Developing an AI controller for Ms. Pac-Man

Final Report for CS39440 Major Project

Author: Joshua Hodges (joh55@aber.ac.uk)
Supervisor: Dr Richard Jensen (rkj@aber.ac.uk)

30th March 2018

Version 2.0 (Final)

This report is submitted as partial fulfilment of a BSc degree in Computer Science (G400)

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB
Wales, UK

Declaration of originality

I confirm that:

- This submission is my own work, except where clearly indicated.
- I understand that there are severe penalties for Unacceptable Academic Practice, which can lead to loss of marks or even the withholding of a degree.
- I have read the regulations on Unacceptable Academic Practice from the University's Academic Quality and Records Office (AQRO) and the relevant sections of the current Student Handbook of the Department of Computer Science.
- In submitting this work, I understand and agree to abide by the University's regulations governing these issues.

Name ……Joshua Hodges

Date ……30th March 2018

Consent to share this work

By including my name below, I hereby agree to this dissertation being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Name ……Joshua Hodges

Date ……30th March 2018

Acknowledgements

I am grateful to Richard Jensen, my Major Project tutor for the guidance that he provided on this project.

Abstract

In recent history there has been a great rise in the research and advancements of Artificial Intelligence and Machine Learning and how it can be used to solve dynamic real time problems, such as being used to classify objects and scenarios when applied to the problem of autonomous vehicles. Machine learning has come a long way because of the great increase in computer processing power and the vast amounts of data now available through sources, such as the blockchain where data is now decentralized.

As is the case with playing the arcade game of Ms. Pacman, because of its increase of difficulty over the popular Pacman arcade game by means of sudo randomness added to the ghosts and an already implemented reward system, it has become a suitable testbed to test the effectiveness of various Artificial Intelligence techniques.

This report will detail the research conducted for the selection of a suitable technique and how the Reinforcement Learning technique of Artificial Intelligence and Machine learning can be applied as a controller for the game Ms. Pacman. For the controller a combination of Q Learning and Function Approximation was used, along with State Generalization. The Agent on the whole learnt quickly on how to navigate the maze and which moves would provide it with the greatest reward.

Two experiments were conducted on the final program, firstly changing the constants of the Q Learning Equation and secondly increasing the amount of training the agent receives before the graphically displayed game.

Table of Contents

1. Background, Analysis & Process

The purpose of this Major Project is to develop an AI controller for the non-deterministic game (agents can act at random) of Ms. Pac-Man, which is a difficult game for human players let alone an AI controller. This is because of the randomness of the ghosts, unlike Pac-Man in which the ghosts follow a specific pattern. This allows the controller and human players to pick up on these patterns and use them to avoid ghosts when playing the game.

1.1. Background

1.1.1. Ms. Pacman

The game of Ms. Pacman was created in 1982 as a sequel to the popular arcade game of Pacman. The main purpose of the game is to eat all the pills in the game while avoiding ghosts to attain the highest score possible. Along with a change in the name, this sequel brought along many improvements, such as better graphics, additional mazes and the addition of an Artificial Intelligence mindset for the ghosts. In Ms. Pacman, the ghosts have the opportunity to move randomly. The red and pink ghosts can move at random during the first few seconds of each round up until the first reversal of the game. Both the blue and orange ghosts remain in the same pattern which they followed in the previous Pacman game, up until (as with the other ghosts) the first reversal.

The latter brought an increase to the difficulty of the game because the ghosts no longer followed the patterns they once did in the previous game and now had a semi-random behaviour implemented. This non-deterministic behaviour is what provides the challenge of creating a well performing agent for Ms. Pacman. Unlike most games, Ms. Pacman has no conclusive ending state and a winning state can be largely different to the last achieved winning state. This introduces the need for smaller goals in the game, such as eating all the pills while avoiding the ghosts.

1.1.2. Interest and Motivation

Having previously done modules in both AI and Machine Learning and with the big boom in interest and functionality of these topics, not only in the general field of computer science but in other industries this project interested me most, as it allowed me to develop skills which will be of great use in the professional environment. This, along with my general fascination and curiosity with these topics and with my interests in Video Games and the Gaming Industry, the Major Projects for selection with elements of these topics were high on my list for favourites and those that would be the most enjoyable.

The projects such as developing a football game with AI for the opposing team and the visual doom agent where just some of those which made my short list. After seeing the advancements in AI and Machine Learning in the industry, such as the work done by Google and OpenAI, I chose a project more heavily based on AI rather than the actual development of a game. This was done so that more time could be spent on the research and implementation of the AI, hence the decision of this project.

1.1.3. Background Preparation

Before the start of the project I conducted research on several different technologies and algorithms in AI and Machine Learning in Games. More specifically, projects using

OpenAI's gym which includes many ways to create AI for various arcade games. However, the method most documented in this area was Q Learning[1] (this is one of the many reasons for using this method). Deepmind used a Q-learner to play Atari games[2], however a neural network was used instead of a linear equation. Microsoft had also achieved a perfect score with their AI[3] and this was done using an aggregator to decide which move to make based on the moves towards pills and away from ghosts and the distances from them.

