Developing a Q-Learning Agent for Solving Grid World Environment Problems using Reinforcement Learning and Python

Omohodion Faith, *Department of Artificial Intelligence Ulster University*, Belfast Northern Ireland.

Abstract

Grid world environment provides a foundational approach to the application of reinforcement learning. This report illustrates the process of exploration and exploitation in solving grid world problems and further details the formulation of policy, environment, observation, and action in the context of grid world problem-solving. A learning agent is created to explore using the epsilon-greedy method for the grid world environment which is configured with a board layout of 5 by 5 and the borders are bounded by four possible actions, North, South, East, and West. The agent is faced with obstacles and rewarded at the JUMP and WIN state with +5 and +10 respectively, while other states give a -1. The agent is trained for 100 episodes to stop whenever the agent earns an average cumulative reward that is at least 10. The state values of each grid cell with the board layout are formulated and visualization is carried out at the execution of the code.

***Keywords*— Deep Learning, Q-Learning, Q-table, Grid World, Reinforcement Learning.**

I. INTRODUCTION

The concept of Deep Learning is best evaluated through the process of reinforcement learning, according to Leslie [13], "Work from statistics, psychology, neurology, and computer science all played a role in the creation of reinforcement learning in the beginnings of cybernetics. It is well known that it substantially increased interest in machine learning and artificial intelligence during the past five to ten years. Its promise includes giving away programming agents by reward and punishment without needing to define how the goal is to be performed". Reinforcement learning demonstrated by Florentin [14] is one of the kinds of artificial intelligence procedures where an agent engages with the environment to learn. The agent is self-taught from knowledge gained during its experience with the environment in the past; simply put, this is called exploitation. Exploration, which is essentially the best way to describe a trial-and-error learning process, is when the agent decides to behave based on prior experiences. Similarly, Yaakov [15] mentioned that supervised and unsupervised learning are two groups of learning problems that are addressed by machine learning research. Of these, reinforcement learning is perhaps best known as the common and realistic class of learning problems, for which a relatively developed theory and effective algorithms exist.

Junjie [15] demonstrated that, in contrast to conventional machine learning, reinforcement learning is a self-supervised training algorithm that enables agents to interact with their surroundings autonomously, observe, and gather feedback from the environment, on the other hand, they can train based on the results of the response and environmental feedback and optimize the action plan. In earlier reinforcement learning techniques based on the theory of optimal control, the sequential choice problem of reinforcement learning was referred to as an adaptive dynamic programming ADP problem. Similarly, Sergey [3] described the decision-making process in reinforcement learning as follows:

that the agent is given some initial environmental observation and is forced to select a course of action from the available options. As the environment reacts, it changes states and produces a reward signal (a scalar number), which is seen as a reliable assessment of the agent's effectiveness. The agent makes decisions on what to do based on observations, and the environment reacts by sending the next state and reward signals. The agent's sole objective is to maximize the total reward. Several important presumptions are already introduced in this presentation of the learning process model. First off, because the agent interacts with the environment sequentially, time and space are discrete. Furthermore, it is expected that the set includes some form of reward system as a controlled measure of achievement. This illustrates the reward concept, often known as the Reinforcement Learning hypothesis. Using this concept provides a distinction between supervised and unsupervised learning in reinforcement learning and traditional machine learning environments. Sergey went on to clarify that, in contrast to supervised and unsupervised settings, where optimal loss functions are typically created by engineers and are not included in the data, reinforcement learning is the only machine learning task that provides an explicit objective function (cumulative reward signal) to maximize. Sergey agreed that the ultimate presumption of the reinforcement learning paradigm is a Markovian property that specifies that transitions merely depend on the previous state and the most recently chosen action and are independent of all prior interaction history.

Reinforcement Learning issues are characterized by a sustained interaction between a learning agent and a dynamic, foreign, uncertain, and possibly hostile environment. This interaction is mathematically represented as a Markov Decision Process (MDP) [15]. According to Garrett [17], Discrete-time decision-making issues can be studied mathematically using Markov decision processes (MDPs). A Markov decision process' formal definition is given by the tuple (S, A, 0, T, r, H), where:

1. S is the state space, which contains all potential states the system could be in.
2. The agent's potential course of action when engaging with the system is included in action space A.
3. The starting state distribution is μ0.
4. T is transition dynamics.
5. R is the reward function.
6. Determining how much future rewards should be "discounted" while making decisions is the role of the discount factor which is denoted by γ
7. The horizon, or the maximum number of time steps in each episode, is denoted by the letter H. It could either be an integer with a positive sign (the finite-horizon scenario) or (the infinite-horizon case).

