AI Assisted Learning in Games
TicTacAI

Date 14/02/2020

Abstract

The problem here is created by the fast paced lifestyle of the modern world. All people that live in this world must make choices for what to allocate the precious finite hours every human has to. This is made even worse by modern society that expects many hours to be invested into a career and preparation for the future. With such little time left to oneself in this world, I aim to create software that can assist all people in doing the hobbies they enjoy doing, which are the things that bring happiness to people.

I will achieve this by creating software that implements a reinforcement learning algorithm in order to gain proficiency in a given environment. This software will be built in a way such that it can be rebuilt for different environments. The software allows the user to acquire examples for how a classical game, such as noughts and crosses or other such settings where this kind of system can be applied could be played. This boosts the user's ability to learn how to play and enjoy these games as a hobby by providing a ready source of material available for them to study, thereby decreasing the amount of time they spend on learning and improvement and maximising the time they have to enjoy their hobbies to the fullest

Table of contents

Provide a table of contents, linking Sections and Subsections to page numbers. If you can, hyperlink the page numbers/sections.

List of figures

Chapter 1. Introduction

1.1 Problem statement

The mastery of anything requires an inordinate amount of time, with 10,000 hours being a commonly quoted timeframe for mastery of a skill. Time is a finite resource in the lives of all people and it would benefit all of us if we could spend less time on the learning of skills so that it may instead be spent applying them

.

One of the reasons why mastery of a skill takes such a long time is the ambiguity of practise. How may you practise for scenarios you could not dream of? Often past early and intermediate stages, there is a lack of easy to find examples and it is often up to the individual to identify what remains to be trained.

Attempts to find a solution to this problem have culminated in the TicTacAI project, A solution that aims to develop AI systems that can implement a machine learning algorithm to learn alongside humans and provide guidelines and goals for development of human skills. These AI systems can help us to understand how exactly it is that practise makes perfect. This will allow humans who follow in its stead to learn from its example, and practise more efficiently, thus reducing the amount of time needed for mastery in a particular subject to be achieved. This project will also have a modular design that allows it to be rebuilt for other areas relatively quickly.

The solution can be broken up into a few areas.
Stage one:
Research on exactly why it is so difficult to find teaching material and examples for subjects not taught in education centres such as universities and schools. To understand what users will need from this project, a survey will be created to gather

information about what users think are the most difficult things when learning skills and what they would like to streamline the learning process.

Stage 2:
Research into current applications that utilise machine learning in game based settings and other chaotic environments. Identification of what systems allow them to function.

Stage 3:
Preliminary development. A prototype system will be developed and then judged by a senior developer.

Stage 4:
Final stage development. The final system will be created using the lessons learnt from the prototype system

1.2 Aim and Objectives

The aim of this project is to create self-learning AI systems that can be used as a guideline for humans to achieve mastery of skills in varying fields, specifically classical games, such as card and board games and simple to learn but difficult to master games such as the Super Mario series. This is achieved by the AI first learning how to play a game or navigate a particular environment at a high level, and eventually achieving mastery. The AI then can be shown as an example of how a skill or set of skills is learnt and mastered or how a particular problem can be solved.
This AI system should then act as a platform upon which a human user can study the data and methods, the AI has gathered in the environment and refine their own skills against those the AI has gained in a competitive setting.
The AI will can then be used to explore theories that the human has as to how a game should be played, such as by inserting it into a chess game at a specific point where it can quickly and thoroughly explore alternative moves, or create new theories that can then evolve into a new set of most effective tactics available for a particular game.

As such my aims and objectives are as follows.

- Create an AI algorithm that is able to select actions in a game or environment to take, initially at random.
- Extend this AI to prefer the best set of actions that it performed in the previous iterations..
- The AI then achieves mastery of a Zero-Sum Game such that it will almost always win or draw. This will enable a learning platform for human players to see reliable play.
- Extend the AI so it can perform in more complex environments e.g. card games and platformer type games.

1.3 How these will be achieved

This project will follow a prototype driven development methodology in order to quickly create many working prototypes of increasing functionality. Prototypes with errors or not passing tests will be further developed until the feature they are prototyping is complete. This is repeated until all functional requirements are met. At this point non essential requirements can be considered and go through the same process.

Requirements will be gathered through surveys and study of existing projects and other material, as well as brainstorming methods of implementation.

Chapter 2. Background

2.1 Survey of Literature and existing applications

Why use Machine Learning in the first place?

Machine Learning grants computers a way to learn instead of merely being programmed to perform specific tasks. It allows computer models to independently adapt to new data rapidly and analyse big data in a way humans or traditional computer programs never can. (Evolution of Machine Learning, 2020)

If I were to approach this problem with traditional programming techniques, it would require a whole new project to be developed for each new setting. It would not be able to accurately model a player in the chaotic environment of a modern game such as DOTA, as the combinations of possible situations number in the millions. It is impossible for a human developer to create a ruleset to accurately model gameplay in these scenarios.

Machine learning solves this by allowing the computer to create its own rulesets through extreme amounts of repetitions that are out of scope for a human player or developer.

Machine Learning in Literature and modern games

"*Classic arcade games, of the type found in the late seventies' and early eighties' arcade cabinets, home video game consoles and home computers, have been commonly used as AI benchmarks within the last decade*" (Justesen, Bontrage, Togelius, Risi1, 2019)

Games being used as environments for AI to work is extremely commonplace in the modern world, and there are many other examples to draw from. One such example is the Google DeepMind AI, which uses neural networks to learn how to play games in a fashion similar to humans (DeepMind, 2019). The similarity between human learning and DeepMind learning is extremely desirable, but the neural network it uses is far too complex for a student such as myself. However, the advances and breakthroughs made by the DeepMind team are still useful to myself, and I can use their knowledge to help develop my own project.

Another game-playing AI project, which was one of the inspirations behind my own, is the MarI/O project by developer Sethbling, which also uses neural networks, in this case to complete a level of Super Mario, (MarI/O - Machine Learning for Video Games, 2015) Whilst again, neural networks are perhaps too advanced for my skill level. He details the development and learning processes of MarI/O, which proved deeply helpful for building my own algorithms, albeit with simpler methods.

In the tushuhei blog there is detailed another AI, that instead this time employs Reinforcement based learning as opposed to neural networks. (Iitsuka, 2019) This is the project that I draw most of the influence for my project from. Reinforcement based learning is something much more available to someone of my level of coding, although lessons learned from neural network-based learning may still apply.

In conclusion, although there are many modern applications and projects that aim to utilise machine learning in order to comprehensively understand a subject, They mostly rely on expensive hardware to run, such as the OpenAI project or complicated systems such as the MarI/O project. I believe it is far more reasonable to take an approach more similar to the tushuhei blog's implementation of reinforcement learning, as it is light enough to be distributed over the internet and onto a webpage and does not require any advanced technologies such as neural net learning.

2.2 Review of projects / applications

MarI/O: (Sethbling, 2015)
MarI/O is a computer program that used neural evolution to learn how to play a level of the Super Mario game. Its neural network based approach allowed it to simulate how a simple brain may gain increasing competence over time. The project is described in detail by its creator on Youtube. (Link in appendix)

MarI/O system of mapping inputs to neural nodes

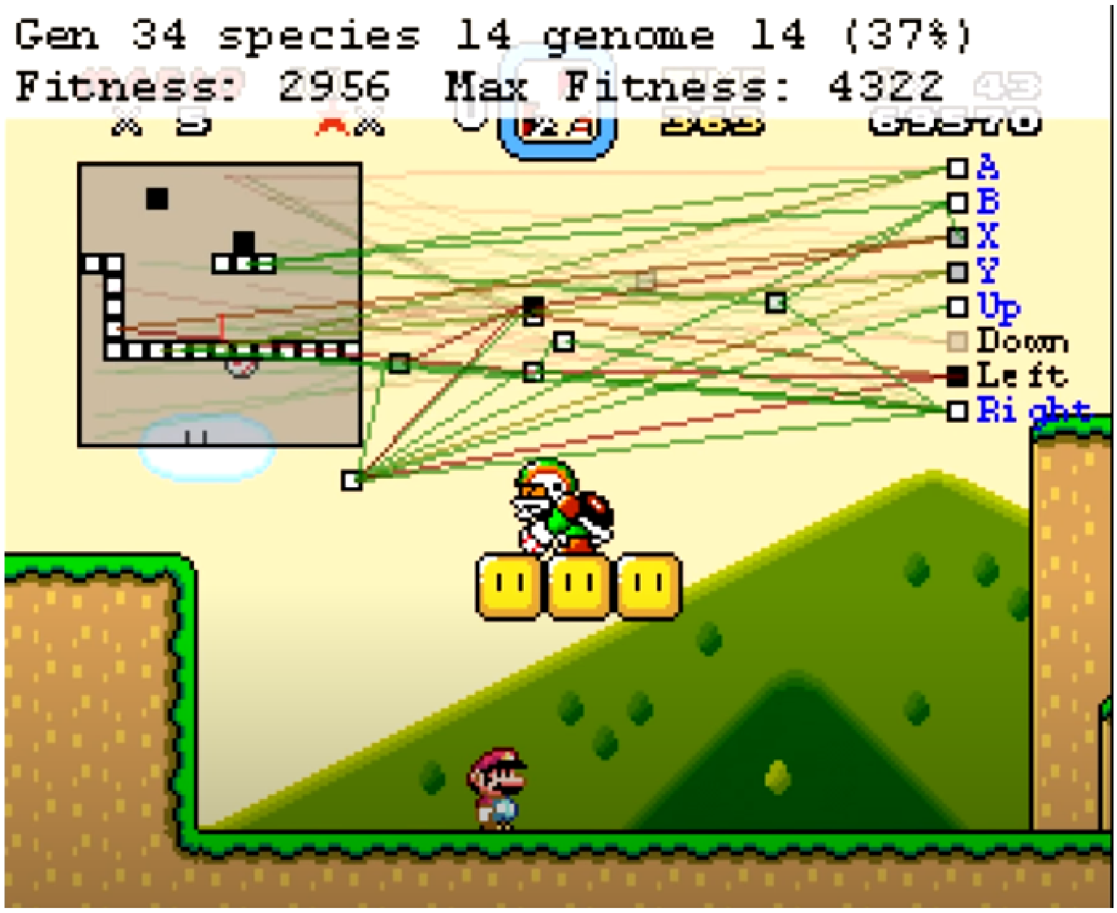

Gen 34 species 14 genome 14 (37%)
Fitness: 2956 Max Fitness: 4322
A
B
X
Y
Up
Down
Left
Right

Figure 2.2.1: A screenshot of MarI/O's learning process. Note the lines that indicate neural connections.

Advantages of the MarI/O project:

MarI/O's increasingly complicated neural network can almost guarantee eventual success and mastery through enough repetitions.

MarI/O can more easily be ported to different environments as it only needs to know what inputs that it can perform and an interface to be able to see the environment it is placed in.

Disadvantages of the MarI/O project:

Creating neural networks is difficult and is likely out of scope for this project.

Takes many hours of practise for such a system to achieve mastery.

Main Features:

Neural Network Learning

Probable Mastery of a given level or environment over enough time

OpenAI (openai, 2018)

OpenAI is a project that aims to create AI systems that benefit all of humanity. A part of it is an AI that surpasses human skill in complicated modern games such as Starcraft 2 and Dota 2, known as OpenAI Five.

Figure 2.2.2: OpenAI’s success in the game DOTA 2.

Advantages of the OpenAI project:

The OpenAI project proves that the reinforcement learning approach to machine learning is capable of creating AI that can reach skill levels that can compete with human professionals in a chaotic environment. It is even capable of outcompeting these professionals, who can have up to a decade of experience, in certain areas.

Disadvantages of the OpenAI project:

It requires huge amounts of computing power. OpenAI runs on 256 GPUs and over 12,000 Computing cores. It accumulates 180 years worth of gametime per day which simply cannot be recreated without matching hardware and development professionals.

Main Features:

Reinforcement Learning on a large scale.

tushuhei blog's Temporal Difference Learning Tic Tac Toe expert:

This project culminated in a blog post with an AI that achieved a skilled level of play in noughts and crosses after around 2000 iterations.

Table of Features:

MarI/O	OpenAI	tushuhei blog AI	My project
Increasingly complicated neural network learning that evolves in a similar fashion to how life evolved in the real world	Large scale reinforcement learning that amasses hundreds of thousands of hours of knowledge in order to surpass human limits.	Online implementation of a temporal difference learning AI that gains knowledge of the noughts and crosses game.	My project uses reinforcement learning on a small scale to gather knowledge on a given environment (this case noughts and crosses)
Eventual Mastery of given environment	Mastery can exceed human professionals in chaotic environments	Mastery moves towards eventual full win/draw situations in zero sum game (noughts and crosses)	My project will be similar to the tsushuhei blog and will move towards eventual mastery of noughts and crosses
MarI/O is programmed in an emulator using lua scripts. It sees a simplified version of the game environment and "knows" the relevant game	OpenAI Five is built specifically to tackle DOTA 2. It is fully immersed in the complicated environment of DOTA 2 and is aware of every	tsushuhei blog's AI is built only for noughts and crosses. it is fully aware of the noughts and crosses board and what it can do.	my project is also only for noughts and crosses but aims to include a degree of modularity in the code in order to more easily facilitate different

inputs	possible input it could make (millions, considering that clicking on the screen in a different area is a different input)		environments in the future. it will be completely aware of its environment and the inputs it can make

Some additional things I have noticed from other unrelated products that might be useful to implement:

1. Addition of user objectives and goals management system
2. Addition of user accounts so objectives/previous data can be persisted over multiple devices
3. Creation of a framework that allows independent developers/users to port classic games over to the AI engine without needing to access the source code of the project.

2.3 Review of possible tools and techniques

Python
Advantages:
1. Powerful Libraries with Machine Learning support, such as TensorFlow and Keras
2. A more modern language than Java that is quicker to produce code in. The development environments that are created for Python almost always consist of Interpreters, and thus Python code can run without having to be completely compiled without errors like Java code.
3. I have used python in the past and am familiar with its syntax.
4. Free and open source object oriented language

5. Cross platform

Disadvantages:
1. Python is seen as a weak language for mobile development, which could make mobile ports of this program difficult to implement.
2. The flexibility of python data types causes it to have high memory consumption
3. The dynamic nature of the language means that it frequently encounters coding errors that only appear at runtime.
4. Database support is limited in python

Java
Advantages:
1. I have regularly been using Java as my main programming language over the last 30 months.
2. Java is an old language with extensive documentation and tutorials on how to do many things using the language. It has increased maintainability over other programming languages and many features have been added over its lifetime.
3. Java is the only language that I have previously used to create multithreaded programs.

TensorFlow:
Advantages:
1. A free and open source platform for machine learning that allows deep learning
2. Backed by Google
3. A reliable platform without many bugs
Disadvantages
1. Not supported on windows unless installed via python package
2. In comparison to competitors, it is slower to run.
3. Only python is supported as a language
4. No support for GPUs other than nvidia GPUs
5. Requires fundamental knowledge of Linear Algebra and Advanced Calculus
6. Has a very steep learning curve

2.4 Graphical Analysis

I have constructed a few diagrams that model the program and how it will be used.