I attended two online AI courses at Udacity[4] and Edx[5] respectively however the only the relevant lectures and quizzes were completed in reinforcement learning. This was helpful for some insight in what can be achieved with reinforcement learning and how using its different techniques such as Q-Learning, SARSA and Neural Networks.
A link to an API was provided, therefore only the AI controller had to be developed and not the rest of the game (allowing for more time and consideration to be taken when developing the AI controller). Some of the research conducted was on the methods which my code would have access to using this API as the backend and what I would have to implement, such as a reward function that was not present in the API.
The provided API used maven hence picking an IDE with maven support was of the upmost importance, and therefore IntelliJ was used.

All the projects that I had seen so far, including the backend I was planning to use, had used a software project management solution called maven. Therefore, I had to allocate time to learn how to use maven for the completion of this project. For keeping track of all the papers that were read and referenced later, a web application called Mendeley was used to store them and provide an easy way for referencing that integrated with Microsoft Word, and was time saving (allowing for more time in which to develop the AI controller).

1.1.4. Similar Systems and Research Techniques

Most of the research conducted was predominantly done using desk research, using papers found on the internet. Google scholar was utilised as a search tool to look for reliable and peer reviewed papers.
Research was conducted after background preparation had been completed on predominantly Q Learning, SARSA and Neural Networks. Many of the papers found were written on previous entries to the competition website which provided the API.

The API provided did come with a few examples on how to control Ms. Pacman, which mostly provided insight in how some methods can be used to fetch information from the current game state. These techniques used various methods that implemented systems which used things from neural networks using features to basic state machines using conditions to return preferable moves to keep Ms. Pacman alive.
The Neural network approach took information from the game, such as the distances to the nearest pill and ghost and used these as inputs for the network. The state machine used if statements to return the move away from the nearest ghost if it was near enough and the nearest pill if not.

However, before picking a method some research was conducted on using other techniques and algorithms to control Ms. Pacman. This was done to introduce a variety in the research and techniques, and to see which one would make the most sense within the allocated time period, and how effectively it could be implemented with the tools at hand.

In a paper by Pepels [6], A Monte Carlo Tree Search Algorithm was used to find the optimal move for Ms. Pacman at each step. The results of many random simulations

were used for the construction of the tree. They used some enhancements over traditional Monte Carlo Tree Search to adapt it for a real time domain, such as the one used in Ms. Pacman. The agent that was developed was then entered into two competitions coming 1st and 2nd. The Monte Carlo Tree Search is a best-first method for searching, with an underlying of random sampling of the states of a domain. Therefore, decisions in the game are chosen based on random selections of moves. The Tree is constructed as the game carries out and each node contains details about the data collected in that state such as the reward.

The recent accomplishment by Microsoft[3] using aggregators to calculate the worth of every pill and ghost was considered as a solution, however, because of the lack of documentation, available research, complexity and timescale this was deemed as unfeasible. Microsoft's team had achieved the maximum score possible of 999,990, using a divide and conquer method. They divided the large problem of the game Ms. Pacman into smaller problems which were then given to AI agents. These agents then worked on their problems separately to achieve a common goal. Each of these agents fed their suggestive moves to a single agent which then decided what move Ms. Pacman should take. This was achieved using multiple reinforcement learning agents with a hybrid reward architecture. [7]

Using high level observations and action modules, rule based policies were automatically constructed in the approach by Szita 2007[8]. They set out to optimize their rule based policy by conducting a policy search of all the allowed rule based policies in the state space using cross entropy. This method has a distribution of all allowed solutions and selects randomly a solution from the distribution [8].

A similar system using reinforcement learning with a combination of Q-Learning and Function Approximation was used by Breen 2014 [9]. This was achieved with relative success because the agent had learnt to interact with the environment. This approach (which is also used in this project) used features extracted from the game state at each time step, which were then linearly multiplied by corresponding weights to calculate the states worth. The weights are then updated using the Q equation for the next time step. This is then iterated till the end state.

Another approach was to use Reinforcement Learning with a Deep Neural Network. This was achieved by the game sending the states, observations and rewards to the neural network. The state representations were similar to the ones used above, as features representing positions of other agents and conditions. These features however, unlike before, were used as inputs into a deep neural network using the TensorFlow library for Python. The Neural Network then gave an output of a move for Ms. Pacman to perform.

1.1.5. Machine Learning

Machine Learning is a subsector of computer science that uses statistics in its techniques to allow computers and systems to progressively improve their performance while carrying out a specific task with supplied data. Essentially, it is the science behind how computers learn from data. This creates an insight by the collections and analysis of this data. Machine Learning is being used more practically all over the business world by allowing companies to more accurately and efficiently analyse data to give them an edge in the industry. Because of the speed and accuracy, it can achieve when classifying data when the algorithm is developed and trained properly. Allowing some companies to process data faster and with more efficiency than the competition.

A more recent development in this area is autonomous cars. Machine Learning can be used to analyse the video input of these vehicles to categorise objects and make decisions such as to break if an object is too close or approaching.