Reinforcement Learning Background: Exploration and Exploitation Processes

According to Leslie [13], reinforcement learning refers to the problem that arises when an agent has to learn behavior through mistakes in a dynamic environment. Although the work described here shares a strong family resemblance with a psychological work of the same name, there are significant differences in the specifics and how the word "reinforcement" is used. Leslie added that there are various ways in which the challenge of reinforcement learning differs from the more popularly researched issue of supervised learning. The absence of input/output pair presentations is the primary distinction. Instead, after deciding, the agent is informed of the immediate benefit and the outcome, but not of the action that would have been beneficial for it in the long run. To act optimally, the agent must actively accumulate useful experience regarding the potential system states, actions, transitions, and rewards.

Grid world problems are frequently solved through exploration and exploitation in reinforcement learning. When an agent chooses to investigate additional possibilities or act differently even after completing a job that either provided a reward or did not provide a reward, we can say that the agent is attempting to explore and better comprehend their surroundings. According to Rogers [1], the way exploration techniques strike a balance between the agent's "want" to perform well given its existing understanding of the environment and its need to learn more about it has a distinct value. In theory, an agent should become more educated to behave effectively and efficiently with the time it spends learning the more it investigates and learns from its surroundings.

[1] Intuitively, one could assume that the best learning strategy would be one of pure exploration. This is not the case, though. As so much time is spent investigating areas of the world that are unrelated to the work at hand, pure exploration can quickly consume computational resources and time. Given that it spends a disproportionate amount of time participating in actions that do not further its goals, this also implies that the agent's performance while learning will be below average. This is consistent with Melanie's [2] theory that one of the challenges in reinforcement learning is striking a balance between exploitation (seeking to gain new knowledge about the world by choosing a poor action) and exploration (using what we already know about the world to get the best results we know of). The key to a good learning capacity for an agent is the ability to recognize when to stop and when to keep going throughout the exploration process. [2] If the agent explores too much, it cannot stick to a path; it is not learning; it cannot use its information, and thus acts as though it is ignorant. To ensure that the agent is learning to make the best decisions, it is crucial to strike a balance between the two.

The major consideration for solving issues with minimal rewards is exploration approaches. The reward is uncommon in issues with scarce rewards; therefore, the agent won't frequently find the reward by acting arbitrarily. The issues in such a scenario's grid world are a fair illustration that, complex exploration methods must be developed since reinforcement learning struggles to learn the rewards and actions they are linked with [5]. According to Rogers [1], the agent's approach to exploration in addressing grid world problems differs. Both undirected and guided exploration strategies fall within this category.

Directed: During directed exploration, the agents take the ecosystem's past into account since it may affect which areas of the environment they will subsequently explore.

Undirected: In contrast, the agent randomly investigates the world using undirected exploration strategies without considering the learning process's previous history.

The sections below go over:
1. Undirected exploration
2. Guided exploration/directed exploration based on watching the learning procedure
3. Further directed strategies based on other learning models.

Undirected Exploration:

With Undirected Exploration, the fundamental characteristic is that no learning is considered, and actions are largely created randomly. Undirected methods frequently change the probability distribution that is used to select actions at random to address the trade-off between exploration and exploitation. In addition, Rogers noted that there are three methodologies used in the undirected exploration process: Boltzmann Distributed Exploration, Semi-Uniform Distributed Exploration, and Random Exploration.

Even though these techniques are frequently employed, it has been shown [16] that undirected exploration techniques are not as effective as directed exploration techniques. To succinctly illustrate these tactics: A method for exploring the state space is random exploration. It entails creating actions at random with a constant probability. If the cost of exploration while learning is not a concern, this method might be used. This strategy might be suitable, for instance, if the learning process is divided into performance and learning phases and the cost of learning is

overlooked. Semi-Uniform Distributed Exploration is an undirected exploration approach that first selects the generated actions using a probability distribution depending on the agent's current utility estimates before generating actions uniformly at random.
Boltzmann-Distributed Exploration, also known as semi-uniform exploration, Leslie [13] discovered that when deciding which action should receive the fixed probability P*best*, only takes the utility of the several courses of action into account. When the values of the actions are near, this strategy struggles but does well when the optimal action is sufficiently distanced from the others. If the temperature schedule isn't manually changed with great care, it could also converge too slowly.
[6] Throughout the game, there is an anticipated benefit for every action. Choosing the course of action with the largest expected payoff is all that remains of the problem if the agent is aware of the expected reward. To acquire a sense of the desirability of each action, though, is advised as the expected rewards for the states in the grid world are unknown. The Agent will have to do an exploration to determine the average of the rewards for each action. It can then utilize this knowledge to make a decision that will yield the greatest potential rewards (this is also called selecting a greedy action).

