Reinforcement Learning for Animal Behaviour

Connor Stewart

2515217

Bruce Graham

April 2020

Dissertation submitted in partial fulfilment for the degree of
Bachelor of Science with Honours Applied Computing

Computing Science and Mathematics

University of Stirling

Abstract

Problem

The problem being tackled is simulating animals each with a very complex state interacting with each other and their environment.

Objectives

- Implementing both fauna and flora, including predator and prey animals.
- Environmental objects such as rocks and water and trees.
- Breeding, the creatures will breed to create the next generation of creatures, passing on what they have learned about their environment.
- Increase the complexity of the simulation, for example, increasing the number of creatures and the complexity of the states the creatures can be in.

Methodology

This project used an agile methodology as this project was susceptible to change as more research was done in the problem area.

Achievements

- A reward table and q-table reinforcement learning framework.
- Agents which use the above framework in order to succeed in their environment.

Attestation

I understand the nature of plagiarism, and I am aware of the University's academic misconduct policy.

I certify that this dissertation reports original work by me during my University project except for the following:

Figure 1 was taken from [1]

Figure 2 was taken from [2]

Signature *Connor Stewart* **Date** *24/03/2019*

Acknowledgements

Thanks to my supervisor Bruce Graham for his help and support during this project.

Table of Contents

List of Figures

1 Introduction

Reinforcement learning is a machine learning method which uses trial and error to teach an agent how to overcome a problem in a dynamic environment. [3] Deep reinforcement learning is the combination of deep learning and reinforcement learning. [4] This project aims to simulate creatures using reinforcement learning in order to create emergent behaviour between them.

1.1 Background and Context

The problem involves simulating animals with a very complex state interacting with each other and their environment.

1.2 Scope and Objectives

The scope of the project will include the following:

Fauna, including predator and prey animals. Flora, including plants to serve as food for prey animals and trees to act as obstacles.

Environmental objects such as rocks and water.

Breeding, the creatures will breed to create the next generation of creatures, passing on what they have learned about their environment.

The project will use reinforcement learning to simulate creatures within an environment. The main objective of the project is to create emergent behavior. A secondary goal would be to push how complex the simulation can become, for example, increasing the number of creatures and the complexity of the states the creatures can be in.

1.3 Achievements

I have achieved:

- An animal taking in its current state.
- An animal learning from its mistakes.
- Q-Learning Implementation.

1.4 Overview of Dissertation

<u>2 – State-of-The-Art</u>

Discussing other relevant projects and how this project relates to them.

<u>3 – Problem Description and Analysis</u>

Discussing Q-Learning and deep reinforcement learning with regards to the project, including implementation details.

<u>4 – Conclusion</u>

Discussing how the project went overall, what could be done to improve it and the results of testing.