It is being used in the medical field with great amounts of success, partially because of the large amounts of data available for training and the recent advancement in the processing power of computers. This might, in the future, be able to diagnose patients at a more efficient rate which could help save money in the sector. This does not come without its drawbacks unfortunately, because no algorithm is one hundred percent accurate there will be cases where it mislabelled data which can cause the incorrect diagnoses of a patient.

Machine Learning comes in two main approaches, Supervised Learning and Unsupervised Learning. In supervised learning, the machine learning function is trained to map inputs to outputs with examples that already have input and output pairs. The algorithm analyses the training data and creates a function to best fit this data, which is then used to analyse the test data to see how well it performs and is eventually used to categorise new unseen data. This is commonly used in situations such as handwriting and image classification. With unsupervised learning, the data it uses to produce the function does not come with any labels, so it is not possible to analyse the accuracy of the function.

1.1.6. Reinforcement Learning

Reinforcement learning is an aspect of Machine Learning that is used in many varying industries, such as the gaming industry. In e-commerce it is used to detect and predict patterns, such as what to recommend on the home page of amazon when you are logged in.[11]

Most of Reinforcement Learning is an interaction between an Agent and an Environment. The Agent performs actions in the Environment in a form called a state, the environment then returns a reward either positive or negative base on the new state which is achieved by performing said action in a specific state. Hence this technique is learning by interaction with the environment. [12]

In reinforcement learning the agent is trying to learn a behaviour, these behaviours have structures. One such structure is known as a plan; a plan is a fixed sequence of actions that an agent is going to perform. However, during learning not all the information might be available for a plan to be constructed. This only works in stochastic environments where everything should act the way you would expect it to act. The next structure is called a conditional plan, this is similar to a normal plan but has if and else statements which can change the rest of the plan if the environment has changed in any way. The final one known is called a universal plan which is a mapping from states to actions. This structure handles stochasticity very well because the mappings are stored in a table and are looked up every time transition is made.

One of such methods is Q Learning. Reinforcement Learning is different to supervised learning in that supervised learning is a method of learning through provided examples. For many Reinforcement Algorithms, a Markov Decision Process is used to represent the environment and a reward function is used to evaluate the worth of a state.

1.1.7. Markov Decision Process

A Markov Decision Process, also known as a controlled Markov Chain, is a way to model decision making situations using a mathematical framework where outcomes can be random. This is useful for observing a large range of optimization problems that can be solved by using reinforcement learning. This is used in many different industries such as robotics and manufacturing. By mapping the state and action pairs for a system to create a model, this model can be studied to make observations about how to traverse through the environment.[13][14]. The MPD is usually denoted as a tuple (X , A, P, r), where X is the state space, A is the action Space, P represents the transitions and r is the reward function.[15]

1.1.8. Q-Learning

Q Learning was created as a relatively easy way for agents to learn how to act efficiently and optimally in an environment which can be described as a Markovian Domain. It works by assigning values for state action pairs that are updated each time the specific action is performed in the specific state.[16]
Each state value is calculated using the following formula in its simplest form of one step Q Learning.

$$Q(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}}$$

In this example, the learned state action value function, directly approximates the best state action value function. However, instead of updating based on the optimal action of the following state, it can be updated based on the action that the policy chooses from the next state in a SARSA update which stands for state, action, reward, next state and next action.

$$Q(s,a) \; += \; \alpha \Big[R(s) + \gamma Q(s', a') - Q(s,a) \Big]$$

Q Learning is a type of model free reinforcement learning or can also be seen as a way of asynchronous dynamic programming. When acting in the Markovian domain, Q Learning provides a way for the agent to learn how to act optimally by experiencing the impact of their actions, negative or positive, so that they do not have to build models or the domain.

Q Learning involves an agent moving in a distinct and limited world choosing an action from a predetermined group at every time step for each state. At each time step values for the state, action, reward, and the next state along with the best future action are taken to decide the value of the current state which is then stored in the Q Table. For a game like Ms. Pacman with a relatively large domain it would require a large state representation is all the Q Values are to be stored in a table, that's where Function Approximation and State Generalization is used to minimize the state space and the amount of storage needed.

However, this requires there to be feature selection which is often done by hand and can restrict the accuracy of the learned rewards.

1.1.9. Function Approximation and State Generalization

Because of the size of the large state space when working with Ms. Pacman, many problems can appear which can cause issues of various sizes depending on various factors. The table of Q Values can become cumbersome which in turn can lead to slow updates of the Q Values and therefore large states can be hard to find.

With function Approximation a reward function can be leant as a linear combination of features. Therefore, instead of having a table full of Q Values, each state can be represented temporarily as a group of features so that a state can be generalised, and similar actions can be performed in similar states.

Features in the Ms. Pacman game can consist of negative and positive features such as the distance to the nearest pill and power pill or the distance to the nearest ghost within reason and if there are any ghosts one step away from Ms. Pacman.
For Function Approximation, weights are used to sum up the features and the whole experience in a few powerful numbers. These are usually initialised as zero.
Q is now represented as,

$$Q(s,a) = \sum_{i}^{n} f_i(s,a) w_i$$

And each weight is updated at the end of each time step as

$$\begin{aligned} w_i &\leftarrow w_i + \alpha[correction] f_i(s,a) \\ correction &= (R(s,a) + \gamma V(s')) - Q(s,a) \end{aligned}$$

Some of the advantages of using Function Approximation and State Generalization is that it greatly reduces the amount of size of the Q Table and with some of the states sharing features, generalization can be made to many unvisited states which allows for more robust behaviour by making similar decisions in similar states.