Exploitation occurs when an Agent uses what it already knows to its advantage by repeatedly taking activities that result in "favorable" long-term benefits. [7] In doing so, the Agent aims to make the most use of what is previously known.
[8] defines exploitation as a greedy strategy in which agents attempt to maximize rewards by relying on estimated value rather than true worth. To maximize their reward, agents in this technique utilize their present estimated worth and make the best decision possible depending on the facts at hand.
Knowing when to conduct exploration and when to halt and resume exploitation is the Agent's biggest challenge. Thomas [9] elaborated on leveraging the exploration and exploitation task process and the epsilon greedy algorithm to solve grid world problems. The procedure is described below:

1. To start, we define an exploration rate called "epsilon," which we initially set to 1. This is the number of steps we'll take at random. Since we don't know anything about the values in the Q-table at the beginning, this rate must be at its greatest level. This means that by picking our activities at random, we must conduct extensive research.
2. A random number is created. We shall engage in "exploitation" if this value exceeds epsilon (this means we use what we already know to select the best action at each step). If not, else an exploration is carried out.
3. The concept is that the Q-function training process must start with a large epsilon. Once the agent is more adept at estimating Q-values, gradually reduce it.

The exploration/exploitation trade-off is what we refer to when an agent understands how to strike a balance between exploitation and exploration.
[8] defined the greedy policy of epsilon as a method for preserving a balance between exploitation and exploration. Therefore, a straightforward technique to decide between exploration and exploitation is to choose at random. This can be accomplished by frequently picking exploitation while doing a little exploring.

Reinforcement Learning in Grid World Problems and Possible Approaches

[10] In the grid world, each state (position) of an agent must be justified for the reward to be distributed appropriately for each state. For the agent to finally learn policy, each action the state performs should have a function that allows it to accept the action and then send back the legal position of the following state. A policy is a simple set of instructions that tells the agent what to perform at each state in a mapping from state to action. A policy specifies what action(s) to conduct at each state based on the Garrett [17] concept. In our example, value iteration will be used to first learn a mapping from state to value (which is the estimated reward). Based on the estimates, at each stage, our agent will select the best action that delivers the highest estimated reward.
Like how Ashwin [18] demonstrated that these benefits are typically taken into account to be cumulative over time, as the policy specifies what must be done in each state, the reward gained at any given time is only a function of the agent's current condition. As a result, this configuration causes a Markov reward process (MRP) on the state space, in which the agent is unaware of the underlying transitions and rewards. Ashwin continued by saying that the agent only observes samples of state changes and rewards and that its objective is to estimate the MRP's value function. This is a problem known as a policy evaluation. Jeremey [10] noted that the agent will simply start with all rewards set to zero if it has no prior knowledge of the grid world (environment). After that, it begins to investigate the world by wandering around. It will undoubtedly encounter many failures at first, but that is okay because those mistakes are a means for it to learn more about the surroundings. The estimated value of all states along the route will be changed depending on the formula below once it reaches the finish of the game, either reward +1 or reward -1. The agent [11] learns to avoid restricted cells that stand as obstacles from numerous observations of the surroundings. As the game progresses the agent figures out that movement to a certain cell is restricted.
[12] Each State s in the grid world indicates the agent's position. A step is taken by the agent in each of the four geographical directions. To represent a policy on a map, the following symbols are employed:

N for the action NORTH
S for the action SOUTH
W for the action WEST
E for the action EAST

To obtain a full policy, unknown symbols are assigned a NONE action. To make the policy file easier to understand, it nevertheless has walls and a target cell. Two aims served as the foundation for the policy's creation:

1. The agent must be capable of achieving the objective. If a policy lacks this quality, the goal's reward will never be attained, making it impossible for policy evaluation to yield a reasonable result.

2. The strategy must be less than ideal. As a result, there are some situations in which the agent does not choose the direct route to the destination. We can observe the results of algorithms that attempt to enhance the original policy using such a strategy.

Matthias [12] further mentioned that the formulation of agent actions takes into consideration the transition function with two types of actions in view, which are illegal actions and legal actions: In the grid world, an action can be prohibited in one of two ways:

1. If performing the task would force the agent to go out of the grid.
2. If the agent would crash into a wall.

This gives us the rules for the transition function:
An agent's action is said to be unlawful when it should be prohibited from changing from one state to another.
Only movements that would move the agent to a neighboring cell are permitted for lawful acts, so if an agent should continue in its current state, the action is unlawful.

2. DESIGN AND METHODOLOGY FOR SOLVING THE CONFIGURED GRID WORLD PROBLEM

The grid world problem is configured to specific rules which can best be illustrated with the diagram below.