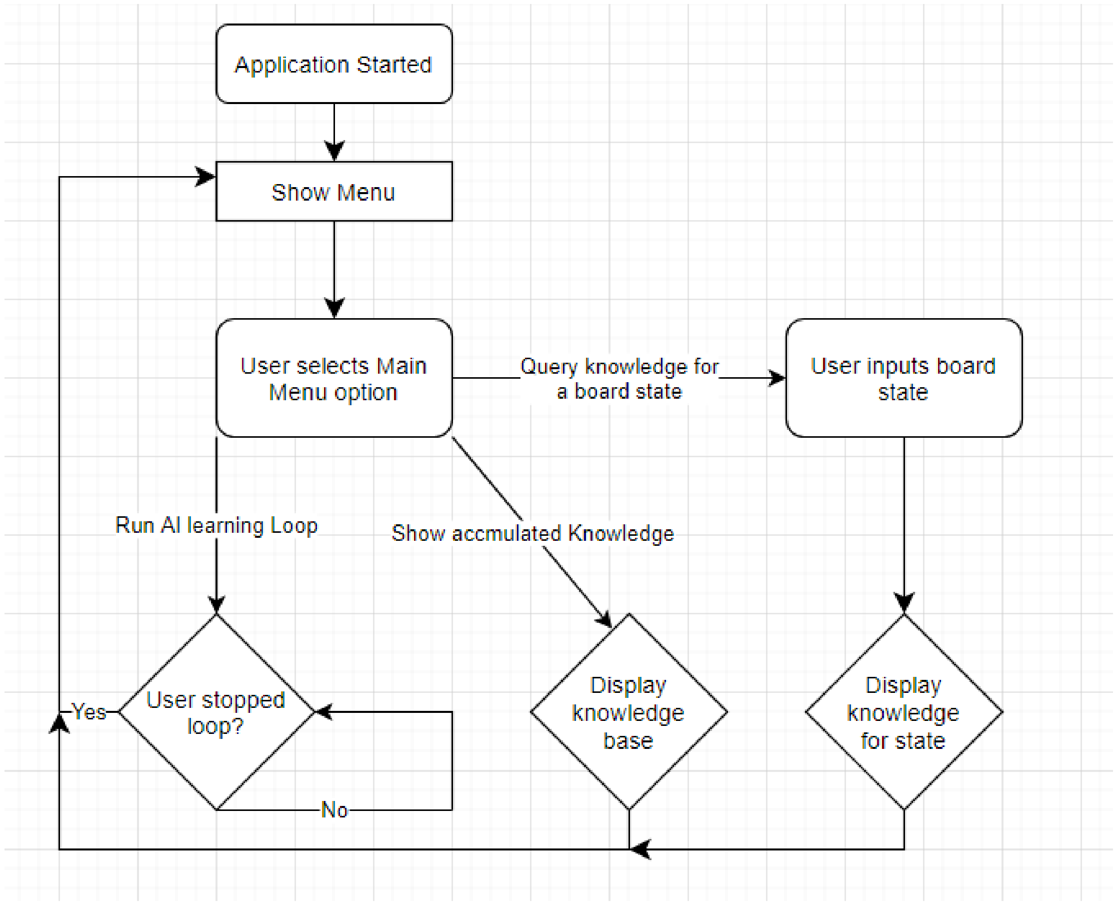


Figure 2.4.1: flowchart showcasing how the user interacts with the program

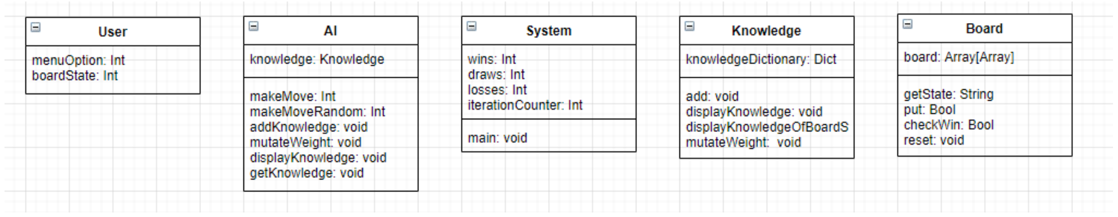


Figure 2.4.2: class diagram showing planned classes and their class variables/methods.

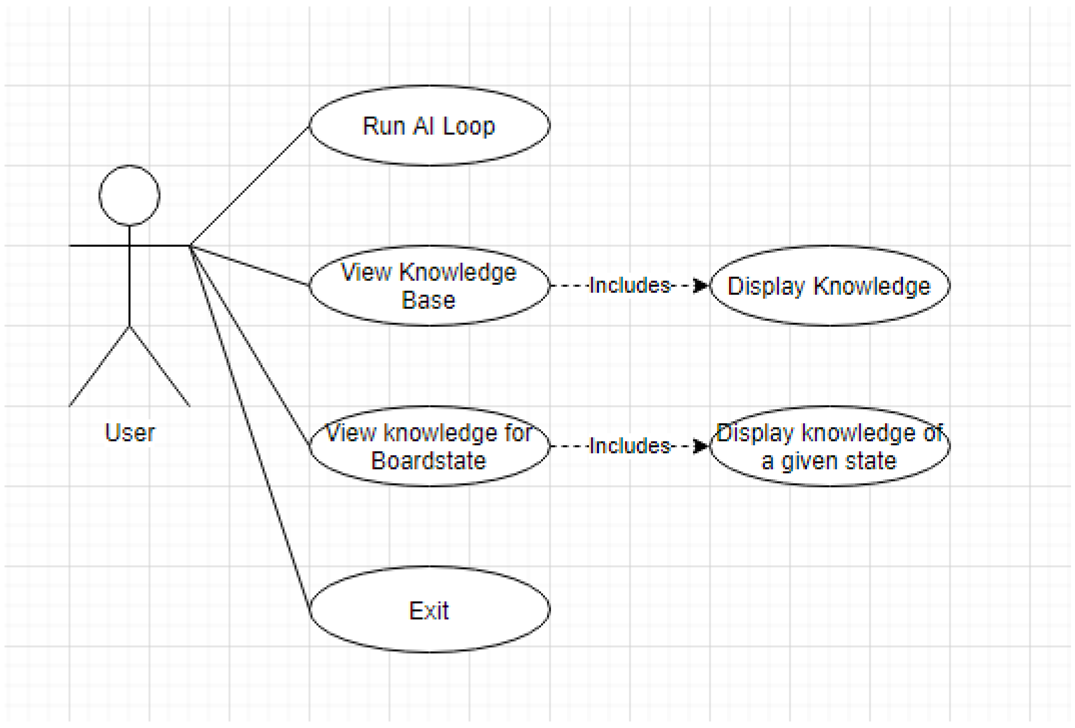


Figure 2.4.3: Usage cases for the user in the system

Please note that the functionality depicted in these diagrams is subject to change at any point during development of the system. Do not take this as a representation of a final product.

Chapter 3. Requirements

3.1 Stakeholders

While I aim for this project to be able to benefit literally anyone who can access it, it is likely to be most useful for salarymen and other working individuals whose time away from work is limited, as well as the organisations they belong to who may use the program to create more efficient training practises. One of the power points of this project is that it's benefits can be reaped by anyone who wishes to learn how to play a card game, or other such similar hobby.

3.2 Gathering requirements

Whilst most of the requirements of my project were either brainstormed from my own thoughts or taken from similar projects, I did conduct a survey of users.

They were asked the following questions.
1. Do you feel like learning something new is usually too hard?
2. Do you feel like the difficulty of learning something new dissuades you from learning something new?
3. Do you feel like learning something new takes too much time?
4. What is it about learning something new that you find difficult?

The results of the surveys conducted showed a strong user preference

This allows me to design the program's interface and functionality more closely to the wants of a user.

The Literature survey also gave me insight into how some of these programs may come to be. Although I did not look at any source code, the features of these projects did help me to realise what may be necessary for my own project to function.

The rest of the functional requirements were brainstormed from my own mind over multiple sessions and discussion with Dimitris, over these sessions the required features of the program were narrowed down to certain features that can be listed and found in the next section.

I followed a logical train of thought to compile the list of requirements, upon thinking "I need to apply a reinforcement learning algorithm in a noughts and crosses environment", I took the problem apart from all angles in my head to see where I could start. In order for the AI to play noughts and crosses, it needed a board to play on. After all, a board can exist without any players but the AI requires a board or it would lack any functionality whatsoever. What would the board require to function? Methods to put pieces on the board, and thus a way to validate locations to put pieces.

Then a way to recognise win states on the board. This formed the basis of the functionality requirements of the program, to go alongside the user requirements gathered earlier. It was these requirements that I initially fulfilled during early development.

Notes: Detailed results/raw data from the process of gathering requirements will be included in an Appendix in the Final Project Report.

3.3 Modelling requirements and relevant diagrams

The program operates in this way. The board handles placement of moves made by the AI and checks for wins. The AI takes the board as an argument and produces a move based on the state of the board. The system serves as a go between that handles sending the board state to the AI and tracks statistics of the AI's play as well as the state of the current game.

The system, AI and board are the actors in this model. The following diagram depicts the actions of the system in a downwards motion.

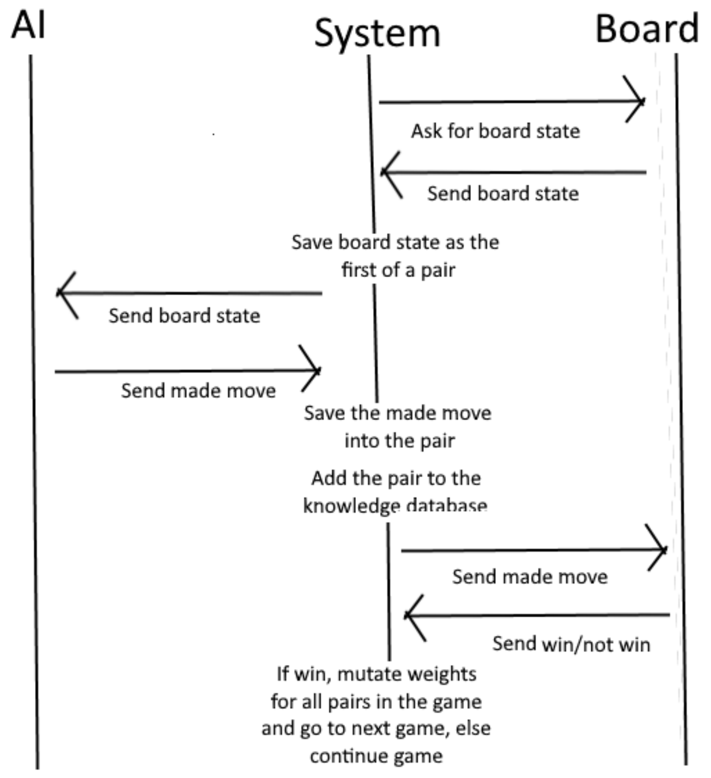

AI
System
Board
Ask for board state
Send board state
Save board state as the first of a pair
Send board state
Send made move
Save the made move into the pair
Add the pair to the knowledge database
Send made move
Send win/not win
If win, mutate weights for all pairs in the game and go to next game, else continue game

Figure 3.3.1: explanation of the inner workings of the program in a downwards motion.

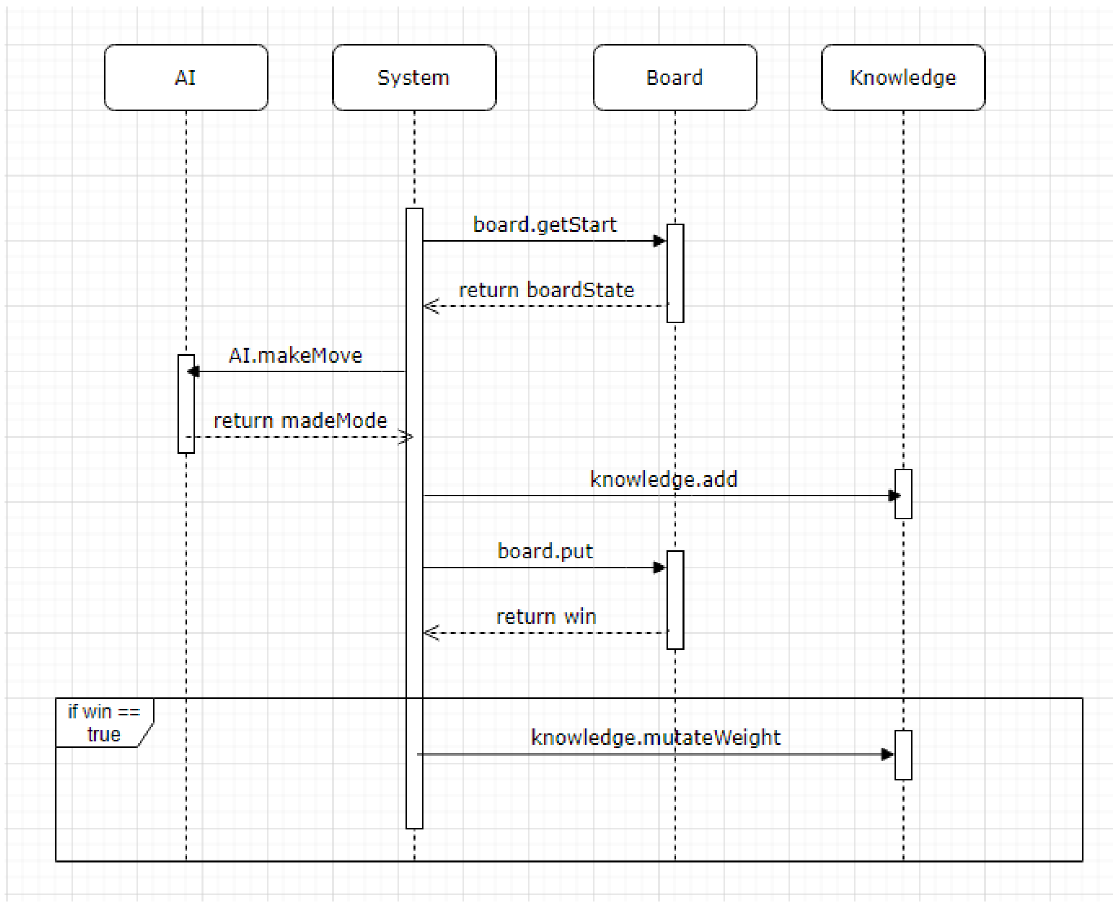


Figure 3.3.2: depiction of the previous diagram as a proper sequence diagram showing how the AI loop operates.

The main loop of the program operates in this way. The system first calls the getState method of Board, and appends it to an array named pair. The makeMove method of AI_test is then called and the board state is passed as its argument. The return value of makeMove is saved and then appended to the pair array that was used earlier. This pair is added to the knowledge database, with a default weighting of 100 if the pair is new. This makes up the system of weighted moves mapped to board states. .The system then calls the put method of Board, and passes it 2 arguments, the symbol that will be written and the move AI_test made earlier. This method returns true or false. True meaning that this was a winning move, and false meaning it was not and the game will continue. Upon a win, the system will notify the user that this AI has won and will print the statistics for this match, increase the weighting of the moves that were made in this iteration and will proceed to the next game. Upon continuation of the game, the system increments the round counter and checks if the round counter is greater than 9. If it is greater than 9 this means the game will end in a draw and the system will print the statistics of the match and end this iteration. If not greater than 9, the system calls the makeMove method of AI_control and passes it the board state as an argument. As with before, this method returns true or false and the same actions as described earlier take place. The loop then resets and the next move by AI_test will be made as described earlier. The loop continues to reset until a win by player X, Player O or a draw is achieved.

The AI system operates via reinforcement learning. When the makeMove method is called, the AI retrieves the board state and calculates all possible moves that can be made, representing them 0 through 8. It then retrieves all knowledge of the possible moves for this board state. This is given as pairs of possible moves with their weighting associated with them. If there is no knowledge for the board state, it creates default weights of 100 for each possible move. The weights are fed into a weighted selection algorithm, with the most weighty moves having a higher chance to be selected. The selected move is then returned to the system where it can then be passed to the board.

The knowledge base of the AI works as such. The knowledge is saved in a system of nested dictionaries. The data is saved in the following format; [boardState [possibleMove : weight]]. Each board state makes for a key in the first layer of the dictionary, and each key's value has another dictionary where the following moves are mapped to their weights. as such, addressing dictionary[boardstate] returns a dictionary of possible moves for that state, and addressing dictionary[boardstate][0-8] returns the weighting for that possible move. When adding knowledge (boardstate - made move pairs) to the knowledge base, it first checks if the knowledge is already there, if not it adds it with a default weighting of 0, if it is there then the values do not change.

As such, I have determined what is required for the system to operate. Viewable in the next section.

