Agent Policy Adaption in Fighting Games using Reinforcement Learning

Gavan Keane[*]
Trinity College Dublin
College Green, Dublin 2
gakeane@tcd.ie

ABSTRACT
The realm of game playing provides a useful framework for research into artificial intelligence. In this paper we take a look at the FightingICE game engine which was designed as a platform for testing fighting game based AIs and also hosting the annual fighting AI tournament. The environment of a fighting game is different to that of a game such as chess in the sense that the environmental state space is much larger. In general the environment of a fighting game is only partially observable and usually there will be no objectively 'best' action. The result of this is that reactive agents (agents with a fixed policy) will be less effective in fighting games than they would be in combinatorial games such as chess [6]. In this paper we propose the use of a reinforcement Learning based agent which will be able to adapt its policy if a chosen action is found to be ineffective.

Keywords
Artificial Intelligence, Reinforcement Learning, Bellman Equations, Fighting Game

1 Introduction
The concept of an artificial intelligence is not recent. The phrase was first coined by John McCarthy in 1955 as a system which would perceive its environment and hence, choose actions to maximize its success [5]. Over the past sixty years there have been many attempts to define, exactly, what an intelligent agent is, this has lead to an equally large number of architectures which aim to implement such an agent. In this paper we will define three broad categories of AI agents. These categories are reactive agents, learning agents and predictive agents. Reactive agents are the simplest form of AI and focus primarily on the development of static strategies. Here a set of actions are mapped to a set of environmental states by a fixed rule base so as to maximise the probability of victory. This type of agent is most effect in combinatorial games such as chess where there is a definite best move which can be mathematically determined [6]. Learning agents also contain an action set and an environmental state set. The difference when compared to reactive agents is that the rule base which maps an action to an environmental state isn't fixed.Instead the rule base (policy) is adapted through some form of optimisation algorithm based on the agent's experiences. The third type of agent, the predictive agent, uses its knowledge of how its actions affect the environment to plan out a strategy to obtain a specific goal. This knowledge can be pre-programmed into the agent, but more commonly it has to be learned through repeated trials similar to how a human player learns which strategies are most effective.

The majority of AI agents used in modern gaming are reactive agents. These agents have a fixed strategy that they will not deviate from even if it has been proven ineffective. While these algorithms computationally efficient, simple to implement and can often fight effectively, they tend to become repetitive and transparent to human players [8]. This often allows for simple exploits by recreating conditions that cause the AI to take an ineffectual action. This sort of exploit is particularly prominent in fighting games, but also in many open world games. A common exploit in such games is to get the AI stuck on some environmental object. To prevent this from happening it would be useful for the AI to be able to learn and adapt its policy based on the circumstances. This would prevent it from being repeatedly caught by the same exploit [2].

In this paper we propose a reinforcement learning AI which uses the Bellman equations to develop an optimal rule base (policy) which can defect simple reactive agents in a fighting game. Three reactive agents are proposed which model the most common strategies adopted by human players within fighting games. The fighting game platform we use in this report is the Fighting ICE game engine [3] which was developed as a testbed for various fighting game agents. The original aim of the agent developed in this paper was to be entered entered into the fighting game AI tournament [3]. However the way the competition is set up makes it difficult for an agent that requires multiple training matches to be competitive.

2 Theory and Method
This section descries the platform used to conduct our research, specifies the winning conditions, the type of environment and the reasons this platform was chosen. We also cover the design of both the Learning AI agent and the reactive agents designed to test it. Finally we cover the theory

[*]MAI undergraduate at Trinity College Dublin

behind the reinforcement learning model used to implement the learning AI agent.

2.1 Fighting ICE Game Engine

The fighting ICE platform [3] is a game engine developed specifically for the fighting AI tournament. The purpose of the platform is to allow for easy development of AI agents within a fighting game environment. The platform is a 2D fighting game in which 2 characters engage in one on one combat. Each match consists of three rounds, with each round lasting sixty seconds. For each round the players split a score of 1,000 points so that a score of over 500 points indicates a win for that round. The characterâĂŹs score is based on how much damage it takes compared to how much damage the opponent took. The exact score is calculated using the equation below, note that a characterâĂŹs hit points start at zero and decrease each time they take a hit.

$$MyScore = \frac{OpponentHP}{MyHP + OpponentHP} . 1000 \qquad (1)$$

As well as their hit points, each character also has energy points. Energy points are gained whenever the character takes or deals damage and are consumed when the character uses a special move.

The Fighting ICE platform consists of three characters, each with a unique set of actions. A character's actions allow it to change its position by moving, crouching and jumping. Allow it to attack at close range and with projectiles and allow it to defend against incoming attacks. Every action a character can use takes a certain number of frames to 'initialise' and a certain number of frames to 'cool down' with some actions being faster than others. The game mimics human response time by providing game information to the agents with a delay of 0.25 seconds. Since there are 60 frames a second this delay corresponds to 15 frames which is greater than the initialisation period of many attack actions thus making them difficult to avoid for a reactive agent. In this report we only use the character Zen. Zen's action set can be seen in Table 9 in appendix A.

