Q-LEARNING: EFFECTS OF ALGORITHMIC IMPROVEMENTS AND DIFFERENT SETTINGS ON BRICK BREAKER GAME PERFORMANCE

INTRODUCTION

Brick Breaker is a game where a player moves a paddle left and right to bounce a ball and destroy bricks. The player wins if all the bricks are destroyed and loses if the ball leaves the screen.

Q-Learning is the reinforcement learning algorithm used in this project to teach the agent to play the game. The game settings, parameter settings and hyperparameters are varied to discover the best settings and best results.

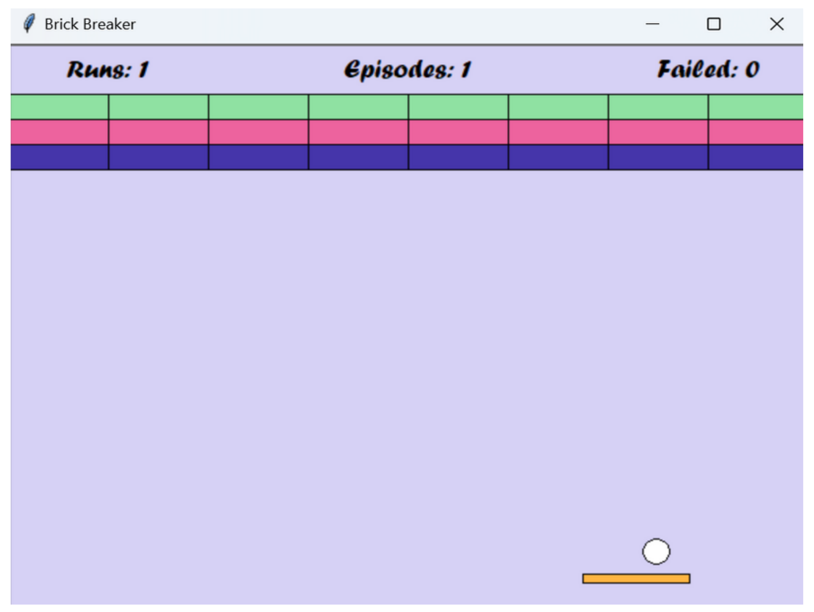


FIGURE 1: BRICK BREAKER

EXPERIMENTS

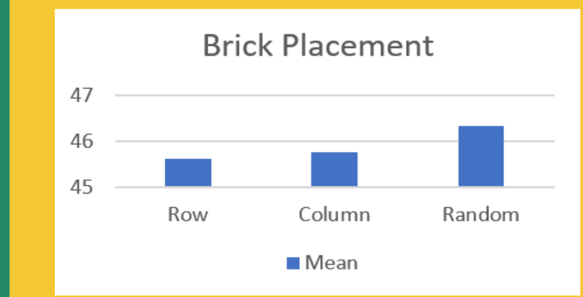
The experiments in this project are run using the GUI shown below. The steps of the experiments are listed below:

- Each game setting is varied individually and the mean is studied.
- Each parameter setting is varied individually and the expected difference in mean is studied.
- Different combinations of parameter settings are tested to discover which combination has the best results.
- Hyperparameter tuning using the best parameter combination is done to find the fittest hyperparameters.
- The best parameter combination, and best hyperparameters are used to find the best results.

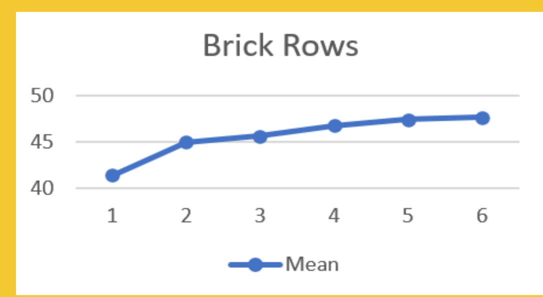
$$d = (m_2 - m_1) - t_{0.01} \times s_p \sqrt{\frac{1}{n_1} + \frac{1}{n_2}}$$