3.4 List of project requirements

Functional:

(All functional requirements I have deemed to be essential. This is due to the fact that each of these support a feature that is required for the whole core systems to operate)

Board:

B.1 Must be able to place moves on only valid spaces:
This is to avoid bugs and errors caused by incorrect placement of moves. If the system can assume that any board state given to it is valid, there would be no need for error checking anywhere other than the initial placement of the pieces, which saves on computation time and development time.
B.2 Must be able to recognise wins:
Self evident. This allows the win detection to be offloaded from the main system and into the board methods, making the system loop easier to understand.

AI:

A.1 Must be able to select a move from all the possible moves:
Self evident again. When giving a list of possible moves the AI has the potential to pick from them. The algorithm for choosing is subject to change over the course of the development of this project.
A.2 Must be able to store knowledge of previously made moves:
A requirement for learning to occur, without knowledge of previous moves, the AI would never be able to learn from them and improve its play.
A.3 Must be able to select a move using a weighted selection algorithm:
Requirement for the actual learning process. Weighted selection algorithm in tandem

with requirement A.1 allows for selection algorithm with a preference for certain moves, in this case, more successful moves.

System:

S.1 Must provide an interface for the Board and AI to communicate:

Since the AI is not built into the board, (as intended, as it is designed to be flexible and be able to adapt to different environments) it is necessary for communication between the AI and the board to be facilitated by the system.

S.2 Must provide handling of match data.

Whilst not exactly being functional, it is a requirement to be able to witness the improvement the AI incrementally makes.

S.3 Must be able to pass knowledge of moves to the knowledge base:

This enables the processing of the knowledge, both by the user and AI. The system facilitates the AI's memory of the past events.

S.4 Must be able to mutate weighting in the knowledge base based on wins/losses/draws

In order to enable the weighted selection to gain an effect, this increases/decreases the weightings in response to win/draw/loss scenarios.

Non-functional:

Essential:

E.1 A way for the user to query the AI's knowledge base, for either the whole base or for one board state.

Required for implementation of user learning features into the project. This allows the user to acquire the examples they might need to help them along a path of progression.

Desirable:

D.1 The system presents the data it has gathered in an easy to see User Interface

System is modular and can be quickly reconfigured to work in different games and environments.

This can be useful to increase the enjoyment the system provides the user.

D.2 The system shows an interface where a user can watch the system's AI Loop as it is running.
For the user to watch the matches in progress. Normally the matches speed along at the maximum speed allowed by the CPU and thus the user would have no time available to watch matches unfold.

Luxury:
L.1 The system can run on multiple threads
Quite a desirable thing to achieve. This would allow the processing capabilities of the program to increase tremendously and perform many times quicker.
L.2 The system can be distributed over multiple devices.
TicTacAI on the move! This is a convenience based requirement that would allow the user to enjoy the benefits of the program on their smartphones or tablets.

3.5 Legal, social and ethical issues

The main legal reason associated with any AI programs or anything that includes machine learning is accountability [Artificial Intelligence: the real legal issues, 2017]. Who should be held responsible for the actions of AI?

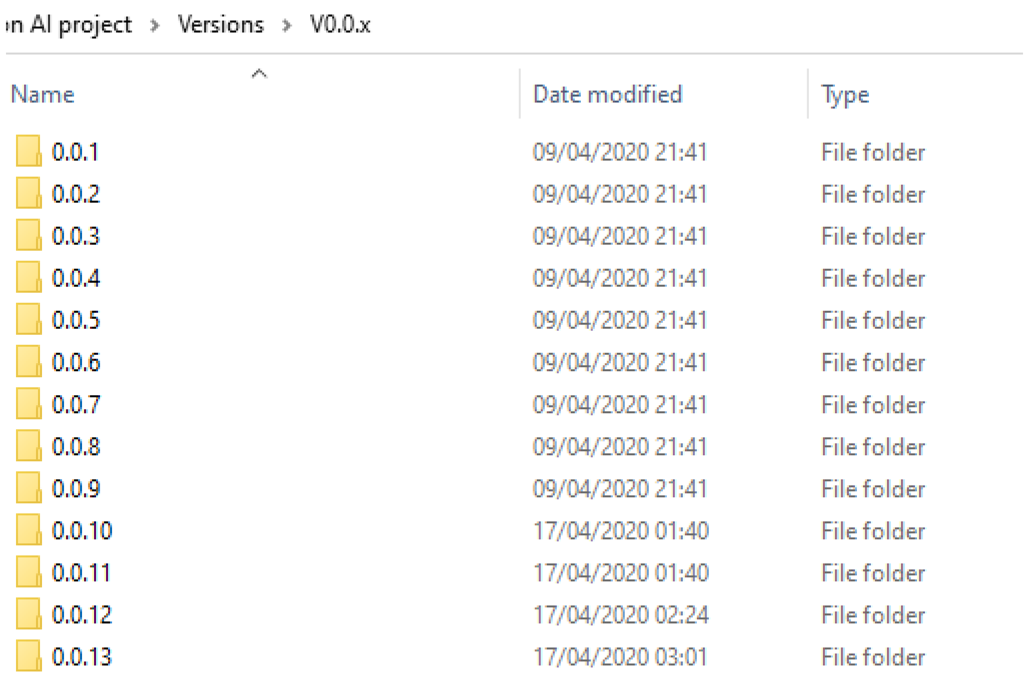
For my project, who is responsible if the AI does not achieve the 100% rate that I aim for? Theoretically it should always reach 100% success rate with enough games played, but the amount can vary wildly between iterations of the system. Throughout countless tests I have run myself, I have determined that the AI does tend towards the 100% success mark, but has yet to reach it and the rate of growth can vary from one test to the next.

Another issue, social in nature, is that AI of this sort can be seen to be pushing teachers and tutors out of business. Programmers have a duty to support society with their creations, and although consumers would benefit immensely from this sort of technology, it would create competition with humans who teach and tutor the same areas. Whilst my program is not designed to be a replacement for a teacher, and is merely a tool to be used for enhancing learning, there will inevitably be people who do not pursue tutoring because of software such as my program. As such, am I and other similar developers to be held socially accountable for any increasing unemployment in hobbyist tutors? However, it could also be said that software such as this is merely outcompeting people in a field where they were all competing against each other anyway.

Chapter 4. Prototyping Methodology

In my research into common development methodologies, I have uncovered a crucial factor. Most commonly used methodologies define ways in which small teams or whole organisations work together in order to create a product. I am a single developer and as such the most effective methodology would be one that matches both my project and my own personal development preferences. I prefer diving into development as soon as possible and not being contained by strict development practises, and as such agile development based methodologies are best suited for me.

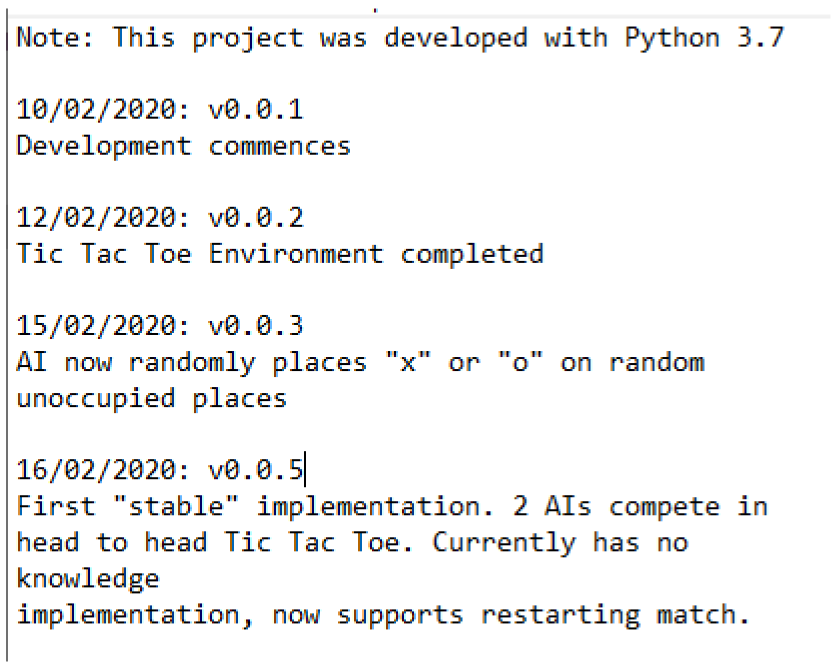
As for my project, they are also well suited because the structure of the program allows later versions to be built upon earlier versions, as can be seen within the project folder packaged with this document.



ın AI project > Versions > V0.0.x

Name	Date modified	Type
0.0.1	09/04/2020 21:41	File folder
0.0.2	09/04/2020 21:41	File folder
0.0.3	09/04/2020 21:41	File folder
0.0.4	09/04/2020 21:41	File folder
0.0.5	09/04/2020 21:41	File folder
0.0.6	09/04/2020 21:41	File folder
0.0.7	09/04/2020 21:41	File folder
0.0.8	09/04/2020 21:41	File folder
0.0.9	09/04/2020 21:41	File folder
0.0.10	17/04/2020 01:40	File folder
0.0.11	17/04/2020 01:40	File folder
0.0.12	17/04/2020 02:24	File folder
0.0.13	17/04/2020 03:01	File folder

Figure 4.0.1: Screenshot of the project folder, showing the different versions of the project.



```
Note: This project was developed with Python 3.7

10/02/2020: v0.0.1
Development commences

12/02/2020: v0.0.2
Tic Tac Toe Environment completed

15/02/2020: v0.0.3
AI now randomly places "x" or "o" on random
unoccupied places

16/02/2020: v0.0.5
First "stable" implementation. 2 AIs compete in
head to head Tic Tac Toe. Currently has no
knowledge
implementation, now supports restarting match.
```

Figure 4.0.2: Development Log screenshot showing how later versions built upon earlier versions and added features along the way.

Thus I decided that a prototyping based methodology was best for this project. This is because the main goal of this methodology is evaluation of an idea. I am a person who is quick to develop ideas and thus this methodology is suited to eliminating problems with the ideas I develop swiftly and creating working prototypes within a short amount of time from conception.

The main steps of a prototyping based methodology are as follows;

1. Determine objective/requirements
2. Initial development
3. Refinement
4. Demonstration
5. Testing
6. Implementation

After narrowing down the functional requirements for the program's core features via analysis of existing systems and my own ideas, I brainstormed how I may develop the project. I did this by narrowing down which requirements would need other requirements to be implemented beforehand. This can be seen in the development log

I have included in the project folder under Python AI Project/Versions/Development Log.txt. There I detail which features were implemented first and used as a basis to build into the other requirements of the project. I used diagrams to try to envision how my ideas of certain requirements could be implemented.

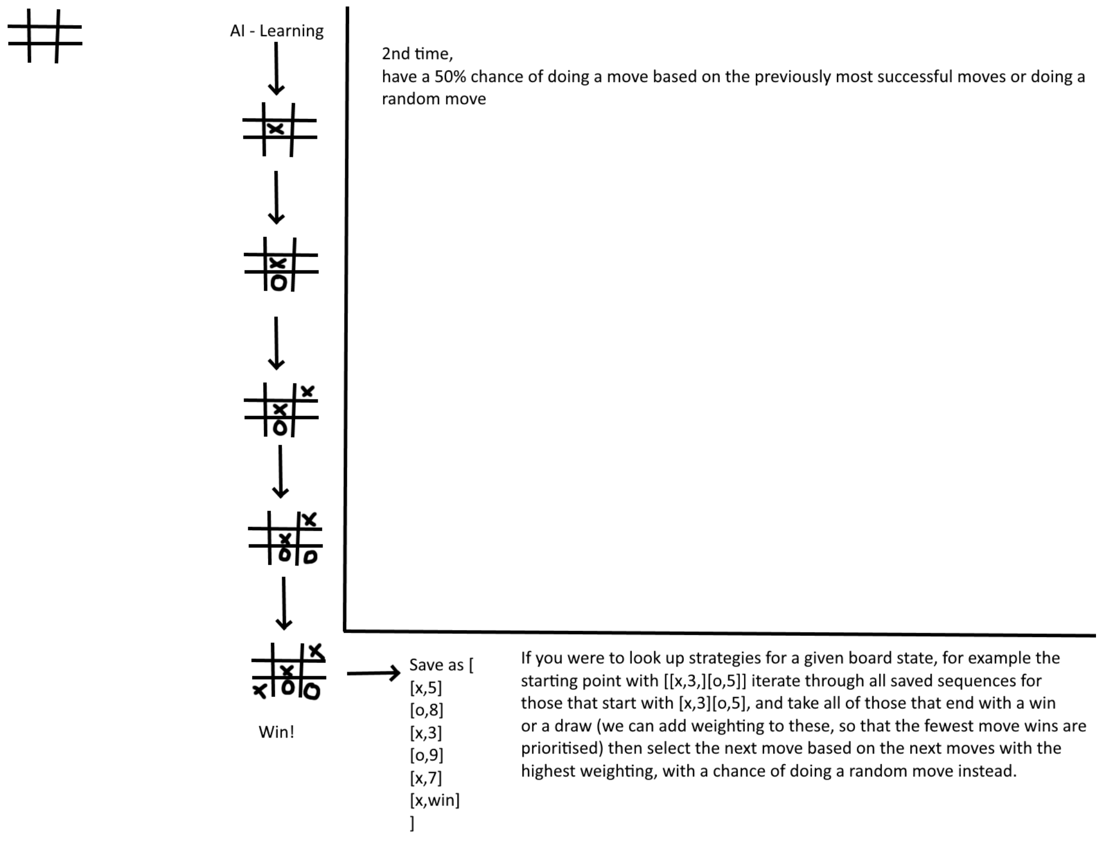

AI - Learning
2nd time,
have a 50% chance of doing a move based on the previously most successful moves or doing a random move
Win!
Save as [
[x,5]
[o,8]
[x,3]
[o,9]
[x,7]
[x,win]
]
If you were to look up strategies for a given board state, for example the starting point with [[x,3,][o,5]] iterate through all saved sequences for those that start with [x,3][o,5], and take all of those that end with a win or a draw (we can add weighting to these, so that the fewest move wins are prioritised) then select the next move based on the next moves with the highest weighting, with a chance of doing a random move instead.

figure 4.0.3: an early diagram showing how the original knowledge system was to work. Whilst being somewhat confusing to interpret in hindsight, it served more of a purpose of allowing myself to figure out how the program could function

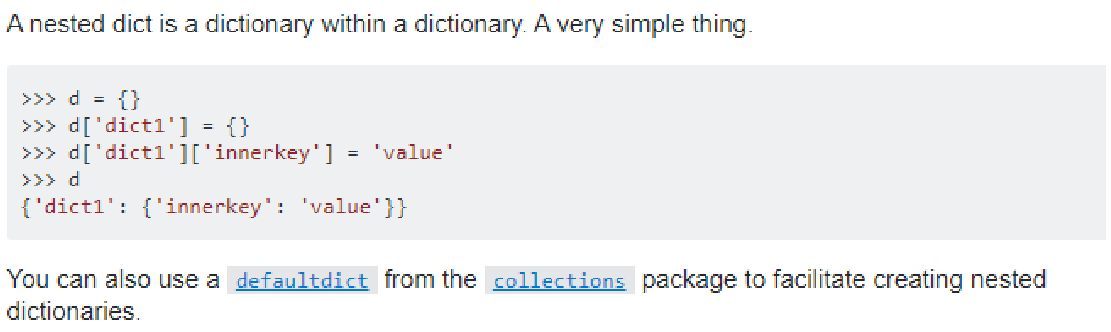
Early versions of the program focused on prototyping the board system, as the AI required an environment to function in. Creating the AI without the board would be meaningless as I would have no way to test if the AI was capable of reading the board state and placing moves upon the board.