For almost all practically conceivable agents the Fighting ICE platform provides a partially observable, stochastic environment. Stochastic in the sense that choosing a certain action in a certain environmental state doesn't guarantee identical outcomes each time. Partially observable in the sense that while an agent could in theory, fully realise its surrounding it's impractical to process this much information in real time [6]. Additionally the agent can never be fully sure of its opponents choice of action. In the majority of cases (Fighting ICE included among these) fighting games can be considered episodic i.e. the chosen action depends only on the current state of the environment and not on any of the past states. This would be different in a fighting game that allowed the use of combination moves as now knowledge of the past states may be required to execute the combination attack. Finally the FightingICE environment can be thought of as a dynamic, continuous environment. For all practical purposes there is an infinite amount of possible states the environment could exist in and these states will change independent of the input from our agent i.e. the states will change based on the opponents actions and the clock running down. A common device used to summarise agents is the PEAS (performance, environment, actions, sensors) descriptor. The PEAS descriptor of a typical fighting game agent is shown in Table 8 in Appendix A.

2.2 Reactive Agents

To test the performance of our learning AI three reactive agents were developed as opponents. These agents model three of the most common strategies adopted by human players within fighting games. The first AI, Charge AI, chases down the opponent as quickly as possible while dodging incoming projectiles. Once in range of the opponent the AI hits with the fastest attacks, not allowing its opponent to recover. This AI mainly aims to deal damage rather than avoid it. The second AI, Jump AI, primarily tries to avoid getting hit by continuously jumping around its opponent. The AI attacks while in the air and upon landing. This AI prioritises avoiding damage over dealing it. The winning AI in the 2015 fighting AI tournament used a similar strategy to the Charge AI for the first half of each fight and a similar strategy to the Jump AI for the second half. The third AI, Projectile AI, attempts to keep its distance from an opponent while continuously shooting projectiles. If an opponent manages to get in close then the projectile AI attempts to throw it away. This AI tries to equally avoid damage and deal damage. Detailed descriptions of the three reactive agents are given below. The implementations for these three AIs are not particularly rigorous and even a fighting game novice would likely be able exploit the various attack patterns. We are hoping that the learning AI will be able to adopt a policy that takes advantage of some of these exploits.

2.2.1 Charge AI

As mentioned above the goal of this AI is to quickly close in on the opponent and then to hit them with high speed attacks that will be difficult to block. The Charge AI is given an action set of six possible actions and can exist in sixteen possible environmental states defined by four variables. The environmental state is describe by a combination of high level variables which we will call state variables. The action set, the state variables and the policy that maps each environmental state to an action are shown below

State Space
For the charge AI the state space is partition by four true or false propositions resulting in 16 possible states. The four propositions that partition the state space are:

1. Is distance (D) to opponent greater than 70 pixels
2. Is the opponent standing (S) or crouching (jumping or knocked down count as standing)
3. Has the opponent launched a projectile (P)
4. Is our energy (E) above 50 points

Action Space
The six actions that the Charge AI can use allow it to move towards its opponent, jump over incoming projectiles and quickly attack the opponent depending on how high the energy level is and whether the opponent is crouching or not

1. Jump Forward (JF)	4. dash forward (DF)
2. Strong High Punch (SHP)	5. Strong Low Kick (SLK)
3. Weak High Punch (WHP)	6. Weak Low Kick (WLK)

Policy
The complete policy for the Charge AI is shown in table 1 below. An important point to note about the policy is that we only use weak attacks when the opponent fires a projectile attack at close range. This is due to the high likelihood of the Charge AI character taking a hit, thus we don't want to waste our energy on an attack that likely won't succeed. We also don't try to dodge the attack since at close range we are more likely to prevent the attack by striking the opponent than we are to avoid getting hit. It should be possible for our Learning AI to develop similar strategies to this through the use of reinforcement learning [2].

	$E > 50$ and S	$E > 50$ and $\bar{S}$	$E < 50$ and S	$E < 50$ and $\bar{S}$
$D > 100$ and $\bar{P}$	DF	DF	DF	DF
$D > 100$ and P	JF	JF	JF	JF
$D < 100$ and $\bar{P}$	SHP	SLK	WHP	WLK
$D < 100$ and P	WHP	WLK	WHP	WLK

Table 1: Policy for Charge AI

2.2.2 *Jump AI*
The goal of the Jump AI is to avoid taking damage while dealing a small amount of damage whenever possible. The agent jumps back and forth over the opponent. On each landing it will attempt to kick the opponent, and then jump again. While in the air the Jump AI will kick if the opponent is nearby and will launch a projectile if the opponent is far away. Again the action space consists of six actions but this time only sixteen possible environmental states defined by four variables. The action space, environmental state variables and policy are shown below.

State Space
For the Jump AI the state space is partition by four true or false propositions. This time however there are only twelve possible states since we cannot be both in the air and having just landed. The four propositions that partition the state space are:

1. Is distance (D) to opponent greater than 100 pixels
2. Are we in the air (A)
3. Have we just landed (L)
4. Is our energy (E) above 20 points

Action Space
The six actions that the Jump AI can use allow it to jump over its opponent, attack both in the air and on landing, and to attack with projectiles while in the air if the opponent is a long distance off.

