Application of Retrograde Analysis on Fighting Games

Kristen Yu
Computer Science Department
University of Denver
Denver, USA
Kristen.Yu@du.edu

Nathan R. Sturtevant
Computer Science Department
University of Alberta
Edmonton, Canada
nathanst@ualberta.ca

***Abstract*—With the advent of the fighting game AI competition, there has been recent interest in two-player fighting games. Monte-Carlo Tree-Search approaches currently dominate the competition, but it is unclear if this is the best approach for all fighting games. In this paper we study the design of two-player fighting games and the consequences of the game design on the types of AI that should be used for playing the game, as well as formally define the state space that fighting games are based on. Additionally, we also characterize how AI can solve the game given a simultaneous action game model, to understand the characteristics of the solved AI and the impact it has on game design.**

I. Introduction

In the fighting game space, there are a few areas of interest that can be addressed, such as creating an AI that will beat any human or computer opponent, creating an AI that is interesting for humans to play against, and scaling the skill of an AI with the skill of a player. Recent work in Fighting Game AI has focused on building strong AI players [1-12] which will beat any opponent, and the current competition has a component which focuses on how to beat opponents as quickly as possible [34]. This raises the question of what it would mean to solve a fighting game, or to build a perfect fighting game AI. The definition of a perfect player depends critically on the definition of the game being played. However, current literature, to our knowledge, does not contain a precise definition of a fighting game, meaning that the notion of building a perfect AI for such games is ill-defined.

Thus, the focus of this paper is threefold. First, the paper builds a broad definition of the state space of a fighting game based on the cross product of finite state machines that determine the possible actions for each player at each stage of the game. Because players have the potential to take actions simultaneously, there are then information sets defined over these states. Next, the paper shows how a game can be solved via retrograde analysis and using Nash equilibria to determine optimal play over the information sets. Finally, the paper builds an optimal strategy for a small fighting AI game and characterizes the solution to the game. Even in a simple game the optimal strategy is complex enough to be non-trivial, such that simple strategies will not perform well against the optimal AI. This work opens the door for deeper study of both optimal and suboptimal play, as well as options related to game design, where the impact of design choices in a game can be studied.

II. Related Work

A variety of approaches have been used for fighting game AI in both industry and academia. One approach in industry has game AI being built using N-grams as a prediction algorithm to help the AI become better at playing the game [30]. Another common technique is using decision trees to model AI behavior [30]. This creates a rule set with which an agent can react to decisions that a player makes. The game designers then create interesting decision trees for every enemy and boss that they want to implement in the game. In Super Smash Bros 4 and Super Smash Bros Ultimate, Nintendo implemented a system where a toy called an Amiibo is trained to play the game. While Nintendo has not officially released the algorithms that these Amiibo are based on, fans have worked out a probable algorithm based on extensive testing. There is an underlying decision tree which governs specific behaviors that would want to be executed under all circumstances, and a learnable section where the AI will learn to favor specific moves over other moves [35]. The probability that a particular action should be used is updated over time in the learnable section to reflect the individual play style of the opponents that the Amiibo plays against. These Amiibos show that a machine learning model can also be effective in learning to play a fighting game, as opposed to just using rule-based AI.

In academia, there are several algorithm classes that have been implemented on fighting game platforms in order to design fighting game AI. At the Conference on Games, there is an annual competition hosted by Ritsumeikan University using a platform called Fighting ICE. In this competition, the AI is playing a fast-paced game where it has a reaction speed of 15 frames at 60 frames per second, and is playing other AI using different strategies to determine a winner with a round-robin tournament. Early work attempted to improve on the existing rule-based AI that are common in published games by implementing a prediction algorithm, K-Nearest Neighbors [1,2,3,4]. These predictions assumed that there is one dominant move at every state in the state space, and

978-1-7281-1884-0/19/$31.00 ©2019 IEEE

aim to predict that single move. Hierarchical Task Networks were also added to the existing rule-based AI as a way of implementing prediction, but was unable to plan effectively [25]. This work was abandoned in favor of Monte Carlo Tree Search (MCTS) algorithms which have consistently placed in the top 3 for the competition [5,6,7,8,9,10]. As such, there has been other research to improve this model by combining it with existing algorithms. Genetic algorithms [11], Action tables [12], and Hierarchical Reinforcement Learning [13] have all been combined with MCTS in attempts to improve on the original algorithm. MCTS works by searching a tree for a set period of time, and returning either the correct answer or the best answer found so far. This algorithm has been validated experimentally to perform well through competing against other AI, but there has been no published work to understand why this algorithm is effective. Another type of algorithm that has been implemented is a multi-policy one, where the rules of the agent is designed to be switched based off of the state parameters [16,17]. These types of algorithms rely on dynamic scripting [23,24] to learn the optimal state with which to execute each individual policy. More complicated algorithms such as genetic algorithms [19], neural networks [20,21,22], and hierarchical reward architecture [26] have all been implemented on the fighting ICE framework and have learned strategies that are stronger than the ”default” rule-based AI.

Other research has been done using AI-TEM framework, which is a Game Boy Emulator that can play a variety of games. The AI reads the components of the chosen fighting game in order to make informed decisions. An agent was trained by learning how a human would play a game [14], in an attempt to learn a viable strategy. A neural network was also trained using human play data [15] on a different game, or the human play data was used to create a finite state machine (FSM) that can be plays to a rule based AI.