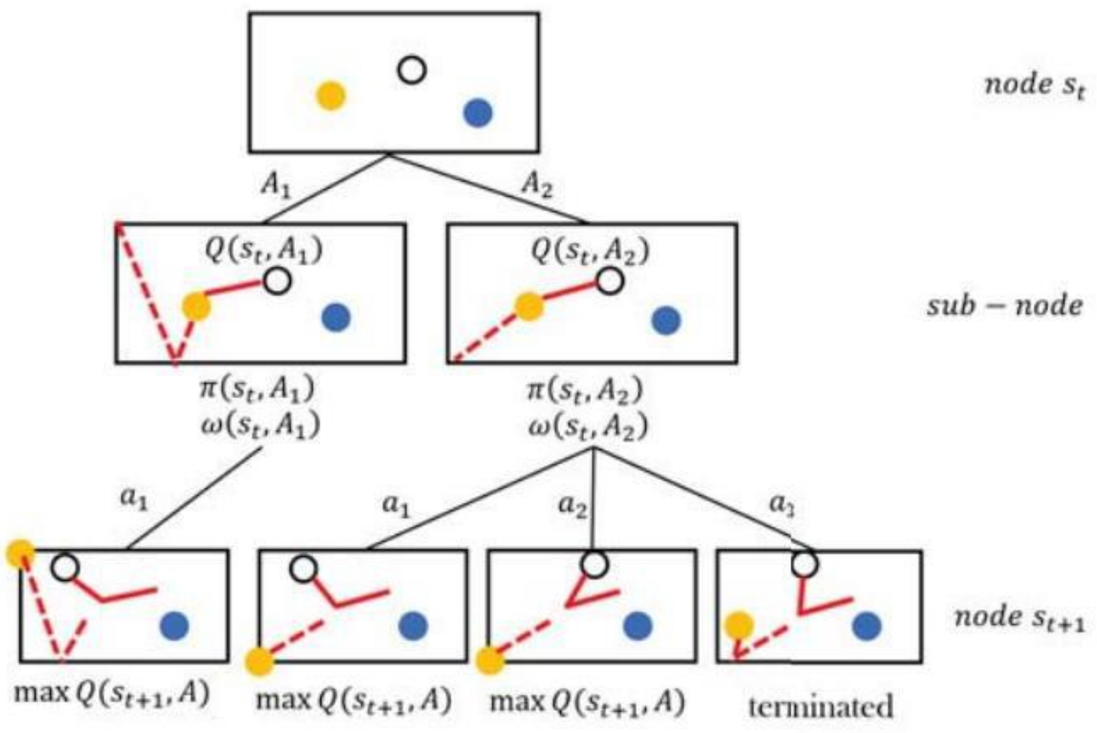
2 State-of-The-Art

2.1 Work of Others

2.1.1 Teaching an AI to play pool with RL [1]

Reinforcement learning has been used to teach an AI how to play 9 ball pool. In this project Yu Chen & Yujun Li use RL to teach an AI to play pool. In this project the state used is the position of the balls and whether they are pocketed. The actions are the angle, speed, position and offset at which the ball is hit. An interesting feature of this project is that it uses a tree to look at the possible states that would result in taking the actions available in its current state. The following is an example of the tree:

Figure 1. Nine Ball Pool Tree Diagram

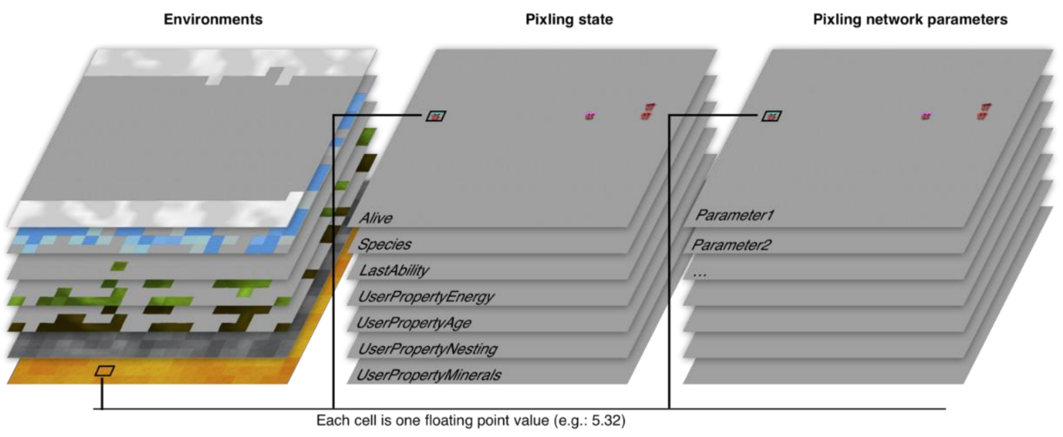


The tree contains both macro(A) and micro(a) steps, the macro actions detail where the ball should be hit to give the best chance of pocketing another while the micro actions give the best position for the ball to be in on the next iteration.