1. Jump Forward (JF)
2. Jump Up (JU)
3. Weak Weak Jump (WJK)
4. Jump Projectile (JP)
5. Strong Jump Kick (SJK)
6. Weak Low Kick (WLK)

Policy
The policy for Jump AI is shown in Table 2. An important point to note about the policy is that when we are a long distance from the opponent we jump forward if we have high energy and we jump straight up if we have low energy. This is due to the fact that if we have high energy it is likely since we are getting hit by projectiles.

	$E > 20$ and A	$E > 20$ and $\bar{A}$	$E < 20$ and A	$E < 20$ and $\bar{A}$
$D > 100$ and L	-	JF	-	JU
$D > 100$ and $\bar{L}$	JP	JF	JP	JU
$D < 100$ and L	-	WLK	-	WLK
$D < 100$ and $\bar{L}$	SJK	JF	WJK	JF

Table 2: Policy for Jump AI

2.2.3 *Projectile AI*
The goal of the Projectile AI is to keep a long distance from the opponent while attacking with projectiles. If the opponent gets too close the Projectile AI will back off. If it canâĂŹt back of any further then it will try to throw the opponent. If the AI doesn't have enough energy for a throw then it will jump away. The action space, state space and policy for the Projectile AI are shown below

State Space
For the Projectile AI the state space is partition by two variables and one true or false proposition. Each of the two variables can exist in three states resulting in 18 possible environmental states. The two variables and proposition are shown below:

1. Distance (D) is greater than 300 pixels, Distance is between 300 and 100 pixels, Distance is less than 100 pixels
2. Energy (E) is greater than 30, Energy is between 30 and 5, Energy is less than 5
3. We are against the Wall

Action Space
The six actions that the Projectile AI can use allow it to back away as its opponent approaches, attack using strong and weak projectiles depending on energy level, throw the enemy when backed into a corner and to escape when unable to throw the opponent.

1. Jump Forward (JF)
2. Jump Backward (JB)
3. Weak Projectile (WP)
4. Strong Projectile (SP)
5. Strong Throw (ST)
6. Weak Throw (WT)

Policy
Table 3 shows the policy for the Projectile AI. An important point to note about the three reactive AIs is that in most cases the Projectile AI beats the Charge AI, the Charge AI beat the Jump AI and the Jump AI beats the Projectile AI. This demonstrates the stochastic nature of the FightingICE environment. Given a deterministic environment two purely reactive AIs should result in exactly the same fight each time.

	E > 30	30 > E > 5	E < 5
D > 300 and W	SP	WP	WP
300 > D > 100 and W	SP	WP	WP
D < 100 and W	ST	WT	JF
D > 300 and $\bar{W}$	SP	WP	WP
300 > D > 100 and $\bar{W}$	JB	JB	JB
D < 100 and $\bar{W}$	JB	JB	JB

Table 3: Policy for Projectile AI

2.3 Reinforcement learning

Machine learning in generally divided into three categories, these being supervised learning, unsupervised learning and reinforcement learning. Regardless of the approach, the core goal of any learning algorithm is to generalise from experience, i.e. preform accurately on new, unseen data after having experienced a training data set. Supervised learning is the most common and most reliable form of learning due to it being trained using labeled data. The essence of supervised learning is that each data point exists as an input/output pair so that for every input the learning system already knows what output it should obtain (this is sometimes called a teacher signal). The system can then adjust its parameters whenever it calculates an output incorrectly. The most famous example of supervised learning is Rosenberg's perceptron which forms the basis of neural networks [7]. Supervised learning is both quick and accurate however, the requirement of known input/output pairs isn't always possible. This type of learning is best used for classification and while it could be used to obtain a very strong static policy it doesn't help us to adjust our policy when faced with new, unseen challenges.

Unsupervised learning is generally more difficult due to the training data being unlabeled, this results in there being no error or reward signal to lead to a potential solution. Unsupervised learning is most commonly used in cluster analysis (such as k-nearest neighbour clustering). The clusters are defined using measures of similarity based metrics such as distance or probability.

Reinforcement learning is a type of machine learning concerned with choosing an action given some environmental state so as to maximize a cumulative reward. The issue with reinforcement learning in terms of artificial intelligence is that it learns which actions to chose without actually learning how these action affect the environment [1]. This makes it unsuitable for use in a predictive (or planning) agent however, it is ideal for our purposes as we're just interested in how effective (how much reward) a certain action will be and not in what it actually does. This simplification makes reinforcement learning ideal for on-line precessing in cases where it would be too time consuming to calculate every possibility [4]. The down side is that the learning process can take a very long time as we are essentially learning through trial and error. It is for this reason that both the action space and the environmental state space of our Learning agent have been kept as small as possible (see section 2.4).

In summary reinforcement learning, also called approximate dynamic programming, is a Markov decision process (information of all the states so far is contained in just the previous state) which operates as a trade-off between exploration (gaining information) and exploitation (using current information). The standard reinforcement model consists of five data structures which are updated on each iteration.