I would create an early prototype to test if a functional requirement was met, and would improve it if the requirement criteria wasn't met. For example, an early version of the checkWin algorithm devised could not recognise if a win was achieved with diagonally placed pieces on the noughts and crosses board. This was corrected in the next versions of the program with pieces of code inspired by an article on Stack

Overflow. (Python programming 3.4.2, Noughts and crosses. Best way to check how to see if game is won? 2015)

4.1 White box test

I used white box testing on the code that was shared on stack overflow that I used as *inspiration* for my own code. It is imperative that all code that is entered into a project must be completely understood by the developer, or unforseen bugs may occur, which is why I tested all code lifted from external sources or derived from external sources. This can be observed in my implementation of the nested dictionary used to store knowledge data. When comparing the code to the example I found at stack overflow (smci, How do you create nested dictionary in python? 2013)

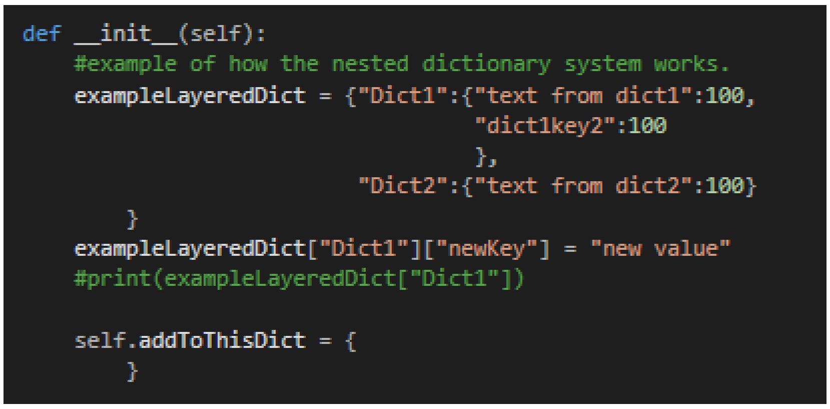


A nested dict is a dictionary within a dictionary. A very simple thing.

```
>>> d = {}
>>> d['dict1'] = {}
>>> d['dict1']['innerkey'] = 'value'
>>> d
{'dict1': {'innerkey': 'value'}}
```

You can also use a `defaultdict` from the `collections` package to facilitate creating nested dictionaries.

Figure 4.1.1: a screenshot of example code taken from stack overflow.



```
def __init__(self):
    #example of how the nested dictionary system works.
    exampleLayeredDict = {"Dict1":{"text from dict1":100,
                                   "dict1key2":100
                                   },
                          "Dict2":{"text from dict2":100}
        }
    exampleLayeredDict["Dict1"]["newKey"] = "new value"
    #print(exampleLayeredDict["Dict1"])

    self.addToThisDict = {
        }
```

Figure 4.1.2: screenshot of how I tested how the nested dictionary idea from stack overflow might work

I tested various inputs to see what could be printed out of the nested dictionaries, as can be seen in the commented out print statement. This statement was altered much over the course of development of versions 0.0.9 through 0.0.12 and was eventually commented out in version 0.0.13, once the functionality of the nested dictionary was fully understood by myself.

This following table shows how tests upon the nested dictionary were carried out.

Input	Expect outcome	Actual outcome	Success?
exampleLayeredDict["Dict1"]	"text from dict1", 100, "dict1key2", 100	{'text from dict1': 100, 'dict1key2': 100}	Yes
exampleLayeredDict.keys()[0]	"text from dict1", "dict1key2"	error: dict_keys is not subscriptable	no
exampleLayeredDict[example LayeredDict.keys()[0]]	"text from dict1", 100, "dict1key2", 100	error: dict_keys is not subscriptable	no
exampleLayeredDict.keys()	“Dict1”, “Dict2”	dict_keys(['Dict1 ', 'Dict2'])	Yes
exampleLayeredDict["Dict1"]["dict1key2"]	100	100	Yes

Figure 4.1.3: white box test of the functionality of a nested dictionary

This testing gave me the knowledge to determine that nested dictionaries were to be referenced by calling dictionary[key][key] and could not be referenced by order

My program versions were based upon incrementally implementing the functional requirements of the program, with new versions coming out sometimes as quickly as every day. Board implementation and win condition recognition were completed by version 0.0.2. Random move generation upon the board came after in version 0.0.3 and the version after that had 2 random move competing “dumb” AI. This pattern of building upon the previous versions continued throughout the whole project.

This cycle of a prototype inspired by online sources and ideas of my own creation, its refinement and subsequent perfection made up the main development loop for this project.

Chapter 5. Design

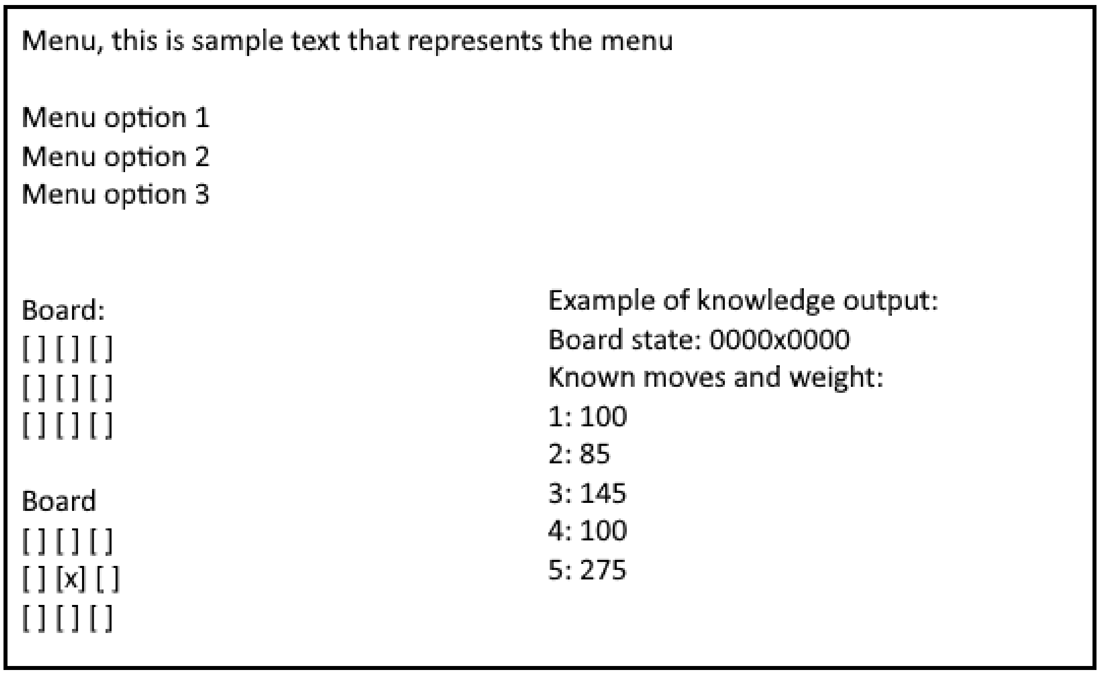
5.1 User Interface

The system requires a main menu in order for the user to interact with the different functions of the program, a view in which the user can see the AI loop running and views for the user to see specific and total knowledge gained by the AI.

I only have one instance of prior experience creating GUI in python, which was using the Display object of the PyGame module several years ago. As such I had no idea how I may initially create a GUI for the program. After considering it, I decided that the functionality of the program should take priority in development, and that a GUI or at least console based UI could be developed later. In order to facilitate this, the code would have to be designed in such a way to make the output from method calls modular, able to be displayed straight onto the console or within a purpose built GUI.

Eventually I decided that the program UI will run in the console. This is because I decided to prioritise development of the AI algorithm and system loop. Also, because this program is essentially a smart data cruncher made for noughts and crosses, its output data (board moves) can be displayed in console with the same fidelity as in a purpose built GUI. The board and menus are easily enough displayed as lines of text.

The requirements merely call for a method for the user to A. see the running of the AI loop, B. see the knowledge base of the AI. This is all able to be shown in the console, as can be seen in the below figure.



```
Menu, this is sample text that represents the menu

Menu option 1
Menu option 2
Menu option 3


Board:                              Example of knowledge output:
[ ] [ ] [ ]                         Board state: 0000x0000
[ ] [ ] [ ]                         Known moves and weight:
[ ] [ ] [ ]                         1: 100
                                    2: 85
Board                               3: 145
[ ] [ ] [ ]                         4: 100
[ ] [x] [ ]                         5: 275
[ ] [ ] [ ]
```

Figure 5.1.1: Diagram of UI structure.

5.2 Infrastructure and Maintenance

How would this program be maintained once distributed? If any updates were to be created by myself, how would the end user be made aware of it? How would bug fixes be distributed? would they fall under the same version number as the version with the bug?

This would be outside the bounds of the requirements discussed in chapter 3, but would pose a problem for the distribution of this program. While different program versions can be hosted online rather easily, the user would have to check for new versions independently. One possible solution would be to contact a server upon starting every time, in order to check for the latest product version and compare it to the version of the current system. Upon the check returning true for newer versions, the user would be prompted to download the latest version at the server address. The user could elect to proceed with the current version with no consequences other than lack of new features however.

As for the structure of the program, the 3 main areas, AI, Board and System are designed with modularity in mind. The board itself has a hardcoded noughts and crosses board, in array format, but is programmed with length variables that can flex and would accommodate boards of different sizes, as long as they could be defined on a 2d plane. This is fine for a noughts and crosses environment, but functionality that allows the user to develop boards for other game types should be created and would be a viable way to extend this program. This requirement for modularity in the system was not fully met, but during development was discovered to be a much larger challenge than was thought.

FULL modularity would require all features of the program to be created programmatically at runtime, without predefined functions. For example, the checkwin method of Board would not be modular as it would only work for the noughts and crosses ruleset.

Thus I have settled for semi modularity. The program will be kept as modular as possible while still developing it in a noughts and crosses environment.

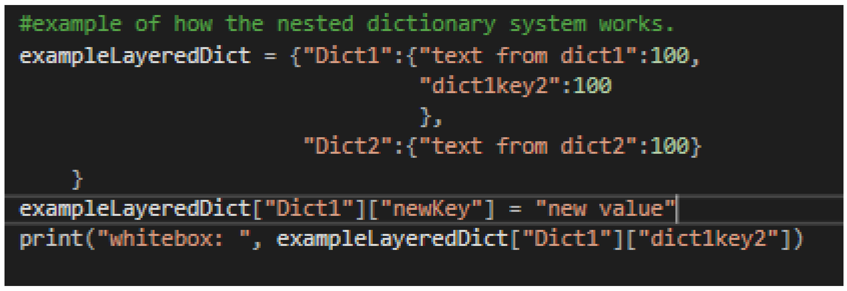
5.3 Functionality

The main issues with the functionality in the program were how far the functionality was to extend. The original requirements were to allow the user to view AI generated knowledge of particular board states. This originally encountered issues with the preliminary method of knowledge storage and injection into the current knowledge bank, as the original method called for knowledge to be stored in sequences of moves. This caused a problem where different moves that resulted in the same board state would not be recognised as the same board state. This would lead to artificial bloating of the knowledge base and reduction in its effectiveness, as the same board states would have to be repeated due to them having different precursor moves. The solution to this was developed during a brainstorming session, where it was suggested to me that board states were saved as text strings and not sequences of tuples, which eliminated the sequence of move based approach to knowledge.

The viewing of AI knowledge is to be implemented as a menu option, upon prompting the user enters a valid boardState and the program retrieves its associated dictionary from the knowledge base. This part of the program can be developed in a modular fashion, as the knowledge dictionary can accept all inputs of a valid board state, even in different fashions of games. as long as the board state can be defined in text string format, ergo, be able to be used as a dictionary key.

5.4 Algorithm Development

Most of the issues concerning development of algorithms used in the development of this program came from the difficulty in their creation. The algorithm that handles knowledge was particularly tricky and took multiple versions to refine to a functional capacity. The knowledge dictionary was particularly difficult because all of its elements were to be added programatically during the system loop, and it started with none. This makes it extremely difficult to visualise, and was the entire reason I included a sample dictionary in the code so that an example could be followed.



```
#example of how the nested dictionary system works.
exampleLayeredDict = {"Dict1":{"text from dict1":100,
                               "dict1key2":100
                               },
                      "Dict2":{"text from dict2":100}
    }
exampleLayeredDict["Dict1"]["newKey"] = "new value"
print("whitebox: ", exampleLayeredDict["Dict1"]["dict1key2"])
```

Figure 5.4.1: Example of the nested dictionary.

The other algorithm, the "weighted selection algorithm" was easier for me to envision. I had the idea for an algorithm that chose larger numbers over smaller numbers from a previous project I worked in. The algorithm worked by assigning weights to the selection targets, adding up the weights and selecting a random number from 0 to the sum of the weights. The weights are then subtracted from the random number until it hits 0 or below. The weight that caused the number to go below zero is selected as the output for the algorithm. Using this method, higher weighted targets are more likely to be chosen (200 weight has 2x higher chance than 100 weight, for example). This can just be implemented as a method that returns the selection target.

5.5 Content Creation

Since this program has no images or anything to speak of, creation of media to use for the program did not pose an issue.

5.6 Tools

The project is mainly based in vanilla code that does not require external modules that are not already included in the base install of Python 3.7, as such tools that will help me develop the program rather than extend the functionality are what will be used. Since Visual Studio is the established IDE to use, all issues can only originate from VS or the python modules I import for the functionality of the program. I love Visual Studio as an IDE and I have little to no problems with it for personal use. It was a little vague in how to switch to a python environment, as I had to consult an online guide by microsoft, which could have limited my capabilities if I were to develop offline, however this was never a situation that occured.

Chapter 6. Implementation

6.1 Tools

I used a Vanilla Python 3.7 (64-bit) Python Environment in Visual Studio Community 2019, with only
I used python mostly for the details already discussed earlier in section 2.3. It is a familiar language with the possibility of expendanded functionality using Python specific libraries and frameworks such as TensorFlow. It is also supported as a development environment in Visual Studio, my IDE of choice for most projects.

I chose Visual Studio as it is a familiar IDE that has the full backing of Microsoft and the Visual Studio development Community. It has an easy to navigate menu and provides all the features I would need to develop most programs. It is inferior for mobile development to Android Studio, but this was not a problem as I chose to forgo mobile development for this program.

I used the Time module of python in order to implement a counter that can measure how fast each iteration of a noughts and crosses game is completed. This allowed me to estimate how long X amount of iterations would take. On my home machine, games tended to last 30-50 milliseconds. On a university machine they tended to last 100-200 milliseconds. However, after a few hundred iterations, I noticed that the number tended to decrease to single digits, such as 8 or 9 milliseconds. I have no idea why this happens.

6.2 Implementation

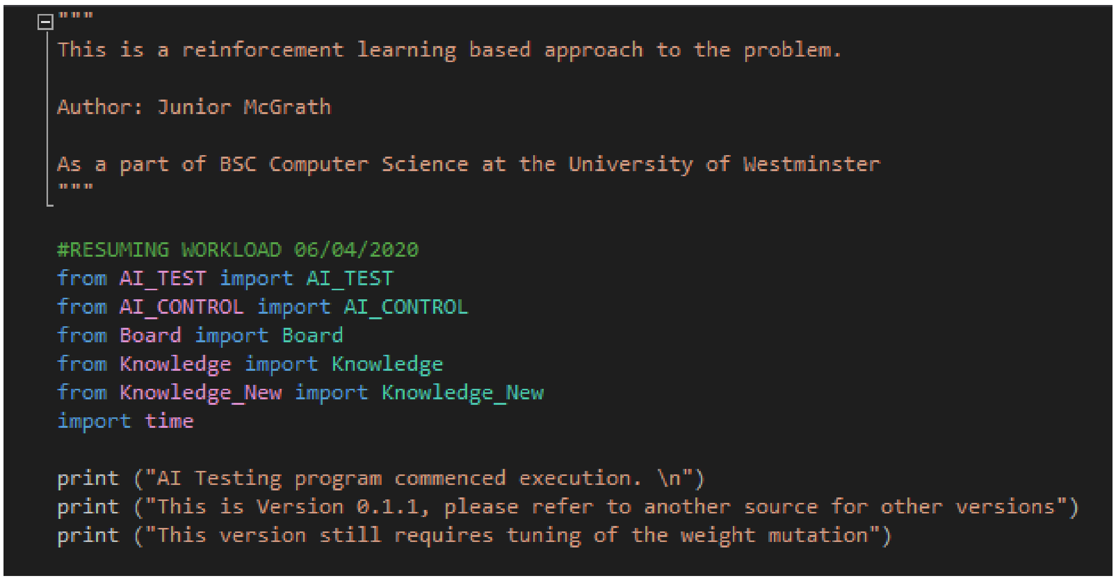
The program is made up of 6 python files.