FIGURE 2: DIFFERENCE FORMULA

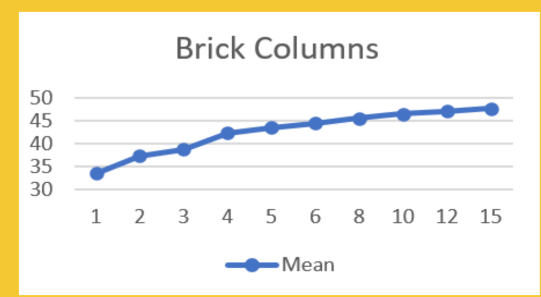
RESULTS + ANALYSIS (GAME SETTINGS)



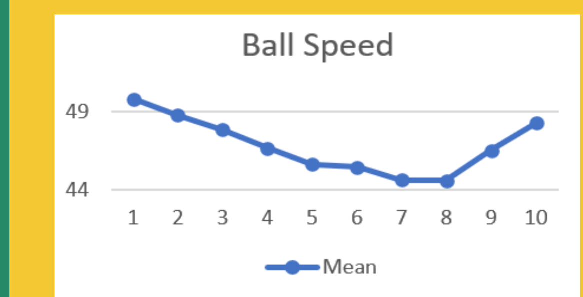
- Minor difference in the results
- The overall objective to break three rows of bricks is same
- Difference is from placement of the different types of bricks



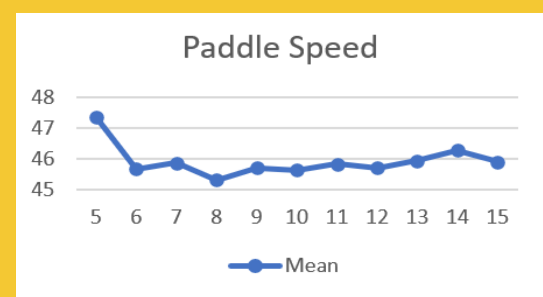
- Mean increases with number of brick rows as the number of bricks increases
- Gradient decreases as game length increases as the agent becomes a better player



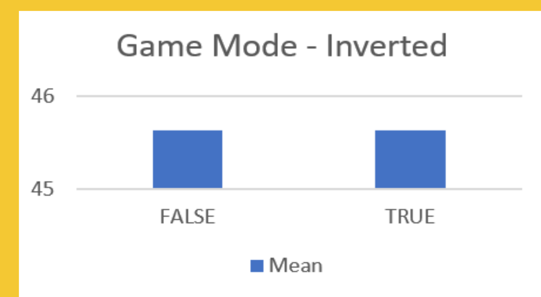
- Mean increases with number of brick columns as the number of bricks increases
- Gradient decreases as game length increases as the agent becomes a better player



- Mean decreases when ball speed increases as the agent learns more quickly
- Mean increases when ball is too fast and the agent loses before learning anything



- Paddle speed is not important as paddle only needs to be fast enough to catch the ball
- Mean is high when paddle is too slow but remains stable when paddle is fast enough

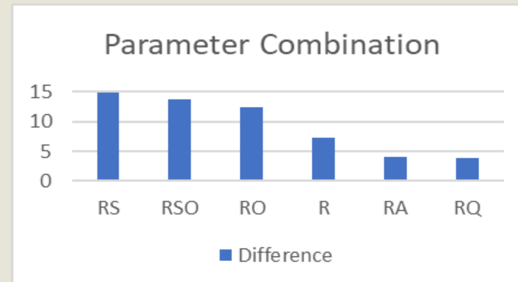


- Both game modes have identical results. Inverting the game has no effect.
- Agent is training against the same three rows of bricks and just has to invert its controls

PARAMETER COMBINATION

Symbol	Parameter Setting
R	Reward – X-Distance-Paddle
S	State Space – 8
O	Opposition Learning – True
A	Action Space – 3
Q	Q-Table Number – 5

FIGURE 7: SYMBOL MAPPING



R is most important so is chosen first.

