UNIVERSITY OF LIVERPOOL

COMP39X

2019/20

Creating a Framework to Teach Key Stage 3 Students Reinforcement Learning with Autonomous (Simulated) Robots

DEPARTMENT OF COMPUTER SCIENCE

University of Liverpool
Liverpool L69 3BX

COMP39X

2019/20

Creating a Framework to Teach Key Stage 3 Students Reinforcement Learning with Autonomous (Simulated) Robots

Student Name:	Brandon Skerritt
Student ID:	201270424
Supervisor Name:	Dr. Dennis

DEPARTMENT OF
COMPUTER SCIENCE

University of Liverpool
Liverpool L69 3BX

Dedication

Dedicated to Megan.

Acknowledgements

Thank you to the cyber security community, for continually supporting me in all that I do.

Abstract

Reinforcement learning has traditionally been a dull topic to learn, especially for students who may not understand programming. With this dissertation, it is shown that reinforcement learning can be joyous to learn, and easier than some might have suspected.

This dissertation shows a framework for teaching reinforcement learning to key stage 3 school students. Using Python to implement reinforcement learning, and using lessons based around the code.

The aim was that by teaching them a "hard" and "cool" computer science topics, the students would become engaged in the world of computer science.

This project implements a Q-learning reinforcement agent in a grid-world 2-dimensional maze to teach the students reinforcement learning. It also has lessons and exercises for the students, that tie in with the Q-learning agent to reinforce what the students can learn.

Contents

List of Figures

1 Introduction

If students or teachers find a subject difficult, the chance that that subject is picked (for GCSE or A Levels) goes down substantially [3]. Difficulty can come in many forms, whether it is:

- Bad teaching.
- Under specialisation of teachers.

However, the question will remain. How do we make reinforcement learning interesting? According to Zahorik [14] some methods include:

- Novelty.
- Emotional Appeal.
- Group Work.
- Computers.
- "Lived experience" (practical knowledge).

2 Aims and Objectives

This project created an engaging set of exercises involving (simulated) robots and reinforcement learning for key stage 3 students to learn from.

The objectives are:

- Create an ASCII based grid-world reinforcement learning playground.
- Implement a reinforcement learning framework in an existing Python based robot simulator.
- Design an API for reinforcement learning.
- Devise a set of challenge problems to be used with the simulator.
- Design the learning exercises.
- Gather feedback about the learning exercises and improve upon it.
- Create an autonomously driving robot vehicle agent.

3 Key Literature and Background Reading

3.1 Reinforcement Learning: An Introduction

This book by Sutton & Barto [**sutton**] was one of the most popular introductions to reinforcement learning. The book starts simple. Introduction, why reinforcement learning is needed, and early history of it.

The book then introduced us to reinforcement learning using the K-armed bandit. This is not a challenge in optimising the algorithm, it's simply a problem to introduce us to the terminology of reinforcement learning and how we can conclude that this problem can be solved with reinforcement learning.

The book then took us through some paradigms we may know that can be considered reinforcement learning paradigms. Such as:

- Dynamic Programming.
- Finite Markov Decision Processes.
- Monte Carlo Methods.

Before taking us into tabular (Q-Learning) based reinforcement learning.

The book ends with the psychology of reinforcement learning. We know the theory, and we know how to implement it, but how does this work with humans? According to Sutton & Barto [12]

> "Reinforcement learning is the closest to the kind of learning that humans".

If reinforcement learning was closet to that of humans, Simon & Barto shows us how humans learn using psychology and neuroscience.

Reinforcement learning is a type of learning that involves repeatedly (and often, failingly) doing things until the agent learns what moves return the highest (or lowest) rewards.

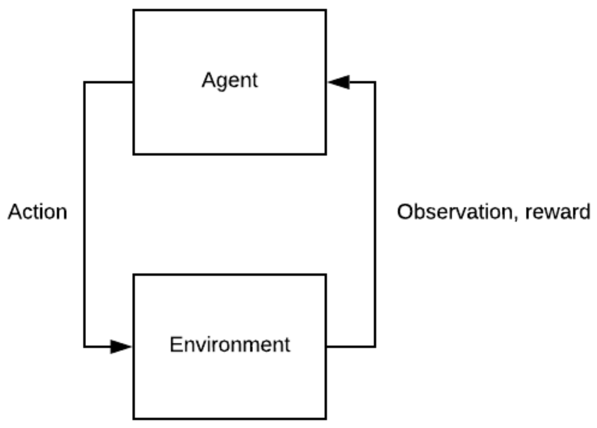


Figure 1: The Agent Reward System.

The agent takes an action. This action has some effect on the environment. If the agent can only see 30cm in front of it and it moves 5cm forward, the environment doesn't change but the agent's perception of the environment changes.

The agent might destroy, create, or move things in the environment.

This action returns an observation and a reward. The observation is what has happened to the agent and the environment. For example, if the agent moved 5cm forward, the agent can now see 5cm more than it previously could.

The reward is either a positive reward or a negative reward. As an example, the agent's goal might be to search for rocks. If the agent sees a rock within 30cm of it now, the agent is rewarded.

The agent would then store this information into a Q-table. The Q-table answers the question: "When the environment is exactly this, and there are these moves available, this is the reward you will get".

However, there are 2 types of rewards. Seen above is the positive reward. The agent is positively reinforced by something. But a negative reward is also possible.

As an example of a negative reward, the agent could be in an environment with one straight line of light. The agent's purpose is to avoid the light. If the agent moves into the light, the agent gets a negative reward.

It is unusual for a reward to be both positive and negative. It is either -1 and 0 or 0 and 1.

The agent uses this Q-learning table to make further decisions. However, we stumble upon the exploration vs. exploitation problem.

Exploitation here means that the agent exploits the Q-learning table to maximise its reward.

Let's say our agent learns to avoid the light by staying on the edge between light and darkness. Our agent realises if it goes into the light, it gains a negative reward. But if it stays between the two, it doesn't gain a negative reward - but it also doesn't gain anything positive.

The agent, since it exploits the Q-learning table, never learns that it can stay out of the light completely and get a much higher score than where it is.

Exploration, regardless of how much reward an action has, is important in this regard as it informs the learning of the agent - it maximises the agents learning.

Although too much exploration and exploitation will no longer offset the value of exploration.

This can be solved by having the reward as a Boolean data type.

3.2 Deep Learning

Deep Learning by Goodfellow et all [6] is an introductory book to deep learning. While Reinforcement Learning by Sutton & Barto is an introduction to reinforcement learning, it assumes prerequisite knowledge of deep learning. More specifically, the mathematics involved in deep learning.

Goodfellow et all start the book off with an introduction to applied mathematics.

- Linear Algebra.
- Probability and Information Theory.

- Numerical Computation.
- Machine Learning Basics.

Once the reader is familiar with the mathematics behind deep learning, Goodfellow et all introduce neural networks. Starting slowly with feed-forward networks and then eventually ends with convolutional networks and recurrent neural networks.

The final chapter is on the research, the more academically rigorous side of this book.

The book proved to be useful in providing the creator knowledge of the mathematics behind artificial intelligence.

3.3 MIT Deep Learning

While this project centred around reinforcement learning, most fundamentally, it was about creating a framework to **teach** reinforcement learning.

For the creator to understand the most effective methods to teach reinforcement learning, they have chosen to study MIT's deep learning course [9].

The reason why MIT was chosen was that they publish every lecture online for free. They are also an Ivy Leauge college, which means they pride themselves on their world-changing research.

Some of the more important things the author learnt whilst watching the lectures were:

3.3.1 Talk Slowly

All of the professors in the lecture videos talked slowly, but not too slowly. They talked clearly, loudly, so everyone can hear.

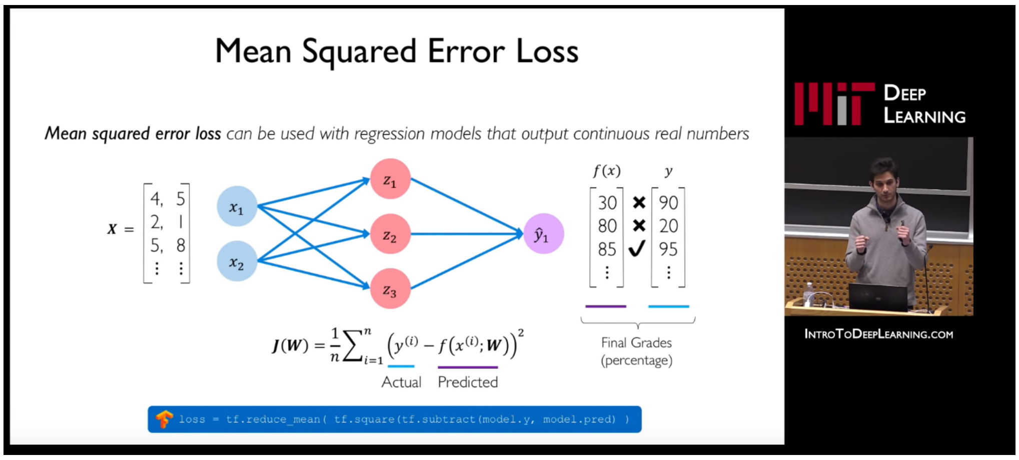
3.3.2 Different Forms of Media

The professors used video, whiteboards, quizzes, live-programming and more. Anything that isn't text on a plain background is appeared to be fair game for them to use.