1. Python Reinforcement Learning AI test.py. The file that contains the Main method of the program and the system loop.

2. AI_CONTROL.py. The twin of AI_TEST, this python file contains an AI class that can only perform random actions and has no knowledge of its own.
3. AI_TEST.py. The twin of AI_CONTROL, this python file contains an AI class that has implemented knowledge and selects actions based on weightings that are influenced by previous play.
4. Board.py. This python file contains a board class that is configured for the game noughts and crosses. It has checkWin methods and an array board that only follow the noughts and crosses ruleset.
5. Knowledge.py. The original knowledge file that was developed. This version only supported knowledge of the first move played and was a prototype for the following Knowledge_New class. It was dropped in v0.0.9 for the newer Knowledge_New that was developed from it.
6. Knowledge_New.py. This is the updated knowledge file that is used in the final version of the program. It contains a class that handles input to a nested dictionary and can query the dictionary for board states, before displaying them to the user.

6.2.1 Main Method and the System Loop

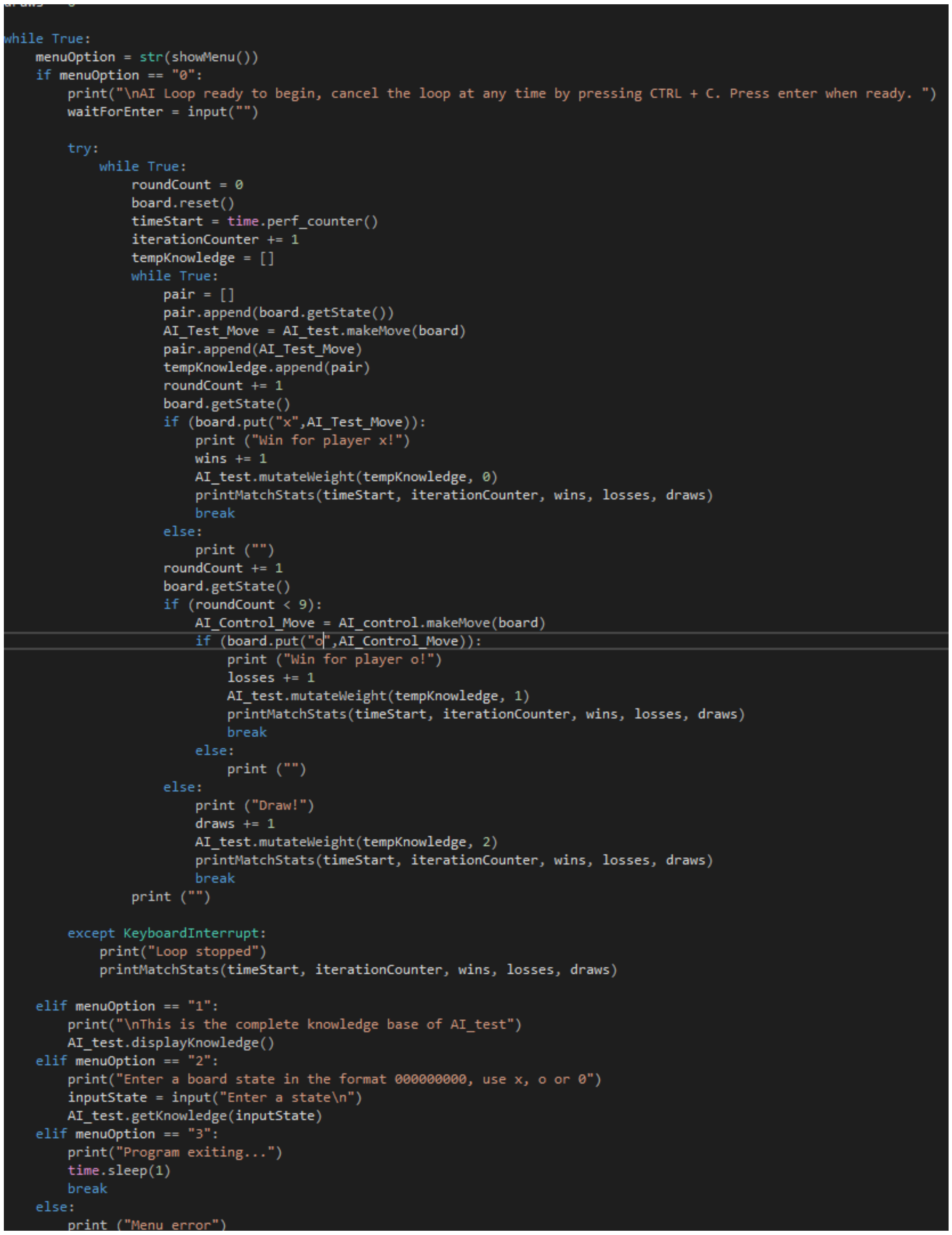
The start of the program.



```
"""
This is a reinforcement learning based approach to the problem.

Author: Junior McGrath

As a part of BSC Computer Science at the University of Westminster
"""

#RESUMING WORKLOAD 06/04/2020
from AI_TEST import AI_TEST
from AI_CONTROL import AI_CONTROL
from Board import Board
from Knowledge import Knowledge
from Knowledge_New import Knowledge_New
import time

print ("AI Testing program commenced execution. \n")
print ("This is Version 0.1.1, please refer to another source for other versions")
print ("This version still requires tuning of the weight mutation")
```

Figure 6.2.1.1: Pre-code. Import of classes from other python files and time module for usage of iteration length tracking.

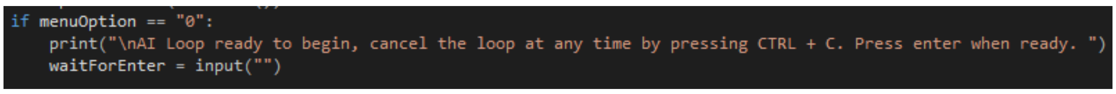
This stores information about the program here, including the version number and user information. Important for knowing which version of the program the user/developer is on.



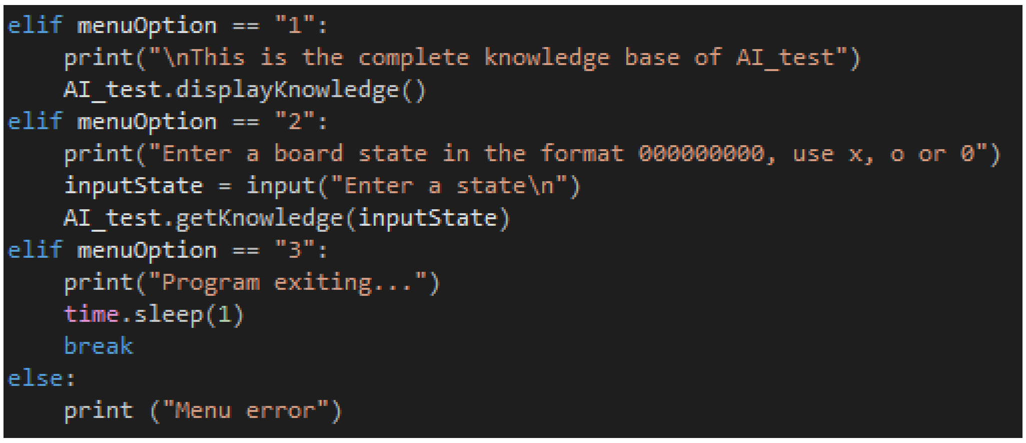
```
while True:
    menuOption = str(showMenu())
    if menuOption == "0":
        print("\nAI Loop ready to begin, cancel the loop at any time by pressing CTRL + C. Press enter when ready. ")
        waitForEnter = input("")

        try:
            while True:
                roundCount = 0
                board.reset()
                timeStart = time.perf_counter()
                iterationCounter += 1
                tempKnowledge = []
                while True:
                    pair = []
                    pair.append(board.getState())
                    AI_Test_Move = AI_test.makeMove(board)
                    pair.append(AI_Test_Move)
                    tempKnowledge.append(pair)
                    roundCount += 1
                    board.getState()
                    if (board.put("x",AI_Test_Move)):
                        print ("Win for player x!")
                        wins += 1
                        AI_test.mutateWeight(tempKnowledge, 0)
                        printMatchStats(timeStart, iterationCounter, wins, losses, draws)
                        break
                    else:
                        print ("")
                    roundCount += 1
                    board.getState()
                    if (roundCount < 9):
                        AI_Control_Move = AI_control.makeMove(board)
                        if (board.put("o",AI_Control_Move)):
                            print ("Win for player o!")
                            losses += 1
                            AI_test.mutateWeight(tempKnowledge, 1)
                            printMatchStats(timeStart, iterationCounter, wins, losses, draws)
                            break
                        else:
                            print ("")
                    else:
                        print ("Draw!")
                        draws += 1
                        AI_test.mutateWeight(tempKnowledge, 2)
                        printMatchStats(timeStart, iterationCounter, wins, losses, draws)
                        break
                print ("")

        except KeyboardInterrupt:
            print("Loop stopped")
            printMatchStats(timeStart, iterationCounter, wins, losses, draws)

    elif menuOption == "1":
        print("\nThis is the complete knowledge base of AI_test")
        AI_test.displayKnowledge()
    elif menuOption == "2":
        print("Enter a board state in the format 000000000, use x, o or 0")
        inputState = input("Enter a state\n")
        AI_test.getKnowledge(inputState)
    elif menuOption == "3":
        print("Program exiting...")
        time.sleep(1)
        break
    else:
        print ("Menu error")
```

Figure 6.2.1.2: The main system loop of the program.

This code follows the principles set forth by figure 2.4.1. The program waits for the user to enter a valid menu option number then proceeds to run the following code denoted in the elif statements.



```
if menuOption == "0":
    print("\nAI Loop ready to begin, cancel the loop at any time by pressing CTRL + C. Press enter when ready. ")
    waitForEnter = input("")
```



```
elif menuOption == "1":
    print("\nThis is the complete knowledge base of AI_test")
    AI_test.displayKnowledge()
elif menuOption == "2":
    print("Enter a board state in the format 000000000, use x, o or 0")
    inputState = input("Enter a state\n")
    AI_test.getKnowledge(inputState)
elif menuOption == "3":
    print("Program exiting...")
    time.sleep(1)
    break
else:
    print ("Menu error")
```

Figure 6.2.1.3: Elif Statements that make up the menu options

Upon entering the MenuOption 1 code block, the system displays a message notifying how the user can exit the AI Loop and executing the AI loop upon user input. The system then proceeds to the AI Loop itself.

The AI loop follows this basic methodology.

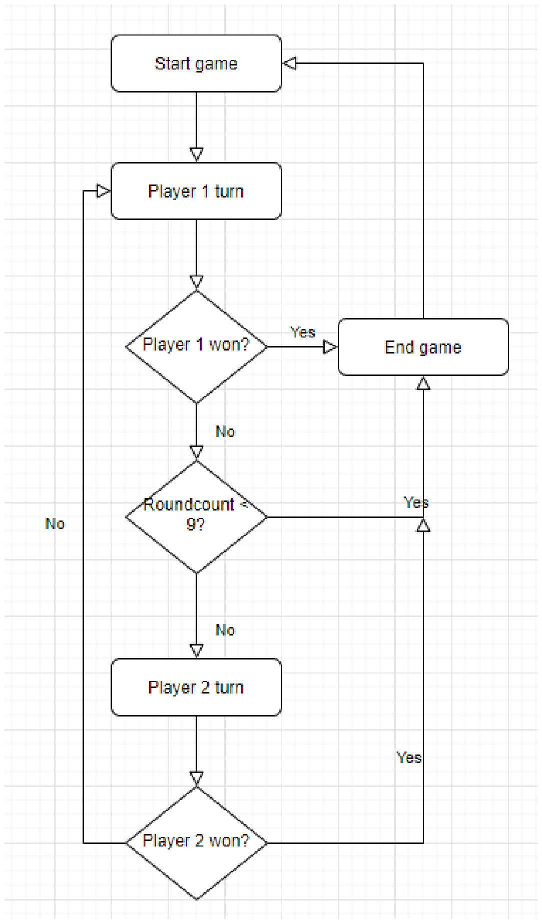
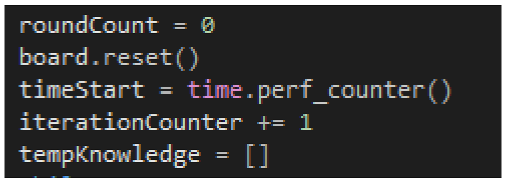


Figure 6.2.1.4: The main system loop

The system first cleans up the board and temporary knowledge and resets ready for a new game to start.



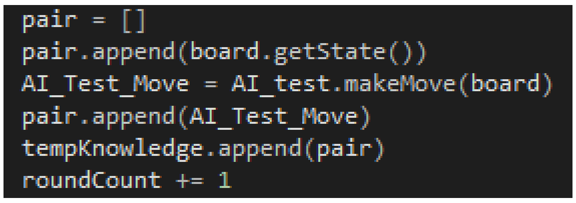
```
roundCount = 0
board.reset()
timeStart = time.perf_counter()
iterationCounter += 1
tempKnowledge = []
```

Figure 6.2.1.5: reset code pre-game

The round count, board state, time counter and temporary knowledge are reset to default values (nothing) and the iteration counter is incremented to serve as storage for how many games have been played.

The game then starts and the system enters a loop that ends when a game ending situation is reached, (Win/Loss/Draw).

The system then moves to the first player's turn, reaching the Player 1 state on the diagram.



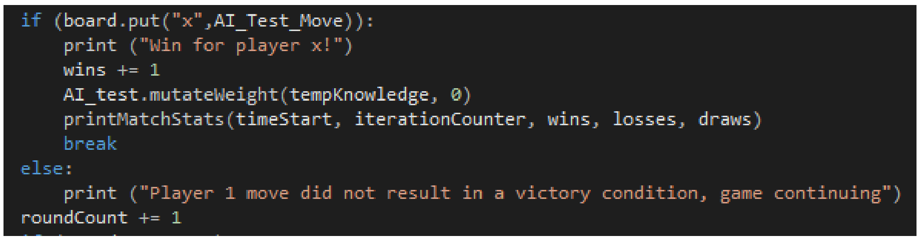
```
pair = []
pair.append(board.getState())
AI_Test_Move = AI_test.makeMove(board)
pair.append(AI_Test_Move)
tempKnowledge.append(pair)
roundCount += 1
```

Figure 6.2.1.6: the making of the player 1 move

An array called pair is created. Each “pair” will contain a move and its preceding board state. These pairs represent a singular move that was taken and will be used for the system to remember which moves were taken in a game. This is used later to mutate the weighting of the moves that led to a defeat.

The Test AI then makes a move, which is then appended to the pair. The pair is then complete with a Board State and Made Move. This pair is appended as temporary knowledge, an array that stores the pairs of moves that were made over the course of this game. The round counter then increments. This is because the round count is used to determine whether a draw state is reached, I.E, all places on the board are filled without a win for any player (A maximum of 9 rounds for a standard noughts and crosses game).

The system then proceeds to place the player’s move upon the board.



```
if (board.put("x",AI_Test_Move)):
    print ("Win for player x!")
    wins += 1
    AI_test.mutateWeight(tempKnowledge, 0)
    printMatchStats(timeStart, iterationCounter, wins, losses, draws)
    break
else:
    print ("Player 1 move did not result in a victory condition, game continuing")
roundCount += 1
```

Figure 6.2.1.7: placing player 1’s move upon the board

board.put is a Board class method that returns True or False, with True denoting a winning move and False denoting a move that did not result in a win. Essentially this code checks to see if the player’s move wins the game if placed upon the board and otherwise proceeds to the next round if not.

