Implementation of an Intelligent Agent for the AIBIRDS Competition

Technical University of Crete

School of Electrical and Computer Engineering

Author: Michail Gemistos

Supervisor: Michail G. Lagoudakis, Associate Professor

Committee

Antonios Deligiannakis, Associate Professor

Georgios Chalkiadakis, Associate Professor

October 10, 2019

Abstract

The broad field of Artificial Intelligence (AI) strives to reproduce human behavior on machines. Machine Learning, as a subfield, and more specifically Reinforcement Learning (RL), enables autonomous agents to take suitable actions under different circumstances through a trial-and-error learning process, without being programmed for every possible scenario they may encounter. Since 2013, the International Joint Conference on Artificial Intelligence (IJCAI) hosts the Angry Birds AI Competition (AIBIRDS), where various AI agents compete on the Angry Birds computer game. The agents compete on unknown game levels without any human intervention. In this thesis, we designed two agents for AIBIRDS following the principles of two well-known RL algorithms, namely Q-Learning and Least Squares Policy Iteration (LSPI). Both of them are model-free RL algorithms, trying to learn the best action at each step (policy) for any given game scene. Since the action and state spaces of the game are extremely large and due to the absence of a model that describes the transition from a state to a next state affected by an action choice, we used an approximation architecture to represent the learned $\mathcal{Q}$ values, which estimate the quality of each action in each state. The approximation uses a set of eight basis functions (features) we designed, which try to describe a game scene effectively, and each one is weighted by its own parameter (weight). In our experiments, the Q-Learning agent is trained for 20,000 iterations updating its *weights* incrementally during the course of that training, concluding to their final values, when the iterations are completed. At each iteration, the Q-Learning agent stores locally each observed sample of interaction with the game, which includes the current state, the action taken, the new state and the reward gained. The LSPI agent is then trained using the stored set of samples to find its own set of *weights* and thus its own policy. When the process of training ends for both Q-Learning and LSPI on the same observed samples, we test each agent on 54 different levels taken directly from the AIBIRDS competition, 34 of those being the levels our agents were trained on and 20 levels being completely new to the agents. The Q-Learning agent is able to complete successfully 68% of these levels and the LSPI agent 81% of them, occasionally performing precise shots with amazing results.

Ανάπτυξη Ευφυούς Πράκτορα για τον Διαγωνισμό AIBIRDS

Πολυτεχνείο Κρήτης
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Μιχαήλ Γεμιστός
Επιβλέπων καθηγητής: Μιχαήλ Λαγουδάκης

Περίληψη

Το ευρύ πεδίο της Τεχνητής Νοημοσύνης (Artificial Intelligence - AI) προσπαθεί να αναπαράγει την ανθρώπινη συμπεριφορά στις μηχανές. Η Μηχανική Μάθηση, ως υποπεδίο, και πιο συγκεκριμένα η Ενισχυτική Μάθηση (Reinforcement Learning - RL), δίνει τη δυνατότητα σε αυτόνομους πράκτορες να επιλέγουν κατάλληλες ενέργειες κάτω από διαφορετικές συνθήκες μέσω μιας διαδικασίας μάθησης δοκιμών-και-σφαλμάτων, χωρίς να προγραμματίζονται για κάθε πιθανό σενάριο που μπορεί να συναντήσουν. Από το 2013, το συνέδριο International Joint Conference on Artificial Intelligence (IJCAI) φιλοξενεί τον Διαγωνισμό Angry Birds AI (AIBIRDS), όπου διάφοροι AI πράκτορες ανταγωνίζονται στο ηλεκτρονικό παιχνίδι Angry Birds. Οι πράκτορες ανταγωνίζονται σε άγνωστες πίστες παιχνιδιού χωρίς καμία ανθρώπινη παρέμβαση. Στην παρούσα διπλωματική εργασία, σχεδιάσαμε δύο πράκτορες για το AIBIRDS ακολουθώντας τις αρχές δύο γνωστών RL αλγορίθμων, συγκεκριμένα του Q-Learning και του Least Squares Policy Iteration (LSPI). Και οι δύο είναι RL αλγόριθμοι χωρίς μοντέλα και προσπαθούν να μάθουν την καλύτερη ενέργεια σε κάθε βήμα (πολιτική) για κάθε δεδομένη σκηνή του παιχνιδιού. Δεδομένου ότι οι χώροι ενεργειών και καταστάσεων του παιχνιδιού είναι εξαιρετικά μεγάλοι και λόγω της απουσίας ενός μοντέλου που περιγράφει τη μετάβαση από μια κατάσταση σε κάποια επόμενη κατάσταση υπό την επιρροή μιας επιλεγμένης ενέργειας, χρησιμοποιήσαμε μια αρχιτεκτονική προσέγγισης για να αναπαριστούμε τις μαθηματικές τιμές Q, οι οποίες εκτιμούν την ποιότητα κάθε ενέργειας σε κάθε κατάσταση. Η προσέγγιση χρησιμοποιεί ένα σύνολο από οκτώ συναρτήσεις βάσης (χαρακτηριστικά) που σχεδιάσαμε, τα οποία προσπαθούν να περιγράψουν αποτελεσματικά μια σκηνή παιχνιδιού, και κάθε μία σταθμίζεται με τη δική της παράμετρο (βάρος). Στα πειράματά μας, ο πράκτορας Q-Learning εκπαιδεύεται για 20.000 επαναλήψεις για την ενημέρωση των βαρών του σταδιακά κατά τη διάρκεια αυτής της εκπαίδευσης, καταλήγοντας στις τελικές τους τιμές, όταν ολοκληρωθούν οι επαναλήψεις. Σε κάθε επανάληψη, ο πράκτορας Q-Learning αποθηκεύει τοπικά το κάθε παρατηρούμενο δείγμα αλληλεπίδρασης με το παιχνίδι, το οποίο περιλαμβάνει την τρέχουσα κατάσταση, τη ενέργεια που έχει ληφθεί, τη νέα κατάσταση και την ανταμοιβή που αποκτήθηκε. Ο πράκτορας LSPI εκπαιδεύεται στη συνέχεια χρησιμοποιώντας το αποθηκευμένο σύνολο δειγμάτων για να βρει το δικό του σύνολο βαρών και επομένως τη δική του πολιτική. Όταν η διαδικασία εκπαίδευσης τελειώσει τόσο για τον Q-Learning όσο και για τον LSPI πάνω στα ίδια παρατηρούμενα δείγματα, εξετάζουμε κάθε πράκτορα σε 54 διαφορετικές πίστες που ελήφθησαν απευθείας από τον διαγωνισμό AIBIRDS, 34 εκ των οποίων είναι αυτές όπου εκπαιδεύτηκαν οι πράκτορές μας και 20 είναι παντελώς άγνωστες στους πράκτορες. Ο πράκτορας Q-Learning είναι σε θέση να ολοκληρώσει επιτυχώς το 68% και ο πράκτορας LSPI το 81% εξ αυτών, εκτελώντας κατά διαστήματα βολές ακριβείας με εκπληκτικά αποτελέσματα.

Acknowledgments

I would like to express my special thanks to my supervisor Assoc. Prof. Michail G. Lagoudakis for his guidance through the course of our study as well as the trust he showed towards me when we participated in the competition, despite the fact that the agent was in his early stage. I would also like to thank my parents, my friends and Sofia for their constant support during my thesis.

Contents

1 Introduction **1**

1.1 Thesis Introduction 1

1.2 Thesis Contribution 2

1.3 Thesis Overview 3

2 Background **5**

2.1 Reinforcement Learning 5

2.1.1 Markov Decision Process 5

2.1.2 Q-learning with Linear Approximation 7

2.1.3 Least Squares Policy Iteration 9

3 Problem Statement **11**

3.1 Problem Statement 11

3.2 Related Work 12

4 Our Approach **17**

4.1 Modeling the Game Scene 17

4.1.1 The State 17

4.1.2 The Action 19

4.1.3 The Reward 21

4.2 Basis Functions - Features 22

4.2.1 Global Features 22

4.2.2 Customized Features 23

4.3 Q-Learning with Linear Approximation 25

4.4 Least-Squares Policy Iteration 27

5 Results **29**

6 Conclusions **41**

6.1 Discussion 41

6.2 Future Work 41

Bibliography **43**

List of Figures

2.1 Step size for 20,000 iterations 8

3.1 2017 AIBIRDS competition 13
3.2 Types of birds and their abilities 14
3.3 Types of building objects 14
3.4 Characteristic samples of levels 15

4.1 State, Action, New State 19
4.2 Different target points for same object and their new state 20
4.3 Different Angles for same target point and their new state 21
4.4 Objects the "imaginary" trajectory meets, after the object-action 21
4.5 "Imaginary" trajectory meets Pigs, feature 23
4.6 Five out of eight features that characterize an action. 24

5.1 Q-Learning converged weights 30
5.2 LSPI converged weights 30
5.3 An Episode of 21 Levels 31
5.4 Level 1 of Episode 1 32
5.5 Level 21 of Episode 1 32
5.6 Level 17 of Episode 2 33
5.7 Level 5 of Episode 3 33
5.8 Level 17 of Episode 3 34
5.9 AIBIRDS 2014 results in red for the AngryBer Team 35

List of Tables

4.1 Impact feature 23

5.1 All episodes gathered 35
5.2 Episode 1 Summary 36
5.3 Episode 2 Summary 36
5.4 Episode 3 Summary 36
5.5 Episode 1 Results 37
5.6 Episode 2 Results 38
5.7 Episode 3 Results 39

Chapter 1

Introduction

1.1 Thesis Introduction

Since the beginning of video games, Artificial Intelligence (AI) has been a vital component of them, serving many purposes. Games are very useful, because we can evaluate how well an AI program performs against human players and are an effective way to develop and demonstrate AI. In role-playing games, like Dark Souls and World of Warcraft, AI is used to design intelligent and strategic enemies, making the game more difficult for the human player. In addition, it has been used to play against human players in complex video/board games, which demand sophisticated reasoning. One milestone of that battle, AI vs Humans, was when AlphaGo, a program developed by DeepMind Technologies to play the board game GO, beat the 18-time world champion Lee Sedol in four out of five games. Go has 10^{172} possible board positions, outlining the importance of that achievement.