III. Fighting Game Design

In its simplest form, all fighting games have these three basic design elements: **Timing**, **Spacing**, and **Effectiveness**. Each of these design elements are parameterized to make up the overall design of the game.

- **Timing** is the duration attributed to the actions that can be executed in the game. This is split into a few categories: lead time, attack duration, lag time, and input time. The input time is the amount of time required for a player to input a move, which is generally shorter than the other timings. Lead time is the time before the attack is executed. The attack duration is the amount of time that an attack has damaging potential. Lag time is the amount of time after a move has been executed that a player is unable to input another move.
- **Spacing** is the distances that are important in the game. Both moves and characters have spacing. Each character has two different dimensions that are important: the hurtbox and the hitbox [54]. The hurtbox is the area that the opponent can attack without taking damage, and the hit box is the area during an attack that can inflict damage to the opponent. Move spacing can be split into two categories: The distance in which a particular action will hit an opposing character, and the area of its hitbox. Similarly, the character spacing can be split into the area of its hurtbox and it’s individual x and y coordinates.
- **Effectiveness** is the performance of a given action. Usually this is the amount of damage assigned to an action, but it can also be affected by its relation to other moves. Suppose there is a block mechanic in the game. If player one blocks and player two attacks, player two’s move can be partially or fully blocked, which decreases the effectiveness of the move.

A. Game Play Mechanics

This paper will analyze the mechanics and dynamics from a Mechanics-Dynamics-Aesthetics (MDA) perspective [31]. The mechanics of any fighting game can be modeled by a (FSM) which determines the outcomes of a players controls for a given state. Each FSM f has a time t associated with it, and list of available actions that the agent can take $m_1...m_k$, where k is finite. This FSM has a starting state f_0 where the player is idle, as shown in Figure 1. When a player selects an action m_i, $i \in \{1...k\}$, the state of the machine transitions to the start of the chosen move, with a lead time of t_{lead}, continues to the action which has an attack duration of t_a, and ends with lag time t_{lag} before returning to idle. Because of how the FSM transitions, once the player has chosen a move, that move typically must be executed to completion before another move can be attempted.

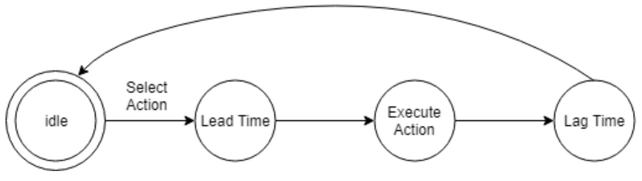


Fig. 1. FSA that governs the players actions

Each player has an individual FSM that is controlling the state in which that player exists. By combining the two FSM, we can reach a state $s_g \in FSM_{P_1} \times FSM_{P_2}$. For the full state of the game, s_g also has attributes $a_1...a_n$, where n is finite, of the current state such as the health and the spacing of the individual players, which will indicate whether an action does damage against the opposing player. If player one successfully attacks player two while player two is not at an idle state, then the FSM for player two will automatically either transition to idle or to lag time, preventing it from executing the remaining part of the machine.

Each player has an individual FSM that is controlling the state in which that player exists. By combining the two FSM, we can reach a state $s_g \in FSM_{P_1} \times FSMP_2$. For the full state of the game, s_g also has the attributes $a_1...a_n$ of the current state such as the health and the spacing of the individual players, which will indicate whether an action does damage against the opposing player. This state s_g transitions to the next state s'_g through both the action of player one and the action of player 2. This transition is deterministic and results in a unique time, location, and states for each attribute, which is determined by the outcome of the interaction of the actions that each player used. If player one successfully attacks player two while player two is not at an idle state, then the FSM for player two will automatically either transition idle or to lag time, preventing it from executing the remaining part of the machine.

B. Game Play Dynamics

Assume both player one and player two have one attack, m_1, with lead time $t_{lead} = k_1$, and their spacing is such that if either player executes the move, they will do damage to their opponent. If player one is in the idle state and player two chooses to execute m_1, then the attack m_1 will be successful and player one will take damage. If both players are in idle, the player that executes m_1 first will do damage. In this situation, whichever player can execute the action first will win.

However, fighting games usually have more than one move, with a variety of lead times. Assume that player one and player two both have action m_1, but now also have action m_2 with lead time $t_{lead} = k_2$, where $k_1 < k_2$. If player one were to execute m_2, then there is a period of time, $k_2 - k_1$, where player two can execute m_1 and counter m_2. Using this model, the agent would simply have to wait for the opponent to select a move, and then respond accordingly.

To simplify the game dynamics, assume that the FSM has no lead or lag time, so that there is a single state with which the FSM can transition to from idle. Additionally, restrict the timing of the game such that each player has to execute the game, so then it becomes purely a simultaneous move game. This can be modeled as Rock Paper Scissors (RPS) because it shares the quality of what fighting game expert David Sirlin calls ”double-blind decisions” [28]. Double-blind decisions are created in a situation where both players are making a decision at the same time, and both of their decisions are revealed at the same time. Double-blind decisions, Sirlin argues, fully capture the essence of fighting games because at the moment a player chooses a move, they do not know exactly what the opponent is doing due to the fast-paced nature of the game. The opponent’s move can be modeled using information sets. An information set is the set of all the states that cannot be distinguished by a player because the full state of its opponent is unknown, and a player must make the same action in every state in an information set. Figure 2 shows a game play tree which creates an information set. The root is the state of player one, and the edges are the different actions available to player one, punch, kick, or idle, which can be taken from that state. Each child of the root represent the state of player two, where player two can also execute actions punch, kick, or idle at that state. Once both players have selected a move, a reward is assigned at a depth of two, and the reward is dependent on both the action from player one and the action from player two. The information set is the set of all nodes at depth one, and is outlined by the box in the figure. This set represents the unknown action of player one when player two must select an action.