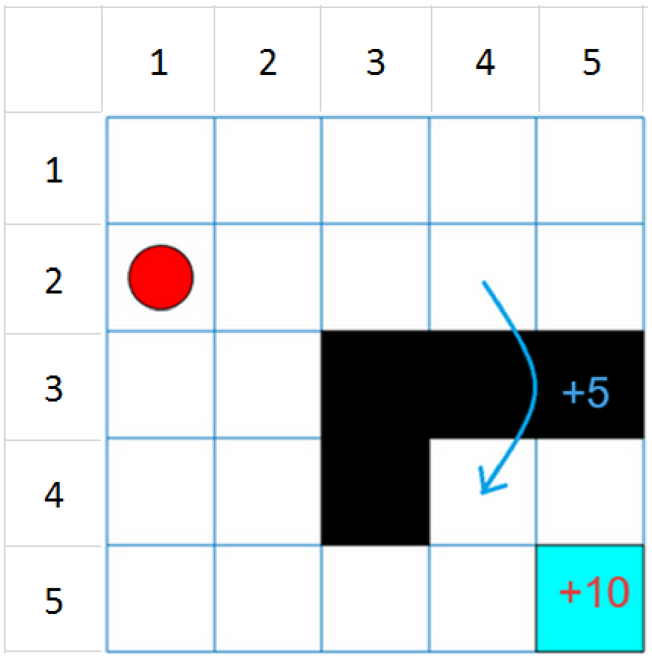
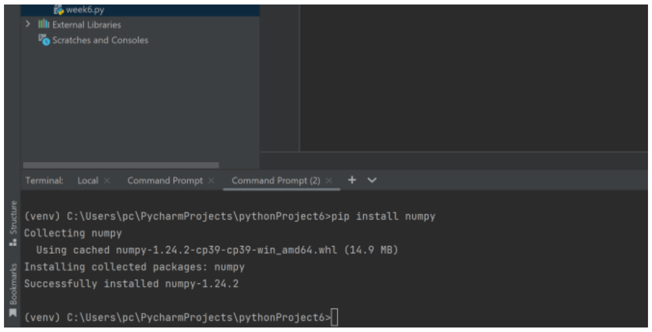


Fig. 1 Mathworks.com, 2023. https://www.mathworks.com/help/examples/rl/win64/BasicGridWorldExample_01.png (accessed Mar. 09, 2023).

According to Fig. 1, the grid world environment is configured with a board layout of 5 by 5 and the borders are bounded by four possible actions, North, South, East, and West. The agent is to start from the second row, first column [2,1], and the agent collects a reward of +10 if it gets to the terminal state also known as the win state which is the fifth row, fifth column [5,5] (blue). The grid world environment also consists of a special jump from the cell second-row, fourth column [2,4] to the cell fourth row, fourth column [4,4] with a reward of +5, and finally, the agent is blocked by obstacles notified with the black cells, all other actions result in a -1 reward.

Sebastian [19] argues that during a succession of discrete-time steps 0, 1, 2, 3, etc., the MDP Agent and Environment may interact at any time. At each time step, the environment delivers the agent a state corresponding to its current condition (MP). The agent selects an action based on the state s, earns a reward r (MRP) for the action and the resulting change in the state's condition, and then ends the MDP. In the mind to make maximum rewards, the agent learns to observe the environment and exploit the best possible ways to move through the states.

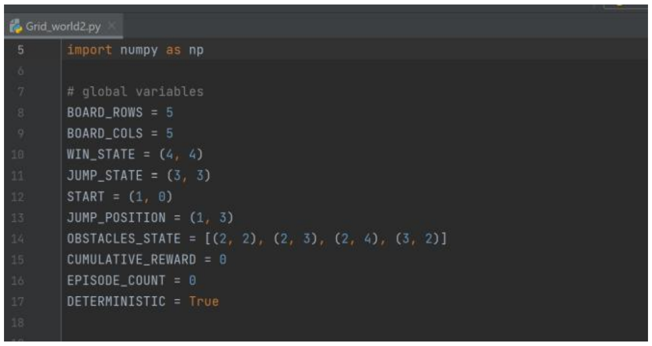
In training the Q-learning agent we use the programming language python to construct the algorithm in solving the grid world problem, firstly we install our package at the terminal command prompt we pip install numpy, then move to the python IDE to import numpy as np and define your variables following the configuration of the grid world environment. Find below Fig 2a and Fig 2b for illustration.



```
(venv) C:\Users\pc\PycharmProjects\pythonProject6>pip install numpy
Collecting numpy
  Using cached numpy-1.24.2-cp39-cp39-win_amd64.whl (14.9 MB)
Installing collected packages: numpy
Successfully installed numpy-1.24.2

(venv) C:\Users\pc\PycharmProjects\pythonProject6>
```