Machine Learning (ML), a subset of AI, offers powerful algorithms to develop a game-playing agent that can approach a video game and its environment sufficiently. An agent designed according to Reinforcement Learning (RL) algorithms, a type of ML, interacts with an environment through an action, which is then evaluated based on the returned reward. Through repetition, the agent learns a policy of actions to perform, when similar environments appear by incorporating the feedback.

The International Joint Conference on Artificial Intelligence (IJCAI), a non-profit corporation for scientific and educational purposes focused on Artificial Intelligence, hosts the Angry Birds AI (AIBIRDS) competition and the Man vs Machine challenge annually. In the AIBIRDS competition

AI agents, from all around the world, compete in the Angry Birds (AB) game on unknown game levels. AB is a two-dimensional game in which multi-colored birds try to save their eggs from pigs, their enemies. Due to the nature of the game, designing an agent that plays well necessitates a careful and sophisticated approach.

During the AIBIRDS competition, many approaches have been employed covering a wide variety of AI fields. Despite the great interest from teams all around the world, no AI agent was able to beat the human players in the Man vs Machine challenge yet.

1.2 Thesis Contribution

Since Reinforcement Learning has proven to be a very effective way to design AI agents for video games, we chose to follow that approach. We implemented two model-free RL algorithms, Q-Learning [6] and Least Squares Policy Iteration (LSPI) [3]. Both of them are easy to implement and try to learn a policy, in other words learn what action the agent should take under what circumstances. Furthermore, model-free RL algorithms have the flexibility to handle environments, like video games, where the information of what the outcome (new state) will be after a certain action at a state is not available. Since AB does not provide us with such information, model-free algorithms are a perfect fit.

Due to the limitation of not having the transition model available, our two algorithms try to characterize a state through basis functions (features). In our thesis we propose a set of eight different basis functions that try to describe any state sufficiently. As mentioned before, RL algorithms develop a policy through repetition. We trained our Q-Learning agent through 20,000 iterations, while storing the information of the state, the action taken, the new state and the returned reward. Having stored our samples locally, LSPI can find a policy in offline mode, given a set of samples, without the limitation of the way they were gathered. During the process of training, the algorithms assign values to the weights of every feature. These values are the ultimate prerequisite for an agent to take an action.

After the training completion of both agents (Q-Learning, LSPI) in 34 different levels, we tested them in 20 more levels that we chose not to be a part of the training set, since the agent will

also face unknown levels during the AIBIRDS competition and we wanted to evaluate how they perform under these circumstances. Alongside the 20 unknown levels, the agents were all tested in the known levels. In addition, we tested the AI agent of the AngryBer team [5], that was placed second in 2014 during the competition, on the same levels and compared our results.

The results are really promising as both agents managed to solve the vast majority of the levels (unknown, known) and outperformed AngryBer agent in terms of level completion. A result worth mentioning, even in this introductory section, is that LSPI agent managed to complete 44 out of 54 game levels when the AngryBer solved 33.

1.3 Thesis Overview

The thesis is organized as follows:

- Chapter 2 will discuss the theoretical framework of reinforcement learning. Furthermore, it will describe Q-Learning and Least Squares Policy Iteration and their key features, thoroughly.
- Chapter 3 presents the motivation for our research, elaborates on the related work for the Angry Birds game and gives a brief introduction to the problem we faced and our approach.
- Chapter 4 describes the way we implemented both of our algorithms. We define what a state, action, new state, reward set is for the AB game. Moreover, key components such as basis functions are discussed thoroughly covering every detail which needs to be mentioned to portray our approach.
- Chapter 5 reviews the results of our agents on unknown and known game levels. In addition, they are compared with former participants of the competition and their overall performance is evaluated.
- Chapter 6 lists possible directions of future research.
- Chapter 7 mentions the contributions of the thesis.

Chapter 2

Background

2.1 Reinforcement Learning

In a reinforcement-learning (RL) problem, the agent is limited to learning the significance of an action through trial and error, while trying to maximize its reward. As the agent performs actions and gains more information, it faces a tough dilemma between exploration and exploitation. While training, the agent may have the knowledge that a certain action yields a decent reward. However, there is a high possibility that a better response exists in the action space, that is not tried yet, producing a higher reward. Finding the perfect balance between exploration and exploitation is a tough challenge that needs to be dealt carefully. The agent expresses a RL problem as a mathematical framework that is called Markov Decision Process (MDP) [4].

2.1.1 Markov Decision Process

A Markov Decision Process (MDP) is an efficient way to model a RL problem and is composed of 6 basic elements $\{S, A, P, R, \gamma, D\}$.

- S is the state space of our environment
- A is the action space of our agent
- P is the transition model $P(s'|s, \alpha)$, the probability that the process moves from state s into its new state s' when influenced by the action α
- R is the reward model, the reward $R(s, \alpha)$ when an action α influences a certain state s
- γ is the discount factor, $0 < \gamma < 1$, a variable the determines how much our algorithm would would care about distant future rewards, rather than the immediate rewards.
- D is the initial state distribution

State Value Function $\mathcal{V}$ and State-Action Value Function $\mathcal{Q}$

In order to describe what a state or state-action value function is, we must define what a policy function $\pi(s)$ is. A policy function is the way of acting when a certain state appears. The state value function $\mathcal{V}$ estimates the significance of the agent to be in a certain state with respect to particular policies. The state-action value function $\mathcal{Q}$ returns the value of an action α influencing a state s. Both functions are described by the following equations.

$$\mathcal{V}^{\pi}(s) = \mathcal{E}_{\alpha_t \sim \pi; s_t \sim P; r_t \sim R} \left(\sum_{t=0}^{h} \gamma^t r_t | s_0 = s \right) \tag{2.1}$$

$$\mathcal{Q}^{\pi}(s, \alpha) = \mathcal{E}_{\alpha_t \sim \pi; s_t \sim P; r_t \sim R} \left(\sum_{t=0}^{h} \gamma^t r_t | s_0 = s, \alpha_0 = \alpha \right) \tag{2.2}$$

However, in real-life environments the reward and transition from a state for an action may involve randomness. One way to overcome that issue and determine the [space: $\mathcal{V}$, $\mathcal{Q}$] is explained in the following paragraph.

Linear Approximation

Linear approximation is often used to estimate the $\mathcal{V}$,$\mathcal{Q}$ values, because of the absence of transition and reward model. To achieve that, it proposes the concept of basis functions-features and weights. Basis functions or features are a way to effectively model and summarize the reinforcement learning environment. Weights are used to measure the significance of each feature. Linear approximation estimates the [space: $\mathcal{V}$, $\mathcal{Q}$] values through the following equations, where ϕ are the features and w are the weights.

$$\mathcal{V}^{\pi}(s) = \sum_{i=1}^{k} w_i^{\pi} \phi_i(s) = \phi(s)^{\intercal} \mathbf{w}^{\pi} \tag{2.3}$$

$$\mathcal{Q}^{\pi}(s, \alpha) = \sum_{i=1}^{k} w_i^{\pi} \phi_i(s, \alpha) = \phi(s, \alpha)^{\intercal} \mathbf{w}^{\pi} \tag{2.4}$$

Temporal Difference Learning with Linear approximation

Since we modeled our RL environment and described the importance and meaning of [space: $\mathcal{V}$, $\mathcal{Q}$] values we then face the challenge to predict them. Temporal Difference [1] with Linear Approximation estimates the weights through the following sequence of steps shown in Algorithm 1.

Algorithm 1 TD with Linear Approximation

procedure TDLA$(D, \pi, k, \phi, \gamma, w_0, \alpha_0, \sigma)$

$//D$: Source of samples (s, α, r, s')

$//\pi$: Policy whose value function is sought

$//k$: Number of basis functions

$//\phi$: Basis functions

$//\gamma$: Discount factor

$//w_0$: Initial parameters

$//\alpha_0$: Initial learning rate

$//\sigma$: Learning rate schedule

$\tilde{w} \leftarrow w_0; \alpha \leftarrow \alpha_0; t \leftarrow 0$

for every sample$(s, \alpha, r, s') \in D(\pi)$ **do**

$\tilde{w} \leftarrow \tilde{w} + \alpha\phi(s)\left(r + \gamma\phi(s')^\intercal\tilde{w} - \phi^\intercal\tilde{w}\right)$

$\alpha \leftarrow \sigma(\alpha, \alpha_0, t)$

$t \leftarrow t + 1$

return $\tilde{w}$

2.1.2 Q-learning with Linear Approximation

Q-Learning with linear approximation updates the parameters of the architecture based on the temporal difference. The algorithm of Q-Learning is discussed in Section 4.3. However, here we will discuss its key components.

Episodic Learning

Q-Learning can be implemented using *episodes* to update its parameters. An *episode* is defined as a set of actions that result in a terminal state.