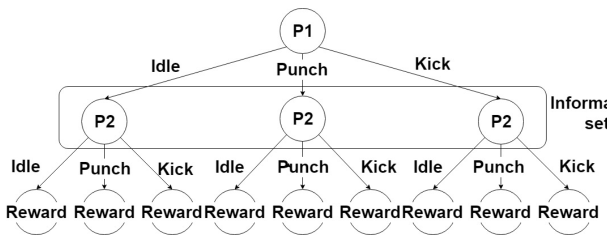


Fig. 2. Information set created by the simultaneous move model

A simultaneous move game can be modeled as a matrix game which maps the n actions $\{a_1, a_2, ..., a_n\}$ of each individual players to their rewards r_i. Each player has its own reward matrix $M = n \times n$, which gives the instantaneous reward r_{ij} of player one taking action i and player two taking action j at M_{ij}. The payoff for a given player R_k, $k \in \{1, 2\}$, is the total instantaneous reward accumulated by playing some strategy. The optimal policy π_1 of player one is a policy that maximizes the payoff of player one given all possible player two strategies. However, player two also has an optimal policy it is trying to execute, so player one wants to minimize the reward that player two gains simultaneously. Thus the optimal policy for player one can be defined as $max_{R_1} min_{R_2} \sum r_{ij}$. RPS can be modeled as a matrix game M_1 by assigning values to the outcomes of player one, as shown in Table 1. If the result of an action is a win, 1 is assigned. If the result of an action is a loss then -1 is assigned. If the result of an action is a tie then 0 is assigned. If the reward assigned for some set of actions a_i and a_j for player two is the negative value of the reward of the reward assigned for the same set for player one, then it is a zero sum matrix game. In that case, the reward matrix for player two is $M_2 = -M_1$.

	P2 Rock	P2 Paper	P2 Scissors
P1 Rock	0	-1	1
P1 Paper	1	0	-1
P1 Scissors	-1	1	0

TABLE I
REWARD MATRIX FOR PLAYER 1 IN ROCK PAPER SCISSORS

By replacing the moves of RPS with moves from a fighting game, a similar reward matrix can be generated. Let move a and move b be attacking moves, which have different effectiveness based on the opponent’s move. Move a will inflict damage against a block, but not against move b. Move b will not inflict damage against a block, but will against move a. Figure 3 shows the relationship between these moves, where the arrow pointing to another move indicates that the move will ”win” and inflict damage against that move.

If the winner of the game is determined only by one action, then the factors that influence whether a player wins are the

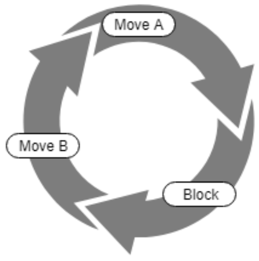
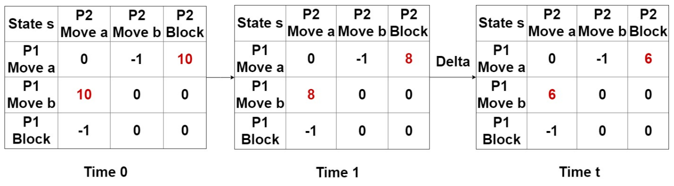


Fig. 3. Effectiveness of moves A, B, and Block in relation to each other



State s	P2 Move a	P2 Move b	P2 Block
P1 Move a	0	-1	10
P1 Move b	10	0	0
P1 Block	-1	0	0

Time 0

State s	P2 Move a	P2 Move b	P2 Block
P1 Move a	0	-1	8
P1 Move b	8	0	0
P1 Block	-1	0	0

Time 1

State s	P2 Move a	P2 Move b	P2 Block
P1 Move a	0	-1	6
P1 Move b	6	0	0
P1 Block	-1	0	0

Time t

Fig. 4. Illustration of reward being propagated through time

spacing between the two players and the effectiveness of the move. If the spacing is such that the two players are out of attack distance, then the entire reward matrix would be zero. However, if the spacing between the two players is such that they are within attack distance of each other, then the moves have the possibility of doing damage and generating an interesting reward matrix. The diagonal of this matrix is zero because both players would successfully execute the move, in which case the reward would be 1 + -1 = 0.

A zero sum payoff matrix can be solved using linear programming which finds the policies π_1 and π_2, for player one and player two respectively. These policies are the set of probabilities with which each move in the game should be used, which forms the strategy for each player. An optimal strategy for a player is one that maximizes its own payoff and minimizes its opponent payoff. In practice, this means that the player cannot unilaterally change their policy to gain more reward, provided that their opponent does not change their strategy. If both players are playing with their optimal policies, it is a Nash Equilibrium. Using the rock paper scissors analogy, if both players are attempting to play optimally, the optimal strategy for player one is to use each move 1/3 of the time, and the optimal strategy for player two is to use each move 1/3 of the time.