1.2. Analysis

The problem at hand was to create a controller that uses Artificial Intelligence to control the Agent Ms. Pacman in the game. The AI controller that was to be created needed to perform well enough to be warranted, the basic rule-based controller performed efficiently therefore further improvement on its score must be achieved. The AI controller would also have to collect pills and avoid ghosts as a primary goal because of the non-deterministic logic of the game the end state can vary.

Considering the background analysis that had been undertaken, one of two techniques where going to be used to solve the problem and with the most research found in approximate Q Learning and the timescale, this would be used to solve the following problem. This would be achieved using the API as a backend to develop my Reinforcement Algorithm onto. This way more time can be spent on developing and refactoring the code for the algorithm.

As previously mentioned, having gained knowledge from some of the example systems, it provided a direction for ways that part of the system could be developed. To start my solution off, I created an exact Q Learner to learn more about both

Reinforcement Learning and Q Learning. This helped with the next step of creating an Approximate Q Learner to better solve the problem.

After the analysis, there were two main tasks that stuck out, namely first trying to achieve a basic Q Learner by using the Q Table to store values to examine the functionality of my knowledge along with the functionality of the API. This was to be make sure the math was possible in how the API interacted with the code that was developed.

For this to be accomplished, firstly, an understanding of the API and its methods was needed to be able to integrate well with the AI controller. After this a good reward function needed to be created for deciding how and when the rewards for actions will be called to input into the Learning Function. Then updating the Q Values for the state and storing it in permanent memory so that between and after training episodes it could be accessed to continue, and this was performed until the terminal state.

Once this had been confidently achieved moving to Function Approximation could be accomplished. This was accomplished by using the previous system as the basis. A way for the extraction of features was needed to be developed, this was needed to collect information at each state about the game and the agents to be used in the update equation. These along with the weights could then be used to linearly calculate the Q Values at each time step. The Q Function could then be calculated using the extracted features and the weights. Weights were updated at the end of each time step.

As mentioned above, various other approaches were taken into account, these approaches included Neural Networks with Q Learning and the Monte Carlo Tree Search. All of these had plenty of documentation and proved to successfully play the game of Ms. Pacman. However, due to the time constraints and the general complexity of some of the methods using Neural Networks, and with a general bias of finding reinforcement learning interesting, the best option was deemed to be to use Reinforcement Learning with a combination of Q Learning and Function Approximation and State Generalization.

Java was picked as the programming language for this project. This was partly because of the framework that was provided. This framework cut down the development time by not having to develop a version of the game to implement my controller which allowed more time to be spent on developing and refactoring the controller itself. Secondly, it was picked because of my greater knowledge using this language which too cut down time from having to learn a new language in turn taking time away from actually developing the controller and not having to learn the basics of a new language.

The hardware that was used for this research was a three year old Lenovo mid spec ThinkPad. This was used instead of a higher end system to check the compatibility of the developed controller on lower end machines, and therefore its efficiency. Because the Function Approximation and Q Learning was done linearly there shouldn’t be a huge impact on the requirements of the hardware needed to run the controller and essentially could be run on anything that was capable of running java and the JVM efficiently.

1.3. Research Method and Development Methodology

1.3.1. Methodology

There are many variations of process methodologies that could be used when developing software, with the two major methodologies being Agile and Waterfall. With the Waterfall method being more linear and sequential, it did not fit the iterative approach that was needed and was not flexible enough if changes did need to be made. In the end an adapted version of FDD or Feature Driven Development was chosen because of the way the Five Processes fits the way the project will naturally flow [17]. The five processes are: develop a model, build a features list, plan by features, design by features and build features. The first process is where a majority of the research was conducted to allow the next process of building the features list. The code is then planned by using these features as the guide and building blocks. Features are then designed and built to complete the methods needed for the execution of the reinforcement learning agent. The adaptation that was made was fitting in the research into developing the overall model.

1.3.2. Features

1. Reward Function
    1.1. Check the game for conditionals such as if a pill was eaten
    1.2. Return a value based on which conditions were true
2. Feature extraction
    2.1. Check the current game state
    2.2. Return distances to:
        2.2.1. Nearest pill
        2.2.2. Nearest power pill
        2.2.3. Nearest ghost
        2.2.4. Nearest edible ghost
        2.2.5. All ghosts in the vicinity
3. Create Q Values
    3.1. Take features
    3.2. Take weights
    3.3. Linearly multiply them together
    3.4. Return the move with the highest value
4. Update Weights
    4.1. Take Q values
    4.2. Take action performed
    4.3. Take reward
    4.4. Take features
    4.5. Multiply them according to the Q Learning function
5. Repeat

2. Experiment Methods

Based on the research previously conducted, a hypothesis has been formed. This being that if the Reinforcement Learning algorithm is correctly implemented with a combination of Q Learning and Function Approximation and State Generalization, that Ms. Pacman should be able to learn how to go after pills and while trying to avoid ghosts. The algorithm should allow Ms. Pacman to learn through experience that eating the pills will provide a positive reward and being eaten by ghosts provides a negative consequence. Because of the methodology that was chosen each feature was needed to be developed separately.