Learning Rate α

The learning rate or stepsize determines to what extent new information of our environment overrides the information already gained. A stepsize of 0 learns nothing and exploits only the old information and a stepsize of 1 makes the agent consider only recent information. To encourage

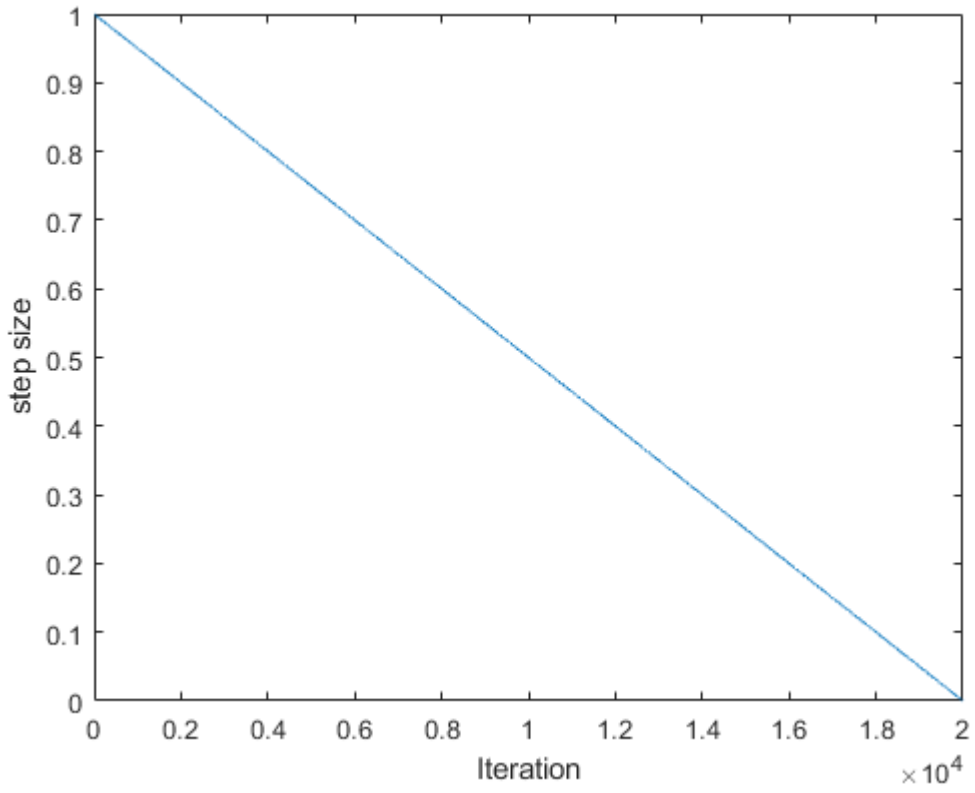

Figure 2.1: Step size for 20,000 iterations

learning in the early stages of our training and exploitation of the old information as the agent gathers information, we compute the step size based on the following equation and Figure 2.1 shows its diagram.

$$stepSize = \left(1 - \frac{Iterations\ completed}{Maximum\ iterations}\right) \tag{2.5}$$

Discount factor γ

Discount factor γ determines the importance of future rewards. A factor of 0 will make the agent consider only the current reward. In the contrary, a factor of 1 will make the agent search for long-term high reward, causing instabilities when our environment lacks terminal states. Typically, the value of the discount factor is between 0.9 and 0.999.

2.1.3 Least Squares Policy Iteration

In order to define how Least Squares Policy Iteration algorithm works, we must take a step back and describe Policy Iteration and Approximate Policy Iteration. Policy Iteration is a procedure that tries to discover the optimal policy by generating a sequence of improving policies. An improved policy in time step m, π_{m+1}, is at least as good as the π_m and is computed by the following equation.

$$\pi_{m+1}(s) = \arg\max_{\alpha \in \mathcal{A}} \mathcal{Q}^{\pi_m}(s, \alpha) \tag{2.6}$$

When there is no difference in the policy between two steps, the iteration has converged to the optimal policy. LSPI is an approximate model-free policy-iteration algorithm. It uses a set of samples to learn an optimal policy. Compared to Q-Learning, LSPI computes its parameters in a more effective way, using all the action-space at once through linear algebra computations. In addition, the algorithm is able to reuse the same set of samples to compute the parameters w until they reach a certain difference $e - tolerance$ in $||w - w'||$. In Section 4.4, we give the exact algorithm of LSPI.

Chapter 3

Problem Statement

3.1 Problem Statement

Angry birds is a video game where the player is challenged to go through a series of different levels, each one of them being composed of four basic elements. The first element is the pigs, whose elimination is the main purpose of the game. The second element is the birds, that fly from a long distance to come into contact with the pigs and kill them. The third element is the sling, where the birds board and fly to the desired target. The last element is the building structures, that are well placed for the best protection of the pigs from their wicked enemies, the Angry Birds. AB abides by the law of physics, which explain the reactions between the different elements, given a certain game state and a certain shot with the sling. Furthermore, birds are divided into five different categories/types, each one of them with a different ability: Red, Yellow, Blue, Black, White (Figure 3.2). Moreover, the building structures are formed of objects of different types with different properties: Wood, Ice, Stone, Hill, TNT (Figure 3.3).

A recent survey [2] concluded that after the prevalence of AI towards humans in various complex games, such as chess, go, dota and starcraft, the Angry Birds is considered the next AI milestone, where AI will beat humans next. Since 2013, the International Joint Conference on Artificial Intelligence (IJCAI) hosts the Angry Birds AI competition and the Man vs Machine challenge. In the Angry Birds AI competition, AI Agents from all around the world participate, capable of playing the game without any human intervention. The agent with the highest score (obtained from every level while playing) is the final winner. The agents with the four highest scores are qualified to the next phase, Man vs Machine challenge, where they come face to face with some of the best Angry Birds players. From the beginning of the competition to this day, the outcome of that battle has been heartbreaking for the AI side with seven consecutive defeats.

The AI agents compete on a particular set of levels, completely unknown before entering the competition. One level can be composed of any possible arrangement of the above-mentioned objects and pigs, creating a set of infinite possible combinations of game states (3.4). Taking this into consideration, it is safe to assume that an agent, which addresses the AB problem by predicting a small set of expected states and assigning a particular action, is naive and will struggle when an unforeseen state occurs. To complete a level successfully (killing all the available pigs) the agent has a finite number of tries (birds). The number and type of birds as well as the number of pigs, is random for every level.

Having many issues to resolve, when someone decides to create an AI agent for the Angry Birds game, the organizers of the competition provide all participants a basic game playing software that is implemented using Java, including a naive AI agent playing randomly and the following two vital components: (a) a computer vision component that can detect parts of the state of the game (object location, score etc.) and (b) a trajectory component which calculates the release point of the bird on the sling given a target location.

The objective of our research is to create an autonomous agent that outperforms the given AI agent of the basic gaming software that plays randomly, is capable of completing a majority of the sample training levels given by the organization and gives a fair battle against human players.

In the AIBIRDS competition of 2017, our team **Vale Fina 007** had participated with an early version of our agent. Despite the fact that our agent was in such a basic form, it managed to compete decently against other teams and was placed 6th among the 10 teams, surpassing the winners of previous competitions, like DataLab Birds and BamBirds. Figure 3.1 shows the detailed results of the AIBIRDS 2017 competition.

3.2 Related Work

During the seven years that the contest is taking place, several approaches have emerged covering a wide variety of scientific fields. Some AI agents worth mentioning are the:

IHSEV (https://bitbucket.org/polceanum/ihsev-aibirds/src/master/)

Eagle Wings (https://github.com/heartyguy/AI-AngryBird-Eagle-Wing

IHSEV team used mental simulations, where their agent creates an "Imaginary World" based on the information provided by the game state using physical laws that govern the object's motion. While the actual game takes place their agent runs simulations internally for the state that it has observed, calculating the best action that maximizes the reward. Their team is one of the most experienced, participating almost every year of the competition and winning the AI competition in August 2013 and being close to winning against humans in the Man vs Machine Challenge.

Grand Final			**Quarter Final 1**	
1. Eagle's Wing	355,700		1. IHSEV	261,600
2. IHSEV	275,110		2. S-Birds	147,120
			3. Condor	94,600
Semi Final				
			Quarter Final 2	
1. IHSEV	415,890			
2. Eagle's Wing	350,900		1. AngryHex	242,980
3. AngryHex	238,040		2. Eagle's Wing	175,510
4. PlanA+	225,780		3. Vale Fina 007	106,930
Quarter Final Ranking			**Quarter Final 3**	
1. IHSEV	261,600		1. PlanA+	172,410
2. AngryHex	242,980		2. DataLab Birds	97,100
3. Eagle's Wing	175,510		3. BamBirds	89,830
4. PlanA+	172,410		4. AngryBNU	0
5. S-Birds	147,120			
6. Vale Fina 007	106,930			
7. DataLab Birds	97,100			
8. Condor	94,600			
9. BamBirds	89,830			
10. AngryBNU	0			

Figure 3.1: 2017 AIBIRDS competition

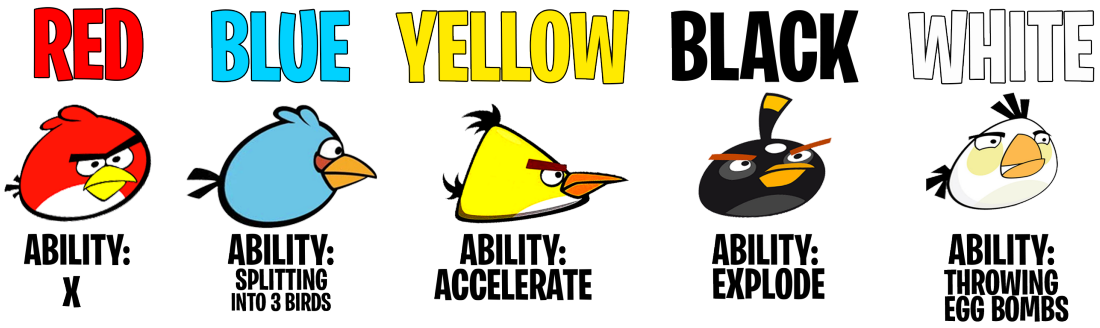

Figure 3.2: Types of birds and their abilities

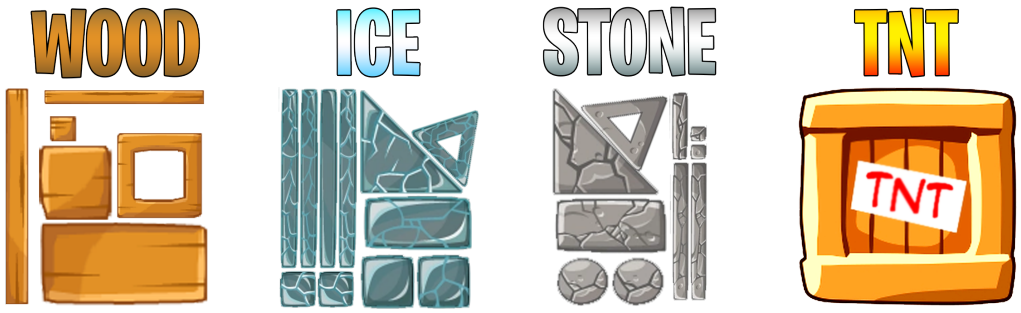

Figure 3.3: Types of building objects

Eagle Wings team won the AI Angry Birds competition for two consecutive years (2017, 2018) setting a very high standard for the other AI agents. Their agent utilizes the machine learning method xgboost (https://github.com/dmlc/xgboost) and deep reinforcement learning. The xgboost method implements the Gradient Boosting, a machine learning technique for regression and classification problems.