Timing still needs to be considered in the fighting game model. Time is not infinite, because there is a time limit imposed on the game itself. Thus, there are two ending conditions for a match, when the timer reaches zero or if one of the player's health becomes 0. If the player's health becomes 0, we can introduce a higher reward value for that move, which incentivizes the agent to win the game. For every state s, each time step has its own reward matrix, and the individual reward matrices for each state are linked through time. The time in which that state has a higher reward has an effect on the future reward matrices, because that high reward is propagated through all future times, as shown in Figure 4. The value in red at time 0 is the initially high reward, and gets propagated to time 1. At the final reward matrix at time t, the final reward is still affected by the initial high reward. The rewards in the matrix are then influenced both by the positions of the players, and the rewards of the previous matrices.

C. Game Balance

Balancing the game is adjusting the different game play mechanics such that they interact in a way that produces the desired outcomes. For fighting games, usually the goal is to prevent a dominating character or strategy from appearing. This can be difficult because there is an inherent level of uncertainty integrated into the gameplay mechanics. While a player is playing the game, they do not know which move the opponent will pick next, which makes the player have to predict what the opponent's move will be. Sirlin describes the ability to predict the opponent's next move as "Yomi" [28,29]. When the opponent's move is uncertain, the player engages in Yomi because they are forced to make their best guess on what move the opponent will make, and then respond appropriately given that prediction. Sirlin argues that using a rock paper scissors like mechanic for the core of action resolution is the optimal way to balance fighting games, because it forces Yomi interactions and prevents any single move from becoming dominant.

Aside from properly balancing the individual game mechanics, fighting games also need to balance the level of skill. Each move is assigned a certain number of inputs that the player must make in the correct order in order to execute that move. These can be simple - such as simply pressing the A button - to extremely complex - such as quarter circle right, quarter circle right, A button, B button. This skill affects the game play because a higher difficulty level would create a higher chance of the human incorrectly inputting the move, which would trigger a random move being executed. A computer has a the ability to correctly input any move at any state, while a human will always have some probability of incorrectly inputting a move.

IV. Solving a Fighting Game

An analysis of how MCTS performs in RPS by Shafiei [33] demonstrated it produces suboptimal play because it cannot randomize correctly for a RPS type of game play model. In a simultaneous move fighting game, the time limit ending condition allows for retrograde analysis to be used to work backwards from the end of the game to the start state. Bellman showed that this technique can be applied to chess and checkers [32] using minimax trees where the max or the min is taken at each state and propagated through the tree. With imperfect information at each state, we instead compute the Nash Equilibrium at each state and propagate the expected value of the Nash Equilibrium. This approach is based on work done by Littman and Hu, which applied reinforcement learning algorithms to markov games to create a nash equillibrium [36,37].

Using a retrograde analysis allows for a quick convergence of the state space. Without it, there would be a need for multiple value-iteration passes to propagate the correct values, which is more computationally expensive. These information sets that appear as matrix games for each state in the retrograde analysis can be solved using linear programming to find the optimal policy and create a Nash Equilibrium.

A. Rumble Fish

The fighting game chosen for the computational model is a game called Rumble Fish, similar to Street Fighter, and is also the game that is used in the Fighting ICE competition. It takes in eight directional inputs and two buttons, which can be executed in sequence to produce moves. These moves can then be strung together into combos to deal more damage. While playing, the character can build up mana, which allows them to execute stronger moves such as projectiles that cost mana.

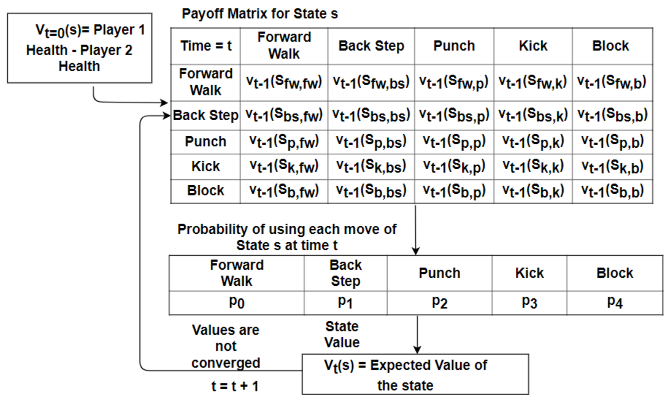
The game was simplified in several ways to solve the game quickly and understand the impact of the game and solver design on the optimal strategy. Only one character was chosen, so that the same character would be playing against itself. This was to eliminate any potential tier differences between characters, where one character could have dominating features and could always win the game. The width of the available space was also reduced, and the jumping and mana mechanics were eliminated. Only five moves were chosen to begin with:

- Move Forward: Move towards the other player by 25 pixels
- Move Backward: Move away from the other player by 120 pixels
- Punch: A short range attack with a damage of 5
- Kick: A long range attack with a damage of 10
- Block: A block will only affect the kick, which reduces the given damage to 0. The block is ineffective against punches and the player will still take 5 damage when punched

The block was modified to introduce an RPS mechanic. Otherwise, the kick becomes the dominating strategy because it has the longest reach and the highest damage. The health was also restricted to be 20, and all distances between 0 and 170 pixels were examined. The maximum distance of 170 was chosen because the kick has a range of 140 pixels. All distances larger than 170 are treated as the state 170 because if the players are out of attack distance and moving forward does not bring you into attack range, it can be treated in the same way. If a punch or a kick is successful, the opposing player will be pushed back. However, if the opposing player was blocking then they will not move. The length of the game was not fixed, but the retrograde procedure in the next section was run backwards until the policy converged.