1. A set of environmental states
2. A set of actions
3. A policy indicating which actions are taken in each state
4. The probability of transitioning from one state to another given a certain action
5. The Reward for transitioning from one state to another given a certain action

2.3.1 *The Bellman Equations*

As mentioned, the reinforcement learning model is driven by a reward signal. This reward signal specifies what the agents goals are. It does not however, specify how the agent achieves these goals, this is the job of the policy. The Bellman equations provide a means of determining an optimum policy for a reinforcement learning model. This is done through the use of two additional data structures [4] given in terms of the current policy: the value function $V^{\pi}(S)$ and the action value function $Q^{\pi}(s, a)$.

Value Function

The value function descries the expected reward of each state given the current policy. If there are n possible states then $V^{\pi}(s)$ is an $nx1$ vector such that:

$$V^{\pi}(s) = E_{\pi}\{R_t | s_t = s\} \tag{2}$$

Where R_t is the weighted accumulation of the expected rewards such that:

$$R_t = \sum_{k=1}^{\infty} \gamma^k r_{t+k+1} \tag{3}$$

From this we get the following expansion of the value function so that:

$$\begin{aligned} V^{\pi}(s) &= E_{\pi}\{\textstyle\sum_{k=1}^{\infty} \gamma^k r_{t+k+1} | s_t = s\} \\ &= E_{\pi}\{r_{t+1} + \gamma \textstyle\sum_{k=1}^{\infty} \gamma^k r_{t+k+2} | s_t = s\} \\ &= \textstyle\sum_a \pi(s,a) \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V^{\pi}(s')] \end{aligned}$$

where $P^a_{ss'}$ is the transition probability matrix and $R^a_{ss'}$ is the immediate expected reward matrix. We now see that the expected value of each state is the likelihood that a certain action will be chosen in the given state based on the current policy multiplied by the reward for transitioning to every other possible state times the probability of that transition occurring. Note that the probabilities off all possible actions occurring given a certain state should sum to one.

Action Value Function
The action value function describes the expected reward for choosing a certain action given a certain state such that:

$$Q^{\pi}(s,a) = \sum_{s'} P^{a}_{ss'}[R^{a}_{ss'} + \gamma V^{\pi}(s')] \quad (4)$$

We can see that the action value function is the same as the value function except it describes a value for every action in every state resulting in an nxm matrix where n is the number of environmental states and m is the number of actions.

Policy Evaluation and Update
On every iteration (i.e. at the end of every match) the Learning AIs policy is updated using two steps. First we determine values of the action value function and value function for each state, this is the evaluation step. We then compare the functions. If for a given state any of the $Q^{\pi}(s,a)$ values are larger than the $V^{\pi}(s)$ values then the policy is updated such that the corresponding action will always be chosen when in that state. In the event of a tie, each of the tied actions has an equal probability of occurring. Note that the expected reward matrix and transition probability matrix are updated after every action rather than every match (see section 2.4 for how these are updated). In order to allow for more exploration early on it may be preferable not to update the policy after the first few matches.

After each match the the value function and action value function are solved using an iterative approach. Simply put we keep solving equations 2 and 4 until the difference in values between two consecutive iterations drops below some threshold.

2.4 Learning Model

Here we outline the model for an agent that is able to learn a policy based on the opponent it is fighting. We have chosen to use reinforcement learning through the use of the Bellman equations as described above and in [1] [2] as our method for determining an appropriate policy. In the case of a fighting game, supervised learning is inappropriate as there's generally no 'best' action that can be used as a teacher signal. Unsupervised learning is similarly unsuitable as it doesn't supply the required structure to reliably lead to a winning policy. Reinforcement learning provides a useful compromise between the two with a reward signal that doesn't explicitly specify the correct action for a given state but provides a metric of how useful that action has been in the past.
As mentioned above we require five things to develop a reinforcement learning model, an action set, an environmental set, a policy, the probabilities of transitioning from one state to another given a certain action and the reward for transitioning from one state to another given a certain action. While it is well within the realms of possibility to dynamically alter the action space and environmental state space we will consider them to be fixed for the purposes of this report. The chosen state space consists of four states defined by two true or false statements and the chosen action space consists of five actions. Both of these are shown below.

2.4.1 State Space

To achieve a low amount of complexity the number of possible environmental states was kept small with only four states being used to ensure fast convergence. Energy isn't considered in any of the environmental states since none of the actions available to the learning AI require it. The result is that there are no restrictions placed on the action set. The two propositions used to define the state space are

1. Is opponent in attack range (R)
2. Is opponent attacking (A)

2.4.2 Action Space

The action space consists of five actions. Again the action space was kept small to ensure low complexity. The five actions were chosen to give a good variety to the learning AI allowing it to move, dodge, block, attack and shoot. The five actions chosen are shown below.