One of the things the author wanted to involve in their teachings is talking over videos. The professor put on a video of a car learning to drive [5], and explains how it is learning. We were engaged in the video, and we learnt how it works.

3.3.3 Using Colour to Learn

All of the mathematical formulae used by MIT were colour coded.



Mean Squared Error Loss

Mean squared error loss can be used with regression models that output continuous real numbers

$$X = \begin{bmatrix} 4, & 5 \\ 2, & 1 \\ 5, & 8 \\ \vdots & \vdots \end{bmatrix}$$

$$\begin{array}{ccc} f(x) & & y \\ \begin{bmatrix} 30 \\ 80 \\ 85 \\ \vdots \end{bmatrix} & \begin{matrix} \times \\ \times \\ \checkmark \\ \end{matrix} & \begin{bmatrix} 90 \\ 20 \\ 95 \\ \vdots \end{bmatrix} \end{array}$$

Final Grades (percentage)

$$J(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n}\left(\underbrace{y^{(i)}}_{\text{Actual}} - \underbrace{f\left(x^{(i)};\boldsymbol{W}\right)}_{\text{Predicted}}\right)^2$$

```
loss = tf.reduce_mean( tf.square(tf.subtract(model.y, model.pred) )
```

Figure 2: An example of the colourful mathematical formulae used by MIT.

In this lecture by Professor Amini [1] the colours have meaning. The purple underline for the second half of the formula corresponds to the purple final grades matrix on the right.

The quotation under each part of the formula stating what that part does (”Actual”, ”Predicted”). The use of the blue box at the bottom of the screen showing what the code is for that formula.

3.4 COMP335

The creator took the module COMP335 - ”Communicating Computer Science”. [13]. This module aimed to enable the students to better teach computer science to key stage 3 students.

The creator specifically learnt these key points from the module:

3.4.1 Students Are Slow

The idea of learning rates, which may seem simple to a university student, is not simple for younger students.

Even the idea of a machine that is capable of learning like a human can be confusing to younger students.

Dr. Thomason has shown that even formulae, as simple as formulae may be, can still be confusing to a younger audience whose only exploration into formulae is Pythagoras’s Theorem.

3.4.2 Do Not Forget Things We Take for Granted

As a university student, the creator took for granted the idea that all schools will have Python installed. That they will have the resources to download packages, or install Python if is it not installed.

This is wasn’t the case. Some schools have restrictive downloading, some teachers may not even know how to install packages.

3.5 Discrete Mathematics with Applications

This book by Epp [**epp**] is typically a first-year university students introduction to discrete mathematics. The book served as a good pointer to not only explaining mathematics well but explaining how to teach well.

Like MIT [9], the book made heavy use of any form that isn’t plain text. Images, quotes, history tidbits, miniature tests, tables, and many, many more.

The book uses colour and syntax-highlighted code to get across how each formula works.

4 Development Process and Method

The Agile Programming Metholdogy was used, more specifically SCRUM [11]. SCRUM allowed the creator to design the end program with the user in mind, using the user stories feature. Since it is agile, the creator had a working prototype early on in the development process which can acted as an ultimate fall back. See the risks & contingencies chapter for more information on this.

The project has to be easy to use for students. It would be unwise to build software using waterfall, because at the end the stduents may not understand how the software works and the project will be a failure. Agile methodology allows us to work towards good code that is easily understandable.

This is because if the project had failed entirely, the Waterfall method would not have provided any working prototypes to use to teach.

The tools the creator used were:

4.1 GitHub

GitHub was used as a version control system, making backups of the code and making it possible to revert to a previous version.

The GitHub Project board was used as a Kanban board to monitor the progress of the project. With 4 sections. ToDo, Testing, Implemented, and finally Issues.

The creator also made heavy use of GitHub actions, the continuous integration part of GitHub to automatically run tests and verify working code.

4.2 Python

The code s written entirely in Python 3.8, making use of small libraries to allow for ease of use for a school.

4.3 NumPy

The project required NumPy, which is used to perform calculations that are required in basic reinforcement learning such as the Q learning algorithm and maintaining the Q learning table.

4.4 Local backups

Local backups of the code were regularly made on the creator's machine and network-attached storage device.

4.5 IDLE & Code Editor

Some schools were not able to use a code editor and instead used IDLE - Python's integrated development environment. The code was regularly tested against both an editor and IDLE.

The creator made a note that the biggest issue was that what worked on the development PC wouldn't always work on the school PCs. They may not have the required packages, or they may not even have Python installed at all.

The software was developed on a separate virtual machine made specifically for the development of this project. On this machine, libraries such as NumPy were automatically installed. Each new install was vetted with the creator.

If the worst came to the worst during the demonstration of this project, the creator would be able to export the virtual machine and install it on the computers in the classrooms. Dr. Thomason had said that this was done before with Kali Linux but is not the preferred way to do things. However, it was still possible.

5 Data Sources

No data sources were used for this project. However, potential data sources might be used by the students.

The maze program set out for the students can be altered, to such a degree that students may search online for mazes that will work with the project. They may download these alternative data sources and attempt to incorporate those into the program.

However, no data sources were used by the creator of the project.

6 Ethical Considerations

Here is a list of the ethical issues discussed between the creator and their supervisor.

6.1 Combining 2 Modules Into 1

The creator combined 2 modules into 1. COMP335 Communicating Computer Science with COMP39x, this project. There were some ethical issues raised due to using the same work for 2 modules.

After the creator discussed this with their supervisor, due to COMP335's deadline of "finish a lesson plan by week 12" this would mean that the project would have to have a working prototype by week 12.

Because of this limitation by combining 2 modules the creator did more work than if I was to do each module separately, which made it ethically & morally okay.

6.2 How to Evaluate the Effectiveness of This Project

The simplest way to evaluate the effectiveness of this project was by asking the students (who are children) what they think of it. However, this brings up a plethora of ethical issues with collecting data from children.

The easiest way to evaluate this project wasn't through the children or the teachers. During the teaching sessions for COMP335 Dr. Thomason will evaluate the teaching. By using this dissertation project in the teaching, Dr. Thomason will partly mark the project (the activity) as well as the teaching.

Throughout the year, the widening participation team at the department hosted events. Dr. Dennis, my supervisor, suggested it might be possible for the creator to bring the project to one of these.

However, due to COVID-19 it wasn't possible to teach, or attend any events. The evaluation chapter will go into more details.

7 Design

7.1 The Grid World

The Grid World was the environment in which most of the players experienced the exercises.

The grid world was a α x β world. The grid world had walls, made of the number 0. The idea behind this was that the number 0 is physically wide, much wider than the number 1. Therefore, wideness equated to a wall and the number 1 equated to a corridor.

```
0000000000000000
0111111111111110
0111111111111110
0000000000000000
```

The environment object created a new grid world (or use a previously generated one) for every run. An example of a simple grid world is:

```
0000000000000000
X111111111111110
01111111111111Y0
0000000000000000
```

Where X is where the agent starts and Y is where the agent ends.

The creator then represented this world in a 2d array.

If the objective was to follow a light, the grid world might have looked like this:

```
0000000000000000
X              0
----------------
0000000000000000
```

Where the line is the light. The agent will then "explore" the world. The agent might decide to move to the right (forwards).

```
0000000000000000
 X             0
----------------
0000000000000000
```

This did not return a reward, so the agent tries again. This time, it moves into the light.

```
0000000000000000
               0
  X-------------
0000000000000000
```

This returns a reward, so the agent knows it is doing the right thing.

Of course, this is a very simple problem with only 2 lanes to choose from. A real-life problem might simulate the disparity of light. The light might not be a perfect line, it might have signs of light all over the place. Such as this:

```
0000000000000000
   ------------0
  X=============
0000000000000000
```

Here we used a double line (equals symbols) to represent the strength of the light. The light in the second lane is weaker, so it only used 1 line.

With a grid world like this, and teaching this to key stage 3 pupils, it is important to wisely choose the symbolic meaning of each item in the grid world so the students can effectively learn and understand what is happening on their screen. Abstract symbols will only make for confused students.

The reasoning behind 0 is that it is physically wide, like a wall. Whereas 1 is physically thin, representing a corridor.