B. Retrograde Analysis

Retrograde analysis is more efficient than forward search for this state space. Time starts at the end of the match, and then runs backwards to the start of the match. The flow through for this algorithm is shown in Figure 5. Moves Forward Walk, Back Step, Punch, Kick, and Block have been abbreviated ”fw”, ”bs”, ”p”, ”k”, and ”b” respectively. This figure shows the payoff matrix for player one for any given state s, where the values at each index i, j is the value of the state at the previous time step when actions a_i and a_j are chosen. When the time is zero, the box on the left indicates the the value of the state is the difference in health for player 1 and player 2. When the time is not zero, the value of the state is calculated differently. The first down arrow shows that the policy for player one is determined, which is then used to find the expected value of that state. This new state value then feeds back into the payoff matrix for the next state. A more precise description of the approach is described below.



$V_{t=0}(s)$= Player 1 Health - Player 2 Health
Payoff Matrix for State s
Time = t | Forward Walk | Back Step | Punch | Kick | Block
Forward Walk | $v_{t-1}(s_{fw,fw})$ | $v_{t-1}(s_{fw,bs})$ | $v_{t-1}(s_{fw,p})$ | $v_{t-1}(s_{fw,k})$ | $v_{t-1}(s_{fw,b})$
Back Step | $v_{t-1}(s_{bs,fw})$ | $v_{t-1}(s_{bs,bs})$ | $v_{t-1}(s_{bs,p})$ | $v_{t-1}(s_{bs,k})$ | $v_{t-1}(s_{bs,b})$
Punch | $v_{t-1}(s_{p,fw})$ | $v_{t-1}(s_{p,bs})$ | $v_{t-1}(s_{p,p})$ | $v_{t-1}(s_{p,k})$ | $v_{t-1}(s_{p,b})$
Kick | $v_{t-1}(s_{k,fw})$ | $v_{t-1}(s_{k,bs})$ | $v_{t-1}(s_{k,p})$ | $v_{t-1}(s_{k,k})$ | $v_{t-1}(s_{k,b})$
Block | $v_{t-1}(s_{b,fw})$ | $v_{t-1}(s_{b,bs})$ | $v_{t-1}(s_{b,p})$ | $v_{t-1}(s_{b,k})$ | $v_{t-1}(s_{b,b})$
Probability of using each move of State s at time t
Forward Walk | Back Step | Punch | Kick | Block
p_0 | p_1 | p_2 | p_3 | p_4
Values are not converged
t = t + 1
State Value
$V_t(s)$ = Expected Value of the state

Fig. 5. The flow through for the modified retrograde analysis algorithm when applied to fighting games

Each time step t is one frame backwards in time and the analysis is run until the beginning of the match is reached. At time $t = 0$, the value $v_0(s)$ of each state $s \in S$ was initialized to be the difference in health from player one to player two. Each possible health value for player one and player two were considered, to account for the situation where time runs out. If the state s was a winning state for player one, meaning that player two's health is zero and player one's health is nonzero, then the value of the state was increased by 100 because those are the most desirable states for player one to be in. The reward for losing is not decreased by 100, in order to encourage the agent to continue attacking even if it is at low health.

To calculate the value of state s at time $t > 0$, a 5x5 reward matrix is created where player one is represented by the rows and player two is represented by the columns. All of the rewards assigned to each outcome is the reward for player one. The rewards for player two are the negative value of each reward, which can be calculated without creating another matrix. Given a set of five actions $A = \{$Forward Walk, Back Step, Punch, Kick, Block$\}$, each row i and each column j is a selection $a \in A$, where a_i is the action for player one and a_j is the action for player two. The next state s' is the state reached by executing actions a_i and a_j at state s. The intersection i x j is the reward of the moves a_i and a_j that were chosen at state s, which is the value of the state $v_{t-1}(s')$. This matrix is then solved using linear programming to produce π_1 and

π_2, which are the policies of player one and player two. Each policy represents the probabilities that each action in $a \in A$ should be executed for state s at Nash Equilibrium. The value of state s is the expected value of that state calculated as follows:

$$v_t(s) = \sum_{i=1}^{5}\sum_{j=1}^{5} p_{a_i} p_{a_j} v_{t-1}(s') \quad (1)$$

This value $v_t(s)$ is stored for the time t. When two actions lead to s at time $t+1$, the value $v_t(s)$ is used as the reward for those actions. In this way, the state values will propagate through the time steps to produce the final state. The iteration process is monitored for convergence for all states s using a δ, where $\delta = V_t(s) - V_{t-1}(s)$. When $\delta < 0.01$, the values are considered converged.

V. Results and Discussion

To understand the characteristics of the Nash Equilibrium model, a few simple rule-based AI were designed to play the game.