Fig 2a

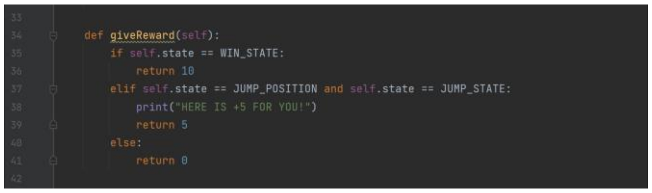


```python
import numpy as np

# global variables
BOARD_ROWS = 5
BOARD_COLS = 5
WIN_STATE = (4, 4)
JUMP_STATE = (3, 3)
START = (1, 0)
JUMP_POSITION = (1, 3)
OBSTACLES_STATE = [(2, 2), (2, 3), (2, 4), (3, 2)]
CUMULATIVE_REWARD = 0
EPISODE_COUNT = 0
DETERMINISTIC = True
```

Fig.2b

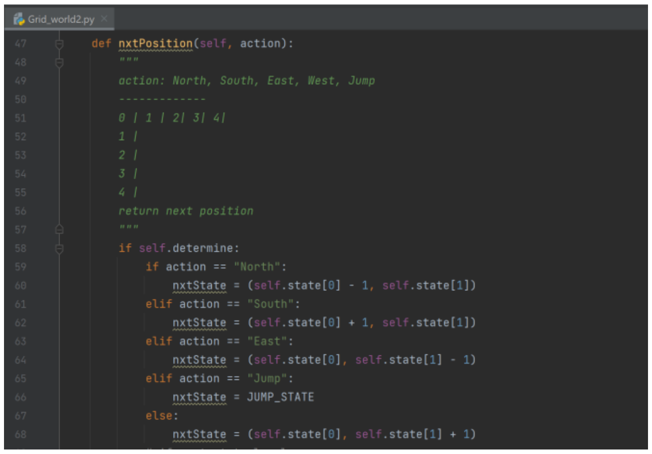
For a grid world environment, the win state, lose state, or Jump state if applicable needs to be formulated for a particular position/state and the reward for achieving that position needs to be defined. Find below Fig 3 for illustration.



```python
    def giveReward(self):
        if self.state == WIN_STATE:
            return 10
        elif self.state == JUMP_POSITION and self.state == JUMP_STATE:
            print("HERE IS +5 FOR YOU!")
            return 5
        else:
            return 0
```

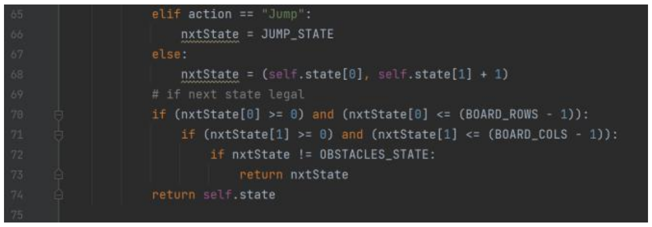
Fig 3

For every action taken by the agent, a function is defined that chooses to accept that action and return to a legal position considered as the next state for the agent to take. Find below Fig 4 for illustration, also the board layout following the configured grid world environment is specified.



```python
    def nxtPosition(self, action):
        """
        action: North, South, East, West, Jump
        -------------
        0 | 1 | 2| 3| 4|
        1 |
        2 |
        3 |
        4 |
        return next position
        """
        if self.determine:
            if action == "North":
                nxtState = (self.state[0] - 1, self.state[1])
            elif action == "South":
                nxtState = (self.state[0] + 1, self.state[1])
            elif action == "East":
                nxtState = (self.state[0], self.state[1] - 1)
            elif action == "Jump":
                nxtState = JUMP_STATE
            else:
                nxtState = (self.state[0], self.state[1] + 1)
```

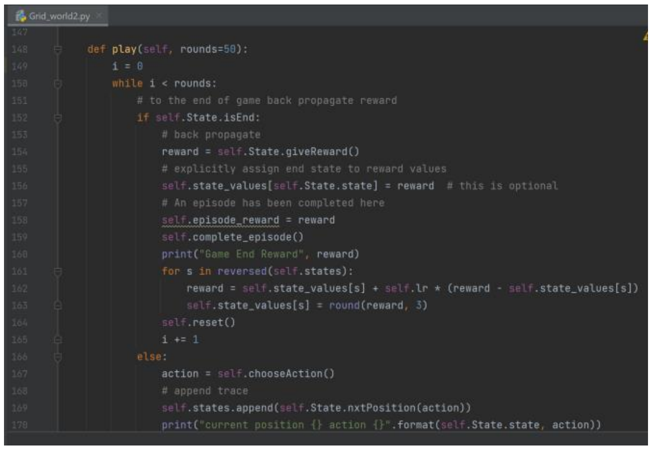
Fig 4a



```python
            elif action == "Jump":
                nxtState = JUMP_STATE
            else:
                nxtState = (self.state[0], self.state[1] + 1)
            # if next state legal
            if (nxtState[0] >= 0) and (nxtState[0] <= (BOARD_ROWS - 1)):
                if (nxtState[1] >= 0) and (nxtState[1] <= (BOARD_COLS - 1)):
                    if nxtState != OBSTACLES_STATE:
                        return nxtState
            return self.state
```