1. Jump Forward (JF)
2. Crouch Guard (CG)
3. Weak Projectile (WP)
4. Move forward (MF)
5. Weak Kick (WK)

2.4.3 Policy

The ultimate goal of this research was for the learning AI to be able to adapt its policy based on the experience obtained fighting its opponent. An initial policy is required in order for the learning AI to gain this experience. The obvious choice for the initial policy is one were for each state, every action has an equal change of being chosen. This corresponds to having no prior information about which action is most beneficial in each state. Given more time it would have been interesting to test the effects of an asymmetric initial policy to see whether it had any influence on the speed of convergence and the final steady state values. The initial policy is shown in Table 4

	$\bar{R}$ And $\bar{A}$	$\hat{R}$ And A	R And $\bar{A}$	R And A
MF	0.2	0.2	0.2	0.2
JF	0.2	0.2	0.2	0.2
CG	0.2	0.2	0.2	0.2
WK	0.2	0.2	0.2	0.2
WP	0.2	0.2	0.2	0.2

Table 4: Initial Policy for Learning AI

2.4.4 Transition Probabilities

The transition probabilities are the probability of changing from one environmental state to another given an action. This is given by an $nxnxm$ matrix where n is the number of environmental states and m is the number of actions. If we consider the environmental state space shown above then no state transition can be ruled as impossible. Initially the agent has no information on the transition probabilities. Instead we gain information about the transition probabilities based on repeated simulation during each match. Initially, in each state, given a certain action there is an equal probability of transitioning to any other state. During simulation an $nxnxm$ matrix contains counts of the number of times a transition form state s to s' given action a occurred. The probability of transition from state s to s' given action a is then the number of times (counts) that this transition occurred divided by the sum of counts of the other possible state transitions given state s and action a. For mathematically clarity all cells of the counts matrix are initialised to one.

$$P(S_{t+1} = s' | s_t = s, a_t = a) = \frac{counts(s, a, s')}{\sum_{s' \in S} counts(s, a, s')} \quad (5)$$

2.4.5 Transition Rewards

As well as the transition probabilities we also require the expected rewards for each transition. Again this is an $nxnxm$ matrix which is initialised to zero. Just like the transition probabilities the expected reward values need to be learned based on experience from fighting an opponent. Every time an action is taken the reward for that action is calculated as:

$$r_t = \triangle(MyHP) - \triangle(EnemyHP) \quad (6)$$

This was chosen as the reward function since winning or losing the fight is based on the amount of damage taken vs. the amount of damage received. The idea is that an action that damages the enemy or prevents damage to you is rewarded. Designing an appropriate reward function is the most important aspect of a reinforcement learning system. It is also one of the most difficult. It should be obvious that the reward function present above is extremely simple. A more complete reward function (such as the one shown below where W_E is a weight between zero and one) would also include the energy levels of the Learning AI and the opponent. This would allow us to reward actions the block opponents attacks and condemn actions that result in our attacks being blocked.

$$\begin{aligned} r_t = & (\triangle(MyHP) + W_E \triangle (MyEnergy)) \\ & - (\triangle(EnemyHP + W_E \triangle (EnemyEnergy)) \quad (7) \end{aligned}$$

It would be further useful if the reward function could inform the Learning agent whether it was more appropriate to close in on the opponent or to keep its distance. This would likely require some sort of sequential reward function so that we could update the expected reward of a movement action based on whether the next attack or dodge resulted in use dealing or taking more damage.
The final total reward matrix is the sum of the rewards for when a given action caused a transition from state s to s'. This is then divided by the number of times (counts) that this transition occurred to get the expected reward for that transition.

3 Results

In this section we detail how the learning AI preformed against the three reactive agents described in section 2. We also provide some criticism and potential improvements for the final Learning AI agent
As mentioned in section 2.4 the initial policy of the learning AI gives all actions an equal chance of occurring in each state. After each match the policy is updated based on how effective each action was in each state. This means that in the next fight only the most effective action is chosen for each state. If a chosen action then proves to be ineffective it will be replaced by another action in the next match. The downside to this is that if a chosen action continues to produce a positive reward then it won't change even if there is a better suited action. In some ways this is a bit of a non-issue since if every state has an action that produces a positive reward signal then we are guaranteed to win the fight. it is however, still desirable to attain an optimal policy rather than just a winning one. A simple way around this is to complete several matches without updating the policy. This should result in enough information so that the first policy update will immediately give the optimum values (or at least something near it). Another possibility is to weight the likelihood of each action being chosen based on its expected reward compared to the expected reward of all the other possible actions for the given state.

Its also worth noting that by updating the policy so that only a single action is chosen for each state the policy update step just needs to consider the action value function and not the value functions. This is due to the value function having the same value as the action value function for the action specified by the current policy. The tables below show the results of the Learning agent against the Charge AI, Jump AI and Projectile AI over ten matches. We also show the policy updates and action value function for each action-state pairing across the ten matches.

A very important point to note is that the stochastic nature of the platform and also the limited amount of exploration (training information) before updating the policy means the policy doesn't always converge to the same outcome. This effect could potentially be reduced by completing several matches before updating the policy as discussed above. The results given below are an example of the most common policy obtained after convergence. These are presented to show that, while flawed, the Learning agent does in the vast majority of cases improve on the simple random policy.

3.1 Charge AI

The values displayed in table 5 are the Learning AI's scores out of 1,000 for each round when fighting the charge AI. A score of over 500 indicates a win. The learning AI starts with an initial random policy which was updated after every match

Round 1	Round 2	Round 3	Round 4	Round 5
383	298	267	479	520
321	265	381	489	524
320	308	386	488	439

Round 6	Round 7	Round 8	Round 9	Round 10
561	561	576	561	521
597	568	542	597	544
624	560	585	590	556

Table 5: Results of ten matches between charge AI and Learning

We can see that as the policy begins to converge to a final value the Learning AI does start beating the Charge AI. Rounds 7 to 10 show that this isn't just a fluke and that the final policy is in fact strong enough to win against the charge AI. We also note the Learning AI is only winning the fights by a small margin. This is because in close combat both AIs use the same attack action so that which character actually gets hit is roughly down to a 50/50 chance. The reason the Learning AI wins all the fights is because it can also shoot projectiles from a distance giving it a slight advantage.