- Random: The agent will randomly select an action from the available move set with equal probability
- Punch: The agent will always punch
- Kick: The agent will always kick
- Rules: The agent will punch when it is at a distance where a punch will cause damage, kick when it is at a distance where the kick will cause damage, and otherwise walk forward

The AI's were then initialized to the starting conditions of the game. They both have 20 health, and all starting distances 0 to 170 were examined, where the starting distance is the initial spacing on the x-axis between the two players and is measured in pixels. The maximum distance of 170 was chosen because the maximum range of the kick attack is 140. This leaves 30 pixels of space where neither player has the ability to do damage. This is sufficient for the player to learn any policies where it is out of range of an attack, and all distances greater than 170 can be modeled as distance 170. Player one is always the Nash Player, which is the policy created by the retrograde analysis as described in the previous section. Then, there are four possible outcomes for the Nash player, player one: win, lose, tie, and run away. A win is the situation where player two's health is zero. A lose is when player one's health is zero. A tie is when both player's health is zero. Run away is the situation where both agents choose to move out of range of the attacks, and will never move forward, so at the end of the game both of the players still have health. Each of these scenarios were run in three trials, to account for randomized play.

Figure 6 shows the results of the Nash player. None of the simple strategies are able to beat this player in any of the simulated situations. The random player is the best strategy because it minimizes the amount of wins for the Nash player. The Nash player is playing most similarly to the Rules AI, because it results in the most number of ties. However, the Rules AI still loses to the Baseline Nash 95 times, so the Nash player is still a better strategy.

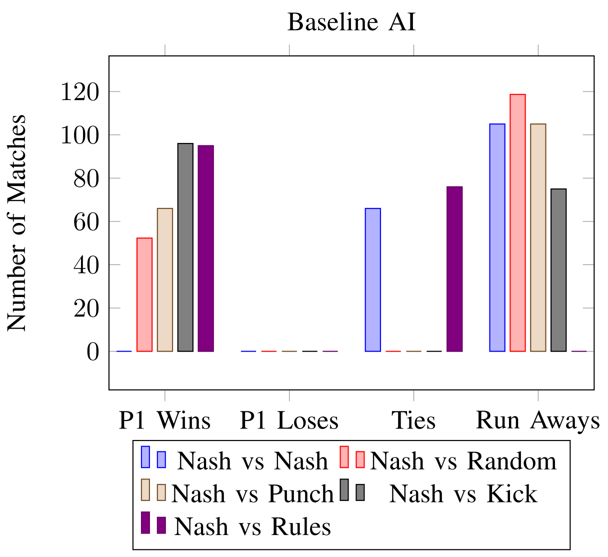


Fig. 6. Results from the Nash AI

The total number of moves selected by the Nash agent while playing these matches was compared to the number of moves where it had a mixed strategy, shown in Table 2. This was done to determine if the Nash player used a deterministic strategy or not. The total number of moves for every match and the total number of moves where the Nash AI selected a mixed strategy was recorded. The percentage of mixed moves was then calculated as the number of mixed strategy moves divided by the total number of moves for the match. The percentage of moves where the Nash player was picking a mixed strategy is very low, at only 2.07%. This means that the Nash player is playing a deterministic strategy in most of the states, even though the game was designed to contain RPS mechanics and lead to mixed strategies. In the case of RPS, if one player is playing at Nash Equilibrium, the other player can use any strategy and still arrive at a Nash Equilibrium. In the fighting game, this is not the case given the Nash player is able to win more than the other strategies

Total Number of Actions	Total Number of Mixed Action	Percentage of Mixed Actions
8637	197	2.07%

TABLE II
Percentage of Mixed Actions out of Total Actions

Some variations on the retrograde analysis were created to model additional behavior such as imperfect human input [39]. These variations had no effect on the overall performance of the Nash player, and had no effect on the percentage of mixed strategies the Nash player chose.

A. *Custom Game*

The results from this simplified version of Rumble Fish were not as interesting as expected due to the presence of a dominating strategy within the game. With this in mind, a custom fighting game was designed to address these issues and create a Nash player that has a mixed strategy. Six moves, 4 attacks and 2 blocks, and two distances were chosen to build the state space, and the effectiveness of the moves were set up such that at each state there was always two of the attacks were always dominated actions. The remaining two attacks and a block were set up in an RPS like wheel, where each action had a specific counter. The remaining block was set to only be effective against some of the attacks, but not all of the attacks. The results from this custom game produced a Nash Equilibrium where it is always beneficial to mix between the actions of the RPS wheel, and never choose a dominated move. The introduction of the dominated actions to the state space was able to successfully create a ”bad” move, which if chosen would produce sub-optimal play.

VI. Future Work

Future work on this project would focus on three main areas, the first of which is scale to larger games. One of the ways to accomplish this is to expand the number of moves available to the agent. Most fighting games have a large variety of moves, more than are currently modeled, so more actions should be incorporated. Another way to increase the size of the game is to increase the range of distances. Each distance value has its own unique effectiveness wheel, which dictates which moves are dominant to other moves in an RPS-like manner.Adding these features to the current model causes for the custom fighting game to become more similar to published fighting games, which allows for a deeper understanding of the design choices on the overall balance and play of the game.

Another consideration is the restrictions on how the players can take actions. In the custom fighting game, each action has the same lead time, attack duration, and lag time. First, the overall attack times of each move can be varied, such that each available action is a multiple of the action that has the shortest attack time. Still restricting to mostly simultaneous moves, one player would then be allowed to choose 1 long action while the other player can choose 2 or more fast actions. Then, the different lead and lag times could be incorporated into the simultaneous move model, so that particular moves can dominate other based not only on the effectiveness of the action but also on timing. Once this model has been built, we will have a comprehensive understanding of the full range of factors that affect actions and the consequences of each action within the game.

