Introduction

Reinforcement learning is a machine learning training method where desired behavior is rewarded while undesirable behavior is penalized. These are called positive and negative reinforcement respectively. A reinforcement learning agent is capable of observing and interpreting its environment, acting then receiving a reward or penalty based on its action. A reinforcement learning agent is said to learn through trial and error.

In this paper, we will be focusing on the topic of positive reinforcement by using the game Brick Breaker as the environment. In Brick Breaker, a player bounces a ball to destroy bricks by moving a paddle left and right. The player loses if the ball leaves the screen and wins if all bricks are destroyed.

Q-Learning is the reinforcement algorithm utilized in this paper. The foundation of the code was written by Matthew Chan and is available at medium.com. The code has been modified to work with the game.

This paper aims to study the effects of different improvements of Q-Learning on the agent performance. Combinations of these improvements will then be tested to determine the best combination and get the best possible results. The effects of varying the game settings on the agent performance will also be studied and documented.

Genetic algorithm will also be employed in this paper and is used to determine the hyperparameters of the Q-Learning agent. Different techniques such as single/double crossover and tournament/roulette selection are used to ensure more variety during the evolution process to produce better results.

A GUI was also created for users to more easily change the settings of the project and streamline the process of hyperparameter tuning and data collection.

Literature Review

The main innovation in this paper is to combine existing improvements of the Q-Learning agent to produce a new, better solution. In the end, 7 research papers of interest were identified and listed below.

Firstly, **genetic algorithm** was used by Wicaksono, A. S., & Supianto, A. A. for hyperparameter optimization of the machine learning methods used in online news popularity prediction.

Secondly, H.R. Tizhoosh has introduced **opposition-based learning** as a new scheme for machine intelligence.

Thirdly, the **granularity of the state space** was shown to affect the results by Jacopo Fior and Luca Cagliero in their study of machine learning-based stock trading.

Furthermore, Michal Gregor and Juraj Spalek have also done research on optimistic exploration value functions. This has been adapted in this project as the **random initialization of the Q-tables**.

Moreover, research into action elimination with deep reinforcement learning has also been done by Tom Zahavy, et al. This has been adapted to varying the **size of the action space** in the project.

Apart from that, research into **different reward functions** on the training performance of a Double DQN has also been done by Stefan Šćepanović.

Finally, Double Q-Learning was introduced by Hado van Hasselt which lacks the overestimation bias of the Q-Learning algorithm. This has adapted to **N-tuple Q-Learning** which is another innovation in this paper where N Q-tables were created for each reinforcement learning agent.

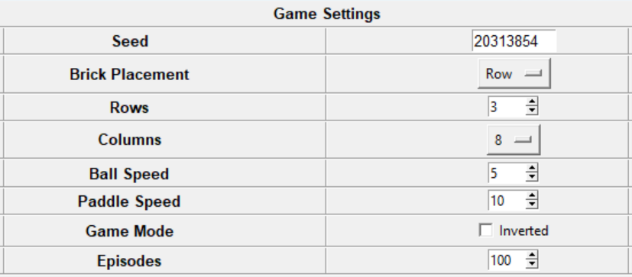
Thus, we can study the effects of combining multiple improvements and ascertain the best combination with the best results.

Graphical User Interface (GUI)

As mentioned above, users can easily change the settings of the project using the GUI created. The different settings of the GUI will be explained below.

## Game Settings

The game settings allow users to customize the different aspects of the game. The different options available to the users are shown below in Figure 1.



***Figure 1: Game Settings***

* **Seed**

|  |  |
| --- | --- |
| Range | 0 – 4294967295 |
| Default | 20313854 |
| Description | Integer used to initialize the pseudorandom number generator to ensure that results are reproducible |

* **Brick Placement**

|  |  |
| --- | --- |
| Options | Row, Column, Random |
| Default | Row |
| Description | Controls how the 3 brick types are arranged in game.  **Row:** by row  **Column:** by column  **Random:** random |

* **Rows**

|  |  |
| --- | --- |
| Range | 1 – 10 |
| Default | 3 |
| Description | Controls the number of brick rows in the game. |