Figure 3.4: Characteristic samples of levels

Chapter 4

Our Approach

Having discussed the theoretical framework of our approach and the reasons we chose them, for the sake of coherence of our thesis, in this chapter we explain how we implemented the Q-Learning and Least Squares Policy Iteration algorithms.

4.1 Modeling the Game Scene

4.1.1 The State

The computer vision module of the basic game playing software provided by the competition, gives us information about the state of the game. By state, we define everything that exists in the game scene. Below there is a list of all the information obtained by the computer vision module that proved to be useful for our implementation.

1. Objects
    - Shape
        - Circle
        - Polygon
        - Rectangle
    - Type
        - Hill
        - Slingshot

  - RedBird
  - YellowBird
  - BlueBird
  - BlackBird
  - WhiteBird
  - Pig
  - Ice
  - Wood
  - Stone
  - TNT
  - Root
  - Ground
  - Unknown
- Area
- Location
- Angle
- Length
- Width

2. GameState
   - Level Selection
   - Episode Learning
   - Playing
   - Lose
   - Win
3. Current Score

To group and rank the objects of the state we used a very effective way designed by Nikolaos Tziortziotis, Georgios Papagiannis and Konstantinos Blekas [5].

The New State

The new state is the derivative of a state affected by an action (see Section 4.1.2). Figure 4.1 shows a random set of {current state, action, new state}.

Figure 4.1: State, Action, New State

4.1.2 The Action

By action, we define the object to be hit by a bird during a shot. To perform a shot, the agent drags the slingshot rubber to a certain release point. To estimate the release point of the slingshot rubber that will result in hitting a desired point in the game scene, the basic game playing software provides the following function (4.1), which abides by the Newton's classical laws of physics and estimates the parabolic path that the bird will take after getting released by the slingshot.

$$\theta = \arctan\left(\frac{v^2 \pm \sqrt{v^4 - g(gx^2 + 2yv^2)}}{gx}\right) \tag{4.1}$$

Where (x, y) are the normalized coordinates of the target point relative to the slingshot (using sling size as the scale), the gravity g is assumed to be 1 unit and v is the initial velocity. The above equation has two solutions for a given $\{v, (x, y)\}$ pair.

As it will be explained in detail in Sections 4.3, 4.4 we will not be thoroughly describing the decision making process at this part of our thesis, but it is important to note that when the decision is made we factor in the following variables.

Choosing the Target & Release Point

Since the objects cover a certain area in a state, hitting two different points in that area may produce different results-rewards. As we can see in Figure 4.2, hitting the same object in different *TARGET_POINTS*, produces a different new state and score. In addition, there are two different angles Eq (4.1) to reach any *TARGET_POINT*. Figure 4.3 shows the outcome of hitting the same *TARGET_POINT* with different *RELEASE_POINTS*. To determine *TARGET_POINT, RELEASE_POINT* we developed a simple, yet efficient, empirical policy.

Considering that eliminating all the *PIGS* is our ultimate goal to win a level, we examine all the *TARGET_POINTS* of an object, alongside with the points of the trajectory of the bird that will hit the object. However, our search of those trajectory points does not end when the bird meets

the object, contrariwise it begins for the points after the object. To avoid any kind of confusion, we must note here, that we do not calculate the trajectory of the bird after it hits an object considering its changes due to the collisions with the objects. We only consider the "imaginary" trajectory of the bird, assuming no collisions happen and the trajectory never changes. To make this more clear, we show in Figure 4.4 the above mentioned points and the information gained by them. With that information available, at this point, we are able to determine whether a trajectory of a {*TARGET_POINT, RELEASE_POINT*} pair meets a PIG or not. In that case we choose that pair to make the shot. If both of the target's point trajectories meet a *PIG*, then we choose the direct one. Finally, if no *TARGET_POINT* meets a PIG then we target the center of the object, considering it to be the safest choice.

Figure 4.2: Different target points for same object and their new state

Figure 4.3: Different Angles for same target point and their new state

Figure 4.4: Objects the "imaginary" trajectory meets, after the object-action

4.1.3 The Reward

In *Angry Birds* the score of a shot is an integer number, where $score \in [0, +\infty)$. The more objects destroyed/hit by a bird the highest the score is. In addition, eliminating pigs gives us a higher score than other objects and when a win occurs, there is a bonus added to the score for every bird that was not used.

4.2 Basis Functions - Features

The most vital component of our research is the basis functions $\varphi(s, \alpha)$. They are used by both of our algorithms Q-Learning (See section 4.3), LSPI (See section 4.4), as a key variable in their equations where $\{\varphi(s, \alpha) \in [-1, +1], \varphi(s, \alpha) \neq 0\}$. These functions are a considerable way of characterizing a specific action by assigning values in their basis functions (features) based on a state. However, this assignment is one of the toughest tasks of our research and constitutes a great challenge. To overcome this challenge, we distinguish basis functions (features) between two main categories. The reason of that categorization, is that an action-object can be hit with five different type of birds, each one of them with a special ability (Figure 3.2), affecting the game scene in a different way. We also include a special feature, the BIAS feature. In case our algorithms decide that all $\mathcal{Q}$ values need to increase or decrease, changing just the bias weight could achieve that. If that feature was not available to our algorithms, they would try to change the weights of all the other features in a strange and unknown way.

4.2.1 Global Features

In this section we will discuss the features that are not affected by the type of bird on the sling, as shown in Figure 4.6.

- **Number of Pigs Above**
  The number of pigs above the object-action in a certain area.
- **Number of Pigs Right**
  The number of pigs to the right of an object-action in a certain area.
- **Objects Before Pig**
  The number of objects between the object-action and its nearest pig.
- **Weight Supporting**
  The number of objects the object-action supports.
- **Distance of the Nearest Pig/TNT to the Right**
  The distance between the object-action and the nearest PIG/TNT to its right.
- **Distance of the Nearest Pig/TNT**
  The distance between the object-action and the nearest PIG/TNT.

4.2.2 Customized Features

Customized features refer to those that are affected by the type of bird on the slingshot.

- **Trajectory Meets Pig/TNT**
  Whether the trajectory of the bird meets a pig after meeting the object-action (Figure 4.2). The *BLACK BIRD* and *WHITE BIRD* have the ability to explode, affecting a big portion of the game scene. As a result, when a pig is inside the radius of that explosion, we assume that the object-action is part of the trajectory that meets a pig, enabling that feature (Figure 4.5).

- **Impact**
  The ability of the object-action to be penetrated/destroyed by the type of bird on the slingshot as summorized in Table 4.1.

	RED	BLUE	YELLOW	BLACK	WHITE
Wood	NO	NO	YES	YES	YES
Ice	NO	YES	NO	YES	YES
Stone	NO	NO	NO	YES	YES

Table 4.1: Impact feature

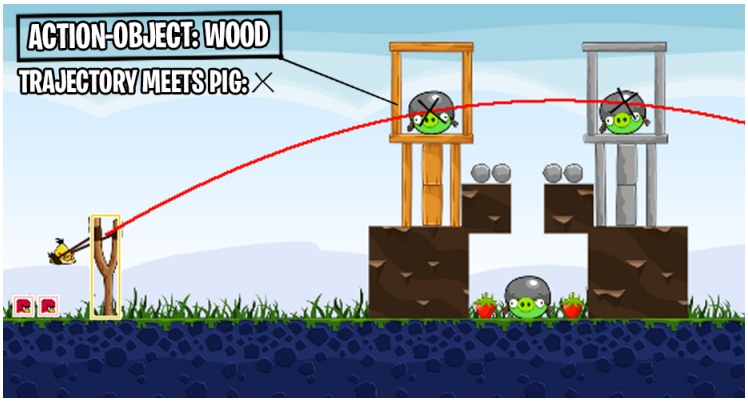

Figure 4.5: "Imaginary" trajectory meets Pigs, feature

In addition, all the features that are dependent on the information concerning the pigs are also customized based on the distance from the nearest pig. The importance of that customization is

shown through the following example. Suppose that we have an object which has the *TRAJECTORY MEETS PIG* feature true, in other words the "imaginary" trajectory meets a pig. In that case, we would assign the maximum value a feature can have, 1.0. However, the pig is in a really far distance from the object, making it impossible to be affected by any collision of the bird to the action-object. The reward of that action would be very low due to their distance resulting in affecting negatively that feature. The reason for that would be that our agent sees a very high value for that feature and really low reward, so when the time of updating the weights comes, the value of the weight for that feature would decrease. To solve that problem, through excessive examination of the training process, we found the spot of balance where features are evaluated for the properties they represent and not for the long distance of the nearest pig. However, that assignment had to be done in a very delicate way, since we did not want to make our architecture completely reliant to that distance.