RS, RO > R > RA, RQ

S and O works well with R while A and Q do not.

RS > RSO > RO

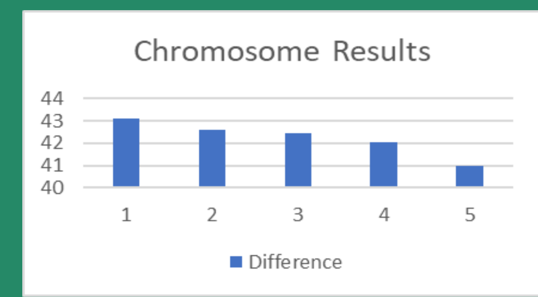
RSO produces worse results as S and O interfere.

The best parameter combination is RS.

HYPERPARAMETER TUNING

No	Learning	Explore	Discount
1	82/92/1.0	19/11/0.7	36/37/0.7
2	82/92/1.0	06/11/0.7	36/37/0.7
3	82/48/1.0	06/11/0.7	36/37/0.7
4	82/92/1.0	06/13/0.6	87/95/0.7
5	82/48/1.0	29/11/0.3	36/37/0.7

FIGURE 8: FITTEST HYPERPARAMETERS



The hyperparameters are tuned using the RS parameter combination. The top hyperparameters and their experiment results are shown above.

The agent prefers a high learning rate, a moderately low explore rate and a moderate discount factor for the best results.

Biggest difference is 43.12 with a mean of only 1.83.

GRAPHICAL USER INTERFACE (GUI)