2.1.2 Pixling World [2]

Pixling world is a simulation of creatures called “Pixlings”. Within the simulation each pixling has a state and a neural network attached to it, the neural network is fed parameters from the state of the pixling in order to choose the next action the pixling will take. This allows the simulation to have over one million pixlings being simulated at 60 steps per second! A difference with pixling world compared to other works discussed is that the pixlings or agents have no defined goals. Instead of going for a defined goal the pixlings use mutation to evolve, each time two pixlings breed their parameters are combined and sometimes mutated. This mutation is what allows for Darwinian evolution to take place, for example a pixling may mutate to have a better trait which favours their environment meaning it will survive to create more offspring.

Figure 2. Pixling World State Diagram



Pixling World is a web-based simulation, it uses React – A JavaScript framework – for its UI. In order to reach the impressive millions of pixlings being simulated at once the GPU is utilised through WebGL which is a framework used to code OpenGL shaders on a webpage. Each pixling in the simulation has its own neural network and a set of attributes, including “Alive”, “Species” and “LastAbility”. Parameters are fed into this neural network which will then give the output of either “move randomly” or “don’t move”.

2.1.3 Playing Atari with Deep Reinforcement Learning [5]

In 2013, DeepMind created a deep learning model capable of playing seven games from the Atari 2600. The model outperforms all previous approaches on six of the games and surpasses humans on three of them. The model is given no other information other than a 210x160 RGB image of the video game being played. No specific data about any of the games was given, yet the model was still able to get these amazing results.

The DeepMind implementation uses an Atari emulator to run the games. The only input that the model gets is a 210x160 RGB image each frame. The model then uses reinforcement learning to pick the best action from a set of actions that differ between games. The implementation uses a modified version of Q-Learning, an action is selected every time step using a neural network, which is trained using the previous experience of the model.

2.2 Extending & Complementing Existing Works

This project is influenced by the work of Fredrik Norén on Pixling World and the projects at DeepMind such as their Atari model. I would like to extend the work of Pixling World by simulating animals with certain goals in mind instead of simply surviving to reproduce. I will use techniques I have learned from projects such as the Atari model by DeepMind.

3 Problem Description and Analysis

3.1 Description

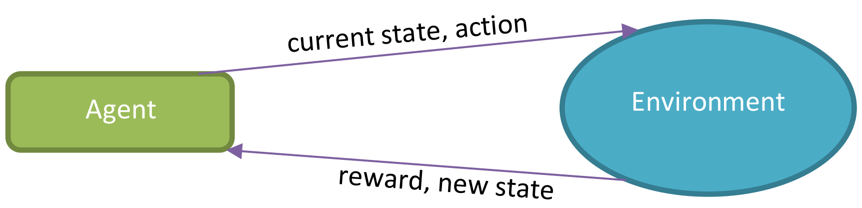
The problem I am trying to solve is simulating a relatively large number of creatures interacting with each other and the environment. The goal of the simulation is to create emergent behaviour in the creatures being simulated while they try to complete their goals of surviving and passing on their genes, The creatures within the simulation will have a number of attributes, such as; a need for both food and water, a vision radius, speed, size and hunger.

I will use deep reinforcement learning to solve this problem, deep reinforcement learning comes in two parts: the reinforcement learning algorithm, in this case Q-learning and a deep learning tool such as a neural network.

3.2 Q-Learning

Q-Learning was first published in 1992 by Christopher Watkins & Peter Dayan. Q-Learning is a form of reinforcement learning. Q-Learning records a set of states and actions which can be taken from those states as well as reward values for taking those actions. [6]

Figure 3. Agent/Environment Diagram.



We will use the following grid as an environment (the numbers indicate the state index):

Figure 4. States Diagram

🟥 0	🟦 1	2	🟩 3	4	5
6	🟦 7	8	🟦 9	10	11
12	13	🟦 14	15	🟦 16	17
18	19	20	21	22	23

Where the Agent is coloured red and can only move in clear and green squares. The agent must collect food which is coloured green.

For Q-Learning to work we must have both a reward table and a Q-table. In our example we have four actions which will be used as the indexes of our tables:

left: 0, Right: 1, Up: 2, Down: 3.

On the next page the rewards table and Q-table are shown. The **x axis** is the index of the **state**, shown above and the **y axis** is the **action** taken from that state.

