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	$q_{status}$	$q'_{status}$	$q'_{status}$

In the case of the simple demonstration, some of the known-transitions that are extracted are found in Table 4.2

s	a	s'
[-6,0,0,0,1,0,0,0,true,true,Stand,Stand]	WalkRight 1	[-6,0,1,0,1,0,0,0,true,true,Moving,Stand]
[-6,0,0,0,1,0,0,0,true,true,Stand,Stand]	WalkRight 10	[-4,0,1,0,1,0,0,0,true,true,Moving,Stand]
[-6,0,0,0,1,0,0,0,true,true,Stand,Stand]	WalkRight 30	[0,0,1,0,1,0,0,0,true,true,Moving,Stand]
[0,0,1,0,1,0,0,0,true,true,Moving,Stand]	Crouch 1	[0,0,0,0,1,0,0,0,true,true,Crouch,Stand]
[0,0,1,0,1,0,0,0,true,true,Crouch,Stand]	JumpNeutral 1	[0,0,0,1,1,0,0,0,false,true,Air,Stand]
[0,0,1,0,1,0,0,0,true,true,Crouch,Stand]	JumpNeutral 45	[0,0,0,-1,1,0,0,0,false,true,Air,Stand]
[0,0,0,-1,1,0,0,0,false,true,Air,Stand]	AirAttack 3	[0,0,0,-1,1,0,0,0,false,true,AirAttack,FreshHit]

TABLE 4.2: Demonstration Known-Transitions

The corresponding action- $\delta$ s are then described in Table 4.3

a	action- $\delta$
WalkRight 1	[0,0,1,0,0,0,0,0,true,true,Moving,Stand]
WalkRight 10	[2,0,1,0,0,0,0,0,true,true,Moving,Stand]
WalkRight 30	[6,0,1,0,0,0,0,0,true,true,Moving,Stand]
Crouch 1	[0,0,0,0,0,0,0,0,true,true,Crouch,Stand]
JumpNeutral 1	[0,0,0,1,0,0,0,0,true,false,Air,Stand]
JumpNeutral 45	[0,0,0,-1,0,0,0,0,false,true,Air,Stand]
Attack 3	[0,0,0,-1,1,0,0,0,false,true,AirAttack,FreshHit]

TABLE 4.3: Demonstration Action- $\delta$ s

Lastly, we extract goal-states from the demonstration. These are simply states  $s'$  found from the transitions where the opponent's status is *FreshHit*. The set of goal states obtained from the demonstration are seen in Table 4.4

$p_x$	$p_y$	$p_{xVel}$	$p_{yVel}$	$q_x$	$q_y$	$q_{xVel}$	$q_{yVel}$	$p_{grounded}$	$q_{grounded}$	$p_{status}$	$q_{status}$
0	0	0	0	0	0	0	0	true	true	LowAttack	FreshHit
0	0	0	-1	1	0	0	0	false	true	AirAttack	FreshHit

TABLE 4.4: Demonstration Goal States

## 4.5 Generating Successors

When trying to figure out the successor of a state-action pair  $(s, a)$ , we have either seen that tuple in the demonstration or we haven't. If we have, we can generate the successor using a known-transition. The resulting successor is the same  $s'$  as the one observed in the demonstration transition  $(s, a, s')$ . By traveling along known-successors, the plan generated by the search closely follows the exact actions taken by the player during demonstration

If  $(s, a)$  has not been seen in the demonstration data, then we have to use a  $\delta$ -transition. When generating a successor using  $\delta$ -transitions, we rely on a predictor function  $\phi(s, a)$ . The predictor works as follows.

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

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

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

$s[i]$  represents the value of field  $i$  in state  $s$  and  $\text{max}_i[i]$  represents the maximum value of  $\text{dist}(s[i], s_\delta[i])$ .

We then create a predicted action- $\delta$  by taking a weighted average over the action- $\delta$ s and the similarity score and then rounding the result.