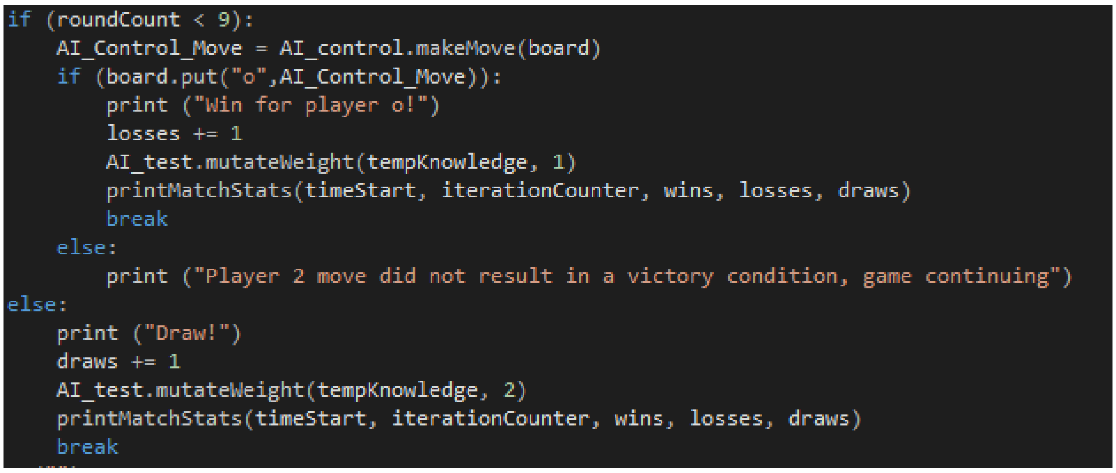
If the put method returns true, the system notifies the user and increments the win counter. The win counter is used to display gameplay stats after the game ends. When a win occurs, the system must mutate the weightings of the moves made in the current game, in order to make the AI more likely to choose winning moves or less likely to choose the losing moves. Thus it calls the mutateWeight method of AI_TEST.

This method takes the input and then calls mutateWeight on the knowledge associated with it. The reason I have developed it this way is so each AI can have its own knowledge, in case in the future 2 Self learning AIs could be put against each other, or player 2 could be self learning while player 1 does not learn. mutateWeight takes tempKnowledge (the moves that were made in this game) and an integer as arguments. The integer represents the win state of the board (p1 win, p2 win, draw).

The system then proceeds to print stats for the match, including the time taken for the match to conclude and the percentage of wins/loss/draw the AI is on, in order to see the progress the AI is making.

Note: Also showing the last 100 games win rate could further show the progression of the AI, as well as providing a chart that shows the change of the last 100 games win rate would, in hindsight, also be helpful for showing the progression of the AI over many iterations.

Upon the put method returning false, the system continues to the next round, player 2's round, and increments the round counter ready for the next round.

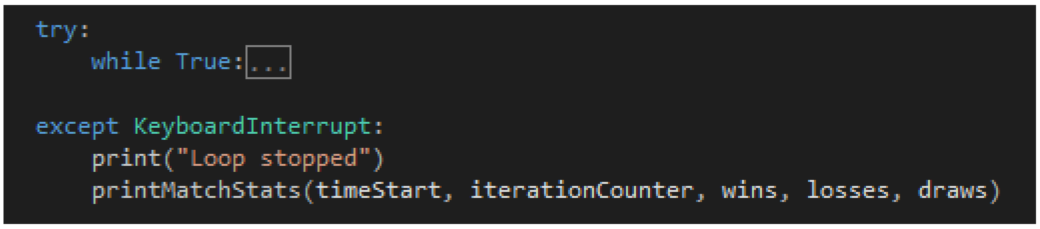


```
if (roundCount < 9):
    AI_Control_Move = AI_control.makeMove(board)
    if (board.put("o",AI_Control_Move)):
        print ("Win for player o!")
        losses += 1
        AI_test.mutateWeight(tempKnowledge, 1)
        printMatchStats(timeStart, iterationCounter, wins, losses, draws)
        break
    else:
        print ("Player 2 move did not result in a victory condition, game continuing")
else:
    print ("Draw!")
    draws += 1
    AI_test.mutateWeight(tempKnowledge, 2)
    printMatchStats(timeStart, iterationCounter, wins, losses, draws)
    break
```

Figure 6.2.1.8: draw check and player 2's turn

Player 2's round begins first with a round check, in order to check that the round limit has not been reached, if it has, the game must've been drawn. It follows the same method as the first player's move. If board.put returns true it proceeds to the player's move code and otherwise will proceed to the next round, (in this case, the end of this iteration of the while loop, and the subsequent reset of the while loop back to the top). This part differs from the previous by incrementing losses for p1 instead of wins, and sending 1 as an argument to the mutateWeight method instead of 0, denoting a loss for player 1. Otherwise it is the same as player 1's round code.

Just like before, the code is the same but flags different events. Draws are incremented and "2", the signal for a draw is sent to the mutateWeight method.

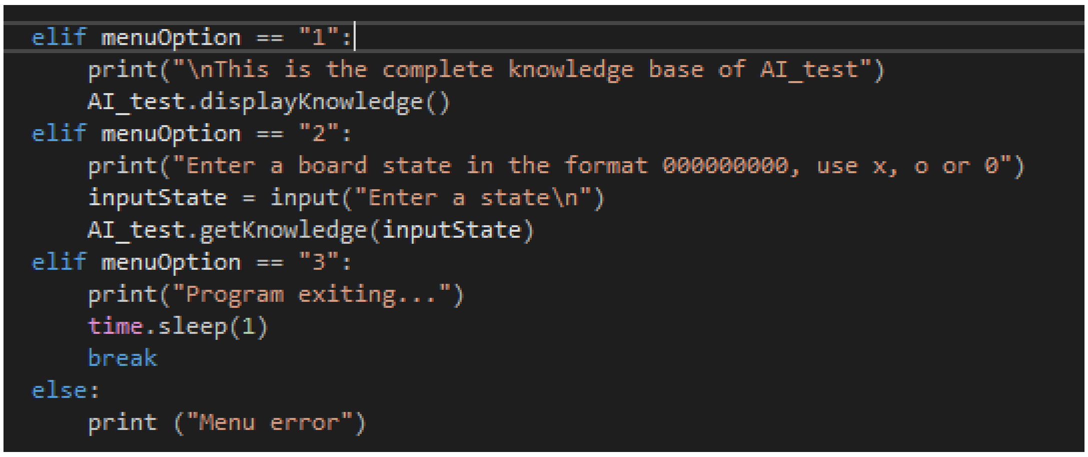


```
try:
    while True:...

except KeyboardInterrupt:
    print("Loop stopped")
    printMatchStats(timeStart, iterationCounter, wins, losses, draws)
```

Figure 6.2.1.9: Try-Except that handles exiting from the system Loop

This Try - Except encloses the code above, and is what allows the user to exit the AI loop. when the loop is in progress. pressing CTRL + C will cause the

KeyboardInterrupt to fire and exit the AI loop, ending the current game and printing the stats. The system then resets to the main menu.



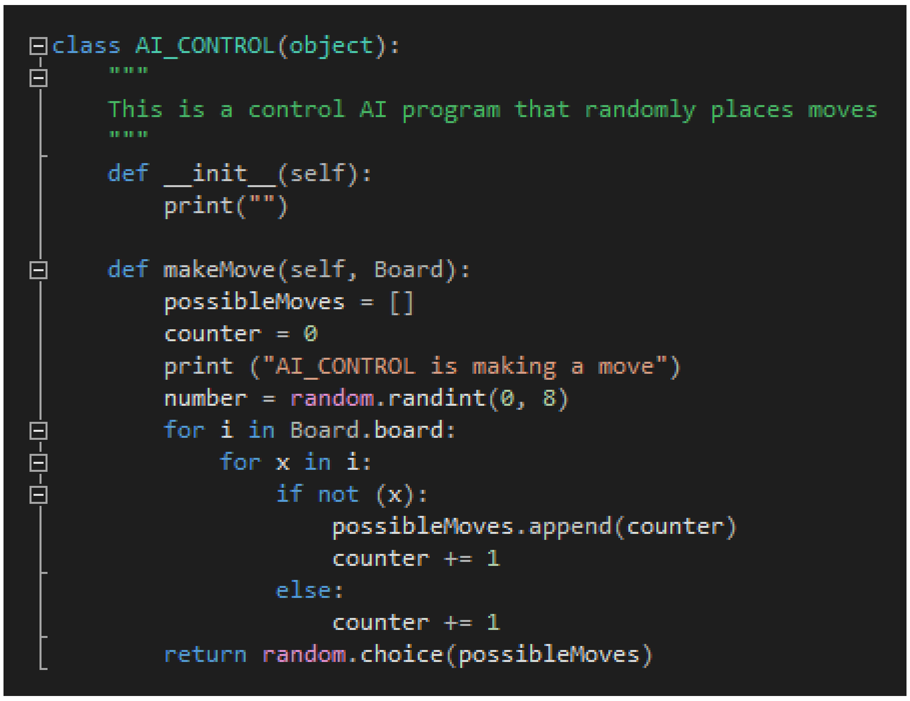
```
elif menuOption == "1":
    print("\nThis is the complete knowledge base of AI_test")
    AI_test.displayKnowledge()
elif menuOption == "2":
    print("Enter a board state in the format 000000000, use x, o or 0")
    inputState = input("Enter a state\n")
    AI_test.getKnowledge(inputState)
elif menuOption == "3":
    print("Program exiting...")
    time.sleep(1)
    break
else:
    print ("Menu error")
```

Figure 6.2.1.10: the other menu options

The other menu options as well as the default menu option. Users can either elect to see all knowledge or see knowledge for a board state.

6.2.2 AI Implementation

AI_Control:

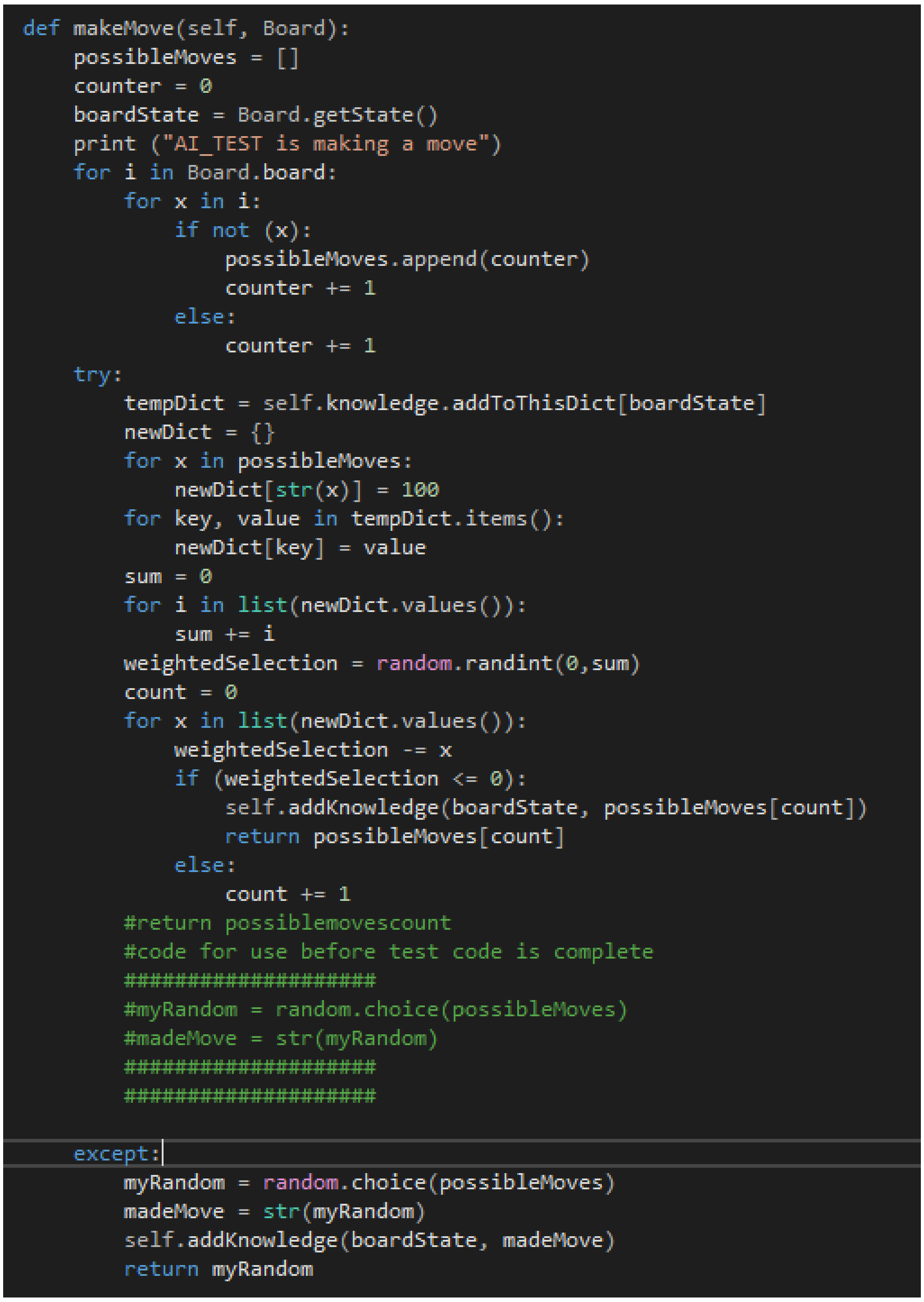


```
class AI_CONTROL(object):
    """
    This is a control AI program that randomly places moves
    """
    def __init__(self):
        print("")

    def makeMove(self, Board):
        possibleMoves = []
        counter = 0
        print ("AI_CONTROL is making a move")
        number = random.randint(0, 8)
        for i in Board.board:
            for x in i:
                if not (x):
                    possibleMoves.append(counter)
                    counter += 1
                else:
                    counter += 1
        return random.choice(possibleMoves)
```

Figure 6.2.2.1: Control AI class

This AI class has no knowledge implementation, and is intended to be the control AI that opposes the learning AI. It only picks random valid moves. It does this by iterating through the board for empty cells and saving the available cells in the possibleMoves array. Random.choice returns a random value from the possibleMoves array, which returns a random move which is then passed to the put method of board back in the main system.

AI_Test:



```
def makeMove(self, Board):
    possibleMoves = []
    counter = 0
    boardState = Board.getState()
    print ("AI_TEST is making a move")
    for i in Board.board:
        for x in i:
            if not (x):
                possibleMoves.append(counter)
                counter += 1
            else:
                counter += 1
    try:
        tempDict = self.knowledge.addToThisDict[boardState]
        newDict = {}
        for x in possibleMoves:
            newDict[str(x)] = 100
        for key, value in tempDict.items():
            newDict[key] = value
        sum = 0
        for i in list(newDict.values()):
            sum += i
        weightedSelection = random.randint(0,sum)
        count = 0
        for x in list(newDict.values()):
            weightedSelection -= x
            if (weightedSelection <= 0):
                self.addKnowledge(boardState, possibleMoves[count])
                return possibleMoves[count]
            else:
                count += 1
        #return possiblemovescount
        #code for use before test code is complete
        ####################
        #myRandom = random.choice(possibleMoves)
        #madeMove = str(myRandom)
        ####################
        ####################

    except:
        myRandom = random.choice(possibleMoves)
        madeMove = str(myRandom)
        self.addKnowledge(boardState, madeMove)
        return myRandom
```

Figure 6.2.2.2: Test AI class part 1

This is the AI that improves its own gameplay It also iterates through the board to save the available cells and acquire which moves are possible to be made. However, instead of picking a random move, it consults its knowledge for the moves that can be made for this board state, and feeds them into a weighted selection algorithm that picks the moves with higher weighting more often.