3.2.1 Rewards Table

The rewards table is used to guide our agent through its environment, it tells it which states are favourable and which states should be avoided.

In this table -1 is used to note that our agent cannot enter this state.

The reward at 4,1 has a value of 10 as this is action leads to a state which has food.

Figure 5. Rewards Table Example

	0	1	2	3
0	-1	-1	-1	0
1	-1	-1	-1	-1
2	-1	10	-1	0
3	0	0	-1	-1
4	10	0	-1	0
5	0	-1	-1	0
6	-1	-1	0	0
7	-1	-1	-1	-1
8	-1	-1	0	-1
9	-1	-1	-1	-1
10	-1	0	0	-1
11	0	-1	0	0
12	-1	0	0	0
13	0	-1	-1	0
14	-1	-1	-1	-1
15	-1	-1	-1	0
16	-1	-1	-1	-1
17	-1	-1	0	0
18	-1	0	0	-1
19	0	0	0	-1
20	0	0	-1	-1
21	0	0	0	-1
22	0	0	-1	-1
23	0	-1	0	-1

3.2.2 Q-Table

The Q-Table stores all the states the agent can be in and every action that can be taken in that state.

The Q-table is updated to allow our agent to learn how to navigate the environment we have put it in. The value of state action combinations is stored here so the agent can eventually find the best combinations for its environment.

Figure 6. Q-Table Example

	0	1	2	3
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0
7	0	0	0	0
8	0	0	0	0
9	0	0	0	0
10	0	0	0	0
11	0	0	0	0
12	0	0	0	0
13	0	0	0	0
14	0	0	0	0
15	0	0	0	0
16	0	0	0	0
17	0	0	0	0
18	0	0	0	0
19	0	0	0	0
20	0	0	0	0
21	0	0	0	0
22	0	0	0	0
23	0	0	0	0

3.2.3 Updating the Q-Table

For our agent to learn from its past actions we must store the result of taking the action within the Q-table. To do this we use the following equation [7] [8]:

$$Q(s_n, a_n) = (1 - \alpha_n)Q(s_n, a_n) + \alpha_n(r(s_n, a_n) + \gamma \max_{a} [Q(s_{n+1}, a)])$$

n = The current timestep of the algorithm.

$a =$ An action taken by the agent.

$s =$ A state the agent can be in.

$Q(s_n, a_n) =$ The Q value of taking the action (a) from the state(s) at timestep n.

$r(s_n, a_n) =$ The reward given for taking action (a) from the state(s) at timestep n.

γ = The gamma, which is the discount factor. This controls how much value the agent gives to future rewards. This value is between 0 and 1, the closer to 1 the higher value the agent places on future rewards and the closer to 0 the higher value the agent places on immediate rewards.

α = The alpha, which is the learning rate. This controls how fast the agent learns from changes in the environment. This value should be between 0 and 1.

The result of updating the Q-Table is that it gives the agent the optimal path to find rewards from every state within the environment, which in turn the agent can use to reach its goal by taking actions with the highest Q value.

3.2.3.1. Example Walkthrough 1

Let's walk through an example of how this would all work together in practice. We'll set our gamma to 1 and our alpha to 0.9.

Figure 7. States Diagram

🟥 0	🟦 1	2	🟩 3	4	5
6	🟦 7	8	🟦 9	10	11
12	13	🟦 14	15	🟦 16	17
18	19	20	21	22	23

In the agent's current state (0) it can only take one action, that's going down (3) which will move it to a new state (6)

$Q(0,3)$ is zero as that is the default Q value.

$r(0,3)$ is retrieved from looking at our rewards table. The value for taking action 3 in state 0 is zero in the rewards table.

$\max_{a} [Q(6,3)]$ is the maximum Q value that could be achieved by taking actions in the next state, in this case our next state would be state 6 as taking action 3 in state 0 would move us to state 6. Since only one action can be taken in state 6 – 3(Left) – we only need to use that Q value which is zero.