Starting with the reward function, each feature was planned and then developed and tested for the correct output before carrying on with the next feature. This way each feature could be tested individually and then the program as a whole at the end. Once all the features had been fully implemented and tested, experimentation begin by running the agent in the environment while changing the values for each of the exploration rate, learning rate and the discount factor. A record of the score for each run was made and compared. The second test change the amount of training done before the displayed run of the game commences and the scores recorded for these tests.

Because of the nature of the program itself and how it was created. Basic tests were sufficient enough because the score of the agent is displayed on the GUI. Therefore, no support tools were needed other than Microsoft Excel for tabulation of the results and the creation of graphs.

All the experiments that where conducted, were done using the provided API and code I had developed myself.

3. Software Design, Implementation and Testing

3.1. Design

Because of the approach that the agile methodology of Feature Driven Development, working software is desired over extensive and comprehensive documentation. Therefore, only a brief but thorough explanation of the design of the developed software is needed. For this project there was only one program created to test the theory and hypotheses conceived by the research conducted. Some documentation to the style of the waterfall method will be created to explain the code that concerns how the algorithm interacts with the API for better understanding of what was created and how it works.

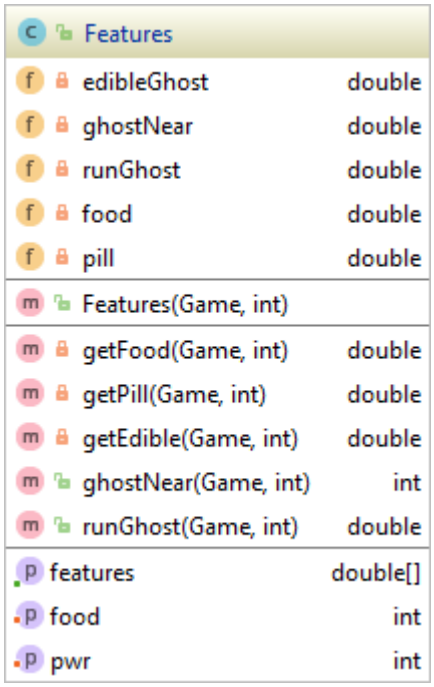
Because the game was supplied beforehand, all of this section will focus only on the algorithm that was developed for the controlling of Ms. Pacman. Not all of the API will be described, only those parts relevant to what is needed by the Q Learning agent.

3.1.1. Overall Architecture

The developed system performs a substantial amount of mathematics in a small amount of methods and classes. There are only two classes, an Agent class which contains most of the main methods and a features class which handles the extraction of features from the game. Because most of the Reinforcement Learning is done in a few equations, not much code was needed to get it to work. The way the API works is that it calls the getMove() method from the agents' class to move the agent in the game world. This is where all the other methods are called for the Reinforcement Learning. Once this method is called it makes a call to the getQ() method which for this time step calculates the Q Values for the possible moves to be made by using the features values and the weights. These Q Value are then used to pick the optimal move. The move is then performed, and a reward and next state received. These along with the function constants and features are used to update the weights for the next move selection.

3.1.1.1. Features

The Features class contains all the information and methods needed to successfully perform the State Generalization and Function Approximation. It has 5 variables that store the numbers associated with the features that are extracted.
The constructor for this class takes in the current state of the game and the current node of the agent to determine with the five methods, the distance to the nearest food, pill, edible ghost, non-edible ghost, and how many ghosts are in the vicinity. These are all determined using a Manhattan distance calculator provided by the API.



3.1.1.2. Reinforcement Learning Agent

This class is the main class for the Reinforcement Learning Agent that is Ms. Pacman. This is where most of the functionality and maths is performed. It contains the three variables to perform the Q Equation that is used to update the weights after each timestep namely Alpha the learning rate, Gamma the discount rate and Epsilon the exploration rate. The myMove variable is where the move is stored before it is returned to the game in order to be used for the calculations of the rewards and finding the max Q Value for the next to use in the Q Equation.

The weights are stored in an array as an instance variable so that it can be accessed between games this way it can store what it learnt. The features are stored in a Features object so that when created the methods in its class does not alter the current game.
In the constructer, the weights are initialised as 0.

The getMove method is required by the API and is one of the superclass methods that is required by all controllers, however it is overridden so that extra functionality can be added. In this method there are calls to the other methods that have been added to perform the selection of the optimal action. It calls the QTable method, maxQIndex method and the updateWeights method before returning the chosen move to the API.

The QTable method takes in the current game state and the node that Ms. Pacman is currently in and creates an array for Q Values for the current possible moves, it calls the getQ method which takes in a move and calls it with each possible move for the current node that Ms. Pacman is currently in.