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

To get the final prediction, we apply  $\delta^*$  to the current state  $s$  to get  $s'$ . For categorical variables,  $s'[i] = \delta^*[i]$  and for everything else,  $s'[i] = s[i] + \delta^*[i]$

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

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

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

## 4.6 Costs

In order to differentiate the qualities of plans, we need a suitable cost function. The cost of taking a known-transition is 1.0, as there is no qualitative way to evaluate one demonstrated action as being more "human-like" than another. For a  $\delta$ -transitions, we apply an additional penalty that is inversely proportional to the confidence returned by the predictor.

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

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

Where  $\lambda$  is a hypervariable. This makes it so that shorter plans which use higher confidence transitions are favored.

## 4.7 The Goal and Heuristics

Before beginning the search, we select a random goal state from the demonstration to target. This the goal states selected are weighted by their similarity to the initial starting state. Goals are selected randomly because it simulates how a player might vary its objective during gameplay. 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 statuses of the player and opponent. The search tries form a plan that results in a state which matches these qualities, shown in Table 4.5.

TABLE 4.5: Qualities Extracted from State  $s$

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

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.

## 4.8 Pseudo-Code

---

### Algorithm 1 Full Pseudo-Code of $\delta$ -search

---

```

1: function  $\delta$ -SEARCH( $s_{start}, demonstrations$ )
2:    $OBS = \{\}$ 
3:    $OBS_{\delta} = \{\}$ 
4:    $KNOWN = \{\}$ 
5:    $UNKNOWN = \{\}$ 
6:    $s_{goal} = GetGoal(demonstrations)$ 
7:    $KNOWN \cup \{s_{start}\}, UNKNOWN \cup \{s_{start}\}$ 
8:   while  $|KNOWN| \neq 0$  OR  $|UNKNOWN| \neq 0$  do:
9:     if  $|KNOWN| \neq 0$  then
10:      Remove the smallest  $[f(s) = g(s) + h(s)]$  from  $KNOWN$ 
11:       $OBS \cup \{s\}$ 
12:      if  $isGoal(s)$  then return  $plan(s)$ 
13:      Expand  $s$  with  $(s, a, s') \in demonstrations$     $s' \notin OBS$  OR  $s' \notin OBS_{\delta}$ 
14:      if  $(s', -, -) \in demonstrations$  then
15:         $KNOWN \cup \{s'\}$ 
16:         $UNKNOWN \cup \{s'\}$ 
17:      else if  $|UNKNOWN| \neq 0$  then
18:        Remove the smallest  $[f(s) = g(s) + h(s)]$  from  $UNKNOWN$ 
19:        if  $isGoal(s)$  then return  $plan(s)$ 
20:         $OBS_{\delta} \cup \{s\}$ 
21:        Expand  $s$  with  $s' = \phi(s, a)$     $s' \notin OBS$  OR  $s' \notin OBS_{\delta}$ 
22:        if  $(s', -, -) \in demonstrations$  then
23:           $KNOWN \cup \{s'\}$ 
24:           $UNKNOWN \cup \{s'\}$ 

```

---

## 4.9 Additional Considerations

### 4.9.1 Dealing with Long Search Times: Real-time 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 an explored state which is the most similar to the current goal state. The idea is that by reaching this intermediate state, it can then resume planning from the position that is closer to the goal, giving the impression of one seamless plan, when in fact generated that plan during execution.

### **4.9.2 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 states which do not match the planned transitions. Because of the short time to plan we enforced, we can seamlessly replan whenever we hit an unexpected state and have the AI adjust accordingly. An example of this is when the opponent moves back while the AI is approaching them. Due to replanning, the AI will then know to continue to move towards the opponent, rather than stopping at its original location and attacking like initially planned.

### **4.9.3 Dealing with Bad Predictions, Updating the Predictor**

One final thing that we do to ensure that our AI is robust is update the predictor. As the game progresses, the AI logs the transitions that don't match up with its predictions and adds it back to the training set. This helps the AI make better predictions in the future and helps avoid local minima plans. This is crucially important as otherwise the AI is likely to get stuck performing the same action repeatedly.