$Q(0,3) = (1 - 0.9) * 0 + 0.9(0 + 1 * 0)$

$Q(0,3) = 0 + 0.9 * 0$

$Q(0,3) = 0$

The outcome of this step is that the Q value at (0,3) stays at zero.

3.2.3.2. Example Walkthrough 2

In this example we'll look at the last step before finding the food. We'll set our gamma to 0.8 and our alpha to 0.9.

Figure 8. States Diagram

0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23

In the agent's current state (4) it can take two actions; go left to state 3 or go right to state 5. Let's assume that the agent has decided to go left – In practice this would be decided by the best possible reward or through random selection.

$Q(4,0)$ is zero.

$Q(4,0)$ is retrieved from looking at our rewards table. The value for taking action 0 in state 4 is ten in the rewards table.

$\overset{max}{a}\,[Q(3,0), Q(3,1)\,]$ is equal to the maximum Q value of all states reachable using actions from the next state (3). From state 3 we can take two actions (we can take actions that aren't -1 on the rewards table.). The Q values of these actions are as follows; $Q(3,0) = 0, Q(3,1) = 0$. The highest Q value is zero.

$Q(4,0) = 0 + \ 0.9(10 + 0.8 * 0)$

$Q(4,0) = 0 + \ 0.9 * 10$

$Q(4,0) = 9$

The outcome of this step is that the value at Q(4,1) is updated to 9.

3.3 Deep Reinforcement Learning

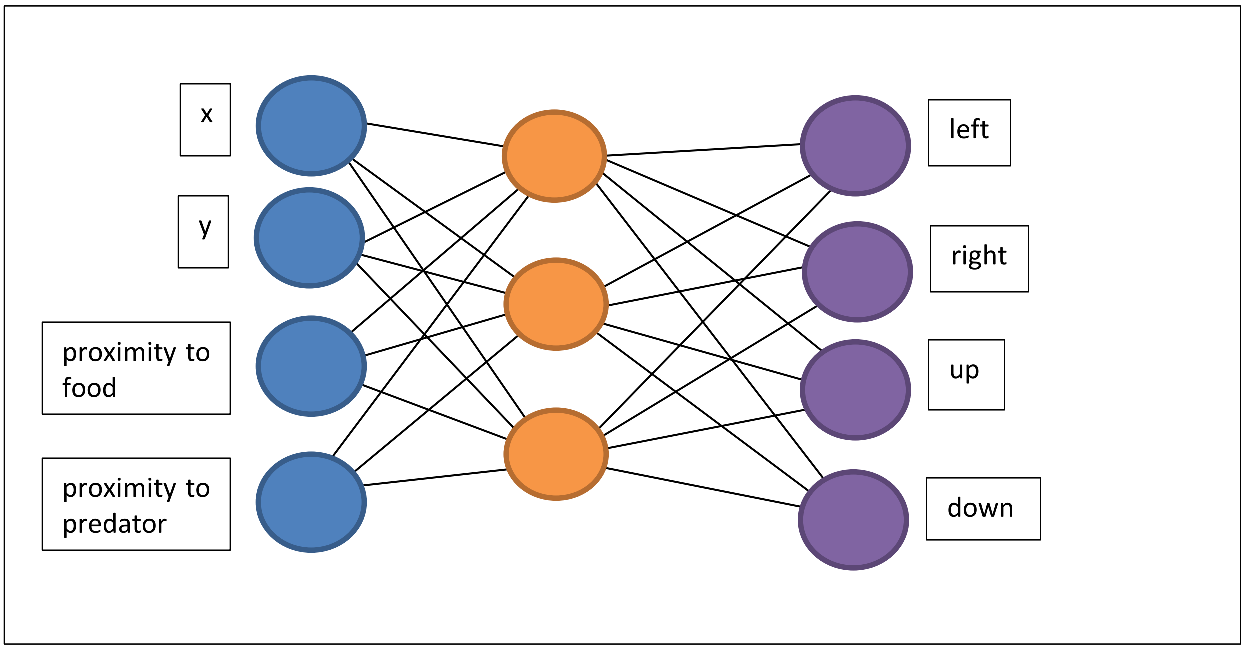


Figure 9. Neural Network Diagram

The above diagram shows an example of the neural network used instead of a q-table in deep reinforcement learning.