* **Columns**

|  |  |
| --- | --- |
| Options | 1,2,3,4,5,6,8,10,12,15 |
| Default | 8 |
| Description | Controls the number of bricks in each row in the game. Can only be a factor of 600 up to 15. |

* **Ball Speed**

|  |  |
| --- | --- |
| Range | 1 – 10 |
| Default | 5 |
| Description | Controls the speed of the ball in the game. |

* **Paddle Speed**

|  |  |
| --- | --- |
| Range | 5 – 15 |
| Default | 10 |
| Description | Controls the speed of the paddle in the game. |

* **Game Mode**

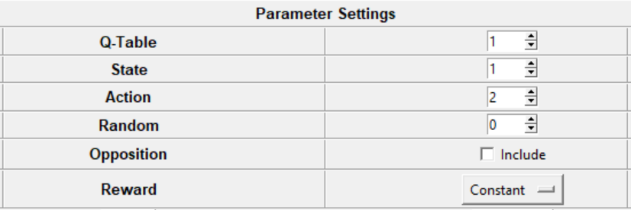
|  |  |
| --- | --- |
| Options | False, True |
| Default | False |
| Description | Allows inversion of the position of the bricks and paddle in the game. |

* **Episodes**

|  |  |
| --- | --- |
| Range | 1 – 200 |
| Default | 100 |
| Description | Controls the number of failed episodes before the game is reset and recorded as a failed run. |

## Parameter Settings

The parameter settings allow users to control the different settings of the Q-Learning agent. The different options available are shown below in Figure 2.



***Figure 2: Parameter Settings***

* **Q-Table**

|  |  |
| --- | --- |
| Range | 1 – 5 |
| Default | 1 |
| Description | Controls the number of Q-tables generated for each learning agent. |

* **State**

|  |  |
| --- | --- |
| Range | 1 – 10 |
| Default | 1 |
| Description | Controls the granularity of the state space of the Q-tables. State is assigned based on distance between the ball and the paddle. |

* **Action**

|  |  |
| --- | --- |
| Options | 2,3 |
| Default | 2 |
| Description | Controls the size of the action space. The 2 default actions are for the paddle to move left and right. The 3rd action is to do nothing. |

* **Random**

|  |  |
| --- | --- |
| Options | None, Normal, Uniform |
| Default | None |
| Description | Controls the initial values of the Q-tables.  **None:** 0s  **Normal:** -10 – 10  (Normal distribution)  **Uniform:** -10 – 10  (Uniform distribution) |

* **Opposition**

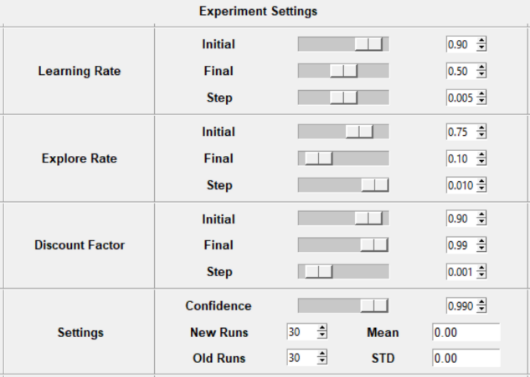
|  |  |
| --- | --- |
| Options | False, True |
| Default | False |
| Description | Allows the agent to also take the opposite action and update the Q-table. The do-nothing action has no opposite action and is only updated once. |

* **Reward**

|  |  |
| --- | --- |
| Options | Constant, Time-Based,  X-Distance,  X-Distance (Center),  XY-Distance |
| Default | Constant |
| Description | Controls the reward function used by the agent. **Constant:** 1 if run not ended else 0  **Time-Based:** (turn / 10) if run not ended else 0  **X-Distance:** 6 – distance (ball.mid.x, paddle.mid.x)  **X-Distance (Center):**  6 – minimum distance (ball.mid.x, paddle.x)  **XY-Distance:** 7.5 – dist (ball.mid, paddle.mid) |

## Experiment Settings

The experiment settings allow users to control the process of comparing different parameter settings. It is accessed by pressing the experiment button. The options available are shown below in Figure 3.