Fig 4b

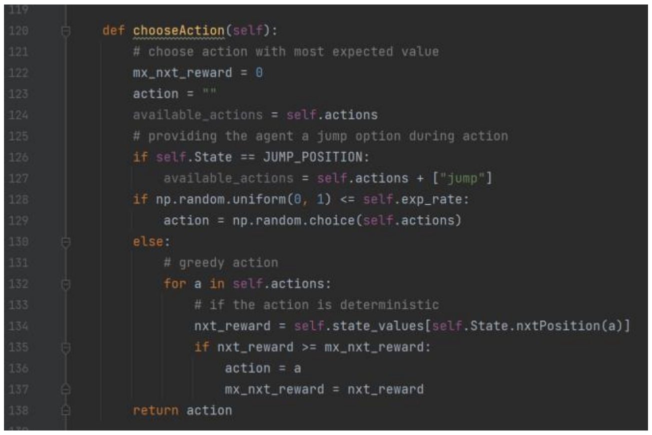
As our agent initially does not know the grid world (environment), it simply initializes all rewards to zero. Then it begins to explore the world by wandering around; undoubtedly, this will result in many early failures, but that is entirely acceptable. The game resets once it reaches the end of the game and the reward propagates backward. This process of learning is formulated in the algorithm illustrated in Fig 5. Also, as the name suggests, the value iteration updates its value (expected reward) at every iteration (end of the game).



```python
    def play(self, rounds=50):
        i = 0
        while i < rounds:
            # to the end of game back propagate reward
            if self.State.isEnd:
                # back propagate
                reward = self.State.giveReward()
                # explicitly assign end state to reward values
                self.state_values[self.State.state] = reward  # this is optional
                # An episode has been completed here
                self.episode_reward = reward
                self.complete_episode()
                print("Game End Reward", reward)
                for s in reversed(self.states):
                    reward = self.state_values[s] + self.lr * (reward - self.state_values[s])
                    self.state_values[s] = round(reward, 3)
                self.reset()
                i += 1
            else:
                action = self.chooseAction()
                # append trace
                self.states.append(self.State.nxtPosition(action))
                print("current position {} action {}".format(self.State.state, action))
```

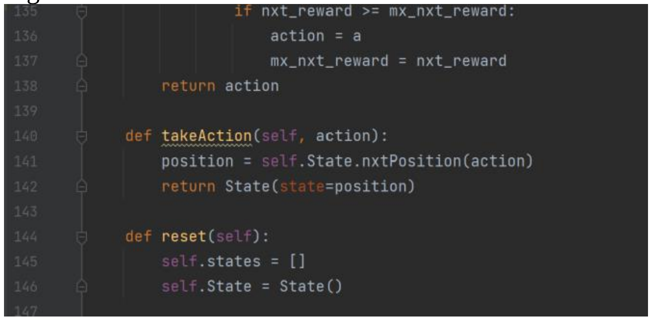
Fig 5

Sebastian [19] also mentioned that the Markov Random Process (MRP) has a scoring system to display how much reward has accrued over a given sequence. The agent now gets paid for every transition from one state to another. Return G is defined as the total of Rewards (R) across a sequence. Based on the findings of Sebastian we can assert that the agent would choose to either explore or exploit the environment to gain more rewards, this decision is left for the agent to take control of with regards to the epsilon-greedy policy so a balance can be maintained. This is illustrated in Fig 6a and Fig 6b below.



```
    def chooseAction(self):
        # choose action with most expected value
        mx_nxt_reward = 0
        action = ""
        available_actions = self.actions
        # providing the agent a jump option during action
        if self.State == JUMP_POSITION:
            available_actions = self.actions + ["jump"]
        if np.random.uniform(0, 1) <= self.exp_rate:
            action = np.random.choice(self.actions)
        else:
            # greedy action
            for a in self.actions:
                # if the action is deterministic
                nxt_reward = self.state_values[self.State.nxtPosition(a)]
                if nxt_reward >= mx_nxt_reward:
                    action = a
                    mx_nxt_reward = nxt_reward
        return action
```

Fig 6a



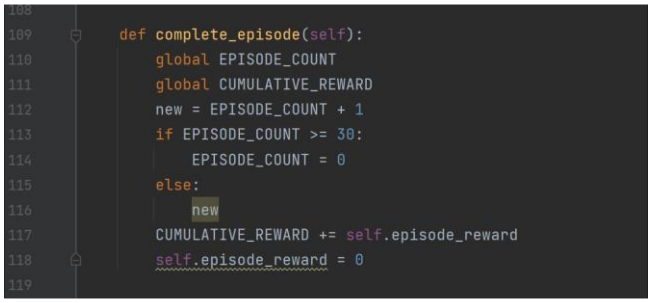
```
                if nxt_reward >= mx_nxt_reward:
                    action = a
                    mx_nxt_reward = nxt_reward
        return action

    def takeAction(self, action):
        position = self.State.nxtPosition(action)
        return State(state=position)

    def reset(self):
        self.states = []
        self.State = State()
```