Instead of storing every possible state inside of the q-table we feed the current state data into the neural network and train it based on past state data and results.

The inputs for this neural network are all the components of the animal's state, in this example its x and y coordinates as well as its proximity to food/predators.

The neural networks output layer consists of the actions the animal can take, each will be given a value based on how likely the neural network thinks that the action will result in a reward.

3.4 Simulation World

The simulation world consists of a grid of tiles, these tiles can contain simulation entities such as flora, fauna or environmental objects.

3.4.1 Simulation Specification

- World
  - The world is represented in an abstract form for the purpose of this project.
  - The world can have any number of animals.
  - The world consists of a finite number of tiles.
  - Actions correspond to an integer number.
    - Left = 0
    - Right = 1
    - Up = 2

        - Down = 3
    - If using Q-Learning, each state is given a unique number, known as a state index. This is used to access information about the state in the q-table and reward table.
- Flora
    - The plants in the world have a limited number of “uses” before they become depleted, the prey animals must then look for new sources of food.
- Animals
    - A predator animal kills a prey animal if it occupies the same tile as the prey animal.
    - A prey animal acquires food if it occupies the same tile as a food source.
    - The animal’s main source of reward is acquiring food.
    - Animals of the same type can breed; this creates a new animal of that type which is a combination of its parents.
    - Animals can move into another tile adjacent to its current tile, except if:
        - It contains another animal, except for predators killing prey.
        - It contains an impassable obstacle.
    - If an animal moves to a tile and does not find food, it will receive a negative reward.
        - This helps to encourage the animals to explore new places to find food.

3.2.3.3. States

The states are generated at runtime, all possible states must be recorded. For example, an animal’s state could consist of its x coordinate, y coordinate, whether it can see a prey animal and whether it can see a predator animal. This could be calculated using:

$$x_{tiles} * y_{tiles} * 2$$

For example, if you had a tile grid which was twenty by twenty tiles you would have a total of 2800 states.

3.2.3.4. Table Structure

Each table stores a two-dimensional array of float values.

Q-Table

The Q-table as seen in the Q-Table section is a two-dimensional array. Each index in the array corresponds to a state index. Each value in the array stores a second array containing the q-values for the actions in that state. For example, if you wanted the q-value for taking the right action in state index five you would get the fourth array in the q-table then read the value from index one within that array (Q-Table[4][1]).

Result Table

The result table as seen in the Rewards Table section is a two-dimensional array as well. The reward table follows the exact same structure as the Q-table, except the reward table stores the reward value for taking a certain action in a state.

3.4.2 World Diagram

Filled green squares specify a plant food source.

Filled black squares specify an impassable obstacle.

Filled blue squares specify a prey animal.

Filled red squares specify a predator animal.

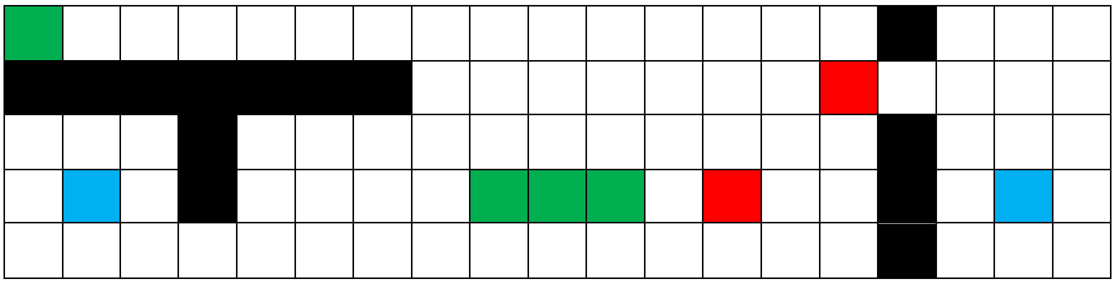


Figure 10. World Diagram

3.5 Chosen Technologies

3.5.1 Programming Language

Most programming languages could be used to complete this project, but two programming languages stood out due to their popularity in the machine learning space. These languages are Python and Java, they both have their pros and cons, listed below:

3.2.3.5. Python

Pros:

- I haven't tried Python before so I think it would be a great chance to learn a new programming language.
- Python is the most popular programming language for machine learning. [9]
- Python has a massive range of machine learning libraries such as TensorFlow, PyTorch and Scikit-learn.