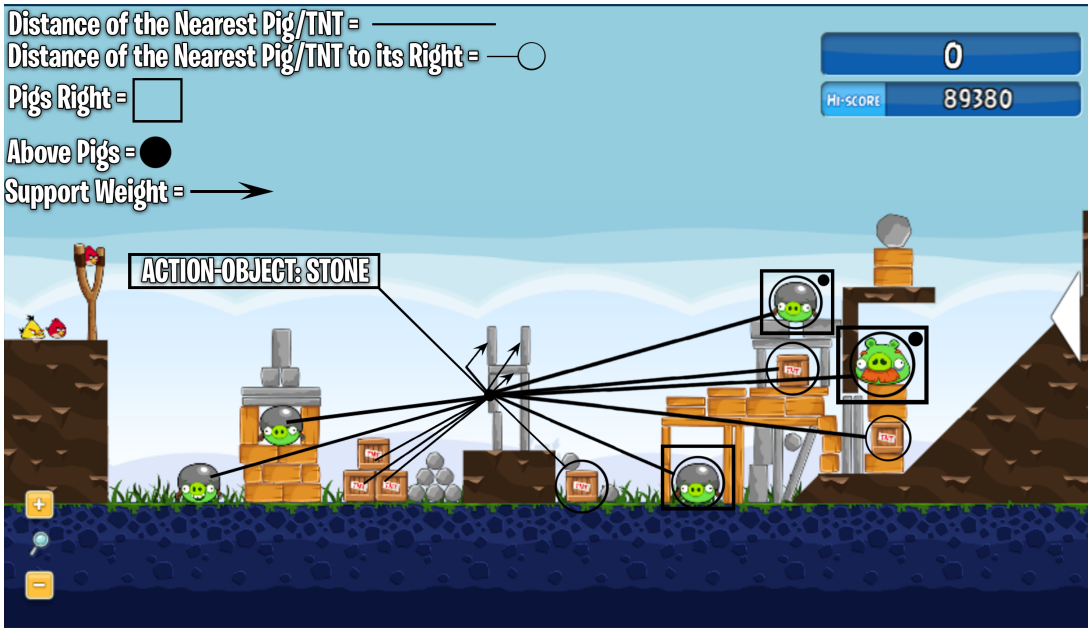

Figure 4.6: Five out of eight features that characterize an action.

4.3 Q-Learning with Linear Approximation

Up to this point, we have designated all key factors of the Q-Learning algorithm and we are ready to apply them. The process of learning starts by initializing our weights with the values {-1,+1}, randomly. Every time an *EPISODE* ends (no birds are left available) our agent incorporates the stored feedback and updates the weights. The *Learning Rate* is calculated by Equation 2.5 and its plot in Figure 2.1 shows the way it changes over time. After extensive experimentation we set $\{Discount\ Factor \in [0,1], Discount\ Factor = 0.95\}$. We train our algorithm for 20,000 time steps and store the information of the {state, action, reward, new state} set locally in order to be used by the LSPI algorithm. One of the reasons we used the stored samples for LSPI is to compare them in a fair way, during the testing process, and none of them gets trained through better samples. In Algorithm 2 there is a pseudo-code for the actions described earlier. In the course of Q-Learning training, the following steps take place, in the following order. Firstly, we detect the game scene through the vision module (provided by the competition) to determine our state. Afterwards, we set the *TARGET_POINT* and *RELEASE_POINT* for every action through the strategy discussed before. Next, we set the features for all the actions. After this assignment, we choose our action using the ϵ-greedy algorithm. That algorithm decides if Q-Learning will try to explore the state-action space by choosing a random action or exploit the information already gained by choosing the action with the highest $\mathcal{Q}$ value. In order to encourage exploration in the early stages of training and exploitation as it proceeds, the ϵ probability is calculated by the following equation. When the action is selected, the agent releases the bird from the slingshot with the right way, in order to hit the object-action. Finally, after the shot is completed the vision module detects the new state, reward and we store that sample for future manipulation by LSPI.

$$\epsilon = \left(1 - \frac{Iterations\ completed}{Maximum\ iterations}\right) \tag{4.2}$$

Algorithm 2 Q-Learning implementation

procedure Q-LEARN($MaxIter, k, \phi, \gamma, w_0, \alpha_0, \sigma$)

//$MaxIter$: The number of total iterations

//k : Number of basis functions

//ϕ : Basis functions

//γ : Discount factor

//w_0 : Initial parameters

Initialize $w = w_0$

for MaxIter **do**

start a new game level

s = detect-game-scene

while s is not terminal state **do**

//Setting our features

for all actions of the state **do**

set $\phi_i(s, \alpha)$ for every $i = [1, 2, ..., k]$

//Getting the *action* using $e - greedy$ algorithm

if $random < \left(1 - \frac{currentIteration}{MaxIter}\right)$ **then**

$action = randomAction()$

else

$action = \arg\max_{\alpha \in A}\{\phi(s, \alpha)^\intercal w\}$

agent.shoots()

reward = game.Score()

s' = detect-game-scene

//Incorporate feedback of the episode and update weights

$stepSize = \left(1 - \frac{currentIteration}{MaxIter}\right)$

for all $i = 1, ..., k$ **do**

$w_i \leftarrow w_i + stepSize\phi_i(s, \alpha)\left(r + \gamma max_{a' \in \mathbb{A}}\phi(s', \alpha')^\intercal w - \phi(s, \alpha)^\intercal w\right)$

$s = s'$

store sample$(s, \alpha, reward, s')$ for LSPI usage

return $\tilde{w}$

4.4 Least-Squares Policy Iteration

The idea of LSPI is that it uses all the samples to generate a certain policy. However, due to the fact that it can not satisfy the $\mathcal{Q}$ function of every sample, it tries to satisfy them in a least-squares sense. In more detail, it tries to find a solution that minimizes their squared error. The key components of LSPI are discussed in Section 2.1.3. Below we analyze how LSPI computes the weights.

Algorithm 3 Least Squares Policy Iteration algorithm

procedure LSPI($\mathbf{D}, \phi, \gamma, e$)

$//\mathbf{D}$: Sample set

$//\phi$: Basis functions

$//\gamma$: Discount factor

$//e$: Tolerance

Initialize $w' = 0$

repeat

$w \leftarrow w', A \leftarrow 0, b \leftarrow 0$

for every sample$(s, \alpha, r, s') \in \mathbf{D}$ **do**

$\alpha' = \arg\max_{\alpha'' \in \mathbb{A}} \{\phi(s', \alpha'')^\intercal w\}$

$A \leftarrow A + \phi(s, \alpha)\left(\phi(s, a) - \gamma\phi(s', \alpha')\right)$

$b \leftarrow b + \phi(s, \alpha)r$

$w' \leftarrow A^{-1}b$

until $||w - w'|| < e$

return w

Chapter 5

Results

After 20000 iterations Q-Learning and LSPI have converged to their final weights through their algorithms. Figure 5.1 and 5.2 show their final weight values and offer a way to understand the significance of each feature.

At this point, as discussed in Section 2.1.1, we are able to find the action that maximizes the expected reward given an action by solving Equation 2.4. On the grounds that the process of training has been completed, we can test both agents (Q-Learning, LSPI) in the angry birds levels to examine the quality of our results.