Policy Update

In the interests of space the policy is only shown for three rounds where π_1 is the initial policy used in round 1, π_3 is the policy used in round 3 and π_7 is the policy used in round 7. By round 7 the policy had converged to a steady state value so that there were no further changes. States 1-4 make up the rows of the matrix and actions 1-5 make up the columns of the matrix

$$\pi_1 = \begin{bmatrix} 0.20 & 0.20 & 0.20 & 0.20 & 0.20 \\ 0.20 & 0.20 & 0.20 & 0.20 & 0.20 \\ 0.20 & 0.20 & 0.20 & 0.20 & 0.20 \\ 0.20 & 0.20 & 0.20 & 0.20 & 0.20 \end{bmatrix}$$

$$\pi_3 = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \qquad \pi_7 = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

The final policy was reached after 6 matches were completed. We can see that the final policy causes the agent to attack with projectiles when the opponent is far away and attack with punches when the opponent is nearby. The final policy here greatly favours attack. Comparatively, the final policy for the jump AI seems to favour defense more. It would appear that the learning AI tends to be aggressive towards aggressive opponents and defensive towards defensive opponents. In fact, in some ways the Learning AI simply mimics its opponent with a slight addition that allows it to come out on top.

Action Value function

Finally we look at the action value function which was used to get the policies for rounds 3 and 7. Again the four rows correspond to the four environmental states (1. enemy far away and not attacking, 2. enemy far away and attacking, 3. Enemy close and not attacking, 4 enemy close and attacking) and the five columns correspond to the five actions.

$$Q_3^\pi(s,a) = \begin{bmatrix} 0.28 & 0.54 & 0.54 & 0.68 & 0.60 \\ -0.04 & -0.06 & 0.01 & 0.01 & -0.13 \\ -0.05 & 0.25 & -0.09 & 0.21 & 0.43 \\ -0.85 & -0.90 & -0.87 & -0.73 & -0.83 \end{bmatrix}$$

$$Q_7^\pi(s,a) = \begin{bmatrix} 0.30 & 0.50 & 0.58 & 0.69 & 0.55 \\ -0.27 & -0.27 & -0.28 & -0.33 & -0.22 \\ 0.18 & 0.5 & 0.15 & 0.86 & 0.69 \\ -0.64 & -0.70 & -0.66 & -0.54 & -0.66 \end{bmatrix}$$

There are a few interesting points to note about the action value function. Firstly we note that some of the values in state 3 are negative. This shouldn't be possible since our opponent doesn't attack in state three. An explanation as to why this is the case is covered in section 3.4. Another point to note is that the action value function changes even if the corresponding action wasn't used. This is due to the action value function being a function of the value function. Since the value function depends on all actions, if a single action is used then all action value functions get updated for that state. This is why the expected value for all actions in state 4 improve between rounds 3 and 7.

3.2 Jump AI

Again the values displayed in Table 6 are the Learning AIs scores out of 1,000 for each round when fighting the jump AI. A score of over 500 indicating a win.

Round 1	Round 2	Round 3	Round 4	Round 5
464	405	267	709	1000
438	397	288	781	871
429	368	285	808	1000

Round 6	Round 7	Round 8	Round 9	Round 10
886	1000	1000	1000	721
1000	956	1000	1000	977
1000	978	1000	1000	1000

Table 6: Results of ten matches between Jump AI and Learning

The results in table 6 show that after round five the learning AI begins winning matches without taking a hit. This is mostly due to the fact that the punch action for the learning AI is faster than the jump AI landing and kicking, thus the Learning AI never gets hit.

Policy Update

Again we only show the policy for three rounds where π_1 is the initial policy used in round 1, π_3 is the policy used in round 3 and π_5 is the policy used in round 5. After five rounds the policy had converged to a steady state value so that there were no further changes. States 1-4 make up the rows of the matrix and actions 1-5 make up the columns of the matrix.

$$\pi_1 = \begin{bmatrix} 0.20 & 0.20 & 0.20 & 0.20 & 0.20 \\ 0.20 & 0.20 & 0.20 & 0.20 & 0.20 \\ 0.20 & 0.20 & 0.20 & 0.20 & 0.20 \\ 0.20 & 0.20 & 0.20 & 0.20 & 0.20 \end{bmatrix}$$

$$\pi_3 = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad \pi_5 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

We can see that the policy for the third match is very poor. This policy results in us trying to punch the opponent when we are out of range and simply walking forward when the opponent is right next to us and attacking. A positive note

is that we try to guard against the opponent when they attack from a distance however, we also try to guard when the opponent is right next to us and not attacking. This would usually be our best change to attack the opponent. As one would expect our results for this round were very low.
The final steady state policy is much stronger. In this case we attack when the opponent is nearby and move toward the opponent when they are far away. By constantly attacking when the opponent is nearby we usually hit the opponent as it lands preventing it from attacking us. This is the reason so many of the rounds obtained perfect scores. This is generally the most common policy for the Learning AI to converge to however, sometimes the policy converges to one where it simply attacks with a punch (action 4) in every state.