***Figure 3: Experiment Settings***

* **Learning Rate**

|  |  |
| --- | --- |
| Range | 0.00 – 1.00 / 0.000 – 0.010 |
| Default | 0.90 / 0.10 / 0.010 |
| Description | Controls how much the new learned value is used. |

* **Explore Rate**

|  |  |
| --- | --- |
| Range | 0.00 – 1.00 / 0.000 – 0.010 |
| Default | 0.50 / 0.01 / 0.010 |
| Description | Controls the chance that a random action is taken instead of the best action. |

* **Discount Factor**

|  |  |
| --- | --- |
| Range | 0.00 – 1.00 / 0.000 – 0.010 |
| Default | 0.90 / 0.99 / 0.001 |
| Description | Determines the importance of future rewards. |

* **Confidence**

|  |  |
| --- | --- |
| Range | 0.500 – 0.999 |
| Default | 0.990 |
| Description | Controls the confidence level of the t-test used to evaluate the results of the different experiments. |

* **New Runs**

|  |  |
| --- | --- |
| Range | 30 – 100 |
| Default | 30 |
| Description | Controls the number of successful runs needed for results collection. |

* **Old Runs**

|  |  |
| --- | --- |
| Range | 30 – 100 |
| Default | 30 |
| Description | Controls the size of the old sample to be compared. |

* **Mean**

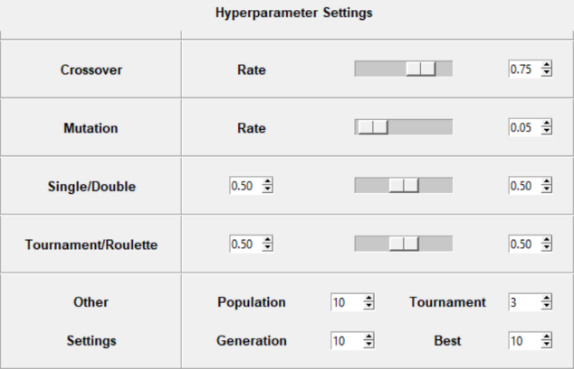
|  |  |
| --- | --- |
| Range | 1.00 – 200.00 |
| Default | 45.63 |
| Description | Controls the mean of the old sample to be compared. |

* **STD**

|  |  |
| --- | --- |
| Range | 0.00 – 140.71 |
| Default | 1.40 |
| Description | Controls the standard deviation of the old sample to be compared. |

## Hyperparameter Settings

The hyperparameter settings allow users to control the process of hyperparameter tuning of the agent. It is accessed by pressing the tuning button. The options available are shown below in Figure 4.



***Figure 4: Hyperparameter Settings***

* **Crossover Rate**

|  |  |
| --- | --- |
| Range | 0.00 – 1.00 |
| Default | 0.75 |
| Description | Controls the percentage of chromosomes that undergo the process of crossover. |

* **Mutation Rate**

|  |  |
| --- | --- |
| Range | 0.00 – 1.00 |
| Default | 0.05 |
| Description | Controls the percentage of genes that undergo the process of mutation. |

* **Single / Double**

|  |  |
| --- | --- |
| Range | 0.00 – 1.00 |
| Default | 0.50 / 0.50 |
| Description | Controls the percentage of single and double crossover when successful crossover occurs. |

* **Tournament / Roulette**

|  |  |
| --- | --- |
| Range | 0.00 – 1.00 |
| Default | 0.50 / 0.50 |
| Description | Controls the percentage of tournament and roulette selection for the selection of the next generation. |

* **Population**

|  |  |
| --- | --- |
| Range | 10 – 100 (±2) |
| Default | 10 |
| Description | Controls the population size of each generation. |

* **Elite**

|  |  |
| --- | --- |
| Range | 0 – 10 (±2) |
| Default | 2 |
| Description | Controls the number of fittest chromosomes that are guaranteed to survive. |

* **Generation**

|  |  |
| --- | --- |
| Range | 1 – 100 |
| Default | 50 |
| Description | Controls the number of iterations of genetic algorithm to be run. |

* **Best**

|  |  |
| --- | --- |
| Range | 1 – 10 |
| Default | 10 |
| Description | Controls the number of fittest chromosomes saved and displayed at the end. |

Experiment

The experiments to be run using the GUI are listed below. The results of the default settings are used as the baseline for comparison. The experiments are run for at least 30 times for the Central Limit Theorem to hold and to ensure statistical significance of the results.