## Chapter 5

# Additional Details

TABLE 5.1: 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 5.2: 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 5.3: 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 low.
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 in the air
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.

## Chapter 6

# Results and Discussion

### 6.1 Results

To test the AI's performance, we set up the following experiment. First, the subject was given time to familiarize and acclimate themselves to the controls and dynamics of the game. Afterwards, the subject played five 20 second long matches against a computer opponent that follows a set defensive behavior pattern. A random set of matches from this pool was selected to be the demonstration sessions, while the rest were grouped into a set of holdout sessions. We then used the data from the demonstration session to create three kinds of agents. One of the agents used our novel search technique. We refer to this agent as Search AI. The other agents implemented an simple N-gram AI and the adaptive Ghost AI and served as a point of comparison. An N-gram AI is essentially a vanilla GhostAI that is not aware of the current game state. We then recorded several sessions where each of these agents was pitted against the same computer opponent that the subject faced.

We performed several kinds of experiments. In one, one session was selected to be the demonstration session. In another, half of the available sessions were selected as the demonstration session. In the last one, the opponent used for the demonstrations and during the demonstration was a human opponent. Half of the available sessions were selected as the demonstration session and the human opponent was kept consistent through all of sessions.

The recorded sessions were evaluated along three criteria: Similarity, Effectiveness, and Qualitative Analysis.

The data on the tables are interpreted as follows. The mean refers to the mean

of all of the similarity scores. The Avg. Standard Deviation refers to the average of the standard deviation within each player’s mimicking agent. So Player A’s N-gram AI may have a similarity standard deviation of 1, Player B’s may have a similarity standard deviation of 2, so the Avg Standard Deviation of the N-gram AI would be 1.5

### 6.1.1 Similarity

To measure similarity, we used a metric based on the work by Hamid et al., 2009. The metric works as follows. Given event sequences  $A$  and  $B$ , let  $H_A$  and  $H_B$  represent the underlying histogram of all  $n$ -Grams where  $1 \leq n \leq 3$ . Then let  $S_A$  and  $S_B$  be the number of unique substrings in  $H_A$  and  $H_B$  respectively, and let  $f(s|H)$  be the number of times substring  $s$  appears in histogram  $H$ . The similarity metric is then defined as follows:

$$PlayerSim(A, B) = \sum_{s \in S_A \cup S_B} \frac{|f(s|H_A) - f(s|H_B)|}{f(s|H_A) + f(s|H_B)}$$

We recorded the similarity between and each AI’s test sessions and the holdout sessions and report them below. We also include the similarity between the demonstration and holdout sessions as a point of comparison.

TABLE 6.1: Similarity Measurements: One Demonstration

	Holdout Session	N-gram	GhostAI	Search AI
Mean	0.447	0.191	0.242	0.237
Avg. Standard Deviation	0.078	0.057	0.033	0.040

TABLE 6.2: Similarity Measurements: Half of the Demonstrations

	Holdout Session	N-gram	GhostAI	Search AI
Mean	0.436	0.210	0.221	0.181
Avg. Standard Deviation	0.085	0.047	0.035	0.028

TABLE 6.3: Similarity Measurements: Human Play

	Holdout Session	N-gram	GhostAI	Search AI
Mean	0.301	0.136	0.168	0.165
Avg. Standard Deviation	0.054	0.033	0.048	0.017



From a similarity point of view the algorithm does decently. When given a single sessions worth of demonstration, the Search AI compares favorably to GhostAI and surpasses the N-gram AI. However, once more data is added to the demonstration data set, Search AI’s performance begins to fall off while N-gram improves and overtakes it. This is likely because of the increased number of demonstrated actions increases the number of state successors when taking  $\delta$ -transitions, slowing down the search. The Search AI’s low standard deviation is expected, as its planning-based behavior should have less of the chaotic randomness shown by the other AI.