It does this by first checking if knowledge of a board state exists. Using a try-except code block allows us to proceed with the line;

tempDict = self.knowledge.addToThisDict[boardstate]

This line attempts to create a new dictionary that's a copy of the dictionary defined by addToThisDict[boardState]. This dictionary is the dictionary of following moves for the board state and its related weights. If there is no dictionary found, the code goes to the except block, which then chooses a random move from the possible moves in the same fashion as the control AI. If a dictionary is found, then we need to load the information from this dictionary into a new temporary dictionary. We do this by first creating a new dictionary, (newDict), and then for each possible move that can be made, we add it to the dictionary with the default value of 100. Then, any moves that are already known in tempDict, overwrite the default weights of the newDict moves with their weights. For example, for possible moves 3,5,6, tempDict would be a dictionary with the form;

{3, 100,
5, 100,
6,100}

If the knowledge dictionary were to contain knowledge of the following moves 3 and 5, then the weights of 3 and 5 would overwrite the default weights of 100, and the dictionary would then look something like..

{3, 130,
5, 240,
6,100}

Once the knowledge has been loaded from the knowledge base of the AI, the weighted selection algorithm commences. It works by taking a sum of the weights of the possible moves, then chooses a random integer from 0 to the sum. The weights of the dictionary are then iterated through and subtracted from the random integer. The value that causes the random integer to drop to 0 or below has its corresponding move chosen as the output for the method. This method gives a proportionally higher chance of higher weighted moves being picked.

The Test AI also contains the makeMove function of the control AI, however renamed to makeMoveRandom, to denote its random nature as opposed to the guided selection of the previous makeMove method. This was used when the guided selection wasn't available yet, but the program still needed to run to troubleshoot other areas.

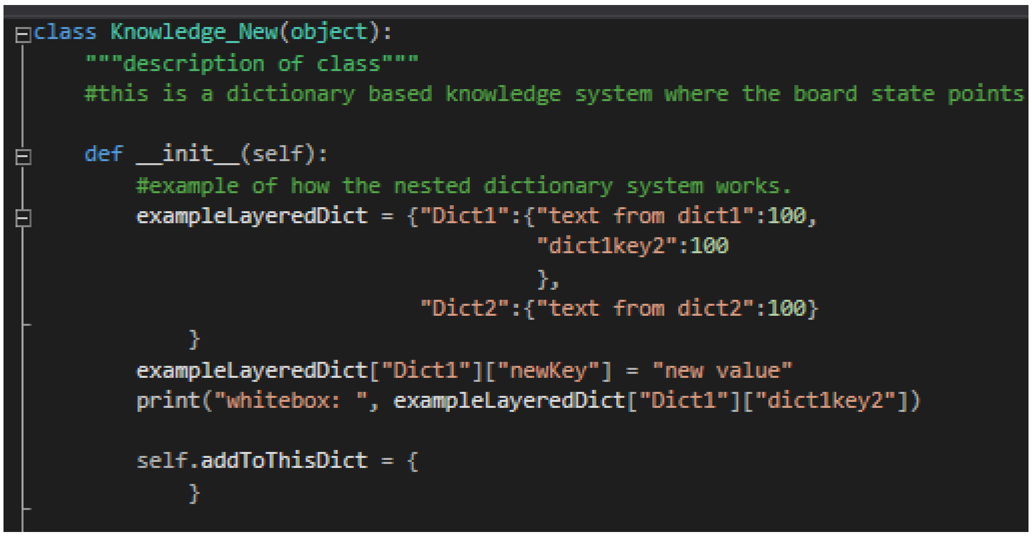
6.2.3 Knowledge Implementation

Knowledge:

The preliminary implementation of the knowledge system, this class is not used in the final versions of the program. It contains no code relevant to the new version of the knowledge system.

Knowledge_new:

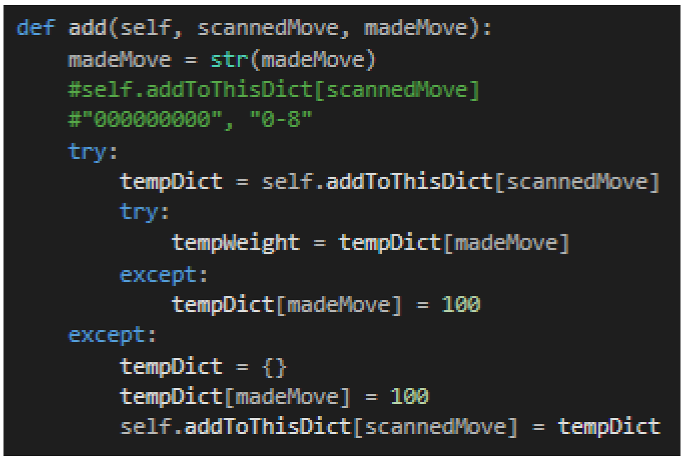
The final version of the knowledge system, it maps boardstates to possible following moves and assigns a weight to the moves based on often they win or lose.



```
class Knowledge_New(object):
    """description of class"""
    #this is a dictionary based knowledge system where the board state points

    def __init__(self):
        #example of how the nested dictionary system works.
        exampleLayeredDict = {"Dict1":{"text from dict1":100,
                                      "dict1key2":100
                                      },
                             "Dict2":{"text from dict2":100}
            }
        exampleLayeredDict["Dict1"]["newKey"] = "new value"
        print("whitebox: ", exampleLayeredDict["Dict1"]["dict1key2"])

        self.addToThisDict = {
            }
```

Figure 6.2.3.1: The constructor of Knowledge_New. You can see the example dictionary that shows the structure of how the knowledge dictionary would look once populated.

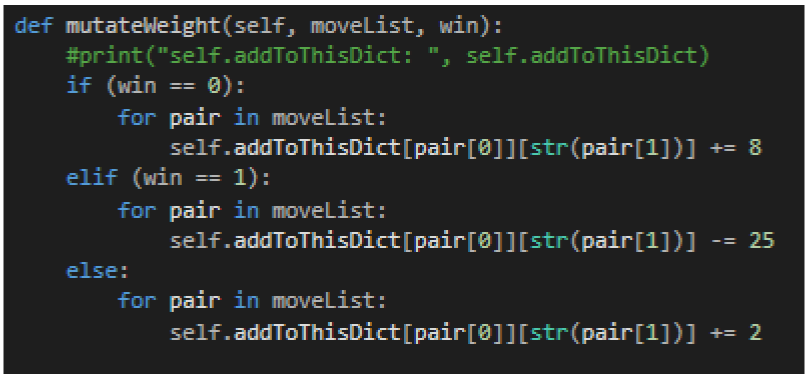
The knowledge in this system is stored in the dictionary addToThisDict. This dictionary holds data in the form of keys named after boardstates, with a dictionary of the possible following moves and their weights as the values for those keys.



```
def add(self, scannedMove, madeMove):
    madeMove = str(madeMove)
    #self.addToThisDict[scannedMove]
    #"000000000", "0-8"
    try:
        tempDict = self.addToThisDict[scannedMove]
        try:
            tempWeight = tempDict[madeMove]
        except:
            tempDict[madeMove] = 100
    except:
        tempDict = {}
        tempDict[madeMove] = 100
        self.addToThisDict[scannedMove] = tempDict
```

Figure 6.2.3.2: The add method of Knowledge_New.

This method is responsible for the addition of new knowledge to the knowledge dictionary. It takes a board state and made move as arguments and attempts to search for the board state within the knowledge dictionary, If the board state is found, then it tries to search for the made move within the sub dictionary, if it finds the made move then it does nothing, however if it does not find the made move, it creates it in the sub dictionary with a default weighting of 100. Similarly, if the board state is not found, then it creates a new temporary dictionary and adds the made move to it with the default weighting of 100. It then creates a new entry in the knowledge dictionary, using the board state as the key and the temporary dictionary as the value.



```
def mutateWeight(self, moveList, win):
    #print("self.addToThisDict: ", self.addToThisDict)
    if (win == 0):
        for pair in moveList:
            self.addToThisDict[pair[0]][str(pair[1])] += 8
    elif (win == 1):
        for pair in moveList:
            self.addToThisDict[pair[0]][str(pair[1])] -= 25
    else:
        for pair in moveList:
            self.addToThisDict[pair[0]][str(pair[1])] += 2
```

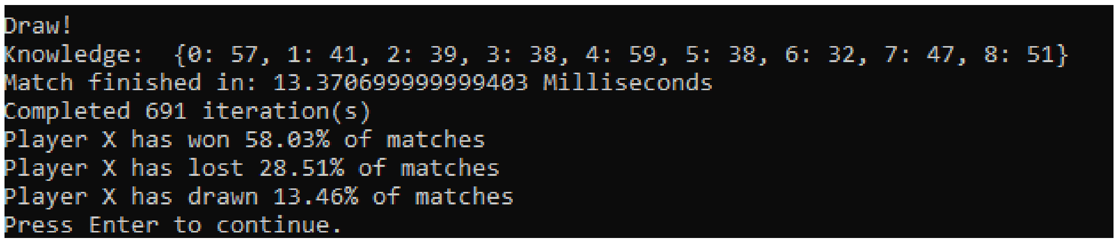
Figure 6.2.3.3: Weight mutation method of Knowledge_New. Note the commented out print statement that was used for testing this method.

This method takes a two dimensional array and an integer win condition as argument. The two dimensional array contains pairs of board states and made moves that comprise the game that was played. If the win condition is 0, a win, then we iterate through the array for all the board states and made moves and increase the weighting of them by 8.

If the win condition is1, a loss, we do the same but subtract 25.

If the win condition is 2, a draw, we do the same but add 2.

The numbers are always subject to change as they drastically affect the performance of the AI's learning algorithm. Negative 25 for a loss is a much steeper penalty than the + 5 win -10 loss that was used in the first iterations of this method. It was changed to +8 - 25 in order to speed up the learning process of the AI so that the effectiveness of the learning algorithm could be more quickly judged. Originally, I had assigned +5 and - 10 as the numbers for win/loss as when the program played completely randomly, player 1 tended to have around 2x the chance of winning than losing, as can be seen below.



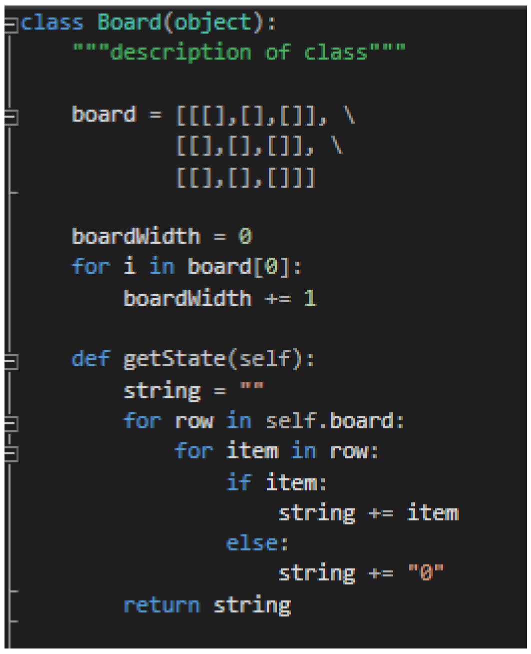
```
Draw!
Knowledge:  {0: 57, 1: 41, 2: 39, 3: 38, 4: 59, 5: 38, 6: 32, 7: 47, 8: 51}
Match finished in: 13.370699999999403 Milliseconds
Completed 691 iteration(s)
Player X has won 58.03% of matches
Player X has lost 28.51% of matches
Player X has drawn 13.46% of matches
Press Enter to continue.
```

Figure 6.2.3.4: Statistics of random play.

Since there is roughly 2x the chance of winning than losing, having losses cost double the weighting than wins gain makes sense. Then, if the AI learning algorithm were to not work and it were to continue making random moves, the weights would stay around 100 for each move. Once it starts learning however, it starts to win more than the baseline amount of twice more than it loses, and thus the winning moves gain weight slowly. This was however a very slow process, and after 2000 iterations the winrate of the AI would stand at around 65%. I have noticed however that there is always a tradeoff between speed of learning and accuracy of learning with the AI. When the numbers are changed to +8 - 25, like in this version for example, if the AI were to make a theoretically good move, but lose on the following moves, then the weighting of the good move would be penalised as well, and thus the good move would lose representation in the next iterations of games. However small increments in weighting such as +5 and - 10 make for an extremely long, but more accurate learning process.

6.2.4 Board Implementation

The board class implements the board that the AI players place moves upon. The board is responsible for outputting the boardstate, validating wins and placing the AI moves upon itself.

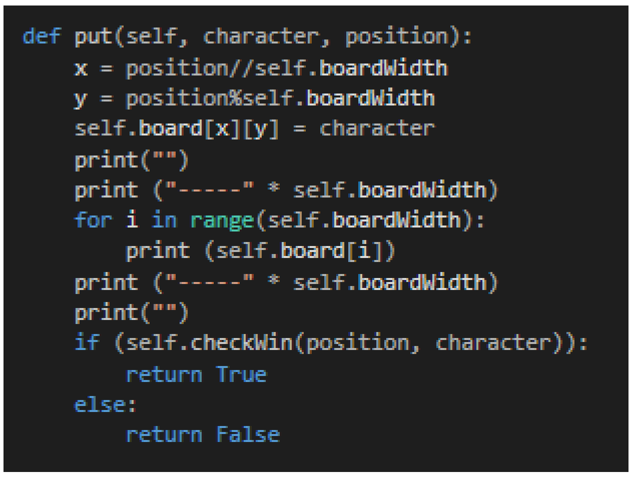


```python
class Board(object):
    """description of class"""

    board = [[[],[],[]], \
            [[],[],[]], \
            [[],[],[]]]

    boardWidth = 0
    for i in board[0]:
        boardWidth += 1

    def getState(self):
        string = ""
        for row in self.board:
            for item in row:
                if item:
                    string += item
                else:
                    string += "0"
        return string
```

Figure 6.2.4.1: The board class and its getState method.

The board here is a 3x3 two dimensional array that represents a typical noughts and crosses board. The boardWidth variable stores the boardWidth for instances where the board needs to be changed, it can be used to align objects with the board or else.

The getState method takes only no external arguments and outputs a 9 character string representing the current state of the board. It does this by iterating the board for arrays and iterating through each array for its elements. It adds the elements to the string or adds 0 to denote an empty cell. This string is returned after the iteration is complete.

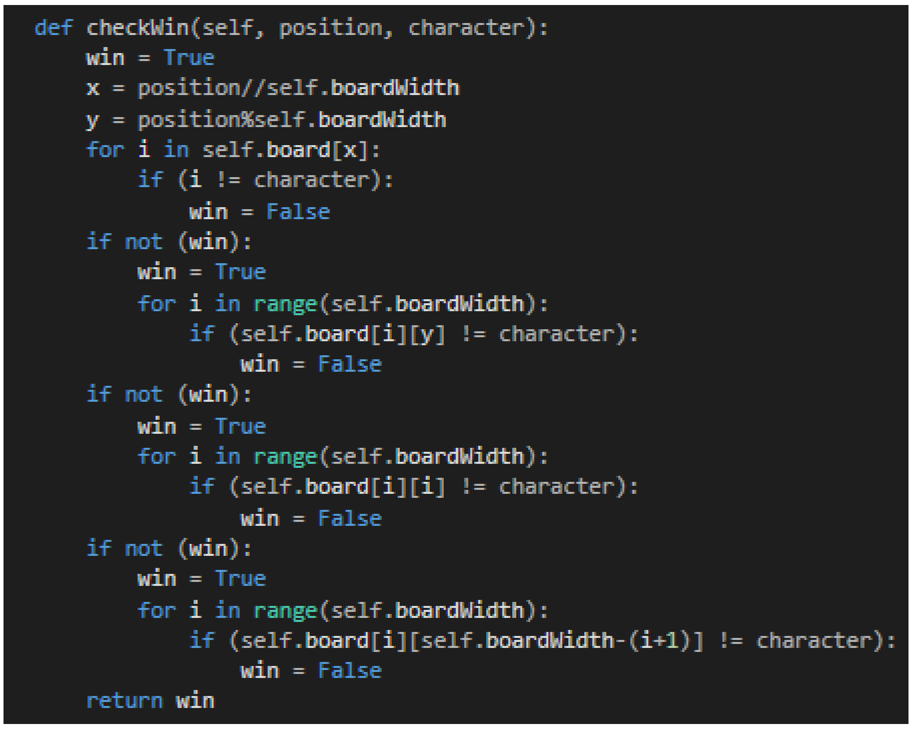


```python
def put(self, character, position):
    x = position//self.boardWidth
    y = position%self.boardWidth
    self.board[x][y] = character
    print("")
    print ("-----" * self.boardWidth)
    for i in range(self.boardWidth):
        print (self.board[i])
    print ("-----" * self.boardWidth)
    print("")
    if (self.checkWin(position, character)):
        return True
    else:
        return False
```

Figure 6.2.4.2: put method of Board. This method also prints the board onto the console.