Action Value function
The action value functions shown below are for rounds 3 and 5. Again the four rows correspond to the four environmental states and the five columns correspond to the five actions.

$$Q_3^\pi(s,a) = \begin{bmatrix} 0.24 & 0.11 & 0.16 & 0.33 & 0.16 \\ -0.05 & -0.01 & 0.04 & -0.01 & 0.01 \\ 0.33 & 0.42 & 0.54 & 0.48 & 0.53 \\ -0.06 & -0.14 & -0.07 & -0.09 & -0.12 \end{bmatrix}$$

$$Q_5^\pi(s,a) = \begin{bmatrix} 0.34 & 0.21 & 0.28 & 0.25 & 0.29 \\ -0.07 & -0.15 & -0.01 & -0.06 & -0.09 \\ 0.68 & 0.72 & 0.59 & 0.89 & 0.56 \\ -0.04 & -0.07 & -0.01 & 0.17 & 0.02 \end{bmatrix}$$

Between matches 3 and 5 there is an increase in the value for all the action state pairs in states 3 and 4. This is due to us doing a large amount of damage without taking a hit. This results in an increase in the expected reward and thus the action value function.
This continues over the next 5 matches and these values become quite large. Interestingly, since we weight more recent rewards more heavily through the use of a gamma factor (see Equation 4) even if the current policy were to suddenly become ineffective it wouldn't take long for the Learning AI to adapt to a new policy. This can be shown by first training the Learning AI against the jump AI and then having it fight the projectile AI.

3.3 Projectile AI

The values displayed in Table 7 for each round are the Learning AIs scores out of 1,000 when fighting the Projectile AI. A score of over 500 indicating a win.

Based on the performance of the random AI we can assume that the projectile AI is the strongest of the 3 reactive AIs used for testing. The results in table 7 show that the Learning AI never actually manages to beat the Projectile AI. The limited action space seems to be responsible for this (at least in part). While the policy does eventually converge to one that allows the Learning AI to move in close to hit the opponent, the the Learning AI gets thrown away most of the time. In order to defeat the Projectile AI, and other more complete reactive AIs, it will likely be necessary to increase the size of the action space and environmental state space. The down side to doing this is that it would take longer for the policy to converge to a final value.

Round 1	Round 2	Round 3	Round 4	Round 5
262	202	305	378	302
254	196	364	361	297
289	236	321	408	281

Round 6	Round 7	Round 8	Round 9	Round 10
453	422	461	417	525
461	459	508	462	446
408	413	439	438	482

Table 7: Results of ten matches between Projectile AI and Learning AI

Policy Update
Once again π_1 is the initial policy used in round 1, π_3 is the policy used in round 3 and π_8 is the policy used in round 8. By this point the policy had converged so that there were no further changes. States 1-4 make up the rows of the matrix and actions 1-5 make up the columns of the matrix

$$\pi_1 = \begin{bmatrix} 0.20 & 0.20 & 0.20 & 0.20 & 0.20 \\ 0.20 & 0.20 & 0.20 & 0.20 & 0.20 \\ 0.20 & 0.20 & 0.20 & 0.20 & 0.20 \\ 0.20 & 0.20 & 0.20 & 0.20 & 0.20 \end{bmatrix}$$

$$\pi_3 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \qquad \pi_8 = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

We note that of the three reactive agents the projectile AI takes the most amount of fights before we obtain a final policy. Additionally we get the widest distribution of final polices when fighting the projectile AI. There are two contributing factors to this. Firstly, the Learning AI must learn how to approach the projectile AI, which is considerably more difficult than approaching the charge and jump AIs. Secondly, as discussed in section 3.4 it is hard to get accurate reward information for projectile attacks. Usually the action that receives the blame for getting hit by the projectile is not the one that was executed in response to the projectile.

Action Value function
Finally we look at the action value function which was used to get the policies for rounds 3 and 8. Again the four rows correspond to the four environmental states (1. enemy far away and not attacking, 2. enemy far away and attacking, 3. Enemy close and not attacking, 4 enemy close and attacking) and the five columns correspond to the five actions.

$$Q_3^{\pi}(s,a) = \begin{bmatrix} 0.28 & 0.35 & 0.33 & 0.25 & 0.26 \\ -0.17 & -0.29 & -0.12 & -1.02 & -0.51 \\ 0.13 & 0.18 & 0.21 & 0.16 & 0.23 \\ -0.33 & -0.19 & -0.12 & -028 & -0.15 \end{bmatrix}$$

$$Q_8^{\pi}(s,a) = \begin{bmatrix} 0.33 & 0.41 & 0.35 & 0.40 & 0.36 \\ -0.11 & -0.13 & -0.13 & -0.23 & -0.17 \\ 0.45 & 0.40 & 0.37 & 0.58 & 0.39 \\ -0.49 & -0.47 & -0.54 & -0.47 & -0.44 \end{bmatrix}$$

3.4 Criticism and Improvements

There are two major issues here that should be immediately apparent. Firstly, when fighting against all three reactive AIs, the Learning AI initially (in the early rounds) preforms worse than the random AI. The second point we should note is the action value function for the cases when we're moving forward, jumping or blocking and the opponent isn't attacking. Since the reward is only based on a change in health, the action value function should be zero. The reason its not zero is a result of the episodic implementation of the reward function (i.e. poor implementation). Take the case in which the learning AI shoots a projectile and then jumps in the air. At this point the opponent is hit by the projectile. Now the expected reward of the jump action is increased rather than the projectile action which is what actually did the damage.