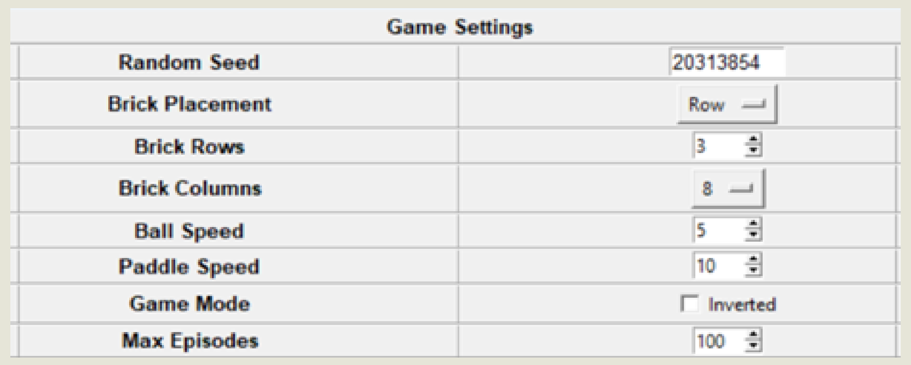


FIGURE 3: GAME SETTINGS

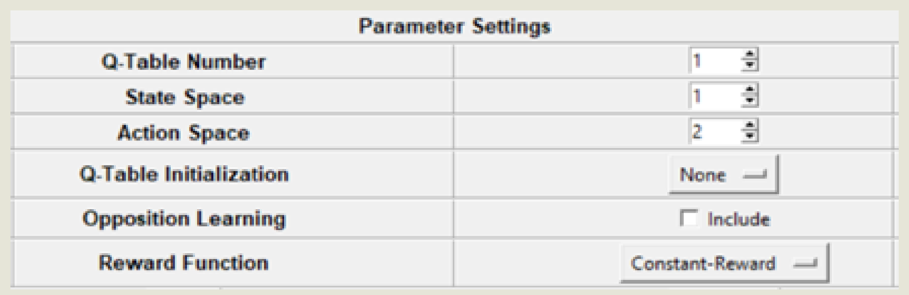


FIGURE 4: PARAMETER SETTINGS

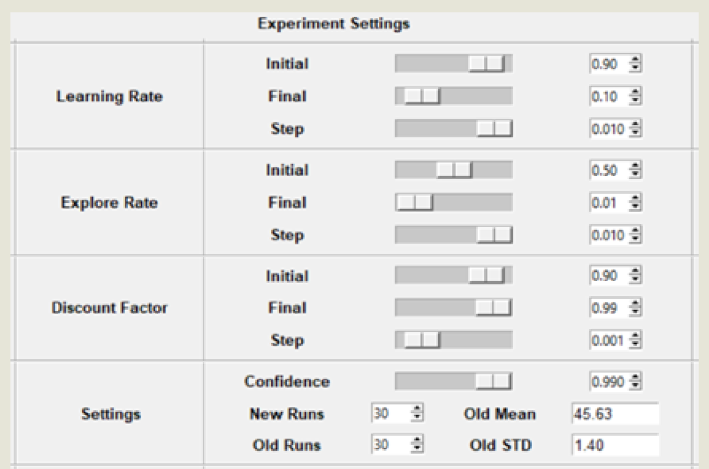


FIGURE 5: EXPERIMENT SETTINGS

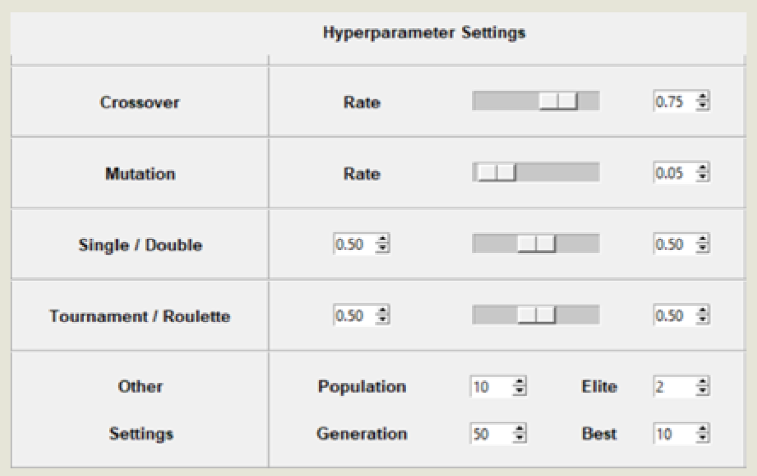
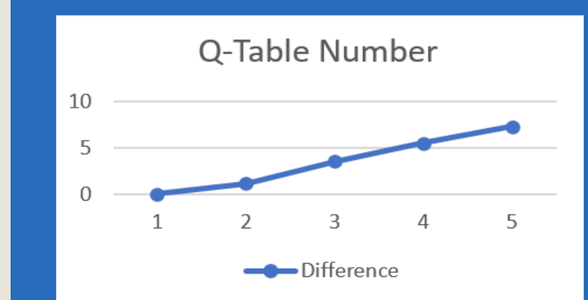
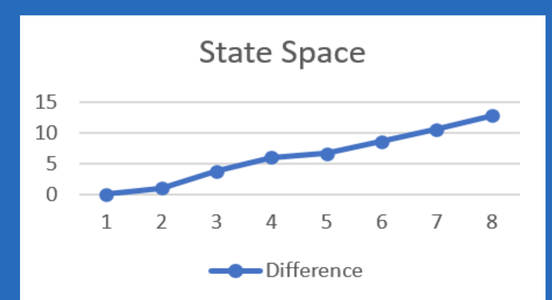


FIGURE 6: HYPERPARAMETER SETTINGS

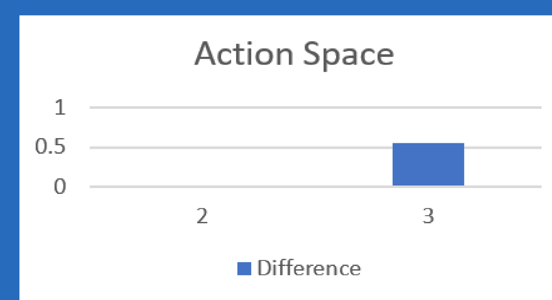
RESULTS + ANALYSIS (PARAMETER SETTINGS)