The put method, aptly named, puts the made move of the user onto the board onto the position defined by the position argument. Something of interest note is that the character to put onto the board is also passed to the method as an argument, so other board games with different characters on the board would be supported without having to change this method. The method takes the position integer and divides it into the boardWidth to find which row of arrays the character will be put into. It then takes the modulus of the boardWidth to find out what index of that particular array the character will be put in.

The method then prints the board out with the character added and proceeds to check the board state if any win conditions are met by calling checkWin on itself. checkWin returns true or false, denoting whether this move caused the game to end or not.



```python
def checkWin(self, position, character):
    win = True
    x = position//self.boardWidth
    y = position%self.boardWidth
    for i in self.board[x]:
        if (i != character):
            win = False
    if not (win):
        win = True
        for i in range(self.boardWidth):
            if (self.board[i][y] != character):
                win = False
    if not (win):
        win = True
        for i in range(self.boardWidth):
            if (self.board[i][i] != character):
                win = False
    if not (win):
        win = True
        for i in range(self.boardWidth):
            if (self.board[i][self.boardWidth-(i+1)] != character):
                win = False
    return win
```

Figure 6.2.4.3. checkWin method of board.

This is the method that flags the end of the game when a game winning move was played. In conjunction with the main system, it is used to identify wins/losses and draws for the Test AI player. If this method returns true on a player’s turn, then that player won the game. If the method continues to return false until the round counter increments past 9, then the game must end in a draw.

Just like the put method, it determines the exact position of the with a division and modulus. This checkWin method starts by assuming a win condition is true. The made move is then evaluated against the current state of the board to check if it causes a win condition to be met.

For example, a move made in the position 4, (the centre of the board) then has the other positions in the same row, (same array) evaluated. If not all of them contain the same character and the move’s character, then win is false for this condition.

This continues for the other conditions. First win is set to true, then the condition is evaluated and if it isn't met, win is set to false. For the y axis win condition, the character is evaluated against the elements of the arrays with the same index. The diagonal condition is checked by checking the elements at the positions defined by the x coordinate -1 and y coordinate -1, and the x coordinate + 1 and y coordinate +1. The other diagonal checks x coordinate -1 and y coordinate +1, and the x coordinate + 1 and y coordinate - 1 instead.

Chapter 7. Testing

7.1 Functional testing

(Reqtest. 2019)

Black box testing tests the usability of the program. It differs from white box testing as the tester does not know the inner workings of the program. It is especially useful as it tests only what the user can do to the system and thus shows what the user can expect from the program.

For my program I conducted manual black box testing of the program with myself and one outside tester.

Test	Expected outcome	Actual outcome	Success?
User enters 0 in main menu	AI Loop begins to run	AI loop confirmation message appears	Yes
User enters 1 in main menu	Knowledge base displayed	knowledge of moves displayed in format: Board state: 00000000 1: 100 2: 100 etc	Yes
User enters 2 in main menu	Knowledge base of one boardstate method called	prompts user to enter board state	Yes
User enters 3 in main menu	program exits	Program displays “Program exiting…” then exits.	Yes
User presses Enter upon AI loop prompt	AI Loop begins	AI loop began	Yes
User enters	program knows	Program displays moves along with their	Yes

000000000 into boardstate prompt	move knowledge for this state	weights in the format 3: -7 8: -5 5: -1 etc	
User enters 4 at main menu	main show tells user its an error	System displays “menu error” and asks for input again.	Yes
User presses CTRL + C during AI loop	loop stops	loop stops, so suddenly it stops it ends the game currently ongoing	Yes
user enters 0 for board state	system produces error message	System writes “there was an error retrieving knowledge” and returns to the main menu	Yes

These tests were successful, there was no user input that could break the system or cause it to crash.

White box testing is like Black box testing, but the tester knows the inner workings of the program and uses this knowledge to attempt to break the program. In this way it contrasts with black box testing, as black box testing is user centric testing that attempts to understand how the user will communicate with the system, white box testing is instead developer centric testing that focuses on determining valid and invalid inputs. There is an example of white box testing that I performed earlier in this report, in chapter 4.1. As the developer of this program, I designed it mostly with the philosophy that all inputs are invalid except for a chosen few. This is reflected in my implementation of the menu for example, where all inputs are invalid except for the integers 0, 1, 2 and 3, anything except causes the else: block to run, which has no functionality except to restart the menu input.

7.2 User testing

I discussed the implementation of my program with my mentor, and used his feedback in order to develop the later versions of the program. The early versions of the program used a version of knowledge that saved games as strings of characters and positions in a two dimensional array. This caused me issues while developing the knowledge past remembering only the first move. My mentor explained that this system could be simplified into a string based system instead, which was much easier to visualise and work with. I took this advice onboard and redesigned the program around this new knowledge system, as well as being more modular.

I also distributed the program during development to my peers and obtained their feedback. They were asked for feedback in the following areas, How well information is conveyed to the user? Was the information enough to use the program? Did the program help you to play noughts and crosses? What features did you like?

Although I received mixed responses, feedback from earlier test versions of the program tended towards there being not enough information available for the user to know how to use the program, and an over exposure to unnecessary information. These older versions of the program tended to show more of the backend variables to the user than was necessary, because these were variables in areas of the program that were currently under development.

In later versions of the program, I removed all unnecessary output from the program to the console. Versions 0.1.1 onwards have a minimum of what the user needs to see to operate the program only. Version 0.0.13 is programatically the same as Version 0.1.1, but has more outputs to allow for easier debugging. Versions 0.1.1 onwards also have more clarification on the usage of the program. Each stage of the program has instructions on what input the user enters and in what format it should be entered in. Invalid inputs are caught by the system and tell the user of the error so that they can do it correctly the next time.

I did this to increase the usability of my program, which was the main concern my users had. Later versions of the program showed an improvement in this aspect, but it did tend to stay as the major concern with the program, even as it improved. This could partially be because the program is designed to help people, and as such if it creates problems using it, it isn’t helping anyone at all.

Chapter 8. Conclusions and reflections

In conclusion, I am satisfied with the functionality of the program. I consider it truly special to see something I created get better at a game, even one as simplistic as noughts and crosses. This project makes me gaze into the future wondering what sorts of possibilities lie ahead in the field of machine learning. I do think the program is not marketable to a wider audience however, and should really have a fully implemented GUI and mobile device port in order to make it an convenient to use as possible, but these are issues that are unrelated to the actual functionality of the program, which remains as something I am proud of.

I am proud of my Tic Tac AI, and will continue to develop it in the future.

Now, the strengths of this program lie in its potential for the future and its semi-modular design. The self learning algorithm can be applied to any situation where a state and the following move can be represented, as it is essentially just a system for mapping a situation to the most effective move available and thus can be rebuilt for any situation given some time. A downside to this however is that the system is not FULLY modular, and requires development time to build versions of the program suitable for learning in different environments. I estimate that full modularisation of the program would be quite difficult and is likely not possible for this program as it currently works. Full modularisation would require acceptance of any situation, and thus a way to represent any kind of situation in a string of text. The system would also require knowledge of inputs it could perform and how to judge the success of the AI in the particular environment in order to suitably affect the weighting system.

I consider this project to be a good base for what could develop into a more advanced AI, but has limitations in its implementation that lead to limits in its potential. The self learning algorithm does have potential to work in many environments still however, but would probably require a new framework to be implemented in.

Lack of a UI also does diminish the usefulness of the program. The console window may intimidate users that are inexperienced with computers and lacks support or anything such as images, video or sound. The original problem is also not entirely addressed. The objective was to develop an AI that could provide a way for the user to learn alongside it. This implementation lacks any way for the user to play vs the AI, or against another player. It can however be used as a reference for a game ongoing, or to establish trends in winning moves quite easily, thus it became more of a tool to help learning, rather than a whole framework to facilitate learning, which I consider the next stage of development for this problem, with further modularisation of the program being the stage beyond that.

The program also only runs on a single thread. Considering it lacks graphics and is solely a CPU based application, this limits how fast it can run on modern CPUs that often have more than 4 CPU cores.

It also lacks a port to any other platforms other than windows desktop. This would be an ideal program to have on a mobile device in order to practise a game when commuting or otherwise between jobs or other responsibilities.

This project taught me a lot about reinforcement learning algorithms and other ways of implementing Machine Learning. I watched and read a lot of content that I didn't use in the implementation of the program in the end, but was interesting and served as inspiration in some areas of my code. I found development of the nested dictionary knowledge management system and reinforcement learning algorithm difficult and a challenge to overcome, but I have now learnt how I could create and apply a reinforcement learning algorithm to an environment, and create and use nested dictionaries.

It also taught me a lot about proper research and development techniques, in particular development methodologies. I had always followed a similar method for developing programs but never read anything about standardising it into methodologies and specific programming and development practises. I learnt how to prepare to code in a way that I enjoyed, and allowed me to best use my talents to swiftly and effectively create and prototype versions of the program from an early stage. Moreover, the act of writing this very report helped me gain time management skills and improved my communication skills.

I think the next step for a program like this would be evolution. There is only so much this code can achieve, and usage of popular frameworks such as TensorFlow could improve the effectiveness of this program beyond the scope of what was originally intended. I believe the next step for this program and myself as a developer is research into Neural Networks and more advanced Reinforcement learning techniques. I could use these advanced techniques to create a more sophisticated self learning AI capable of operating more effectively in more chaotic environments, and learning at a faster speed.

9. References

https://www.osborneclarke.com/insights/artificial-intelligence-the-real-legal-issues-an-article-by-john-c-buyers-osborne-clarke-llp/

Sethbling. (2015). MarI/O, Machine Learning For video games. Last accessed 20/04/2020. Viewable at: https://www.youtube.com/watch?v=qv6UVOQ0F44

Justesen, Bontrage, Togeliis, Risil. (2019) Deep Learning for Video Game playing. New York University, IT University of Copenhagen. Last accessed 21/04/2020, Viewable at https://arxiv.org/pdf/1708.07902.pdf

Iitsuka, S. (2017). Training tic tac toe expert with temporal difference learning. Tushuhei Blog. Last accessed 21/04/2020. Viewable at http://blog.tushuhei.com/posts/training-tic-tac-toe-expert-with-temporal-difference-learning/

OpenAI Blog. (2018) OpenAI Five. OpenAI. Last accessed 23/04/2020. Viewable at https://openai.com/blog/openai-five/

scmi. (2013) How do you create nested dictionary in python? Stack Overflow. Last accessed 03/05/2020. Viewable at https://stackoverflow.com/questions/16333296/how-do-you-create-nested-dict-in-python

Sas insights. (2020) Evolution of Machine Learning. SAS Institute. Last accessed 04/05/2020. Vieawable at https://www.sas.com/en_gb/insights/analytics/machine-learning.html

Callum Overy. Python programming 3.4.2, Noughts and crosses. Best way to check how to see if game is won? (2015) Stack Overflow. Last accessed 04/05/2020. Viewable at https://stackoverflow.com/questions/33178684/python-programming-3-4-2-naughts-and-crosses-best-way-to-check-how-to-see-if-g

Reqtest. 2019. Black box testing. Reqtest. Last accessed 14/5/2020. Viewable at https://reqtest.com/testing-blog/black-box-testing/

10. Bibliography

https://www.osborneclarke.com/insights/artificial-intelligence-the-real-legal-issues-an-article-by-john-c-buyers-osborne-clarke-llp/

Sethbling. (2015). MarI/O, Machine Learning For video games. Last accessed 20/04/2020. Viewable at: https://www.youtube.com/watch?v=qv6UVOQ0F44

Justesen, Bontrage, Togeliis, Risil. (2019) Deep Learning for Video Game playing. New York University, IT University of Copenhagen. Last accessed 21/04/2020, Viewable at https://arxiv.org/pdf/1708.07902.pdf

Iitsuka, S. (2017). Training tic tac toe expert with temporal difference learning. Tushuhei Blog. Last accessed 21/04/2020. Viewable at http://blog.tushuhei.com/posts/training-tic-tac-toe-expert-with-temporal-difference-learning/

OpenAI Blog. (2018) OpenAI Five. OpenAI. Last accessed 23/04/2020. Viewable at https://openai.com/blog/openai-five/

scmi. (2013) How do you create nested dictionary in python? Stack Overflow. Last accessed 03/05/2020. Viewable at https://stackoverflow.com/questions/16333296/how-do-you-create-nested-dict-in-python

Sas insights. (2020) Evolution of Machine Learning. SAS Institute. Last accessed 04/05/2020. Vieawable at https://www.sas.com/en_gb/insights/analytics/machine-learning.html

Callum Overy. Python programming 3.4.2, Noughts and crosses. Best way to check how to see if game is won? (2015) Stack Overflow. Last accessed 04/05/2020. Viewable at https://stackoverflow.com/questions/33178684/python-programming-3-4-2-naughts-and-crosses-best-way-to-check-how-to-see-if-g

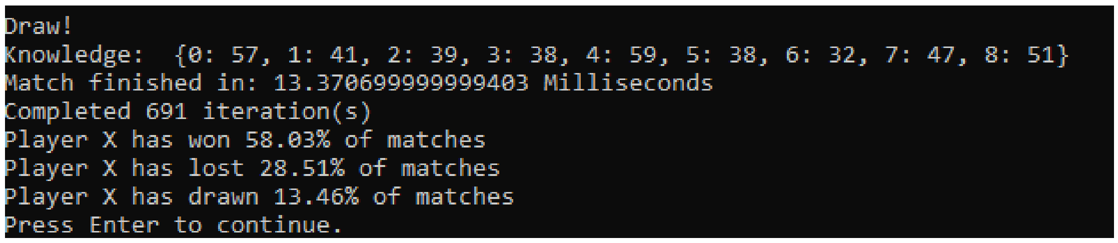
Appendix I

Provide additional material, if appropriate, in separate appendices.

Detailed overview of the MarI/O project:

https://www.youtube.com/watch?v=qv6UVOQ0F44

Appendix II



```
Draw!
Knowledge:  {0: 57, 1: 41, 2: 39, 3: 38, 4: 59, 5: 38, 6: 32, 7: 47, 8: 51}
Match finished in: 13.370699999999403 Milliseconds
Completed 691 iteration(s)
Player X has won 58.03% of matches
Player X has lost 28.51% of matches
Player X has drawn 13.46% of matches
Press Enter to continue.
```

Random AI win statistics

Appendix III

Link to Video Demonstration of Program:

https://www.youtube.com/watch?v=SGQaExPb11w&feature=youtu.be

Chapter 8a. Coming back to this project a year later.

I chose this project to upload to github as an example of a project I produced in University. In hindsight now, I realise that a project in which I was not familiar with the methods and techniques involved would not make for a good example of professional development. However the project will continue to remain on GitHub as an example of a project in which I had no precursor knowledge.

Still, I have learnt much over the course of the covid months and will endeavour to develop new projects that better showcase the level of professionalism I can display in my development.

The code was slightly modified and low level documentation added. I do not believe that this project can be significantly improved without full redesign however, coming in a future project.