Fig 6b

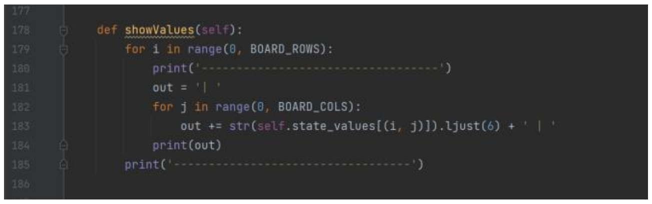
We then train the agent for 100 episodes, where each episode should have the ability to control unlimited.
paths from starting position to the terminal or jump-end position, when the agent earns an average cumulative reward of more than or equal to 30, we then discontinue training. This is illustrated in Fig 7 below.



```
    def complete_episode(self):
        global EPISODE_COUNT
        global CUMULATIVE_REWARD
        new = EPISODE_COUNT + 1
        if EPISODE_COUNT >= 30:
            EPISODE_COUNT = 0
        else:
            new
        CUMULATIVE_REWARD += self.episode_reward
        self.episode_reward = 0
```

Fig 7

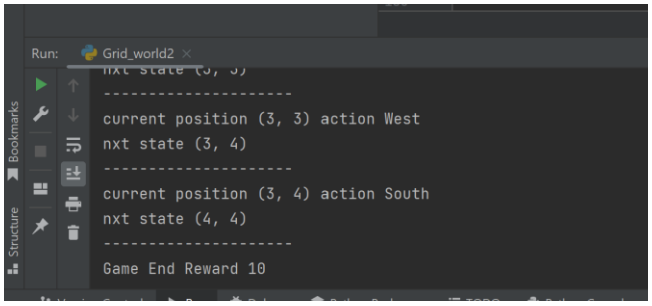
Finally, we visualize the state values in each of the grid cells with the board layout. This is defined with the function showValues, illustrated in Fig 8 below.



```
    def showValues(self):
        for i in range(0, BOARD_ROWS):
            print('----------------------------------')
            out = '| '
            for j in range(0, BOARD_COLS):
                out += str(self.state_values[(i, j)]).ljust(6) + ' | '
            print(out)
        print('----------------------------------')
```

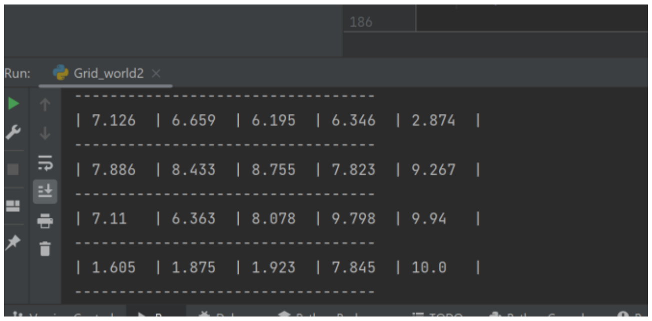
Fig 8

The agent was trained up to 4 times before it could best explore/exploit the environment in the best way to achieve a cumulative average reward. The visualization of the state values is shown in Fig 9a and Fig 9b, whereby the agent obtained the winning reward in the last state cell.



```
nxt state (3, 3)
--------------------
current position (3, 3) action West
nxt state (3, 4)
--------------------
current position (3, 4) action South
nxt state (4, 4)
--------------------
Game End Reward 10
```

Fig 9a



```
----------------------------------
| 7.126  | 6.659  | 6.195  | 6.346  | 2.874  |
----------------------------------
| 7.886  | 8.433  | 8.755  | 7.823  | 9.267  |
----------------------------------
| 7.11   | 6.363  | 8.078  | 9.798  | 9.94   |
----------------------------------
| 1.605  | 1.875  | 1.923  | 7.845  | 10.0   |
----------------------------------
```

Fig 9b

3. CONCLUSION

Reinforcement learning in general has proven to be a well-embraced part of the learning process of a machine, as findings show that an agent can be trained and conditioned to a particular learning environment, also based on Junjie's [15] idea an agent can be trained based on the reward value of the action and environmental feedback it gets and with that it will then optimize the action strategy on next action. However, there has been concern about how the agent can best balance its exploration and exploitation process so as not to waste more time and resources but based on studies [8] showed that epsilon's greedy policy as a technique can maintain a balance between exploitation and exploration, that technique has brought about an improvement in Q-Learning.