References

[1] K. Asayama, K. Moriyama, K. Fukui, and M. Numao, ”Prediction as Faster Perception in a Real-time Fighting Video Game,” Proc. the 2015 IEEE Conference on Computational Intelligence and Games (CIG 2015), pp. 517-522, 2015.

[2] Yuto Nakagawa, Kaito Yamamoto, Chu Chun Yin, Tomohiro Harada, and Ruck Thawonmas, ”Predicting the Opponent’s Action Using the k-Nearest Neighbor Algorithm and a Substring Tree Structure,” Proc. of the 2015 IEEE 4th Global Conference on Consumer Electronics (GCCE 2015), Osaka, Japan, pp. 531-534, Oct. 27-30, 2015.

[3] Yuto Nakagawa, Kaito Yamamoto, and Ruck Thawonmas, ”Online Adjustment of the AI’s Strength in a Fighting Game Using the k-Nearest Neighbor Algorithm and a Game Simulator,” Proc. of the 3rd IEEE Global Conference on Consumer Electronics (GCCE 2014), Tokyo, Japan, pp. 494-495, Oct. 7-10, 2014.

[4] Kaito Yamamoto, Syunsuke Mizuno, Chun Yin Chu and Ruck Thawonmas, ”Deduction of Fighting-Game Countermeasures Using the k-Nearest Neighbor Algorithm and a Game Simulator,” Proc. of 2014 IEEE Conference on Computational Intelligence and Games (CIG 2014), Dortmund, Germany, pp. 437-441, Aug. 26-29, 2014.

[5] Shubu Yoshida, Makoto Ishihara, Taichi Miyazaki, Yuto Nakagawa, Tomohiro Harada, and Ruck Thawonmas, ”Application of Monte-Carlo Tree Search in a Fighting Game AI,” Proc. of the 5th IEEE Global Conference on Consumer Electronics (GCCE 2016), Kyoto, Japan, pp. 623-624, Oct. 11-14, 2016.

[6] Makoto Ishihara, Taichi Miyazaki, Chun Yin Chu, Tomohiro Harada, and Ruck Thawonmas, ”Applying and Improving Monte-Carlo Tree Search in a Fighting Game AI,” Proc. of the 13th International Conference on Advances in Computer Entertainment Technology (ACE 2016), Osaka, Japan, Nov. 9-12, 2016. DOI: http://dx.doi.org/10.1145/3001773.3001797

[7] Makoto Ishihara, Suguru Ito, Ryota Ishii, Tomohiro Harada and Ruck Thawonmas, ”Monte Carlo Tree Search for Implementation of Dynamic Difficulty Adjustment Fighting Game AIs Having Believable Behaviors,” Proc. of 2018 IEEE Conference on Computational Intelligence and Games (CIG 2018), Maastricht, The Netherlands, pp. 46-53, Aug. 14-17. 2018.

[8] Ryota Ishii, Suguru Ito, Makoto Ishihara, Tomohiro Harada and Ruck Thawonmas, ”Monte-Carlo Tree Search Implementation of Fighting Game AIs Having Personas,” Proc. of 2018 IEEE Conference on Computational Intelligence and Games (CIG 2018), Maastricht, The Netherlands, pp. 54-61, Aug. 14-17. 2018.

[9] Simon Demediuk, Marco Tamassia, William Raffe, Fabio Zambetta, Xiaodong Li and Florian Floyd Mueller, ”Monte Carlo Tree Search Based Algorithms for Dynamic Difficulty Adjustment,” Proc. of 2017 IEEE Conference on Computational Intelligence and Games (CIG 2017), New York City, USA, Aug. 22-25, 2017.

[10] Suguru Ito, Makoto Ishihara, Marco Tamassia, Tomohiro Harada, Ruck Thawonmas, and Fabio Zambetta, ”Procedural Play Generation According to Play Arcs Using Monte-Carlo Tree Search,” Proc. of the 18th International Conference on Intelligent Games and Simulation (GAME-ON’2017), Carlow, Ireland, pp. 67-71, Sep. 6-8, 2017.

[11] Man-Je Kim and Chang Wook Ahn, "Hybrid fighting game AI using a genetic algorithm and Monte Carlo tree search," GECCO' 18 Proc. of the Genetic and Evolutionary Computation Conference Companion, Kyoto, Japan, pp. 129-130, July 15 - 19, 2018.

[12] Man-Je Kim and Kyung-Joong Kim, "Opponent Modeling based on Action Table for MCTS-based Fighting Game AI," Proc. of 2017 IEEE Conference on Computational Intelligence and Games (CIG 2017), New York City, USA, Aug. 22-25, 2017.

[13] Ivan Pereira Pinto and Luciano Reis Coutinho, "Hierarchical Reinforcement Learning with Monte Carlo Tree Search in Computer Fighting Game," IEEE Transactions on Games (Early Access, Date of Publication: 11 June 2018), doi:10.1109/TG.2018.2846028

[14] S, Lueangrueangroj and V. Kotrajaras, "Real-time Imitation based Learning for Commercial Fighting Games," Proc. of Computer Games, Multimedia and Allied Technology 09, International Conference and Industry Symposium on Computer Games, Animation, Multimedia, IPTV, Edutainment and IT Security, 2009.

[15] S.S. Saini, C.W. Dawson, and P.W.H. Chung, "Mimicking Player Strategies in Fighting Games," Proc. of the 2011 IEEE International Games Innovation Conference (IGIC), pp. 44-47, 2011.