7.2 Components of the System

There were 2 separate modules to the system. Agent.py, and Maze.py. Agent.py controlled the agent and what the agent does, as well as the

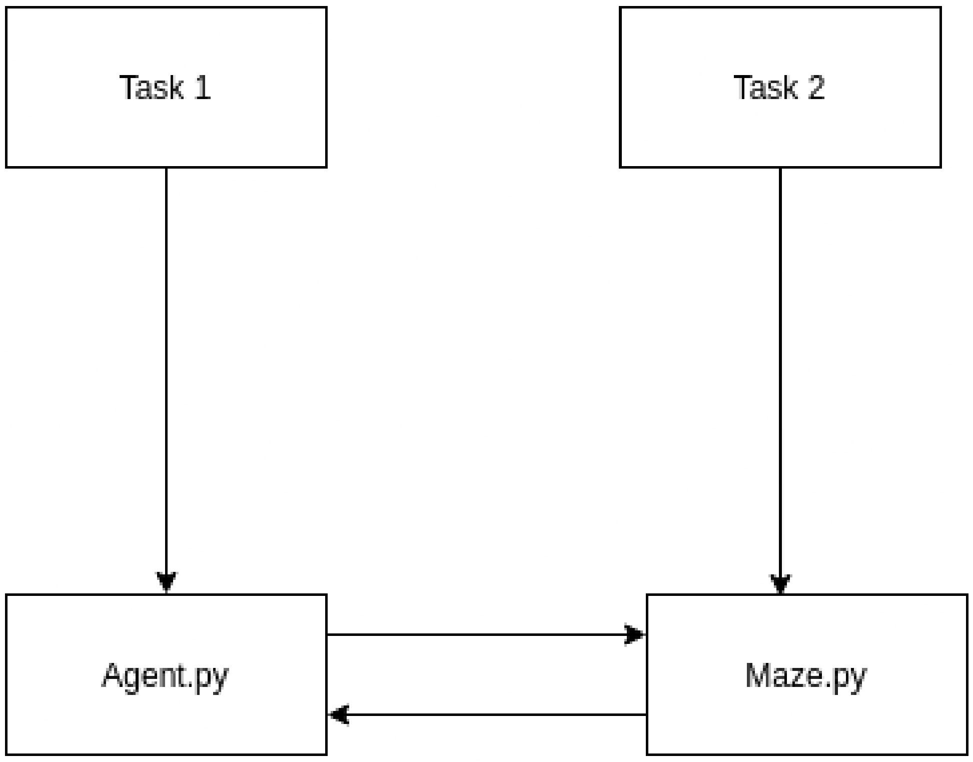


Figure 3: Organisation of the system

Q-table. Maze.py controlled the maze, and decided on what was a legal move for the agent to take.

Each task focussed on one part of the system, so the students effecitvely learnt how they integrate togeter. If the task called for a change to the rewards system, the student would explore Agent.py. If the task called for a change to the legal moveset or the maze itself, the students entered into Maze.py.

By seperating 2 distinct parts of the system this way, students saw how the 2 parts work with eachother.

7.3 Data Structures and Algorithms

7.3.1 Arrays

The most important datastructure used was multi-dimensional arrays, used for the Q-table.

To explain this datatructure, look at the below example.

```
{0: [0.5, 421, -1, False)],
 1: [(1.0, 228, -1, False)],
 2: [(1.0, 348, -1, False)],
 3: [(1.0, 328, -1, False)],
 4: [(1.0, 328, -10, False)],
 5: [(1.0, 328, -10, False)]}
```

Where the dictionary has the structure:

```
{action: [(probability, nextstate, reward, done)]}
```

This is an example of a rewards table in a multi-dimensional array.

7.3.2 Q-Learning

Q-learning is a reinforcement learning algorithm. The goal is to learn something which tells the agent what action to take under what circumstances. It does this using the reward table, as seen in the previous section.

It is the simplest of all reinforcement learning algorithms, as understood by [12] in their book. This is useful because a simpler algorithm will be exponentially simpler to teach to key stage 3 students.

7.4 User Interface

There is no direct user interface for this project. It is a code library. However, the design of the code library will matter.

All the important variables were represented near the top of the file, clearly marked for the students to understand. Each function had full documentation, as well as the thought process behind why that function exists.

However. the lessons themselves were designed intentionally well. Therefore this document details the design of the lessons, as they are what the user interfaced with when they used the project.

7.4.1 Exercise 1

The first exercise was designed to be the easiest. Given this code:

```
learning_rate = 0.5
```

The user would change the learning rate to observe what will happen to how the agent acts on the maze.

More specifically, the learning goal of this lesson was to learn what the learning rate was, and see for themselves how the learning rate affected how the agent behaved in the environment.

7.4.2 Exercises

Each exercise does not require knowledge of the previous lessons. This meant that the students could skip ahead without worrying about missing some vital information.

Each exercise contained a learning goal, to maximise the students learning. The exercises contained succinct knowledge, with clearly marked Python code and line number references to enable the students to easily find the exact variables and functions they needed to change or observe.

The agent runs with an epoch of 1, meaning it is easy for a student to observe the changes in the terminal as they come in.

For more information on exercises, look at the implementation chapter where every exercise created is detailed.

7.5 UML

Python had no public or private classes which contradicted the Unified Modelling Language.

The environment and agent interact with each other and using a simple AI API (which is the files themselves, as they are objects and can be interacted with like an API) the tasks can be completed. The preferred method was to directly edit the classes themselves, so the students could get a direct feel of how the system works. But the API still exists.

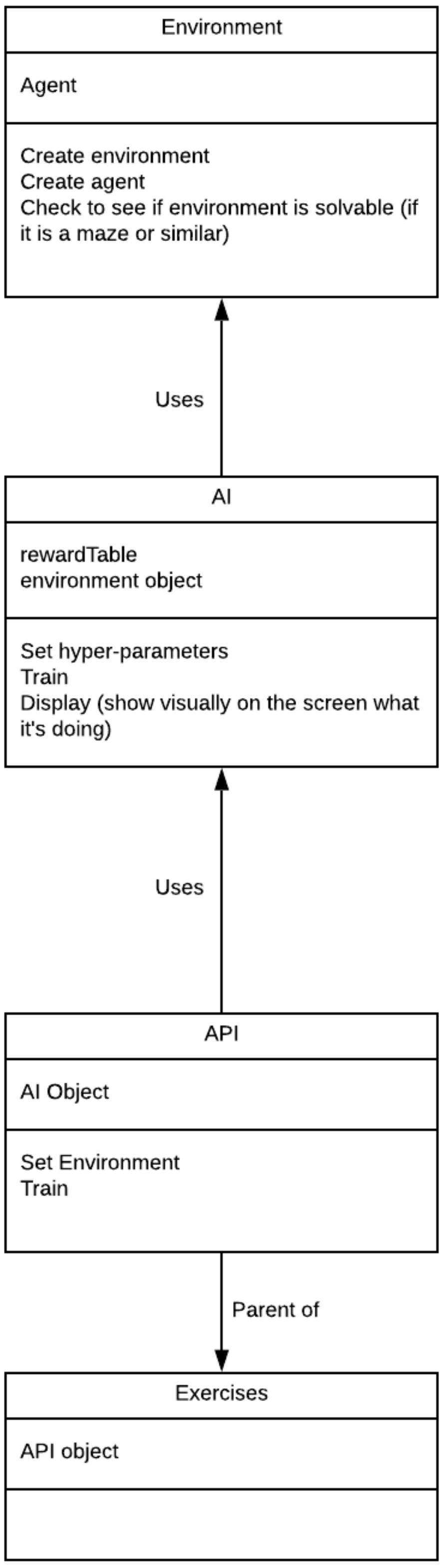


Figure 4: UML Diagram

7.6 Simplification

The code contains simple variables which can be changed before run-time to enable users to learn through real-life reinforcement learning. These variables can be changed:

- The maze itself. Which is defined as self.maze. By changing self.maze, it will change the maze the agent learns on.
- The agent's starting location.
- The allowed legal moves of the agent. In the default example, the agent can move up, down, left or right.
- The goal of the maze. What does the agent touch to know it has reached the end of the maze?
- The rewards table itself. It is possible to manually input into the rewards table.
- Gamma. The value of the future reward. How much the agent values future rewards vs how much the agent values current rewards.
- Learning rate. The speed at which the agent learns.
- Max_epochs. How many epochs will the agent perform?
- Explore. In Reinforcement Learning, it is a balance between exploratory rewards and exploitionary rewards. Setting this to 50, for example, would mean half the time the agent explores and the other half the agent exploits, taking the highest possible reward move.
- AgentReward - how much reward the agent starts with. Traditionally 0.
- PenaltyMoving - how much of a penalty does the agent receive for moving? This prevents the agent from going around in circles.

In the code, these variables were very clearly explained and were at the top of the code. This makes it easier for younger students to find the variables to play around with. Here is a copy of that code.

The code was encapsulated in a class, so students could call the class in Python's IDLE. As an example:

```python
# Where does the agent start>
self.start = (0, 0)

# Where is the goal of the agent?
# self.maze.getSizeOfMaze() sets
# the agent goal to the very bottom right hand corner
# Can use coordinates such as (15, 15)
self.goal = self.maze.getSizeOfMaze()

# The reward table the agent starts with.
# You can manually insert rewards like:
# self.rewardsTable = { (1, 1): {"State": (1, 1),
# "Action": (0, 1), "Reward": 0.75},
# (0, 0): {"State": (0, 0), "Action": (0, 1),
"Reward": 0.75}}
self.rewardsTable = {}

# What actions can the agent take?
self.possibleActions = {"Up": (1, 0), "Down": (-1, 0),
                        "Left": (0, -1), "Right": (0, 1)}

# How much the agent expects to gain from future value.
self.gamma = 0.5

# How fast can the agent learn?
self.learningRate = 0.5

# How many times will the agent complete
# the maze before it stops?
self.max_epochs = 1000

# The probability that the agent will
# explore on that round,
# instead of exploiting the rewards table
self.explore = 0.15

# The agent's current reward when it starts
self.Agentreward = 0.00

# The agents penalty for moving. Prevents
# the agent from running around
# in circles
self.penaltyMoving = -0.05
```

Figure 5: Changable Variable Code

```python
import Qagent
agent = Qagent.agent()
```

Figure 6: Calling the API via IDLE

```python
>>> agent.learningRate = 0.97
```

Figure 7: Changing the variables with the API

From here, it was possible to change the variables as simple as calling them in IDLE. And then calling the agent's run() method.