Cons:

- Since I don't know Python, I'll have to take time to learn it.
- The amount of choice in libraries in Python may be overwhelming for a beginner.
- Python is an interpreted language so it will be slower than a compiled language.

3.2.3.6. Java

Pros:

- Javas execution is faster than Pythons due to it being compiled instead of interpreted.
- I could use the Weka machine learning tool which I've had some experience with.

Cons:

- Java is quite a verbose language when compared to others such as Python.
- Seems to have less documentation online compared to Python.

3.2.3.7. Decision & Justification

The Python programming language was chosen for this project. The main reason for this choice was the massive amount of high-quality neural network libraries it offered. As an added benefit it would allow me to learn Python and increase my skill set. Since Python is the most popular machine learning language it has more resources available online compared to Java when it comes to machine learning.

3.5.2 Neural Network Library

Two popular libraries for implementing neural networks include TensorFlow [10] and Keras [11]. Keras was chosen over TensorFlow as Keras is a high-level library that uses TensorFlow, providing a high-level abstraction which should be enough for the projects needs while being easier to understand.

3.6 Solving the Problem

3.2.3.8. Simulating many creatures at once

Simulating many creatures at once comes with the added problem of efficiency, to simulate a relatively large number of creatures each creature will have to be efficiently stored in memory and efficiently updated during each tick of the simulation. Currently the solution is to store all the states, actions and rewards for taking those actions in tables (Q-table and rewards table). Storing all states and actions for every creature will cause the memory requirements of the program to grow massively. A potential solution to this is to use a neural network which takes the state and action and produces the value of taking that action. The rewards for taking certain actions would still need to be stored as they would be used to train the neural network. This solution would take away the need to store the Q-table replacing it with a neural network for each creature.

3.2.3.9. Recording the creatures current state

The issue of state is one of the biggest problems with this project. The state of a creature can involve many things such as what is around it, for example other creatures or predators. The state would also have to record the position as well as the attributes that keep the animal alive such as food and water. The more information we store in the state the harder it becomes to solve our problem; this is the curse of dimensionality [12] which means that as you add more dimensions to your data – in this case; increasingly complex states - the harder it becomes to find patterns within it. This is solved by using neural networks as they can process data without falling prey to the curse of dimensionality, they do this by removing the irrelevant data and only focusing on the data that is relevant to the outcome. [13]

3.2.3.10. Varying the environment

There will have to be a suitable way for the environment to be defined within the simulation, this will be in the form of a config file, either in XML or JSON format. The file will contain creature definitions as well as a definition for the environment.

4 Conclusion

4.1 Summary

In conclusion, the project achieved an implementation for both deep reinforcement learning and Q-learning. Animals can use either of the implementations to succeed in their environment.

4.2 Evaluation

4.2.1 Language Choice

In the Programming Language section I discussed the choice of programming language for this project, I believe that my choice was the incorrect one, for multiple reasons;

- The dynamically typed nature of Python made it unsuitable for such a large project, even though I used type hints [14] to help alleviate the problem.
    - Without having static types, reading the code base as it got larger and keeping track of what values variables held was very challenging.
    - This problem became even worse when using the Keras library, it was very hard to tell what types of variable that methods wanted in their parameters. Especially when libraries referenced each other. For example, Keras uses arrays from the NumPy library.
- While Python has a large amount of machine learning libraries it has very little to choose from when it comes to displaying sprites to the user as Python has no inbuilt was to do this.
    - The only popular library for displaying sprites was the PyGame library which isn't a bad library by any means but is more complex to use than others when compared to inbuilt offerings from other languages such as Java.