[16] S. Saini, P. W. H. Chung and C. W. Dawson, "Mimicking Human Strategies in Fighting Games using a Data Driven Finite State Machine," 2011 6th IEEE Joint International Information Technology and Artificial Intelligence Conference, vol 2, P. Yan and B. Xu, Eds. Chongqing: IEEE Press, pp. 389-393, 2011.

[17] N. Sato, S. Temsiririkkul, S. Sone. and K. Ikeda, "Adaptive Fighting Game Computer Player by Switching Multiple Rule-based Controllers," Proc. of the 3rd International Conferenceon Applied Computing and Information Technology (ACIT 2015), pp. 52-59, 2015.

[18] Yoshina Takano, Suguru Ito, Tomohiro Harada, Ruck Thawonmas, "Utilizing Multiple Agents for Decision Making in a Fighting Game," Proc. of the 7th IEEE Global Conference on Consumer Electronics (GCCE 2018), Nara, Japan, pp. 562-563, Oct. 9-11, 2018.

[19] Giovanna Martinez-Arellano, Richard Cant and David Woods, "Creating AI Characters for Fighting Games using Genetic Programming," IEEE Transactions on Computational Intelligence and AI in Games, Dec. 2016.

[20] Nguyen Duc Tang Tri, Vu Quang and Kokolo Ikeda, "Optimized Non-visual Information for Deep Neural Network in Fighting Game," Proc. of 9th International Conference on Agents and Artificial Intelligence(ICAART 2017), Porto, Portugal, pp. 676-680, Feb. 2017.

[21] B.H. Cho, S.H. Jung, Y.R. Seong, and H.R. Oh, "Exploiting Intelligence in Fighting Action Games using Neural Networks," IEICE Transactions on Information and Systems, vol. E89-D, no. 3, pp. 1249-1256, 2006.

[22] Seonghun Yoon and Kyung-Joong Kim, "Deep Q Networks for Visual Fighting Game AI," Proc. of 2017 IEEE Conference on Computational Intelligence and Games (CIG 2017), New York City, USA, Aug. 22-25, 2017.

[23] K. Majchrzak, J. Quadflieg, and G. Rudolph, "Advanced Dynamic Scripting for Fighting Game AI," Proc. of Entertainment Computing (ICEC 2015), pp. 86-99, 2015.

[24] Yasutomo Kanetsuki, Ruck Thawonmas, and Susumu Nakata, "Optimization and Simplification of Dynamic Scripting with Evolution Strategy and Fuzzy Control in a Fighting Game AI," Proc. of the 2015 IEEE 4th Global Conference on Consumer Electronics (GCCE 2015), Osaka, Japan, pp. 310-311, Oct. 27-30, 2015.

[25] Xenija Neufeld, Sanaz Mostaghim, and Diego Perez-Liebana, "HTN fighter: Planning in a highly-dynamic game," Proc. of 2017 Computer Science and Electronic Engineering (CEEC 2017), Colchester, UK, pp. 189-194, Sep. 2017.

[26] Yoshina Takano, Wenwen Ouyangy, Suguru Ito, Tomohiro Harada and Ruck Thawonmas, "Applying Hybrid Reward Architecture to a Fighting Game AI," Proc. of 2018 IEEE Conference on Computational Intelligence and Games (CIG 2018), Maastricht, The Netherlands, pp. 431-436, Aug. 14-17. 2018.

[28] Sirlin, David. "Designing Yomi." Sirlin.Net - Game Design, Sirlin.Net - Game Design, 22 Aug. 2014, www.sirlin.net/articles/designing-yomi.

[29] Sirlin, David. "7) Spies of the Mind." Sirlin.Net - Game Design, Sirlin.Net - Game Design, 3 Aug. 2014, www.sirlin.net/ptw-book/7-spies-of-the-mind.

[30] Millington, Ian, and John Funge. Artificial Intelligence for Games. 2nd ed., Morgan Kaufmann Publishers, 2009.

[31] Hunicke, Robin, et al. "MDA: A Formal Approach to Game Design and Game Research." Conference on Artificial Intelligence, 2004.

[32] Bellman, R. "On The Application Of Dynamic Programing To The Determination Of Optimal Play In Chess And Checkers." Proceedings of the National Academy of Sciences, vol. 53, no. 2, 1965, pp. 244247., doi:10.1073/pnas.53.2.244.

[33] Shafiei, Mohammad, et al. "Comparing UCT versus CFR in Simultaneous Games." Comparing UCT versus CFR in Simultaneous Games, 2009.

[34] "Fighting Game AI Competition." Welcome to Fighting Game AI Competition, 2013, www.ice.ci.ritsumei.ac.jp/ ftgaic/.

[35] Whoa1Whoa1. "r/Smashbros - AMIIBO INFORMATIONAL - They Do Not Learn the Way You Think They Do, Dispelling the Magic of How They Work ." Reddit, 16 Dec. 2014, www.reddit.com/r/smashbros/comments/2pi7zr/amiibo_informational_they_do_not_learn_the_way/.

[36] M.L. Littman: Markov games as a framework for multi-agent reinforcement learning, Proc. ICML, 157-163, 1994.

[37] J. Hu and M.P. Wellman: Nash Q-Learning for General-Sum Stochastic Games, Journal of Machine Learning Research, 4:1039-1069, 2003.

[38] Kristen Yu, "Application of Retrograde Analysis on Fighting Games" Masters Thesis, University of Denver, Jun 2019.