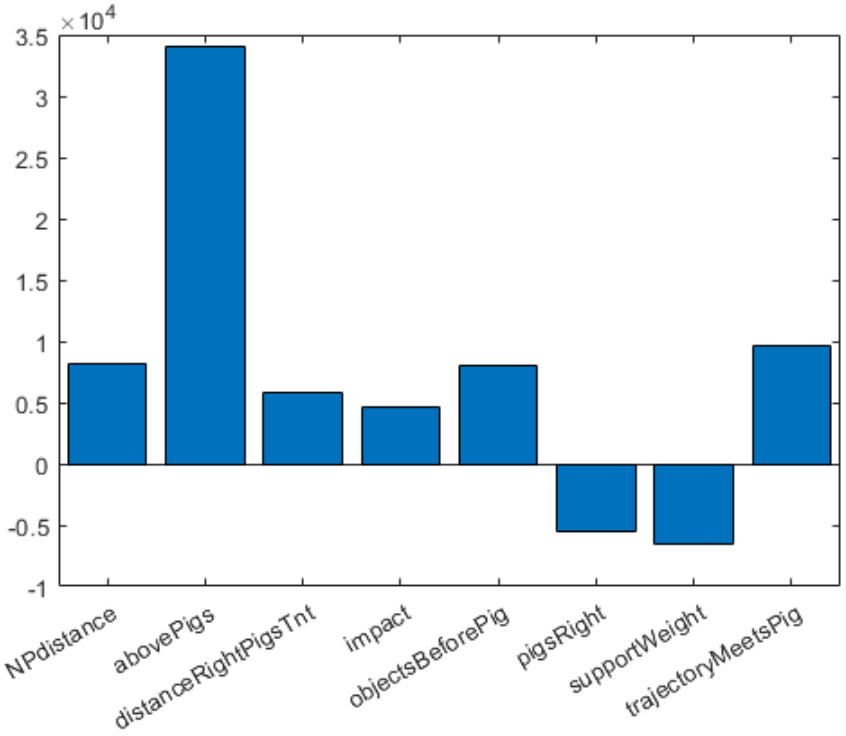

Figure 5.1: Q-Learning converged weights

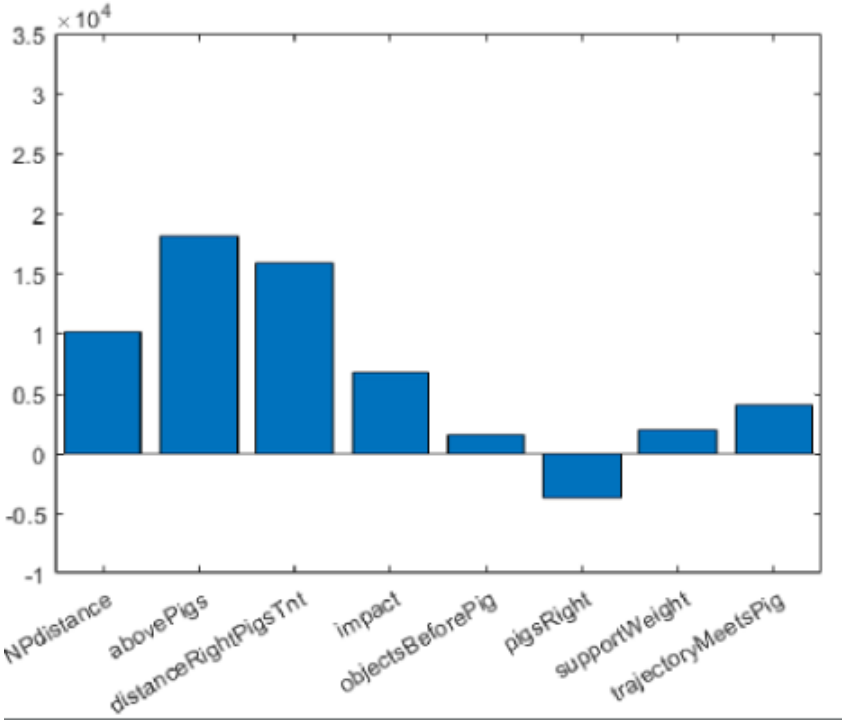

Figure 5.2: LSPI converged weights

Angry Birds game has three *EPISODES* each one of them consisting of 21 levels, as shown in Figure 5.4. *EPISODE 1* game levels tend to be easier, forgiving bad shots that do not help the agent clear the level faster, due to the fact that most of the levels in that *EPISODE* can be cleared with one shot. However, as the levels progress, difficulty goes up.

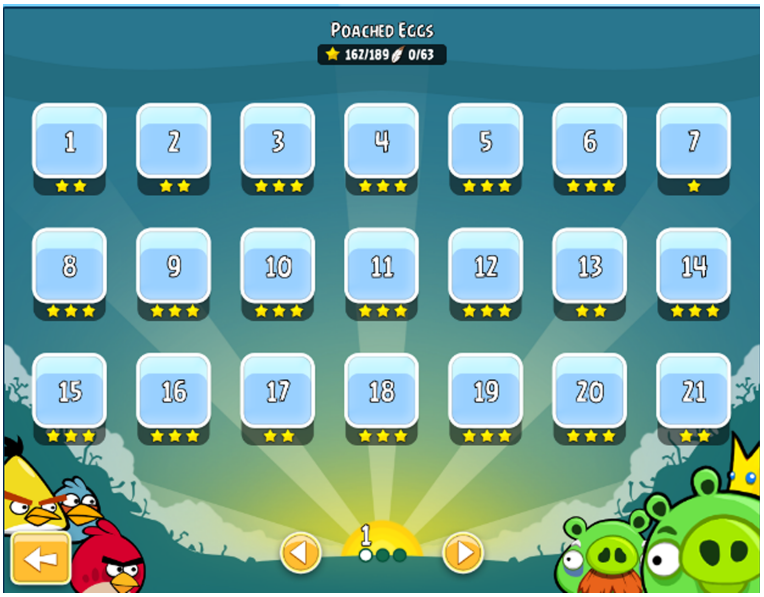

Figure 5.3: An Episode of 21 Levels

EPISODE 2 game levels are really harder. In most levels, birds are placed and shot by the slingshot in such order that if one of them targets a "bad" object, then the remaining birds are not able to recover from that bad action and complete the level. Fiure 5.6 is a perfect example of that. Due to far distance between the three pigs and the number of birds available (three), if the first bird fails to eliminate a pig, then the remaining two are not able to eliminate the three pigs of the scene considering the structure of the objects.

EPISODE 3 is composed of the most difficult levels in the Angry Birds game. Game levels of that *EPISODE* share the same difficulty described before, but in this *EPISODE* a "bad" object is not the only issue that can result to a defeat. Supposedly, we have chosen a "good" object, if the bird hits the "good" object in a "bad" spot, then this can be enough for the agent to fail the level. Figure 5.7 shows that exact case. In addition, *EPISODE 3* game levels demand great reasoning on what object to shoot based on the type of bird on the slingshot. In the level of Figure 5.8 if the agent wastes the *BLACK BIRDS* to destroy the pigs protected by wood, then it will be left with *YELLO WBIRDS* trying to destroy pigs protected by stone. This is extremely ineffective, since *BLACKBIRD* can destroy everything, but *YELLO WBIRD* can not penetrate stone and is really effective towards wood.