Taking a closer look at the data points where the AI’s have more demonstrations, the instances where Search AI did dramatically better than the N-gram AI and the instances where it did dramatically worse were split fairly evenly. In the instances where we saw worse performance, the Search AI would repeatedly attack in place until the predictor could update and have create a more feasible plan. This shows the Search AI may be able to achieve high similarity with larger data sets if it can learn to overcome these planning local minima.

Lastly, the similarity measurements for human-play are mostly in-line with the similarity when pit against an computer opponent. Despite having more demonstration data, the Search AI is able to keep pace with the GhostAI and outperform the N-gram AI. This likely happens because the human-opponent rarely lets the Search AI form poor local minima plans, as they will hit and interrupt it before it can get stuck for an extended period.

### 6.1.2 Effectiveness

To measure Effectiveness, we compared the number of hits that were landed in the training session to the average number of hits recorded for each other category. We also compared the average number of actions that needed to be performed to land a hit. We rec

TABLE 6.4: Hits Landed Per Session: : One Demonstration

	Original Player	N-gram	GhostAI	Search AI
Mean	6.75	2.6	2.35	4
Avg. Standard Deviation	1.55	1.162	0.683	1.368

TABLE 6.5: Actions required per Hit: One Demonstration

	Original Player	N-gram	GhostAI	Search AI
Mean	10.54	22.49	36.83	11.45
Avg. Standard Deviation	3.77	11.54	19.39	2.66

From an effectiveness point of view, the search AI is definitely more effective compared to the other 2. Not only does it on average land more hits per session, it also uses fewer actions to achieve those hits. This makes sense because the Search AI tries to create plans that let it reach a goal state relatively efficiently, whereas N-gram and GhostAI are constantly performing actions. It is also good to see that the Search AI's requires roughly the same number of actions to land a hit as the original players. It shows that the AI is not just taking an optimal path to the goal, but is in some way matching the success-rate of the player it is emulating.

One thing that is odd is that the Ghost AI seems to perform worse than the N-gram AI in this category. This is probably because the Ghost AI does not receive a penalty if an opponent blocks an attack, meaning that in one instance it could hit the opponent and then be tricked into thinking that action is always the best one to take to maximize reward.

TABLE 6.6: Hits Landed Per Session: Half of the Demonstrations

	Original Player	N-gram	GhostAI	Search AI
Mean	6.75	3.15	2.4	2.7
Avg. Standard Deviation	1.546	2.041	1	1.186

TABLE 6.7: Actions required per Hit: Half of the Demonstrations

	Original Player	N-gram	GhostAI	Search AI
Mean	10.541	20.079	37.835	16.259
Avg. Standard Deviation	3.776	7.200	10.670	2.610

When we increase the amount of demonstration data, the Search AI's performance worsens by a significant margin. This is likely due to the fact that we are able to expand less states in the limited amount of time given due to the increase number of  $\delta$ -transitions.

Against a human player it's surprising to see that the Search AI is actually more effective than even the original player. This is probably because the Search AI is potentially able to stitch together the original players actions to create more effective

TABLE 6.8: Hits Landed Per Session: : Human Player

	Original Player	N-gram	GhostAI	Search AI
Mean	6.8	4.4	3.8	9.4
Avg. Standard Deviation	5.418	5.953	2.482	4.224

TABLE 6.9: Actions required per Hit: Human Player

	Original Player	N-gram	GhostAI	Search AI
Mean	9.588	17.727	12.684	5.809
Avg. Standard Deviation	0.540	2.197885296	6.929140424	0.384684368

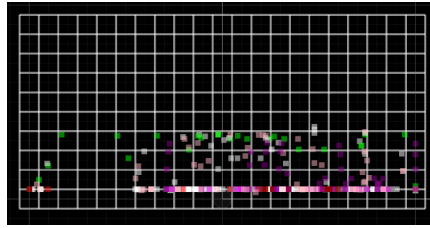
strategies that the original player might not even have considered. Additionally, the Search AI is able to pick the best parts of all of the mimicked players demonstrations and use them to greater effect against the opponent. Lastly, the human-player takes offensive actions unlike the defensive AI, which make them more vulnerable to attacks in general.