## Game Settings

Firstly, each **game** setting is varied individually and the results recorded to study their effects and determine the significance of each game setting to the results of the reinforcement learning agent.

## Parameter Settings

Secondly, each **parameter** setting is varied individually and the results recorded to study their effects and determine the significance of each parameter setting to the results of the reinforcement learning agent.

Different parameter settings are then combined to determine which combination produces the best results before moving onto the final step in the experiment.

## Hyperparameter Tuning

The genetic algorithm is then run using the best parameter settings combination found above. It is run for 50 generations with a population size of 10 with the default settings. The fittest 10 chromosomes are recorded and shown at the end.

The fitness value of each chromosome is determined by running the experiment only once for the sake of time. To reduce the overestimation bias, if the same chromosome is evaluated more than once, the worst result is taken as the fitness value.

Finally, the fittest chromosomes are then combined with the best parameter settings combination. The results are recorded to determine the best hyperparameter settings which produces the best results.

Results

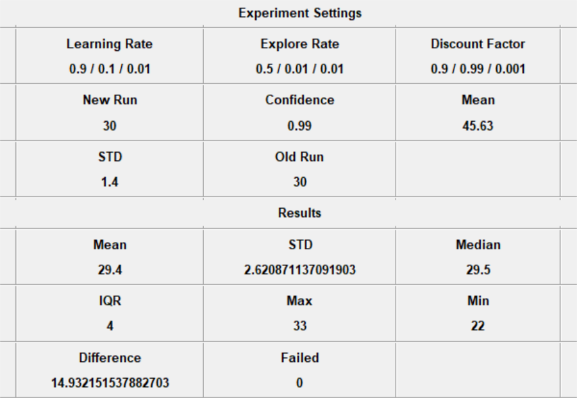
Metrics such as the mean, median, STD, IQR, max, min, difference and failed are recorded. Only relevant metrics are shown but the full list can be seen in the appendix.

The most important metric is the difference. It records the expected difference in mean between the 2 samples. It is calculated using the mean and STD with a t-test with 99% confidence level. A higher difference means the new sample is objectively better.

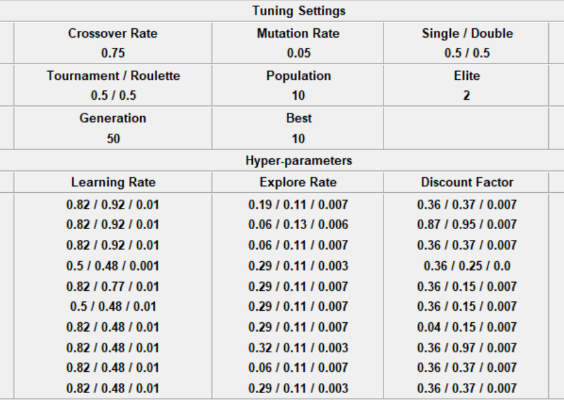
Examples of the result screens are given below in Figure 5, 6 and 7.



***Figure 5: Initial Settings***



***Figure 6: Experiment Results***



***Figure 7: Hyperparameter Results***

## Game Settings

The results of varying the game settings are shown below. The difference is not recorded as it is not relevant to the results. We only focus on studying the mean.

* **Brick Placement**

There are only minor differences in the results of different brick placements.



* **Rows**

As the number of row increases, the mean of the results increases.



* **Columns**

As the number of columns increases, the mean of the results also increases.



* **Ball Speed**

As the ball speed increases, the mean of the results decreases before increasing again along with the number of failed runs.



* **Paddle Speed**

After an initial high along with a high number of failed runs, the mean of the results remains relatively stable.



* **Game Mode**

Both game modes have identical results. Inverting the bricks and paddle position seems to have no effect on the results.



## Parameter Settings

The results of varying the parameter settings are shown below. We study the difference to discover which parameters has the greatest improvements on the results.

* **Q-Table**

As the Q-table increases, the difference also increases along with the failed runs.



* **State**

As the number of state space increases, the difference also increases.



* **Action**

When the action space increases, there is minimal difference between the results.