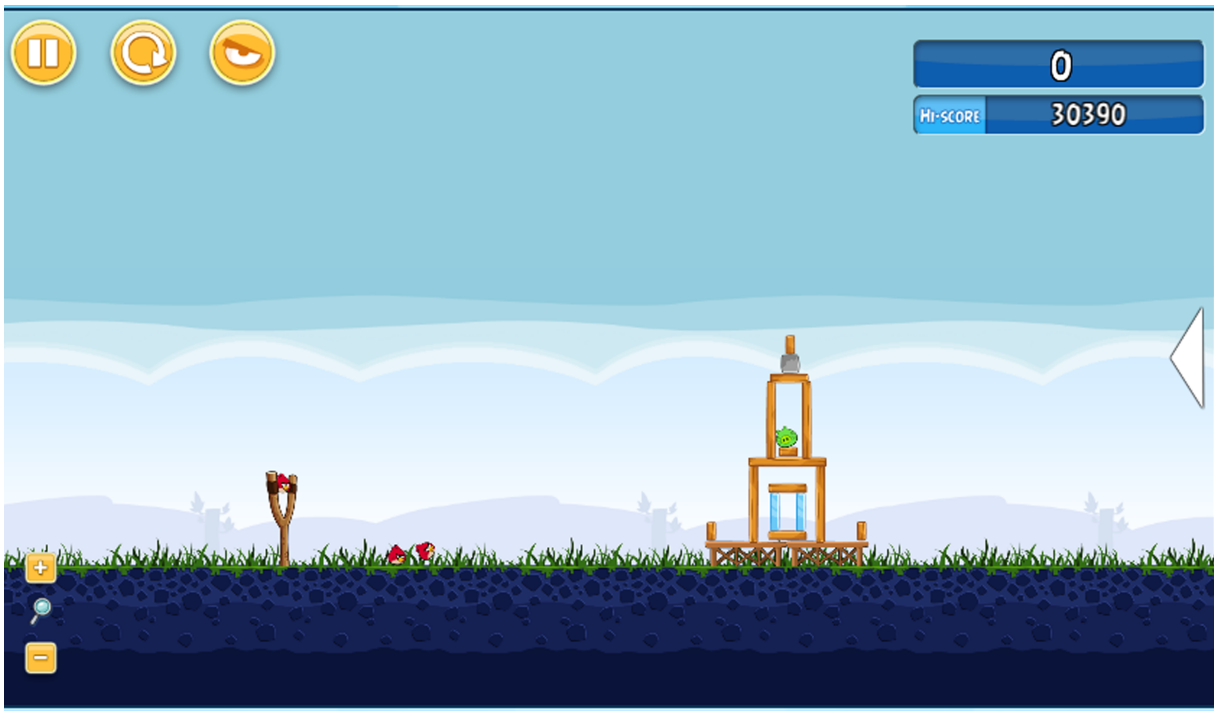

Figure 5.4: Level 1 of Episode 1

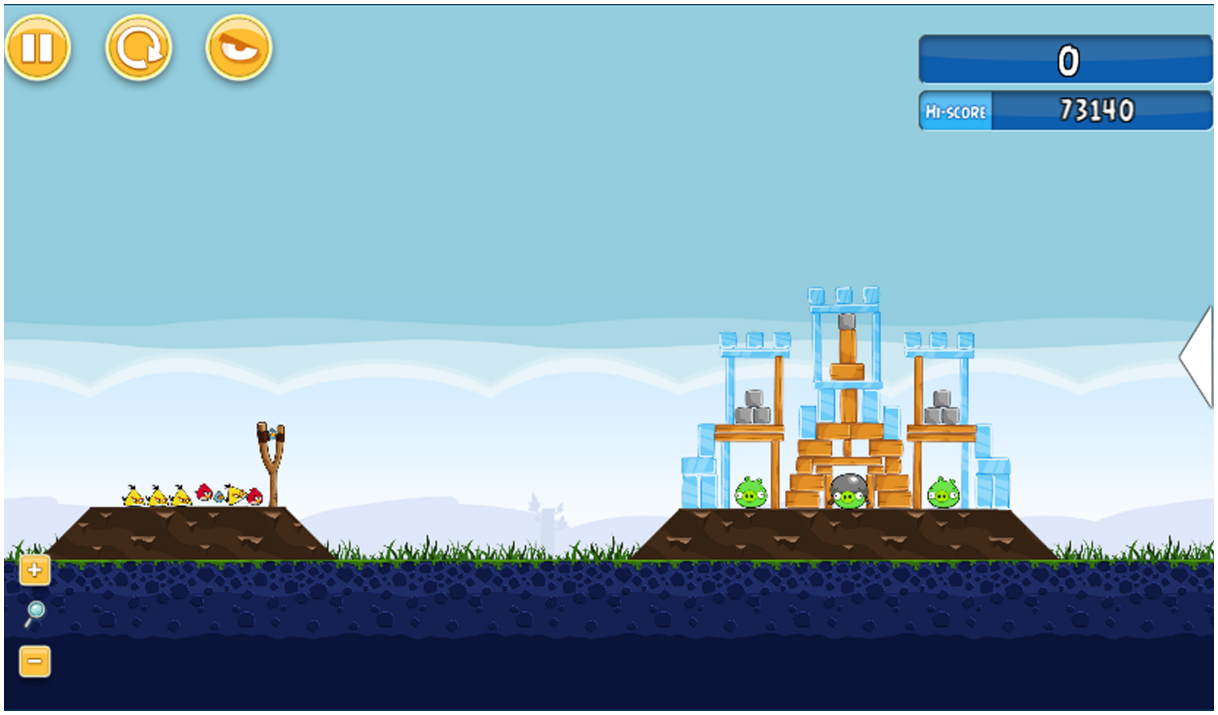

Figure 5.5: Level 21 of Episode 1

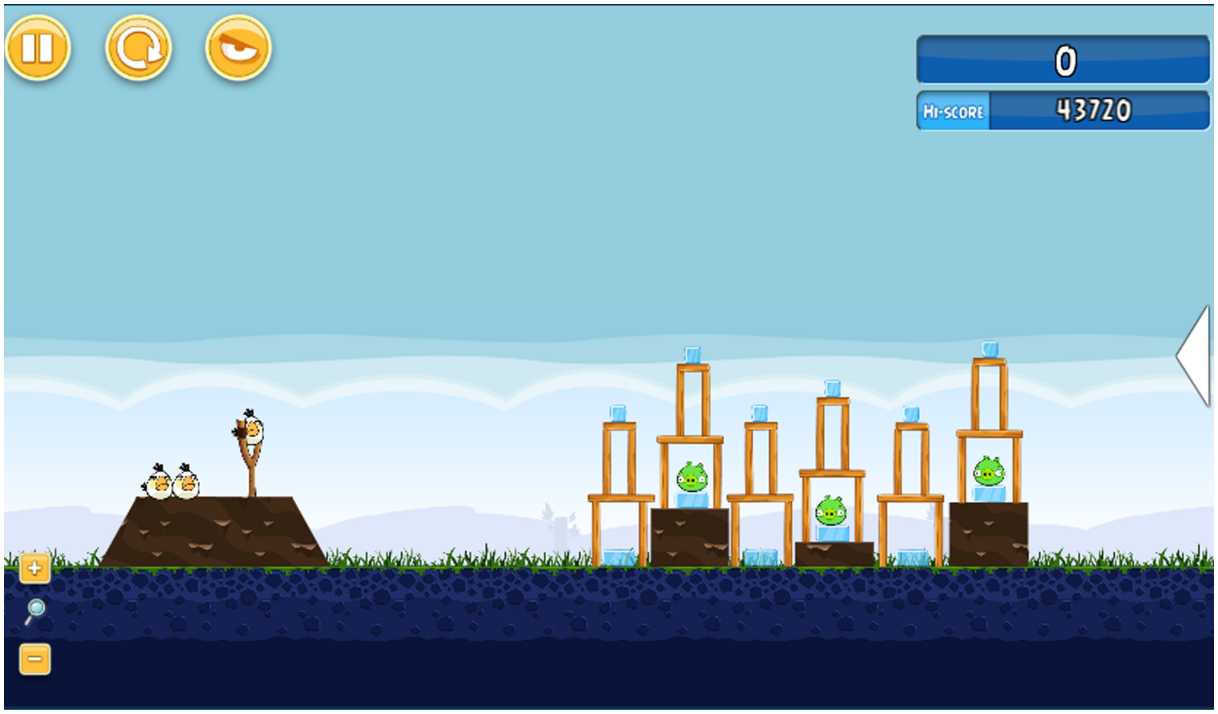

Figure 5.6: Level 17 of Episode 2

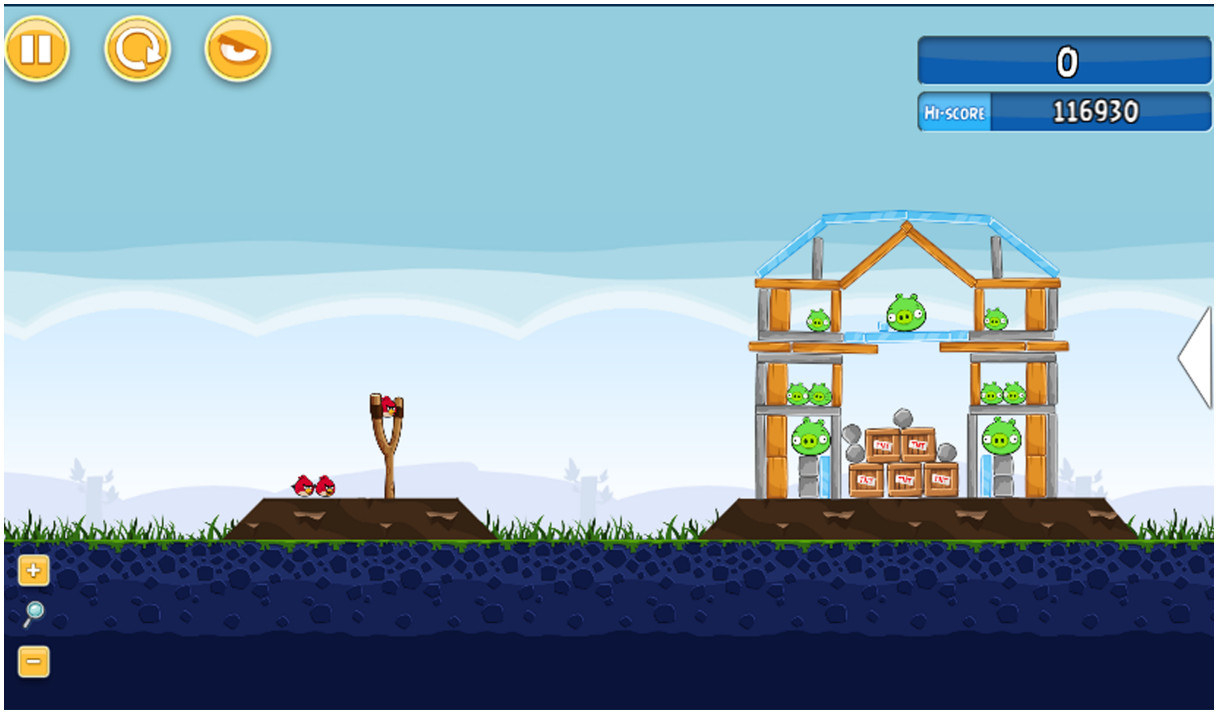

Figure 5.7: Level 5 of Episode 3

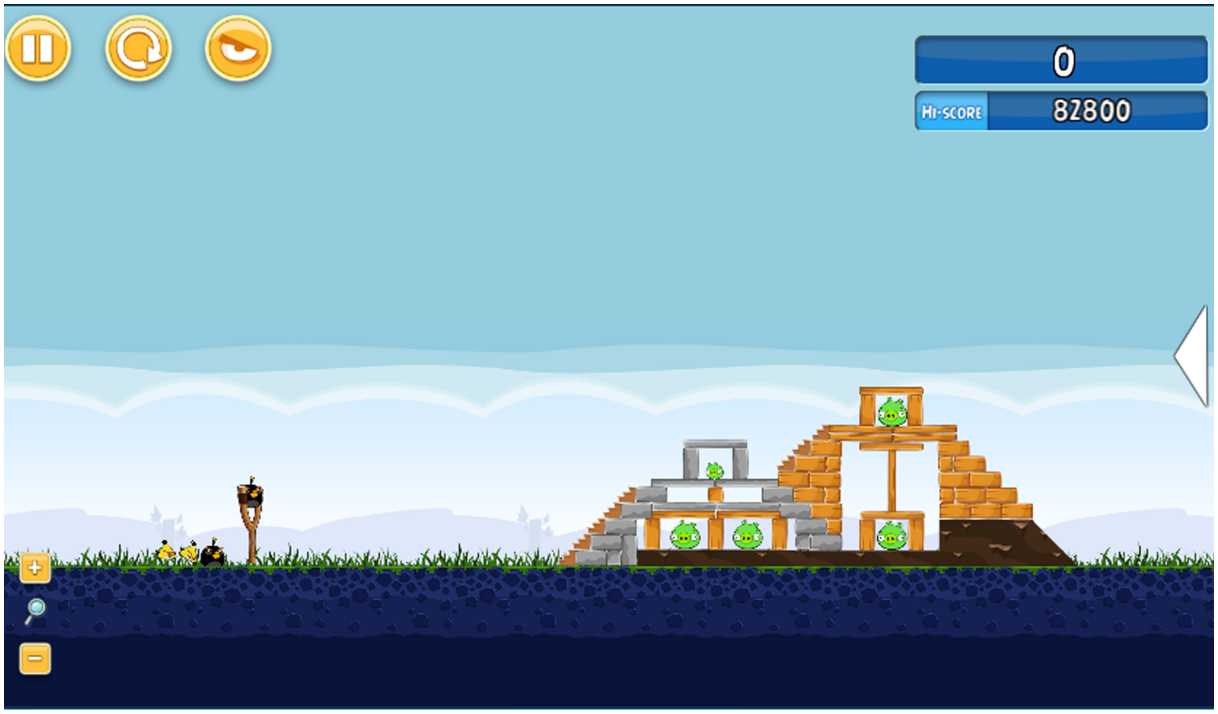

Figure 5.8: Level 17 of Episode 3

One thing worth mentioning at this point, is that during the training, we deliberately chose not to use some of the game levels, since we wanted to evaluate how our agents perform on completely unknown environments (states). These levels are marked, in the following result tables, with the (*) symbol above their number. In addition, we tested our agent on the same levels of the AngryBer's team, which is uploaded on the AIBIRDS forum site for anyone interested, and compared our agents to it. The AngryBer agent participated in the competition in 2014 and lost in the grand final placing second (Figure 5.9).