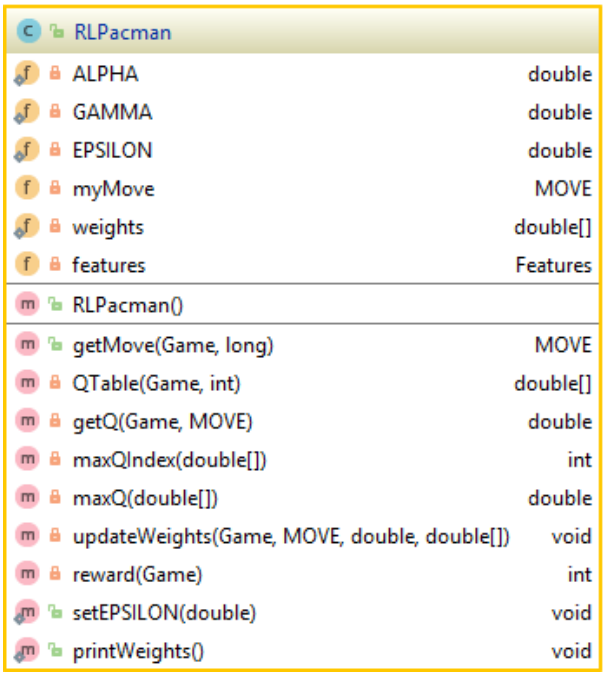
The getQ method takes in the current game state and a move to be made, it then takes a copy of the game and performs the move selected then extracting the features for the new state of the game. It then multiplies these features with the weights using the dot product method to work out the value for Q at this current state.

The maxQ and maxQIndex both take in the array of Q Values and turns them into ArrayLists. This is useful because of the Collections class in java which allows the selection of the greatest item and or its index, reducing the amount of coded needed and the possibility of any errors.

The updateWeights method takes in the current game state, the move to be made by the agent, the Q value corresponding to that move, and the array of features for the current game state before the move. It then updates the game with the move to be made and gets the node that Ms. Pacman will be in when the move is made. Bellman's Equation is then used with this data along with the learning rate and the features to update the weights linearly. The equation used can be seen previously referenced in the background research section of this report.

The reward function takes the game state after a move has been taken and checks with methods from the API what the associated reward would be for taking that move at the state such as returning a positive five for a move that allows Ms. Pacman to eat a pill.

The setEPSILON method is there for the experimentation on how randomness can affect the learning capability of the agent and the score it can achieve in explore vs exploit.



3.2. Implementation

When implementing the Reinforcement Learning agent there were a few problems that were encountered during the development section. Firstly, because of the way the API is setup the code can only interact with the game through the getMove method and therefore all the method calls to the others had to be done through this when it was called by the game. This proved troublesome when retrieving the rewards for the move made when because once the move was made it carried on with the game and called the getMove method again for the next move disregarding all the information from the last go. This was overcome by creating a copy of the game each time a move was needed to be made and the move was done on the copy to see the outcome of various moves to calculate the Q values for each move at the current state.

When first developing the feature extraction and Q value calculation methods, an error occurred that once Ms. Pacman knew that going for the pills would produce a positive result it would actively go out and seek the pills. However, when Ms. Pacman was in the node next to the pill and when the code was calculating the Q values for each move, it updated the game to the point that the pill that was one step away before did not exist anymore which would make the agent go in the opposite direction to another near pill. This would then produce the error that when the move in the opposite move was taken the pill will appear in the original game again and Ms. Pacman would move towards it again and get stuck in this loop of going back and forth right next to the pill. This was overcome by taking a list of the locations of the pills before each move was made in calculating the Q values and checking if the node that Ms. Pacman was in after the move was made corresponded to a move in the list which would mean that it would eat a pill in that move and then the features where altered to show that. This solved the problem of going back and forth. This problem also occurred for the power pills and was solved using the same solution.

When the method for calculating the Q values for each state and action pair was created, Ms. Pacman would exhibit the strange behaviour of trying to go to the top right hand corner each playthrough of the game. At first it was thought that this was what it learnt was the optimal path, however, it did this regardless of the actions of the ghosts and whether or not it was eaten in that corner. On closer inspection it was found the when calculating the Q values, the copy of the game was not reset and therefore all the Q Values where the same. This meant that Ms. Pacman always took the first available move available which lead it to the corner where it got stuck going back and forth in the corner. This was solved by extracting the updating of the game and the calculation of the Q value for the new state after the move was made to another method so that the changes made to the game was not present in the main function. This then allowed the proper function of the Q Values and movement of Ms. Pacman.

For the methods that returned the highest Q values and the index, arrays of doubles were used and iterated through to see what was the highest number was. This was replaced with Collections from the java util package which already comes with a max method. This simplified what was already done but provided security that the correct values were selected.

Calculating the distance to the ghosts and nearest pills was originally was done by getting the coordinates of them and Ms. Pacman and using calculations to work out the distance. Although, after reading up on the API and looking at more examples, a method was found that calculated the Manhattan distance between Ms. Pacman and the other objects. This worked much better than the way it was done previously, as this took the walls into consideration too.

As mentioned above, there were a few problems that were encountered and that a solution had to be developed for. Most of these problems encountered incurred small delays that were quickly solved. However, the most complex issue that was encountered was the issue of the agent seeing not seeing a pill because it was deemed as already eating before the Q Values could be calculated. This issue did take up quite a substantial amount of time compared to the previous smaller problems.