Since there are many epochs, and each epoch can have 10,000+ moves, it was unreasonable to assume the students can monitor the maze in real-time. It was also unreasonable for the students' computers to be able to display that many frames a second of a maze changing.

Therefore, it was proposed that students either perform 1 epoch to monitor how the maze changes over time and thus how the agent will go in the wrong direction before the rewards table is fully fleshed out, or to look at the Q table and the successful route of each epoch.

If given as a presentation, it was recommended that the teacher uses the former method. The students being able to see that the agent is, in fact, dumb, and repeatedly goes the wrong way, but only finds the correct route after brute-forcing all possible combinations is an invaluable lesson.

The latter, watching the successful route of the maze, didn't look nice on a low-resolution monitor. It was understandable that most schools do not have a high-resolution monitor, and especially those with dyslexia may not be have been able to tell the difference between edges of the maze or the route used. Therefore, it was expected that the latter is used more often than the former.

7.7 Design of the documentation

For more advance students, merely playing with the variables may not be enough to learn. The more advance students may have wished to play around with the code itself, to understand how to build their own reinforcement learning agent. For that reason, almost all lines of the codebase were commented. Extensive documentation was given, not only in the form of comments but in the form of docstrings for each function.

The thought process behind that function and how it behaves as it does was also included. So not only do the students learn how a function behaves as it does, but also how they can develop the thought process to create their own functions for their own reinforcement learning agents.

As an example, take the documentation for the reward() function at figure 8.

"agentLocation = current location of the agent oldLocation = previous location of the agent Steps = all the steps the agent has taken

Record the path the agent takes to get to the end result, once it reaches the end we can update the Q learning table to reflect this. Reaching the reward means that path incurs a reward of 1, but each action also generates a reward. To generate this reward, we use the Q learning algorithm.

To do that, we need to do a few things: * The agent chooses the maximum reward at each point that is exploitationary * Reward is added after each action

But this doesn't solve the issue of the agent mindlessly wandering.

We can apply a negative reward to the agent to prevent this.

But again, this doesn't solve the problem - the agent just gets a lower reward.

We have another problem - the agent may choose to revisit squares it has already been to.

Therefore, the agent takes a negative reward when visiting a square it has been to.

To avoid infinite loops and mindless wandering, it is not enough that the agent takes negative reward. The game should end (the agent loses) when the total reward is below a negative threshold. We assume that when the agent reaches this threshold, it has lost its way and already has made too many errors from which the agent has learned enough. The agent should start a fresh new game.

Eventually, the best path to the maze will be the one which incurs the least penalty after having explored the entire maze (or the highest reward).

The rough numbers I should use are:

* -0.25 negative reward for visiting a square it has already been to * all squares it visits when it reaches goal gets +1 reward * -0.5 reward every time it moves, encourages it to get to end faster * Agent keeps internal state of reward it has collected thus far

Game ends when: * Agent has -0.5 * 10 (size of maze) points (stops infinite loops / senseless wandering)

the rewards table should include the previous state, and the action it took, with the reward gained from it. Every time the agent wants to move, it should consult the reward table"

Figure 8: Documentation for reward()

The end goal of having such extensive documentation was to enable students of all levels the ability to learn from the code. The students starting at a lower level would have been able to read the documentation and understand how the function works and what its rule was in reinforcement learning. The students starting from a higher level will be able to see how the creator's thought process solved the problems solved by the functions, and how they can integrate this own thought process into their own code to better solve problems.

8 Implementation

The implementation of this project was difficult. Not only did the creator have to wrestle with reinforcement learning, but they had to create lessons for key stage 3 students to learn from.

And the code written by the creator would have to be simple enough for a key stage 3 student to manipulate and understand, for them to learn.

Ultimately, this proved many problems for the creator. This section on the implementation will detail the multiple problems the creator had with the implementation, as well as how this project was implemented.

8.1 Python 2 vs Python 3

When the creator first started out writing the code for this project, they had to decide on what version of Python to use. Python 2 would more likely be universally used in schools, as it was much older. But Python 3 was the newest version of Python, and the creator believed that Python 3 would be better to teach.

However, it is not down to the creator of this project to decide on what Python versions schools use. The creator cannot simply show up and demand Python 3 when the school has been teaching with Python 2 for so long. It was unreasonable to expect the school to install a newer version of Python too. Since on Windows, this can create compatibility problems if the Python versions aren't properly sandboxed from each other.

However, in late 2019 the Python Software Foundation, the overseeing body of Python, announced that they will officially sunset Python 2 on the 1st of December 2020 [4].

The creator took this into account and realised that most teachers will likely choose to upgrade to the latest version of Python 3, as that will be the only supported version from 2020 onwards.

However, the creator knows that what a teacher should do and what they are doing are 2 different things. The teacher may not have the resources or the time to manage the switch over and to upgrade all of their own teaching material in time.

For this reason, the creator worked hard to ensure Python2 compatibility as well as Python3 compatibility.

However, it also depended on what version of Python 2 the schools were using. The latest version Python 2.7 worked fine, but any earlier versions that are no longer supported may not have worked.

For this reason, there were 3 options the teacher could have taken to ensure the lesson could continue.

Firstly, the teacher could have installed Python 3 on their own work laptop and used a projector to project the program, and give a lecture on each task of the exercises. But this required the teacher to have a work laptop that they can install software on, and even have a projector. Not all schools will have this privilege.

The 2nd method was to manually teach the principles of reinforcement learning without the use of code. Simply, a lecture was given on learning rates, gamma, and so on.

Finally, the 3rd option was to open an online Python 3 sandbox program. These programs allow any user to create, run, and test Python 3 applications on the web. What's more is that these online sandboxes can install third-party software, such as NumPy which was required for this project.

The final option was a lot harder than it may seem to implement. To use these online code sandboxes the web browser would have to have been the latest update, allowing use of WebGL and more.

8.2 Black

Black [2] was a Python formatter. Run Black on Python code and it will automatically format it according to PEP8 [8], the Python Style Guide. It was created and owned by the Python Software Foundation, the organisation behind the language.

By running the code through Black, it was guaranteed to look similar to other Blackend code, or any code that strictly follows PEP8. Because of this enhancement, the code is easier to be read to someone that has seen Blackened code before or even to someone that hasn't heard of PEP8.

8.3 External Packages

There were many ways to install the external package NumPy, all of them are easy to do and required little resources. The NumPy installation was 20mb, and there are 4 methods mentioned in the documentation.

1. pip install NumPy
2. python setup.py install
3. pip install -r NumPy
4. run install.py

Pip was in the standard library for Python, so every Python installation would have Pip installed. And using Pip does not require administrator rights or a superuser account.

Setup.py will automatically install NumPy. Setup.py is recommended by the Python Software Foundation to be packaged with all public releases to allow anyone to run the code.

```python
from setuptools import setup

setup(
    name='Reinforcement Learning Agents',
    version='',
    packages=[''],
    url='https://github.com/Author/diss',
    license='',
    author='Author,
    author_email='',
    description=''
    install_requires=['NumPy']
)
```

Figure 9: Setup.py

Install.py is the final option, a Python file that calls the command ”pip install NumPy”. Below is the source code for install.py:

```python
import sys
sys.os("pip install NumPy")
```

Figure 10: Install.py

On every version of Python, Pip was automatically installed. It does not require admin or superuser rights to run Pip. The NumPy module is only 20mb large, which is tiny in comparison to other modules. The hardest part about installing NumPy will be running the pip install script, hence 4 options were given to the user.

8.4 Rewards

One of the most daunting tasks of this project was the implementation of a rewards system. It was simple to imagine that the Q algorithm would take care of all the learning, and while it did provide a large relief for the programmer, it was not completely up to scratch with how the rewards system should work.

$$\underbrace{\text{New}Q(s,a)}_{\text{New Q-Value}} = Q(s,a) + \underset{\text{New Q-Value}}{\alpha}\Big[\underbrace{R(s,a)}_{\text{Reward}} + \underset{\text{Discount rate}}{\gamma}\overbrace{\max Q'(s',a')}^{\substack{\text{Maximum predicted reward, given} \\ \text{new state and all possible actions}}} - Q(s,a)\Big]$$

Figure 11: The Q learning algorithm.

More specifically, the largest problem the creator encountered was deciding on what the reward should be.