The effect of this is that when using the random policy we actually get a lot of misinformation, due to the reward value being updated for a later (incorrect) action. This explains the initial drop in performance of the learning AI compared to the starting random AI as mostly wrong actions are deemed successful. Interestingly once we start using only a single action for each state we begin to get a lot more correct information. This is because, with the low number of possible environment states, we don't actually change state all that often. This means we continue using the same action, resulting in the reward value for the correct action being updated even if it was the previous action that actually had the effect on character health. There are still cases where the wrong action is updated however, provided the state space is small, these are few enough that the policy does in general converge to a strong steady state value.

This issue with the wrong actions being rewarded prevents us from implementing most of the potential improvements discussed through out this paper. The idea of increasing the size of the state space and action space to provide more variability (see section 3.3) goes out the window. This would greatly increasing the amount of fights required for convergence and also cause far more state changes resulting in an increased likelihood of the wrong action being rewarded. Similarly the idea of weighting the actions based on there expected reward (see start of section 4) compared to other actions for a given state also increases the likelihood that an action will be incorrectly rewarded. Finally the idea of completing several matches without updating the policy from the random AI in order to gain more information has a highly negative effect on performance as the majority of rewards are given to the wrong action. In some cases the initial misinformation of the random AI causes the Learning AI to converge to a poor policy. Simply put, the best thing that could be done to improve this learning AI is to fix it so that correct actions are given the correct rewards.

4 Conclusion

In this paper we've demonstrated the limits of conventional reactive agents for use in fighting games. While these types of agent can be effective, its often a simple matter for a human player to find some exploit of the AI which can then be repeated without the AI ever changing strategy. We proposed the use of a Learning AI based on reinforcement learning and the Bellman equations. This agent would be able to change its chosen action for a given state if that action had proven to be ineffective in the past through the use of a reward function. Three simple reactive agents where designed so that they would mimic common strategies employed in fighting games. These reactive agents were used to train and test the Learning agent. While there were some issues with the implementation of the learning AI, it still showed an improvement over the initial random policy.

5 References

[1] L. Baird and A. W. Moore. Gradient descent for general reinforcement learning. *Advances in neural information processing systems*, pages 968–974, 1999.

[2] T. Graepel, R. Herbrich, and J. Gold. Learning to fight. In *Proceedings of the International Conference on Computer Games: Artificial Intelligence, Design and Education*, pages 193–200, 2004.

[3] F. Lu, K. Yamamoto, L. H. Nomura, S. Mizuno, Y. Lee, and R. Thawonmas. Fighting game artificial intelligence competition platform. In *Consumer Electronics (GCCE), 2013 IEEE 2nd Global Conference on*, pages 320–323. IEEE, 2013.

[4] M. Martin. Bellman equations and optimal policies. *Universitat politÃĺcnica de Catalunya Dept. LSI*, page 15, 2011.

[5] J. McCarthy, M. L. Minsky, N. Rochester, and C. E. Shannon. A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955. *AI Magazine*, 27(4):12, 2006.

[6] A. Ricciardi and P. Thill. Adaptive ai for fighting games. *December*, 12(200):8, 2008.

[7] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[8] K. Yamamoto, S. Mizuno, C. Y. Chu, and R. Thawonmas. Deduction of fighting-game countermeasures using the k-nearest neighbor algorithm and a game simulator. In *Computational Intelligence and Games (CIG), 2014 IEEE Conference on*, pages 1–5. IEEE, 2014.

6 Appendix A

Agent	Performance	Environment	Actions	Sensors
Fighting Game Agent	Damage taken Vs. Damage dealt	Current Health Current Energy etc.	Attack Defend Dodge	Distance to enemy, enemy attack

Table 8: PEAS descriptor of the FightingICE environment

Skill	Damage	Startup	Active	Recovery
Throw A	10	5F	1F	24F
Throw B	20	20F	1F	9F
Stand A	5	3F	3F	11F
Stand B	10	5F	3F	15F
Crouch A	5	3F	3F	12F
Crouch B	10	10F	3F	5F
Stand FA	5	5F	3F	28F
Stand FB	15	12F	3F	28F
Crouch FA	5	3F	3F	31F
Crouch FB	10	12F	3F	45F
Stand D DF FA	10	20F	-	40F
Stand D DF FB	40	15F	-	45F
Stand D DF FC	300	15F	-	33F
Stand F D DFA	10	10F	20F	40F
Stand F D DFB	40	10F	10F	40F
Stand D DB BA	10	30F	5F	20F
Stand D DB BB	20	5F	10F	45F

Table 9: Moveset for the Character Zen, The start up, active and recovery time are the amount of frames it takes to preform each action