3.3. Experimentation and Results

For the first experiment, a change in the constants of the Q Learning function was done, namely the Learning Rate and the Discount Factor. All these tests were run with no training and on the same machine to keep the variation minimal. The weights and scores could then be collected in a table for ease of view. From the results we can see that the impact of the future reward has the highest change in score which states that the more Ms Pacman thinks about future rewards the higher Ms Pacman will score. Each test row was run ten times with no training to take an average of the score for the results.

ALPHA	GAMMA	AVG. SCORE
0.2	1	1850
0.4	1	1850
0.6	1	1850
0.8	1	1850
1	0.2	200
1	0.4	1550
1	0.6	1730
1	0.8	1990
1	1	2640

4. Conclusions and Future Work

From the results of the test and general usage of the developed code, it is clear to see that even with limited knowledge from the start and a relatively small timeframe it is possible to implement an Artificial Controller for the arcade game of Ms. Pacman with relatively good results. The results show that Ms. Pacman is able to learn about the correctly implemented features which is evident by the score that it is getting by eating the pills and edible ghosts, even though the agent might not learn that it needs to stay away from normal ghosts.

If this experiment was further expanded on a change could be made for the linear function and could be replaced with a deep neural network. Additional research will have to be conducted on neural networks. This has been shown to achieve much higher scores than the maximum attained in this project. Although using a neural network would have achieved higher scores the time and performance needed to train the neural network would have been much higher compared to the Q Learner. Further work would be put into better feature extraction and to figure out how the API functions according to how the ghosts eat Ms. Pacman and how to tell the feature that they are on the same space.

If the experiment was to be conducted again, I would most likely change the technique that was used to introduce a bit of variance. I would conduct research into how to implement Evolutionary algorithms with neural networks and try to apply it to the problem of the Ms. Pacman game. I would try to find a different API to use as the game itself to make it easier to access information without having to access it in an obscure way, like with the pills being four nodes apart and not actually right next to each other. This along with picking another language to see if the knowledge is transferable or whether what I have learnt is language specific. For this I would chose python, because during the research nearly all the similar systems where developed using python with relative ease because of its scripting language behavior and the available machine learning plugins which would have sped up the whole process and might have allowed for more features and testing to be done.

5. Critical Evaluation

The purpose of this project was to develop an AI controller for the game of Ms. Pacman using the approach of Q Learning technique of Reinforcement Learning along with Function Approximation for the State Generalization. This was somewhat successfully achieved and in testing Ms. Pacman knew that going after pills, power pills and edible ghost would return a positive reward. However, the knowledge that normal ghosts were bad was only sometimes learnt, depending on the experience of the agent in the current playthrough. This was to do with the way the features were extracted and therefore the weights that were assigned to these features. Because the way the API functioned in that when Ms. Pacman was eaten by a ghost the game was reset before the ghost was sent back to the lair before then features could be extracted and therefore the negative reward was then associated with the other features and not the ghost

Implementing the actual Q Learning function was fairly easy, once the math behind it was figured out. The only issue with this was the way information had to be extracted from the current game state which made things more difficult than it had to be. Because the exact data that was needed to perform the calculations where available at that time step and we were only able to access it from one method in your class it became difficult to access it after the move had been made. Therefore, actions had to be performed on a clone of the world to see what the outcome would be before actually performing it in the original game world.

Choosing Java as the programming language for this project helped create more much needed time for the development of the controller itself because of the provided API that already had a version of the game implemented. However, because of the way the API was structured this added to the time trying to avoid the pitfalls of not being able to access any part of the code from the agents' class. This hindered the way the code had to be developed because to get around this all the other methods had to be called from a single method which exited once a move was sent to the main code. This issue was eventually fixed by using clones of the current game and executing the methods there, however, this did take up a substantial amount of time that could have been used somewhere else in development.

Personally, I think I should have chosen another programming language such as python. Even though I would have had to either code the game myself or look for another API other than the one provided, after much research I found that python had much more libraries that dealt with Artificial Intelligence and specifically machine learning. This might have solved the way I had to design my code around it all being accessed through one method and could have used some of the other libraries that were part of my research.

With the selection of the Agile methodology of Feature Driven Development more time was spent on the code than on extensive documentation which proved itself useful along with the flexibility and iterative nature of this approach when problems in the code had occurred. This allowed for the changing of how some of the methods worked to better suit the API while still achieving the feature it was set out to. However, because of the relatively small amount of planning compared to a method like Waterfall, some of the methods did not return the results it should have done first time around and had to be iterated over a couple of times.

Despite doing modules in both Artificial Intelligence and Machine Learning, I think my eagerness for this major project topic was a bit misguided. All of the techniques I ended up using were not taught in these modules and therefore a substantial amount of research had to be done before any of the planning or coding could be done. This was one of the downfalls of picking Q Learning with Function Approximation is that there was almost no prior knowledge and therefore more time than I liked had to be spent researching and less time for implementation and refactoring.

6. Appendices

A. Third-Party Code and Libraries

http://www.pacmanvghosts.co.uk/

This is the website to the competition of the API that was used to develop my controller on top of.