8.4.1 Positive or Negative Rewards

The agent has 2 options for how their reward progresses. Take the maze example. The agent gains a positive reward for reaching the goal, or the agent incurs a negative reward every action until it reaches the goal.

In the former, the agent is encouraged to find the goal, but it does not take into account the amount of time it takes. The agent gets there in the end. In the latter, the agent is encouraged to move faster to reach the end goal.

However, we must take into account how students, especially younger students, understood a rewards system. In the traditional world, students get rewarded for the "good" things they do, and punished for the "bad" things they do.

The creator of the program decided that attempting to teach a new rewards system to students may be too confusing. Especially trying to explain how incurring a punishment every second was still a rewards system, as the connotation of reward meant good rewards.

For this purpose, the agent's rewards are positive (more than 0), but the agent can incur penalties depending on if the agent has done something the creator does not wish it to do.

The agent reaches a positive reward for reaching the end goal, but the agent incurs a penalty every time it moves. This penalty encourages 2 things of the agent:

- No senseless wandering.
- To hasten the time it takes to find the goal.

It was possible for the agent to become stuck in an infinite loop if it never received a punishment for senseless wandering. And the punishment encourages the agent to move faster to the goal.

However, on some occasions the agent decided to take the punishment as is and do nothing about it. It was possible for the agent to become stuck in an infinite loop, continually receiving negative rewards. It was not enough to merely give the agent a punishment, the agent had to learn to avoid the punishment.

For this reason, the "game over" variable was invented. This variable stated that when the agent should declare game over, and stop learning. The variable is there so the agent cannot get stuck in an infinite loop. If it does, we start the games afresh and retry until the agent learns. The game over variable is based on the agents own personal reward, not the reward from each square in the grid world maze.

The exact numbers chosen for these variables were randomly selected by the designer as being appropriate. It is expected that students will fiddle with these numbers in the tasks created.

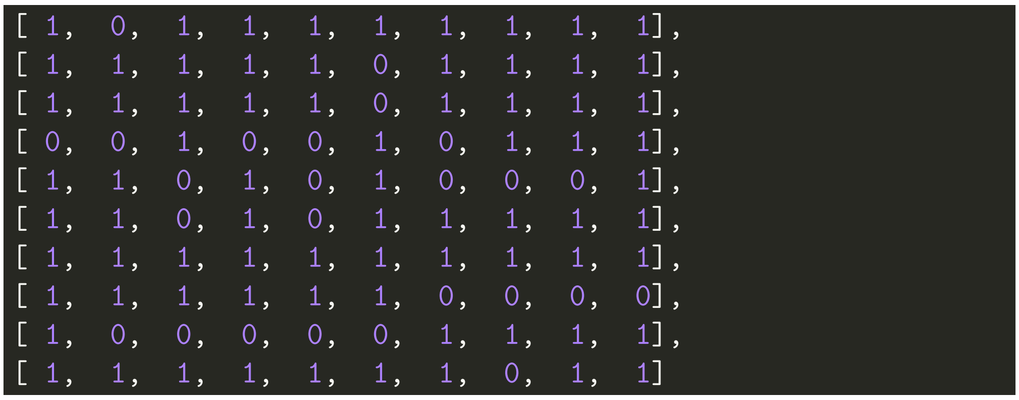
Finally, the agent incurs a further penalty if the agent re-visits a square it has already been to. This penalty is the largest penalty of all, in the hopes that the agent learns that visiting a place it has already been to is severely bad because it should find the route in one go, rather than having to double back on itself.

8.5 The Maze

The maze class controlled the operation of the maze. It controlled how the agent moved around the maze, and what the maze itself was.

The maze itself was defined in a 2-dimensional array. In this array, the number 0 represented a wall (because 0 is physically wide, and looks closet to a wall) and the number 1 represents openness (because 1 looks like a corridor, a thin strip of openness).

The analogies were to help the students understand what each bit of the maze represented.



```
[ 1,  0,  1,  1,  1,  1,  1,  1,  1,  1],
[ 1,  1,  1,  1,  1,  0,  1,  1,  1,  1],
[ 1,  1,  1,  1,  1,  0,  1,  1,  1,  1],
[ 0,  0,  1,  0,  0,  1,  0,  1,  1,  1],
[ 1,  1,  0,  1,  0,  1,  0,  0,  0,  1],
[ 1,  1,  0,  1,  0,  1,  1,  1,  1,  1],
[ 1,  1,  1,  1,  1,  1,  1,  1,  1,  1],
[ 1,  1,  1,  1,  1,  1,  0,  0,  0,  0],
[ 1,  0,  0,  0,  0,  0,  1,  1,  1,  1],
[ 1,  1,  1,  1,  1,  1,  1,  0,  1,  1]
```

Figure 12: The maze.

The start location was defined by agent.startLocation, and the goal was defined by agent.goal.

8.6 Movement of the Agent

The agent chose exploratory or exploitatory moves. There was a 15% chance it would have picked an exploratory (random) move, but this number was expected to be changed by the students.

If the agent chose an exploratory move, it would pick a random legal move.

If the agent chose an exploiatory move, it would calculate the legal moves and work out which one has the highest reward to choose from.

The function getLegalMoves() calculated the legal moves of the agent. The maze got the current location of the agent, performed a move on those coordinates (for example, to go up is to add (1, 0) to the vector of the agent's current coordinates) and then it would calculate whether that move was legal or not.

A legal move needs to meet 3 conditions:

1. The new coordinates were not negative, as there is no negative plane on the 2-dimensional grid world.
2. The new coordinates are not physically outside of the 2-dimensional grid world.
3. The move was possible to do.

The creator made it impossible for the agent to leave the 2-dimensional grid world, by visiting coordinates that did not exist in the planes. For example, starting at (0, 0) and travelling up by (1, 0) would result in coordinates that do not exist in the grid world.

This function is called before the agent tries to move to determine what the legal moves are. This function meant that the agent cannot make an illegal move.

```
legalMoves = {}
for move in self.moves:
    newCoords = self.vectorAddition(self.agentLocation,
                                    moves[move])
    # checks to see if there is an illegal move
    negativeCoords = True if any(y < 0
    for y in newCoords) else False

    outsideMap = True if any(y > len(self.maze[0])
    - 1 for y in newCoords) else False

    illegalMove = True if negativeCoords + outsideMap
    == True else False

    if not illegalMove and newCoords != 0:
        legalMoves[move] = moves[move]
```

Figure 13: Legal Moves Function.

8.6.1 Vector Addition

One of the problems the designer had with agent movement was vector addition. The coordinate system is based on vectors (x, y) where x is the latitude and y is the longitude. To move around on this vector-based coordinate system the agent had to add or take away other vectors depending on the appropriate action. For example, to move physically up by 1 the agent had to add (1, 0) to its current vector represented coordinates.

But, in Python adding 2 vectors together resulted in a vector representation that was unsatisfactory for this project.

$$(0, 0) + (1, 1) = (0, 0, 1, 1)$$

Whereas the creator wanted the vector addition to being element-wise, as seen below:

$$(0, 0) + (1, 1) = (1, 1)$$

To accomplish this, the designer created the function in figure 14. This function called operator.app on the 2 tuples (vectors), and then mapped this function over both vectors, eventually turning it back into a tuple (from a list).

```python
def vectorAddition(self, tup1, tup2):
    return tuple(map(operator.add, tup1, tup2))
```

Figure 14: Vector addition function.

8.6.2 Printing of Data

When the program runs, it prints data to the terminal. For example running the program for the first time and letting it run for a loops results in this:

> ”The steps taken so far are [(0, 0), (0, 0), (1, 0), (2, 0), (1, 0), (0, 0)]”

Here we can see the agent is repeatedly looping back on itself. As the agent learns, the steps it takes changes.

The agent also prints out the rewards table.

> ”(0, 0): ’Action’: 0, ’Reward’: -0.075, ’State’: (0, 0), (1, 0): ’Action’: 0, ’Reward’: -0.075, ’State’: (1, 0), (2, 0): ’Action’: 0, ’Reward’: -0.075, ’State’: (2, 0)”

As the agent runs, the rewards table and taken steps change. Students have been able to follow the agent as it learns.

In the exercises and by default, the epochs of the agent are set very low so the students could observe the agent changing in finer detail. The program also, by default, sleeps for 1 second between every action. This enabled the students to read the print statements and figure out where the agent is going.

8.6.3 Globe Maze vs Flat Maze

In some other programs replicating a maze solving algorithm [10], the maze was built like a globe. That is, if the agent travelled north they would eventually end up on the southern side of the maze.

Whereas in others, a flat maze was used [1]. The agent can't end up on the other side of the map by travelling outside of the border.

It was chosen to use a flat maze to represent the maze. This is because a 3-dimensional global maze represented in 2-dimensions might be too confusing for a student to worry about. The project was designed to help students understand reinforcement learning, not to help them understand how to transpose a 3-dimensional globe into a 2-dimensional flat space.

In some other mazes, the agent receives a harsh negative reward for trying to visit outside of the square. Logically, the author does not believe that a student should have to worry about the agent attempting to travel outside of the maze. To humans, the idea of walking backwards out of a maze and finding the end by walking around the maze, instead of through it would be cheating.