4. REFERENCES

[1] M. Roger, "A survey of exploration strategies in reinforcement learning," 2018, Available: R McFarlane - McGill University, 2018 - researchgate.net.
[2] M. Coggan, "Exploration and exploitation in reinforcement learning.," McGill University, Montreal, Canada, 2004. Available: https://neuro.bstu.by/ai/To-dom/My_research/Papers-2.1-done/RL/0/FinalReport.pdf
[3] S. Ivanov and A. D'yakonov, "Modern Deep Reinforcement Learning Algorithms," *arXiv:1906.10025 [cs, stat]*, Jul. 2019, Available: https://arxiv.org/abs/1906.10025
[4] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement Learning: A Survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, May 1996, doi: https://doi.org/10.1613/jair.301.
[5] P. Ladosz, L. Weng, M. Kim, and H. Oh, "Exploration in deep reinforcement learning: A survey," *Information Fusion*, Mar. 2022, doi: https://doi.org/10.1016/j.inffus.2022.03.003.
[6] E. Bisong, "The Exploration-Exploitation Trade-off," *dvdbisong.github.io*. https://ekababisong.org/the-exploration-exploitation-trade-off/#:~:text=Exploration%20is%20when%20an%20Agent
[7] G. Dam and K. Körding, "Exploration and Exploitation During Sequential Search," *Cognitive Science*, vol. 33, no. 3, pp. 530–541, May 2009, doi: https://doi.org/10.1111/j.1551-6709.2009.01021.x.
[8] "Exploitation and Exploration in Machine Learning - Javatpoint," *www.javatpoint.com*. https://www.javatpoint.com/exploitation-and-exploration-in-machine-learning#:~:text=in%20technical%20terms- (accessed Feb. 27, 2023).
[9] T. Simonini, "Diving deeper into Reinforcement Learning with Q-Learning," *We've moved to freeCodeCamp.org/news*, Jun. 10, 2022. https://medium.com/free-code-camp/diving-deeper-into-reinforcement-learning-with-q-learning-c18d0db58efe
[10] J. Zhang, "Reinforcement Learning — Implement Grid World From Scratch," *Medium*, Aug. 24, 2019. https://towardsdatascience.com/reinforcement-learning-implement-grid-world-from-scratch-c5963765ebff
[11] H. Prabhu, "Applying Reinforcement Learning Algorithms to solve Grid World Problems," *MLearning.ai*, Sep. 30, 2022. https://medium.com/mlearning-ai/applying-reinforcement-learning-algorithms-to-solve-gridworld-problems-29998406dd75 (accessed Feb. 28, 2023).
[12] "Navigating in Grid world using Policy and Value Iteration," *www.datascienceblog.net*, Jan. 10, 2020. https://www.datascienceblog.net/post/reinforcement-learning/mdps_dynamic_programming/ (accessed Feb. 28, 2023).
[13] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement Learning: A Survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, May 1996, doi: https://doi.org/10.1613/jair.301.
[14] F. Woergoetter and B. Porr, "Reinforcement learning," *Scholarpedia*, vol. 3, no. 3, p. 1448, 2008, doi: https://doi.org/10.4249/scholarpedia.1448.
[15] J. Zhang, C. Zhang, and W.-C. Chien, "Overview of Deep Reinforcement Learning Improvements and Applications," *Journal of Internet Technology*, vol. 22, no. 2, pp. 239–255, Mar. 2021, Accessed: Mar. 11, 2023. [Online]. Available: https://jit.ndhu.edu.tw/article/view/2485/2501
[16] S. Thrun, "THE ROLE OF EXPLORATION IN LEARNING CONTROL." Accessed: Mar. 11, 2023. [Online]. Available: https://www.ri.cmu.edu/pub_files/pub1/thrun_sebastian_1992_3/thrun_sebastian_1992_3.pdf
[17] L. C. Thomas, "Markov Decision Processes," *Journal of the Operational Research Society*, vol. 46, no. 6, pp. 792–793, Jun. 1995, doi: https://doi.org/10.1057/jors.1995.110.
[18] A. Pananjady and M. Wainwright, "Value function estimation in Markov reward processes: Instance-dependent ∞ -bounds for policy evaluation," 2019. Available: https://people.eecs.berkeley.edu/~wainwrig/BibPapers/PanWai19.pdf
[19] S. Dittert, "Reinforcement Learning and the Markov Decision Process," *Analytics Vidhya*, Apr. 11, 2020. https://medium.com/analytics-vidhya/reinforcement-learning-and-the-markov-decision-process-f0a8e65f2b0f#:~:text=A%20Markov%20Reward%20Process%20(MRP (accessed Mar. 11, 2023)