Results after Q2		Quarter Final 1	
1. DataLab Birds	423280	1. DataLab Birds	346260
2. PlanA+	372810	2. Angry BER	224860
3. s-Birds Avengers	361770*	3. Impact Vactor	173710
4. Angry Dragons	317300	4. s-birds Avengers	167860
(5. Naïve	302710)		
6. Impact Vactor	298390		
7. Angry-Hex	294170	Quarter Final 2	
8. IHSEV	292380		
9. Angry BER	253820	1. PlanA+	360920
10. BeauRivage	238080	2. IHSEV	277530
11. RMIT Redbacks	188890	3. Angry-HEX	129610
12. Auto Lilienthal	0*	4. Angry Dragons	78970
Semi Final		Grand Final	
1. DataLab Birds	232790	1. DataLab Birds	406340
2. Angry BER	206680	2. Angry BER	243880
3. PlanA+	206620		
4. IHSEV	93100		

Figure 5.9: AIBIRDS 2014 results in red for the AngryBer Team

Below there are summary tables of every *EPISODE* for the overall wins, overall defeats of an agent. Tables also show the number of wins and defeats of the agents in the unknown levels (levels not used in training). Since the training was performed on Q-Learning and LSPI agent, we assign values only to those two.

Episode 1,2,3	Random	Q-Learning	Lspi	AngryBer
wins	14	37	44	33
defeats	40	17	10	21

Table 5.1: All episodes gathered

As shown in Table 5.1 LSPI proved to be the best algorithm, completing more levels than any other algorithm. Following, Q-Learning completed 37 and AngryBer completed 33. In addition, Tables 5.2, 5.3, 5.4 show the results of every episode separately and the wins and defeats of unknown levels for Q-Learning and LSPI.

Tables 5.5, 5.6 and 5.7 give a detailed description for the reward of every game level in every episode. Rewards marked as "HIGH" mean that a high-score was achieved during that try, surpassing our human effort, when we tried to unlock every level manually to be available for the training of our agents. Levels marked as "SOFTWARE ERR." are those that we were unable to test our agents due to vision module failure.

Episode 1	Random	Q-Learning	Lspi	AngryBer
overall wins	12	17	19	19
overall defeats	9	4	2	2
unknown lvl wins		5	6	
unknown lvl defeats		2	1	

Table 5.2: Episode 1 Summary

Episode 2	Random	Q-Learning	Lspi	AngryBer
overall wins	2	12	17	11
overall defeats	16	6	1	7
unknown lvl wins		6	8	
unknown lvl defeats		2	0	

Table 5.3: Episode 2 Summary

Episode 3	Random	Q-Learning	Lspi	AngryBer
overall wins	0	8	8	3
overall defeats	15	7	7	12
unknown lvl wins		3	3	
unknown lvl defeats		2	2	

Table 5.4: Episode 3 Summary

Episode 1	Random	Q-Learning	Lspi	AngryBer
Level 1	win 29150	win 27600	win 27630	win 28360
2	win 43330	win 53600	win 52530	win 52180
3	win 40410	win 40500	win 40500	win 41190
4*	win 28440	win 19360	win 19460	win 28650
5	win 39770	win 55300	win 41500	win 65380
6*	defeat 9860	win 26370	win 36470 high	win 35290
7*	defeat 17830	defeat 18580	win 29530 high	win 30770
8*	win 29240	win 28380	win 23320	win 36490
9	win 30050	win 41500	win 28460	win 23030
10	defeat 26000	defeat 8300	defeat 27670	win 34500
11	win 29640	win 46830	win 47380	win 42930
12*	win 56970	win 54300	win 53700	win 58050
13*	win 29450	defeat 13530	defeat 7990 S.eR.	defeat 18970
14	defeat 35560	win 65640	win 55640	win 70050
15	win 46520	win 45400	win 50060 high	win 47850
16	defeat 39530	win 65310	win 64460	win 56780
17*	defeat 30090	win 41130	win 47130	win 43420
18	defeat 33600	win 46980	win 43750	win 35800
19	win 27660	defeat 11600	win 34980	win 36060
20	defeat 17740	win 52430	win 39950	win 35220
21	defeat 44750	win 67440	win 63140	defeat 42490

Table 5.5: Episode 1 Results

Episode 2	Random	Q-Learning	Lspi	AngryBer
Level 22	defeat 40110	win 55620	win 56180	win 52370
23*	software err.	software err.	software err.	software err.
24	defeat 68140	defeat 81440	win 88810	win 90060
25*	defeat 15340	defeat 24950	win 52940	defeat 20060
26	defeat 43330	win 83740	win 67490	win 69600
27	defeat 17840	win 59580	win 50350	win 54340
28	win 42620	win 47700	win 43990	win 49420
29	defeat 33710	win 47400	win 49700	win 55990
30*	defeat 6220	win 46740	win 20080	defeat 19200
31*	win 40800	win 43870	win 43870	defeat 10100
32	defeat 32400	defeat 90120	win 78540	defeat 74800
33*	defeat 35020	win 32920	win 35910	defeat 24850
34*	defeat 41650	win 66910	win 69640	win 64040
35*	defeat 29860	win 42790	win 44730 high	win 51510
36*	software err.	software err.	software err.	software err.
37	defeat 31760	defeat 19490	defeat 37020	win 55750
38*	defeat 26740	win 34520	win 30540	defeat 23130
39*	defeat 22680	software err.	software err.	software err.
40	defeat 8860	win 35120	win 36450	win 48560
41*	defeat 15500	defeat 14400	win 43340	defeat 24340
42	defeat 32040	defeat 43760	win 59090	win 65960

Table 5.6: Episode 2 Results

Episode 3	Random	Qlearning	Lspi	AngryBer
Level 43*	defeat 20980	win 57850	win 58190	win 47060
44	software err.	software err.	software err.	software err.
45*	software err.	software err.	software err.	software err.
46*	software err.	software err.	software err.	software err.
47	defeat 22390	win 108040	defeat 3170	defeat 26340
48	defeat 35980	defeat 35280	win 52700	win 49830
49	defeat 27500	win 46410	defeat 32500	defeat 44130
50	defeat 42830	win 79890	win 79460	defeat 63690
51*	defeat 44130	win 46510	win 46510	win 36880
52*	defeat 35060	win 45290	win 44410	defeat 24910
53*	software err.	software err.	software err.	software err.
54*	software err.	software err.	software err.	software err.
55	defeat 20470	defeat 22600	win 28980	defeat 17120
56*	defeat 30940	defeat 33770	defeat 32690	defeat 31120
57*	defeat 7130	defeat 37860	defeat 58550	defeat 11790
58	defeat 38070	win 59060	win 56140	defeat 43370
59	defeat 19530	defeat 31110	defeat 38330	defeat 48410
60	defeat 46460	defeat 26640	defeat 32560	defeat 49160
61	defeat 48210	defeat 22690	defeat 25270	defeat 21280
62*	software err.	software err.	software err.	software err.
63	defeat 40130	win 87140	win 100830	defeat 89710

Table 5.7: Episode 3 Results

Chapter 6

Conclusions

6.1 Discussion

Our results indicate the prevalence of LSPI in terms of level completion. The advantages of LSPI compared to Q-Learning are the following:

- Manipulation of the set of samples repeatedly generating a better policy each time
- More effective usage of a state by examining all the action-space.
- Independent of the stepsize, factor causing that causes instabilities, as well as the learned approximations diverging to infinity.
- Unaffected by the distribution and the order of the samples presented to the agent.

Due to the fact that LSPI had a 81% percentage of level completion and Q-Learning 68% it is safe to conclude that the reasons above caused that difference in the percentage of success.

6.2 Future Work

As for future work, we propose three important directions for future studies, aiming to achieve the ultimate goal of an AI agent to win the best human players in the Angry Birds game.

Vision Module

Although the competition provides us with an excellent vision module to detect the game scene, in some occasions due to the disorder of the game scene when affected by a shot, the vision module is unable to detect certain objects. For example, when an object is hit by a bird, but is not destroyed, there is a slight change in its color and there are a few cracks on its surface. Those cracks are misinterpreted by the vision module as the edge of an object and considers that object as two objects. That issue causes wrong states of the game scene, affecting negatively the training of an agent. Another issue was the slingshot not being detected effectively, resulting in pausing our training process, since the agent was trying to detect its right place. Our solution to that issue was to avoid that kind of levels for training with the trade-off of limiting our variety of different states. For that reason we suggest future works to focus on that matter.

Level Generation

Reinforcement-Learning algorithms benefit from the diversity of the states that they are trained on. In the Angry Birds game the available set of training is 54 game levels. In order to develop an RL agent that completes any unknown level, the set of levels has to enlarge by a great volume. The AIBIRDS organization also hosts the AIBIRDS CIG Level Generation Competition which is a good starting point to that direction.

Bibliography

[1] Klaas Apostol. *Temporal Difference Learning.* SaluPress, 2012.

[2] Katja Grace, John Salvatier, Allan Dafoe, Baobao Zhang, and Owain Evans. When will AI exceed human performance? evidence from AI experts. *CoRR*, abs/1705.08807, 2017.

[3] Michail G. Lagoudakis and Ronald Parr. Least-squares policy iteration. *J. Mach. Learn. Res.*, 4:1107–1149, December 2003.

[4] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994.

[5] Nikolaos Tziortziotis, Georgios Papagiannis, and Konstantinos Blekas. A bayesian ensemble regression framework on the angry birds game. *CoRR*, abs/1408.5265, 2014.

[6] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992.