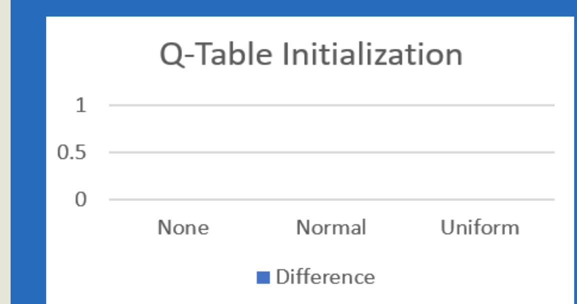
- Difference increases as the overestimation bias decreases
- Failure rate also increases with more spaces to fill while bad moves are not immediately apparent to the agent



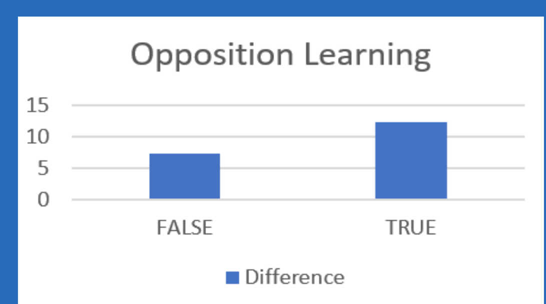
- Difference increases as the granularity increases
- Too coarse granularity causes bad state space separation
- Too fine granularity causes redundancy in state spaces



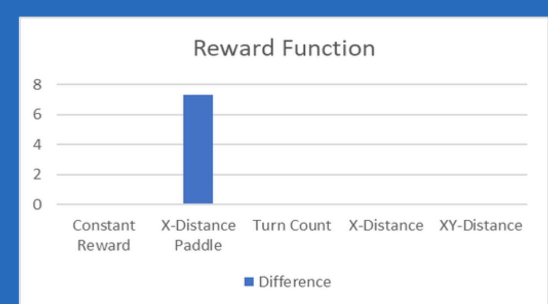
- Actions: Left, Right, Do Nothing
- Action space needs to be meaningful for good results
- Minor improvement in results due to uninformative reward function and randomness



- There is no difference in initializing the Q-tables with 0s or random values
- Game is too simple and short for exploration to have any noticeable effects



- Opposition learning shows visible improvement in results as twice the information is available to train the agent
- Reward function needs to be informative for good results



- Reward function needs to be informative for good results
- Only the shortest horizontal distance from the ball to whole paddle shows a visible improvement in results

CONCLUSION

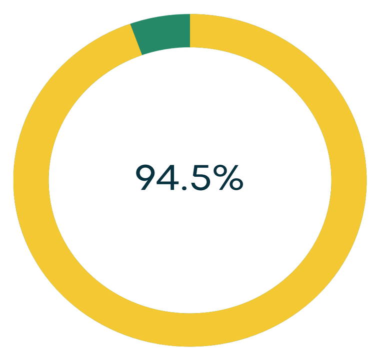
In conclusion, the Q-Learning algorithm was successfully used to teach an agent to play the game Brick Breaker.

The effects of varying the game settings, parameter settings and hyperparameters settings were studied.

The best parameter combination found was to use the correct **Reward Function** and a high **State Space**.

The best hyperparameter settings found was to use a high learning rate, a moderately low explore rate and a moderate discount factor.

The best result was a difference of 43.12 with a mean of only 1.83. This represents an improvement of 94.5% over the original mean of 45.63.



REFERENCES

The code referenced in the project:

- https://www.studytonight.com/tkinter/brick-breaker-game-using-tkinter-python-project
- https://medium.com/@tuzzer/cart-pole-balancing-with-q-learning-b54c6068d947

The authors referenced in the project:

- Wicaksono, A. S., & Supianto, A. A.
- Tizhoosh, Hamid
- J. Fior and L. Cagliero
- M. Gregor and J. Spalek
- Zahavy, T., Haroush, M., Merlis, N., Mankowitz, D. J., & Mannor, S.
- S. Šćepanović
- Hasselt, H.