Another unfortunate outcome of choosing python was the time I had to spend learning it as I hadn't had a lot of experience with it before this project. This accounted for quite a bit of lost time during the project which could have been avoided if I used a programming language that I had more experience in.

4.3 Results

4.3.1 Deep Learning

I couldn't get good results with my deep learning implementation due to the inefficiency of the code; a single tick of the simulation took around three seconds. Since I was using Keras to run my neural network I had little control over the speed of the training which was the main cause for such a long time to run a single tick. The extremely long time between ticks made it impossible to test this implementation.

4.3.2 Q-Learning

The Q-learning implementation had decent results, with the prey animals being able to find their way to a fixed food location, this stopped working as soon as the food began moving, this was due to the limited state of the animals as they only considered their x and y position.

This meant that the animals had nothing to rely on in order to find food other than wandering until they found it. This could be improved by adding a more complex state.

4.4 Future Work

4.4.1 Deep Learning

The deep learning section of my project has a good base but unfortunately it requires more work to be run in an efficient way, currently it trains the neural networks per simulation tick using the Keras library. I think that the project would benefit from someone with a better knowledge of the mathematics behind neural networks, they might be able to write a more efficient piece of code instead of using a library.

4.4.2 Q-Learning

The Q-learning section of the project was reasonably successful, the algorithm along with the reward and q tables have been implemented, unfortunately the algorithm doesn’t seem to allow the creatures to find their way to food consistently in its current implementation. I think this section of the project could be improved with more time to test the algorithm and figure out why it isn’t producing good results.

References

[1] Y. Chen and Y. Li, “Macro and Micro Reinforcement Learning for Playing Nine-ball Pool,” *IEEE Conference on Games,* 2019.

[2] F. Norén, “Under the hood of Pixling World,” Hackernoon, 14 10 2019. [Online]. Available: https://hackernoon.com/how-to-run-1m-neural-network-agents-at-60-steps-per-second-in-a-browser-183c6213156b.

[3] L. P. Kaelbling, M. L. Littman and A. W. Moore, “Reinforcement Learning: A Survey,” *Journal of Artificial Intelligence Research 4,* 1996.

[4] D. Silver, “Deep Reinforcement Learning,” Deepmind, 17 6 2016. [Online]. Available: https://deepmind.com/blog/article/deep-reinforcement-learning.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra and M. Riedmiller, “Playing Atari with Deep Reinforcement Learning,” *DeepMind Research,* 2013.

[6] S. Paul, “An introduction to Q-Learning: Reinforcement Learning,” FloydHub Blog, 15 4 2019. [Online]. Available: https://blog.floydhub.com/an-introduction-to-q-learning-reinforcement-learning/.

[7] Mnemosyne Studio, “A Painless Q-Learning Tutorial,” 2013. [Online]. Available: http://mnemstudio.org/path-finding-q-learning-tutorial.htm.

[8] C. J. Watkins and P. Dayan, “Q-Learning,” *Machine Learning,* vol. 8, no. 3-4, p. 279–292, 1992.

[9] T. Elliott, “The State of the Octoverse: machine learning,” 24 1 2019. [Online]. Available: https://github.blog/2019-01-24-the-state-of-the-octoverse-machine-learning/.

[10] TensorFlow, [Online]. Available: https://www.tensorflow.org/.

[11] Keras Team, [Online]. Available: https://keras.io/.

[12] C. Godbout, “What Killed the Curse of Dimensionality?,” Hackernoon, 14 10 2019. [Online]. Available: https://hackernoon.com/what-killed-the-curse-of-dimensionality-8dbfad265bbe.

[13] C. Bupe, “How does a deep neural network escape/resist the curse of dimensionality?,” Quora, 14 4 2017. [Online]. Available: https://www.quora.com/How-does-a-deep-neural-network-escape-resist-the-curse-of-dimensionality.

[14] “Python Documentation,” [Online]. Available: https://docs.python.org/3/library/typing.html.