While other creators decided to negatively punish the agent, the author deemed it to too much for the agent to attempt to even travel outside the maze. After all, the students in the classrooms themselves wouldn't travel outside of a maze to try and beat it. Therefore, in getLegalMoves() it is impossible for the agent to even consider travelling outside of the maze.

8.6.4 The Legal Move Set

Another problem the creator faced was "what moves should the agent be allowed to perform?".

One option would be to allow the agent to move Left, Right, Up, and Down. Another would be to allow diagonals such as North-West, South-East, South-West, North-East.

Another option would be to disallow any upward movements, as the goal tended to be downwards. However, at the time the creator decided this didn't make sense. What if the goal was moved? Similarly, what if we banned left movements but the goal was on the left-hand side but there was a wall right below the agent? Therefore, the creator stuck to the traditional up, down, left, and right movements.

When deciding on diagonal movements or not, the creator had to take into account the simplicity of the program. The creator wanted to make the code as simple as possible to allow the students to focus on the reinforcement learning aspect of the project.

If the creator added in diagonal movements, the vector addition would have to change. One cannot simply add one vector to another to get a diagonal movement, the program would have to add at least 3 vectors to get to the right spot. Overcomplicating the code meaning it is harder for students to learn from.

In the end, the creator chose to use the normal move set of up, down, left, and right.

8.7 Creation of the learning materials

The project was not intended to be 100% code-based. More so it was intended to create learning material to teach reinforcement learning to students that use code as a means to teach.

For this reason, the majority of the project sits in designing an easy to understand code interface for students, as well as lessons that teach students reinforcement learning.

8.7.1 The Learning Materials

Here is each part of the lesson that was created. It features a small explanation, along with a task. It was expected that the teacher would give a short presentation on the task, as well as setting everyone up to use the code. It was estimated to take 15 - 25 minutes to get everyone to use the code, and 10 - 20 minutes to explain what's happening and why it's happening. Hence why the tasks are short. Also, the creator of the project was tasked with creating a program with some example tasks to teach reinforcement learning, not lesson plans themselves.

8.8 The Lesson

8.8.1 What is Reinforcement Learning?

Reinforcement learning is similar to how we humans learn. When we want to learn how to open a jar, we repeatedly try to open it in many different ways. First clockwise, then anti-clockwise, maybe we'll use a tea towel or our shirts to get a better grip. Eventually, after trial and error, we know how to open a jar.

Reinforcement learning is the same. The machine repeatedly tries one thing after another until it gets it right.

For example, if we have a maze where the machine has to go from one side to the other, the machine will bump and get lost millions of times before it eventually learns how to complete the maze and even the fastest way through the maze.

8.8.2 What is an Agent?

The agent is the little person that explores the maze! We traditionally call them agents, because we tell them to accomplish something and they do it. And also, universities prefer agents over "tiny computerised humans".

8.8.3 What is a Rewards Table?

The way agents learn is that they store all the information they gather in a table. Specifically, the information gained from an agent is:

- Where did the agent start?
- What is the new location of the agent?
- What is the reward of moving this way?

Take, for instance, the agent is at the start at (0, 0). If the agent moves right, that might give a reward of 0.2. If the agent moves down, it might give a reward of 0.5. The agent now knows the best move is to move down.

The agent has to write these things down. As humans, we also use memory to remember things. "This door opens inwards", "to unlock the front door you have to push the lockdown and the door handle at the same time" and so on. The agent has a memory, just like us.

8.8.4 Task 1

We're going to change how fast the agent learns, and we're going to explore what this will result in.

The learning rate is the speed at which the agent learns. Set it to 1.0, and the agent learns extremely fast. Set it to 0, and the agent doesn't learn at all.

The learning rate can only be between 0 and 1, think of increments like 0.25, 0.7831 and so on.

However, simply increasing it to 1 does not mean the agent finds the best route through the maze. Because the agent learns so fast, it might be rushing through the maze missing all the import steps to finding the fastest way through it.

But setting it too low such as 0.01 means the agent is very slow, and it will take a longer time for the program to complete.

Open the file 'Qagent.py', and find this part in the file (it's on line 13):

> " How fast can the agent learn? self.learningRate = 0.25"

1. Set the learning rate to 0.5.

Now, run the file 'maze.py' and observe the output. **Note: every time we want to run the agent, we have to run maze.py**

Look at the path the agent took, and the rewards table. How big is the path? How small is the rewards table?

Play around with the learning rate. Set it to any number between 0 and 1, and see for yourself how changing the pace of learning the agent's outputs are different.

8.8.5 Task 2

We're going to play around with the 'self.explore' variable now.

The explore variable is a probability between 0 and 1 (where 0.15 is 15

In reinforcement learning, we have a trade-off of exploiting what we know or trying something different.

Take, for instance, buttering bread.

If we butter bread like a circle, going clockwise around the bread it's not very efficient. But if we never explore, we would never find the most efficient way to butter the bread.

If the agent only does what it knows, it will never find the most efficient route through the maze.

The exploratory variable means that every move the agent takes, it has a 15% chance to randomly select a move, regardless of whether or not it understands how rewarding the move is.

Find the exploratory variable at line 37 of Qagent.

> "The probability that the agent will explore on that round, instead of exploiting the rewards table self.explore = 0.15"

1. Set the exploratory percentage to 100 ('self.explore = 1') See how the agent has lost its memory? It has a memory, but it doesn't use its memories! Every move it makes is random.

What if we set explore to 0? Will the agent find the best route through the maze or not?

8.8.6 Task 3

Now we’re going to understand the randomness aspect of exploratory. Because the agent explores, that means every time the agent attempts the maze will be different from the last time. Run your programs and have a look at the person next to you. See how they’re different routes? Different rewards? One of your agents is smarter than the other!

But, you might see a problem. There is only 1 route which is the fastest way through the maze. But if there are 1 route and every agent is slightly different because of the random explorations, how does it find the best route?

The answer is using epochs.

Epoch is a fancy word for ”time”. Specifically, we might tell the agent: ¿ ”Do this maze 10 billion times” If we make 2 agents do the maze 2 or 3 times, their answers will be completely different. But if we make the agents do the maze 10 billion times, they will $_{c}onverge_{o}nthecorrectanswer - thefastestroutethroughthemaze$!

Set the epochs variable to a high number, such as ‘10000‘. However! Your screen might go crazy, so try not to read every single line that comes out of the program.

You can find epochs on line 34 of Qagent.

”How many times will the agent complete the maze before it stops? self.max$_{e}pochs = 1$”

Note: You will need to set max$_{e}pochsbackto1fortherestofthistutorial.$

8.8.7 Task 4

Let’s talk about gamma!

Gamma is the value of the future reward. If Gamma is set to 1, the agent sees reaching the goal in 10 moves and reaching the goal in 1 move as the same value.

If gamma is set to 0, the agent values direct moves (what it's doing on the next turn) more than it does in the future.

When we humans try to solve a maze, we know where the exit is and we try to tend towards the exit. If the agent's gamma was 0, the agent wouldn't tend towards the exit. The agent would simply try to collect as much reward as possible until it accidentally stumbles upon the exit.

Find gamma on line 31 of Qagent:

> "How much the agent expects to gain from future value.
> self.gamma = 0.5"

1. Set the gamma rate to 1. 2. Set the gamma rate to 0.

See how they differ?

8.8.8 Task 5

Now, we're going to work on punishing the agent. To prevent the agent from wandering around in circles, the agent takes a punishment everytime it moves.

If the punishment is too much, the agent gives up and we get a new agent to try (but with the same rewards table as the last agent).

Increasing the punishment means the agent gives up faster, but also means the agent is more encouraged to find the goal faster.

Too low of a punishment and the agent will think "why bother?" and will reach the goal in a much slower method.

Find this on line 43 of Qagent:

> "The agents penalty for moving. Prevents the agent from running around in circles self.penaltyMoving = -0.05"

1. Set the penalty to -1. 2. Set the penalty to 0.

Observe the outputs.

8.8.9 Task 6

This is a more advanced task, so be prepared!

The maze the agent uses is located in ‘maze.py‘ (line 12), specifically this is it:

> ”self.maze = np.array([[1, 0, 1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 0, 1, 1, 1, 1], [1, 1, 1, 1, 1, 0, 1, 1, 1, 1], [0, 0, 1, 0, 0, 1, 0, 1, 1, 1], [1, 1, 0, 1, 0, 1, 0, 0, 0, 1], [1, 1, 0, 1, 0, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 0, 0, 0, 0], [1, 0, 0, 0, 0, 0, 1, 1, 1, 1], [1, 1, 1, 1, 1, 1, 1, 0, 1, 1]])”

The 1 represents openness, a straight corridor for the agent. The 0’s represent a wall, something the agent can not pass.

What if we make the goal impossible to reach? The goal is in the bottom right-hand corner.

1. Change some of the 1’s to 0’s (walls) to prevent the agent from reaching the goal. What will happen then?

Play around with changing the layout of the maze and see how the agent reacts.

8.8.10 Task 7

Now we’re going to change what moves the agent knows.