The scores for the N-gram and GhostAI are expectedly low. This is because neither the N-gram or GhostAI have the proper means for taking actions after being hit. The N-gram AI may try to take actions that are no longer available to it while it is knocked down, and though the GhostAI has some awareness of game context, if the world state is not the same as something encountered in the demonstration it is forced to pick from a uniform random distribution.

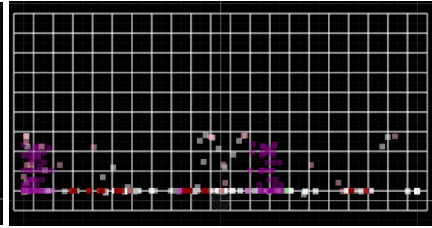
### 6.1.3 Qualitative Analysis

For our qualitative analysis we also visually inspected heatmaps of the demonstrations like follows. We also analyzed the video footage of the different AI's

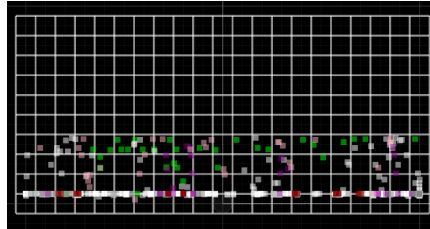
Lastly from a qualitative standpoint, it is hard to infer much from the heatmaps. They clearly show that Search AI and GhostAI are a step above the N-gram AI, but beyond that there is no clear delineation. Search AI seems to more accurately represent the grounded actions of the original player compared to GhostAI, but it is hard to tell the difference. However, in the video analysis, the Search AI's movement was the smoothest. This is because planning allows it to take broad motions before deciding to take a different action. That said, both the Ghost AI and Search AI were prone to getting stuck into certain repeated patterns, a trait which to most people signifies a non-human player. This is likely because both GhostAI and Search AI try



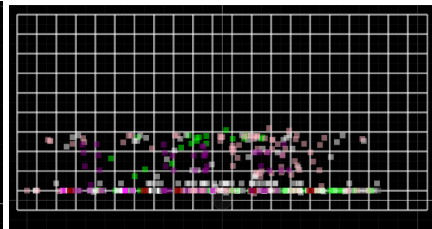
(A) Original Player Heatmap



(B) N-gram AI Heatmap



(C) GhostAI Heatmap



(D) Search AI Heatmap

to reach the goal of hitting the opponent in an efficient manner. This adherence to an "optimal" kind of play can result in situations where the agent behaves greedily to a fault and comes off as non-human-like.

## Chapter 7

# Conclusion

Among competitive multiplayer games, fighting games are among the most expressive of the player's playstyle. The tight dynamics of the game combined with the fast-paced close-quarters combat means that it's vital for a player to understand the opponent's behavior in order to secure an advantage. Replicating human behavior is a difficult task for AI, as it has to generalize the actions of a player across uncountably many different game states. Any illogical movement can break the player's suspension of disbelief, and if it plays it too safe it's behavior patterns will be quickly figured out and exploited by human opponents.

In this paper we explored a new technique to create human-like behavior for fighting games. By using search, we have enabled the AI to plan and execute long strategies to reach its goal. By using action- $\delta$ s, we enable the AI to learn the results of actions and to understand how to use them in any context. These tools helped give our AI the capability of expressing the attributes of a human fighting-game player.

We then compared our approach to repeat human demonstrations and other common implementations of fighting game AI in this field. The technique showed to be effective at emulating certain aspects of human behavior. Specifically, it did a good job in replicating the qualitative feel of a human player and was the best at replicating the effectiveness of a human player. However, it was not able to truly capture the player's behavior, recording a similarity score that was comparable to the other kinds of AI.

Moving forward, there are a few additional avenues that this work could go down. One is augmenting it with a better predictor function  $\phi(s, a)$ , as this would allow it to better understand how actions affect the state around it. This sort of project

would be an entire undertaking in of itself, as it would require more advanced Machine Learning techniques to be successful. In addition, improving the adaptability of the predictor would improve the human-likeness of the AI, as it would capture a player's ability to alter their strategy in response to different circumstances.