B. Ethics Submission

AU Status
Undergraduate or PG Taught

Your aber.ac.uk email address
joh55@aber.ac.uk

Full Name
Joshua Brian Hodges

Please enter the name of the person responsible for reviewing your assessment.
Reyer Zwiggelaar

Please enter the aber.ac.uk email address of the person responsible for reviewing your assessment
rrz@aber.ac.uk

Supervisor or Institute Director of Research Department
cs

Module code (Only enter if you have been asked to do so)
CS39440

Proposed Study Title
Developing an AI controller for Ms. Pacman

Proposed Start Date
29/01/2018

Proposed Completion Date
04/05/2018

Are you conducting a quantitative or qualitative research project?
Mixed Methods

Does your research require external ethical approval under the Health Research Authority?
No

Does your research involve animals?
No

Are you completing this form for your own research?
Yes

Does your research involve human participants?
No

Institute
IMPACS

Please provide a brief summary of your project (150 word max)
The purpose of this project is to develop an AI controller for MS. Pacman which will avoid ghosts and collect pellets and fruit to achieve the highest possible score. To achieve this, extensive research will have to be performed to improve on the knowledge I have already learnt in my time at university taken the relevant AI and Machine Learning modules.

Where appropriate, do you have consent for the publication, reproduction or use of any unpublished material?

7. Annotated Bibliography

[1] “Gym.” [Online]. Available: http://gym.openai.com/. [Accessed: 23-Apr-2018].
This webite helped me understand how to translate my knowedge of reinforcement learning into code.

[2] V. Mnih *et al.*, “Playing Atari with Deep Reinforcement Learning,” 2013.
This paper helped understand the more complex ideas behing intergrating Q Learning and neural networks and how they can be used to play games.

[3] A. Linn, “Divide and conquer: How Microsoft researchers used AI to master Ms. Pac-Man,” 2017.
this article helped shed light on how microsoft accomplished a maximum score in the game.

[4] “Free Online Classes & Nanodegrees | Udacity.” [Online]. Available: https://eu.udacity.com/. [Accessed: 23-Apr-2018].
this website helped me with its free online course to understand the basics of reinforcement learning

[5] “edX | Free online courses from the world’s best universities.” [Online]. Available: https://www.edx.org/. [Accessed: 23-Apr-2018].
this webiste had helped me understand function approximationa and state generalisation.

[6] T. Pepels, M. H. M. Winands, and M. Lanctot, “Real-time monte carlo tree search in Ms Pac-Man,” *IEEE Trans. Comput. Intell. AI Games*, vol. 6, no. 3, pp. 245–257, 2014.
this paper helped me understand about the adaptations that would have to be made for a MCTS to work in Ms. Pacman

[7] H. van Seijen, M. Fatemi, J. Romoff, R. Laroche, T. Barnes, and J. Tsang, “Hybrid Reward Architecture for Reinforcement Learning,” 2017.
This paper helped with my understanding of Reinforcement Learning

[8] I. Szita and A. Lorincz, “Learning to play using low-complexity rule-based policies: Illustrations through Ms. Pac-Man,” *J. Artif. Intell. Res.*, vol. 30, pp. 659–684, 2007.
This paper was used to research alternate approaches to the controller

[9] C. R. Breen -, Y. Lai Moderator, and F. C. Langbein, “Artificial Intelligence for Pac-Man,” 2014.
This paper was done using the same API and help with the feature extraction

[10] R. C. Deo, “Machine learning in medicine,” *Circulation*, vol. 132, no. 20, pp. 1920–1930, 2015.
This helped find other use cases for machine learning and provided more definition to some of the functions

[11] “Amazon Machine Learning Documentation.” [Online]. Available: https://aws.amazon.com/documentation/machine-learning/. [Accessed: 01-May-2018].
This showed how amazon used machine learning with its online store

[12] P. R. Montague, “Reinforcement Learning: An Introduction, by Sutton, R.S. and Barto, A.G.,” *Trends Cogn. Sci.*, vol. 3, no. 9, p. 360, Sep. 1999.
this book helped with a large maajority of the knowledge obtained of machine learning, reinforcement learning and Q learning

[13] E. Altman, “Constrained Markov Decision Processes,” *Oper. Res. Lett.*, vol. 19, no. 1, p. 260, 1999.
this paper helped with understanding the uses of MPDs

[14] R. Bellman, “A Markovian decision process,” *Journal Of Mathematics And Mechanics*, vol. 6. pp. 679–684, 1957.
this paper helped with understanding MDPs and the equations behind the processes

[15] F. S. Melo, “Convergence of Q-learning: A simple proof,” *Inst. Syst. Robot. Tech. Rep*, pp. 1–4, 2001.
This paper helped understand more of the mathematics behind Q Learning

[16] C. J. C. H. Watkins and P. Dayan, “Q-learning,” *Mach. Learn.*, vol. 8, no. 3–4, pp. 279–292, May 1992.
This paper help with defining what Q Learning is and what it was all about

[17] S. R. (Stephen R. Palmer and J. M. Felsing, *A practical guide to feature-driven development*. Prentice Hall PTR, 2002.
This paper helped refresh my memory on Feature Driven development.