On line 28 of ‘Qagent.py‘ the agent defines the moveeset as:

> ”self.possibleActions = ”Up”: (1, 0), ”Down”: (-1, 0), ”Left”: (0, -1), ”Right”: (0, 1)”

The agent can move up, down, left, or right.

It does this using vector addition. On a 2 dimensional map, adding (1, 0) to our current coordinates (15, 15) means we would go up 1 square to (16, 15).

1. Try to remove some moves. You can do this by deleting the entire dictionary entry. For example, to reduce Up do this:

> ”self.possibleActions = ”Down”: (-1, 0), ”Left”: (0, -1),
> ”Right”: (0, 1)”

Don’t forget to remove the comma!

8.8.11 Task 8

Now we’re going to change the agents start location. Currently, it starts at the top left (0, 0). But it can start anywhere!

The way the coordinates system works is (row, column). ”Along the corridor, up the stairs”.

Find the code on line 26 of ‘maze.py‘:

> ”self.agentLocation = (0, 0)”

1. Try setting the agent location to (5, 5) and seeing what the agent does! Does it walk to the top left again? Or go straight to the goal.

8.8.12 Task 9

Change the goal.

The goal variable is located at line 21 of ‘Qagent.py‘.

> ”self.goal = self.maze.getSizeOfMaze()”

1. Change the goal to a location somewhere on the maze. For example, (15, 15) like so:

> ”self.goal = (15, 15)”

2. If you set the goal and start location to the same place, what happens?

8.8.13 Task 10

This is an advanced task. 1. Change how the agent learns

Some things you may want to do: * Remove the penalty completely (by setting it to 0.0) * Remove the agent giving up when it is punished too much * Play with the learning rate * Play with the gamma rate

Try to find the most optimal settings which result in the most optimal agent. Often, we would program a computer to automatically find the most optimal settings. But it is possible to deduct and use logic to work out roughly what the most optimal settings are.

Google things you don't understand such as "learning rate" or "gamma q learning" to get the answers you're looking for.

9 Testing and Evaluation

Throughout the project, the creator used many means of testing to ensure the program was of high quality. Furthermore, the creator regularly asked for feedback to enhance the project.

9.1 Continuous Integration

Every time the code is committed to the master branch on GitHub, a GitHub action automatically tests the code. Firstly, for syntax compliance. Secondly, for unit testing. The syntax testing used was Black, and the testing module used was Pytest.

More specifically, the GitHub action in figure 15 was executed.

```yaml
name: Python application

on:
  push:
    branches: [ master ]
  pull_request:
    branches: [ master ]

jobs:
  build:

    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v2
    - name: Set up Python 3.8
      uses: actions/setup-python@v1
      with:
        python-version: 3.8
    - name: Install dependencies
      run: |
        python -m pip install --upgrade pip
        pip install black pytest
        if [ -f requirements.txt ]; then pip install -r
        requirements.txt; fi
    - name: Lint with black
      run: |
        # stop the build if there are Python
        # syntax errors or undefined names
        black . --count
        --select=E9,F63,F7,F82
        --show-source --statistics
        # exit-zero treats all errors
        as warnings. The GitHub editor
        is 127 chars wide
        black . --count --exit-zero
        --max-complexity=10
        --max-line-length=127 --statistics
    - name: Test with pytest
      run: |
        pytest
```

Figure 15: GitHub Action.

9.2 Unit Tests

Because the project was based on reinforcement learning, the exact maze, outputs, rewards, and other variables will differ for every time the program was run. Thus, there are little unittests involved other than to make such the Q learning algorithm operates correctly and the program doesn't break under pressure.

The testing software used was PyTest, rather than the standard library UnitTest. This was because:

- PyTest required less code to create tests.
- PyTest provided more information on the errors.

Although, the biggest benefit to PyTest was that it was the default testing suite in GitHub Actions.

The tests were written in the style of boundary testing. Where a test is written for one extreme, and then the other. For instance, the Q Learning Algorithm tests were written using negative numbers, extremely large numbers, and the number 0. See figure 16 for some of the tests used.

```python
def test_q_algorithm_normal():
    """
    Testing to see if the q algorithm operates correctly
    """
    q = Qagent.agent(m)
    result = q.qAlgorithm(0.54, 1, 0.67)
    assert result == 0.835

def test_q_algorithm_negative():
    """
    Testing to see if it breaks on negative
    """
    q = Qagent.agent(m)
    result = q.qAlgorithm(-50000, 60, 0.9999999)
    assert result > 1

def test_q_algorithm_big():
    """
    Testing to see if it breaks on large numbers
    """
    q = Qagent.agent(m)
    result = q.qAlgorithm(5000000000000000000
    00000000000000000000000000000000, 5000000000000000
    00000000000000000000000000000000000, 50000000000
    0000000000000000000000000000000000000000)
    assert result > 1

def test_q_algorithm_zero():
    """
    Testing to see if it breaks on 0 input
    """
    q = Qagent.agent(m)
    result = q.qAlgorithm(0, 0, 0)
    assert result == 0.0
```

Figure 16: PyTest's tests,

At the time this test was run, PyTest outputted this:

```
....
4 passed in 0.09s
```

Figure 17: PyTest results.

Where the four full stops represent each test as seen above passing.

More tests were written for the maze program. The maze class was responsible for creating the maze and deciding on what moves are legal. See figure 18 for the maze program's tests.

```
def test_maze_movement():
    """ Testing the movement of the agent
    in the maze"""
    m = maze.maze()
    # move to the right starting at 0,0
    result = m.vectorAddition((0, 1), m.getAgentLocation())

    assert result == (0, 1)

def test_maze_movement_up():
    """ Testing the movement of the agent
    in the maze"""
    m = maze.maze()
    # move to the right starting at 0,0
    result = m.vectorAddition((1, 0), m.getAgentLocation())

    assert result == (1, 0)

def test_maze_movement_legal_moves():
    """ Does legal moves work?"""
    m = maze.maze()
    # move to the right starting at 0,0
    result = m.getLegalMoves()

    assert True if "Right" in result else False
```

Figure 18: PyTest Maze.

```
test_maze.py ...
test_program.py ....
7 passed in 0.10s
```

Figure 19: PyTest Maze results.

All the tests ran successfully.

9.2.1 Continuous Manual Testing

After a new function was written, that function would be copied to Python's IDLE, where it was evaluated. Take the function of element-wise vector addition. After the function was written, it was copied and pasted into a terminal and manually tested.

The function did not rely on any external libraries or code, it merely added 2 vectors element-wise and returned it. As such, the chance of this function containing future errors after testing was none, unless Python's default operators changed drastically, which was unlikely in the near future.

Other functions were also the same, where it was unfeasible to imagine that in the future they would break. Thus, no unit tests were written for these functions.

After each function was added and integrated into the core code, the code was run. The programmer evaluated the code, making sure that the integration of the new function did not break any important parts of the code.

If the code did break, no major harm came to the already working code. The programmers utilised branches on GitHub, creating a new branch for each change to the code. This meant that the master branch was always fully tested and worked, whereas the other branches could be broken.

Once the programmer committed code to the master GitHub branch, the code was evaluated using the GitHub action. If GitHub failed that testing, it would email the creator to let them know.

9.3 Evaluation

Earlier in this projects life cycle, it had been planned that the project would be demonstrated to teachers in classrooms or even used in classes, and the creator would use this feedback to better the product.

However, due to COVID-19, the creator couldn't enter a classroom.

Therefore, the creator opted to ask other people who work with children (who are around the same age as year 7's to year 11's).

The creator sent the program and the learning materials to these people. However, most of them weren't in ICT and didn't understand Python, or artificial intelligence.

One clear comment was that if the teacher had decided to google "gamma reinforcement learning", the explanation given is that which would require a university degree to decipher. Because of this feedback, the creator tried to create a simpler explanation of some of the core concepts. So not only students can understand, but teaching staff may understand as well.

Throughout this project, the creator spoke regularly with Dr. Dennis and Dr. Thomason, who were in the outreach program at the University of Liverpool for Computer Science.

Dr. Thomason's most critical comment was that the project was too ambitious for a classroom. They stated that in a class, the creator can expect up to 20 minutes of solely installing the program, and without clear guidance, the students were likely not to learn. Thus it was needed for learning material to be created alongside, which specifies exactly what the students should do.

Another comment made by Dr. Dennis was that the students are all at different levels. It may be a GCSE Computer Science class, but some of the students will speed through the resources while others might spend the entire lesson on the first task. It proposed an age-old question, "what do we do about the difference in students' abilities?".

The answer was to create many tasks, progressively getting harder with each task. The idea is that the first few tasks are easy, and can be used to encourage the slower students to get through them. While the last tasks are very difficult, which given the limited amount of time in a classroom should prove worthwhile to the faster students.

Another comment made that impacted the work of the creator was that reinforcement learning was simply too hard. The original idea was to teach students how to design an autonomous driving vehicle. Dr. Thomason told the creator that what may seem easy to a university student isn't easy to a secondary school student. For this reason, the simplest task of them all is changing a variable and seeing how it affects the agent's run.