* **Random**

There is no difference between initializing the Q-tables with 0s or random values.



* **Opposition**

Opposition learning fails to work with the constant reward function.



* **Reward**

Only one of the reward function, X-Distance (Center) show a significant difference while the rest have none.





* **Combination**

First, we decide which parameters to include in the combination. **Random** was excluded from the combination as there was no difference in any of the results. All other parameters were included. **Opposition** was included as the results were inconclusive and were not a definite failure.



We first choose the correct **R** as it is the most important part of the agent. We then combine **R** with the other parameters.

We observe that only **RS** and **RO** have better results while **RA** and **RQ** have worse results. We then combine both **RS** and **RO** and see if there are any improvements.

We find that **RSO** is better than **RO** but worse than **RS**. So, we conclude that **RS** is the best parameter settings combination.



## Hyperparameter Tuning

The **RSO** parameter settings combination is used in genetic algorithm to tune the hyperparameters. The fittest chromosomes after 50 generations are given below and are arranged in the order Initial / Final / Step and the values are given in percentages (%).

* **Final Chromosomes**

These are the final 10 fittest chromosomes.



* **Chromosome Results**

These are the results of using the **RSO** parameter settings combination and the hyperparameters of the chromosomes.



* **Chromosome Results Sorted**

These are the chromosome results sorted by the difference in descending order.



* **Final Chromosomes Sorted**

These are the final chromosomes sorted based on the sorted chromosome results. The combination of the **RSO** parameter settings combination and hyperparameters given here will produce the best results for the game that we were able to find.



Discussion

The results of the experiments above will now be discussed so that we may gain better understanding and insight into the different settings of the experiments.

## Game Settings

First, we discuss the effects of varying the game settings on the results.

* **Brick Placement**

As seen above, there is minimal difference in the results of different brick placements. This is because the overall objective of the game is still the same which is to break 3 rows of bricks without losing the ball. The difference in results comes from the positioning of the 3 different types of bricks.

* **Rows**

As the number of row increases, the mean of the results also increases. This is fairly obvious as when the number of row increases, the number of bricks also increases. This increases the length of the game and gives the agent more chances to commit mistakes and lose the game. So, the number of episodes naturally increases.

However, it can also be observed that the rate of increase of the episodes slows down as the number of rows increases. This can be attributed to the fact that as the length of the game increases, the agent becomes an even better player and makes less mistakes. In theory, there is a point where increasing the number of rows does not increase the number of episodes as the agent has already learned to play the game perfectly.

* **Columns**

As the number of columns increases, the mean of the results also increases. The reasoning for this is also similar to the one above where increasing the number of bricks increases the length of the game and gives the agent more chances to lose. The rate of increases also slows down for the same reason that the agent slowly becomes a better and better player at the game.

* **Ball Speed**

As the ball speed increases, the mean of the results decreases before increasing again along with the number of failed runs.

As a reinforcement learning agent, it needs to win and lose in order to learn the correct moves. When the ball speed is low, the ball takes more turns to move and to lose. This causes the agent to take longer to learn the correct moves and thus the number of episodes needed is higher. So, increasing the ball speed decreases the mean.

After a certain point, the ball moves and loses so quickly that the agent has no time to learn. This causes the failure rate to increase as the ball speed increases. So, the perfect ball speed for this agent is around 7.

* **Paddle Speed**

After an initial high along with a high number of failed runs, the mean of the results remains relatively stable.

For the paddle, its movement speed is not very important. It just has to be fast enough to catch the ball. So, increasing the paddle speed does not affect the mean very much.

However, issues arise when the paddle speed is too low. The paddle may not be able to traverse the screen to catch the ball and this causes failure rates to increase. So, any paddle speed above 5 is usable.

* **Game Mode**

Both game modes have identical results. Inverting the bricks and paddle position seems to have no effect on the results. This is due to the fact that the agent is training against the same 3 rows of bricks and thus the only difference is to invert the controls.

## Parameter Settings

Next, we discuss the effects of varying the parameter settings on the results.

* **Q-Table**
* **State**
* **Action**
* **Random**
* **Opposition**
* **Reward**
* **Combination**

## Hyperparameter Tuning

This part discusses about