Another problem that should be tackled is the collection and usage of useful player data. As is, data collection is time-consuming and expensive because of the time that subjects would have to spend testing. In addition, the AI is not able to use large data-sets well due to looking through the action- $\delta$ s of all of the demonstrated actions. Devising some way to compactly obtain more useful information from small data sets could drastically improve the AI's expressive capabilities.

# Bibliography

- Cho, Byeong Heon, Chang Joon Park, and Kwang Ho Yang (2007). "Comparison of AI Techniques for Fighting Action Games - Genetic Algorithms/Neural Networks/Evolutionary Neural Networks". In: *Entertainment Computing – ICEC 2007: 6th International Conference, Shanghai, China, September 15-17, 2007. Proceedings*. Ed. by Lizhuang Ma, Matthias Rauterberg, and Ryohei Nakatsu. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 55–65. ISBN: 978-3-540-74873-1. DOI: [10.1007/978-3-540-74873-1\\_8](https://doi.org/10.1007/978-3-540-74873-1_8). URL: [https://doi.org/10.1007/978-3-540-74873-1\\_8](https://doi.org/10.1007/978-3-540-74873-1_8).
- Firoiu, Vlad, William F. Whitney, and Joshua B. Tenenbaum (2017). "Beating the World's Best at Super Smash Bros. with Deep Reinforcement Learning". In: *CoRR* abs/1702.06230. arXiv: [1702.06230](https://arxiv.org/abs/1702.06230). URL: <http://arxiv.org/abs/1702.06230>.
- Hamid, Raffay et al. (2009). "A novel sequence representation for unsupervised analysis of human activities". In: *Artificial Intelligence* 173.14, pp. 1221 –1244. ISSN: 0004-3702. DOI: <https://doi.org/10.1016/j.artint.2009.05.002>. URL: <http://www.sciencedirect.com/science/article/pii/S0004370209000629>.
- Julian Togelius, Renzo De Nardi and Simon M. Lucas (2007). "Towards automatic personalised content creation for racing games". In: *R2007 IEEE Symposium on Computational Intelligence and Games*.
- Lee, G. et al. (2014). "Learning a Super Mario controller from examples of human play". In: *2014 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1–8. DOI: [10.1109/CEC.2014.6900246](https://doi.org/10.1109/CEC.2014.6900246).
- Lueangrueangroj, Sarayut and Vishnu Kotrajaras (2009). "Real-Time Imitation Based Learning for Commercial Fighting Games". In: DOI: [10.5176/978-981-08-3190-5\\_301](https://doi.org/10.5176/978-981-08-3190-5_301).
- Ortega, Juan et al. (2013). "Imitating human playing styles in Super Mario Bros". In: *Entertainment Computing* 4.2, pp. 93 –104. ISSN: 1875-9521. DOI: [https://doi.org/10.1007/978-3-540-74873-1\\_8](https://doi.org/10.1007/978-3-540-74873-1_8).

org/10.1016/j.entcom.2012.10.001. URL: <http://www.sciencedirect.com/science/article/pii/S1875952112000183>.

Thunputtarakul, Worapoj and Vishnu Kotrajaras (2007). "Data Analysis for Ghost AI Creation in Commercial Fighting Games." In: *GAMEON*. Ed. by Marco Rocchetti. EUROSIS, pp. 37–41. ISBN: 978-90-77381-37-3. URL: <http://dblp.uni-trier.de/db/conf/gameon/gameon2007.html#ThunputtarakulK07>.

Van Hoorn, Niels et al. (2009). "Robust Player Imitation Using Multiobjective Evolution". In: *Proceedings of the Eleventh Conference on Congress on Evolutionary Computation*. CEC'09. Trondheim, Norway: IEEE Press, pp. 652–659. ISBN: 978-1-4244-2958-5. URL: <http://dl.acm.org/citation.cfm?id=1689599.1689684>.