Overall, the creator would have liked to get the software in front of a classroom, to ask teachers directly and see how the students learn from it. But due to COVID-19, it simply wasn't possible.

10 BCS Project Criteria

10.1 Practical and Analytical Skills

This relates to BCS criteria:

> "An ability to apply practical and analytical skills gained during the degree programme."

This project is an AI system designed to be taught in schools. The creators' modules over their degree which relate to this project are:

- Artificial Intelligence.
- Advanced Artificial Intelligence.
- Communicating Computer Science.
- Software Engineering.
- Software Engineering 2.

The skills gained from artificial intelligence were used to create a reinforcement learning agent. Without those modules, the creators understanding of artificial intelligence would have acted as an inhibitor for finishing this project.

The skills gained from communicating computer science were used to better design effective exercises for the students, as mentioned in the key literature and background reading.

The skills gained from software engineering helped the creator write better code, better in the sense that it runs faster but also better in the sense that users can read it, that it is tested, and it is generally good code.

10.2 Innovation and creativity

As Isaacson [7] says in his book, The Innovators:

> "innovation comes from being able to stand at the intersection of art and science."

The project is scientific in the sense of creating a reinforcement learning agent. However, the intersection with art comes from creating a project to use it to teach key stage 3 students.

On the technical side, the innovation came from creating a Python implementation of a reinforcement learning agent using basic code, without relying too heavily on external libraries. In most examples, the code was written to only be understood by other programmers. Whereas the code for this project was written to be understood by anyone.

10.3 Synthesis of Information

This relates to BCS criteria:

> "Synthesis of information, ideas and practices to provide a quality solution together with an evaluation of that solution."

This project was created by Dr. Dennis to solve a real-world problem. When the creator took up this project, they read books such as [12] as discussed in Key Literature and Background Reading.

The creator took in this information and decided on what were the best parts to take from each resource. The maze problem from Sutton, the teaching skills from MIT and COMP335. The creator took all of this information and combined the best of it into one quality solution.

Then, the creator critically evaluated this project in the Evaluation section. As well as continually evaluating it throughout the projects life cycle. The creator regularly sought guidance, asked for help, and watched as other people attempted to use the project to learn. Using this information, the creator continually improved the project until the end of its life cycle.

10.4 Real World Need

This relates to BCS criteria:

> ”That your project meets a real need in a wider context”

Dr. Dennis, the creator’s supervisor, created this project because of a gap in the market. More specifically, computer science taught in classrooms wasn’t fun, or approximately close to what computer scientists were working on at the time. Sorting algorithms such as bubble sort were taught for multiple lessons, as an example.

Dr. Dennis decided that reinforcement learning would be fun, as it is a current subject in artificial intelligence still being worked on, and can be explained easily to a younger student using analogies such as how that student learns.

10.5 Self-Manage a Significant Piece of Work

This relates to the BCS criteria:

> ”An ability to self-manage a significant piece of work.”

Throughout this project, the creator regularly stuck to the Gantt chart to monitor how well they were doing and the time schedule of code. As well as the Gantt chart the creator planned for emergencies, any issues that might arise and took care of them accordingly.

This dissertation is one example of many that the creator can self-manage a significant piece of work.

10.6 Critical Self-Evaluation of the Process

At the start of the project, the creator went to work quickly. Having finished a prototype before the first month finished. The creator asked Dr. Thomason for support in creating the educational texts and even took up online classes to better learn how to teach this subject to the students.

The creator followed the Gantt chart originally made, and all was set to be well.

Unfortunately, the creator suffered an illness in December that went on until late February or early March. During this time, the creator carried on coding but could not communicate effectively with their supervisor or the university.

Shortly after the creator became well again, COVID-19 stormed the world. Causing panic in the creator, and worry about what's going to happen.

Unfortunately, the creator did not succeed in all the original tasks of the dissertation. One of the original tasks was to implement this reinforcement learning agent into a small robot. This task failed, mainly due to the illness but partly due to COVID-19 not allowing the creator to physically pick up the small robot to try and implement it.

In the future, the creator will take extra care and will try to assume the worst - including a potential global pandemic. If the creator had been in a more communicative state during their December - February illness, they might have just avoided COVID-19 and may have been able to implement the agent into the small robot.

The creator, unfortunately, felt like they didn't need to tell anyone what was wrong with them, which led to their supervisor and university becoming worried and asking if they were okay. The creator did not want to admit their own defeats, so they refused to communicate with the university from December to February.

The dissertation portion of the final year project was also delayed. The creator felt like they could create more tasks, more agents that could help aid the students. The creator felt bad that they couldn't complete the small autonomous robot task, as it was impossible to pick up during COVID-19. Therefore, the creator, instead of spending time working on the dissertation, spent most of their time trying to create more tasks and improving the robot.

What would have been a better choice would be to discuss with their supervisor what the best use of their time could be, which was the dissertation itself.

Overall, the creator has learnt that to self-manage such a large project means that the creator should regularly communicate with those supporting them. Without that support in December - February, the creator was lost and this led to the failure of the mini robot car task.

The creator should have put their own health first, and then the university studies second. This would have no doubt caused the illness the creator to suffer from to have been reduced, and it would have let the creator continue to work much sooner. The creator was scared that their supervisor wouldn't allow them to look after themselves and they would have been forced to work full speed on their project. When in actual fact, looking at what has happened it seems more likely that the creator would have done more work if they had been honest and open with those actively supporting the creator.

It would have been nice for the creator to show the agent moving through the maze on the map, clearly. However, this caused issues at higher epochs and it was hard to see the agent moving through the maze amongst the 0's and 1's, again the creator focussed too much on the design and creation of the project rather than spending it on working what matters most to the project.

11 Conclusion

Although this project failed in one of the more major tasks, it was a success in that the creator still managed to fulfil the original goal of creating a reinforcement learning system that can teach key stage 3 students about artificial intelligence and encourage them.

There would always have been something extra to have done or something new and exciting to add to the project. Merely finishing the project on time was a blessing for the creator, and it is exciting to know that the project will potentially be used to excite the next generation of computer scientists.

Looking back at the introduction, these were the tasks assigned to the creator:

- Create an ASCII based grid-world reinforcement learning playground.

- Implement a reinforcement learning framework in an existing Python-based robot simulator.
- Design an API for reinforcement learning.
- Devise a set of challenge problems to be used with the simulator.
- Design the learning exercises.
- Gather feedback about the learning exercises and improve upon it.
- Create an autonomously driving robot vehicle agent.

The creator did make a grid-world reinforcement learning playground, in maze.py. The creator did design an API to be used in IDLE, they did devise a set of problems to be used with the simulator as seen in the exercises section. The creator gathered valuable feedback from the exercises and used that feedback to improve upon the exercises.

Overall, the creator succeeds in 6 out of the 7 tasks, which is positive as most software projects tend to fail or over-run.

References

[1] Alexander Amini. *MIT 6.S191: Introduction to Deep Learning.* https://www.youtube.com/watch?v=5v1JnYv$_y Ws$. Accessed 19/10/2019. Feb. 2019.

[2] Mariano Anaya. *Clean Code in Python: Refactor your legacy code base.* Packt, 2018.

[3] Benjamin M. P. Cuff. "Perceptions of subject difficulty and subject choices: Are the two linked, and if so, how?" In: *Ofqual* (October 2017). Published for Ofqual.

[4] Python Software Foundation. "Sunsetting Python 2". In: *Pycon.* 2019.

[5] Lex Fridman. *MIT 6.S094: Introduction to Deep Learning and Self-Driving Cars.* https://www.youtube.com/watch?v=1L0TKZQcUtA. Accessed 19/10/2019. Dec. 2017.

[6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* The MIT Press, 2016. ISBN: 0262035618, 9780262035613.

[7] Walter Isaacson. *The Innovators: How a Group of Hackers, Geniuses, and Geeks Created the Digital Revolution.* SIMON & SCHUSTER, 2014. ISBN: 147670869X, 9781476708690.

[8] Tanya Schlusser Kenneth Reitz. *The Hitchhiker's Guide to Python: Best Practices for Development.* O'Reilly, 2016.

[9] MIT. *MIT Deep Learning.* https://deeplearning.mit.edu/. Accessed 19/10/2019.

[10] D. Osmanković and S. Konjicija. "Implementation of Q — Learning algorithm for solving maze problem". In: *2011 Proceedings of the 34th International Convention MIPRO.* 2011, pp. 1619–1622.

[11] Ken Schwaber and Mike Beedle. *Agile software development with Scrum.* Vol. 1. Prentice Hall Upper Saddle River, 2002.

[12] Richard Sutton and Andrew Barto. *Reinforcement Learning: An Introduction.* MIT Press, 2018.

[13] Dr Stuart Thomason. *Communicating Computer Science.* University of Liverpool Computer Science Module. 2019.

[14] John A. Zahorik. “Elementary and Secondary Teachers’ Reports of How They Make Learning Interesting”. In: *The Elementary School Journal* 96.5 (1996), pp. 551–564. DOI: `10.1086/461844`. eprint: `https://doi.org/10.1086/461844`. URL: `https://doi.org/10.1086/461844`.