**Reinforcement Learning in 2-D Space with Varying Gravitational Fields**

**Timothy Rousseau1, David Jose Florez Rodriguez2,3**

1 Menlo-Atherton High School, Atherton, California, USA

2 SHTEM, Stanford University, California, USA

3 Papaya Institute, Phoenix, Arizona, USA

**Student Authors**

Timothy Rousseau, high school

**SUMMARY**

Reinforcement learning is incredibly useful for things like navigating a predefined space. This is the topic we choose to focus on. The problem we set out to solve is evaluating various methodologies for training a reinforcement learning model to navigate a plane in space with a defined gravitational field set by two randomly placed planets. In a plane let the rocket start in the bottom left quadrant, aiming to reach a target in the opposite corner. For the reinforcement learning, we used a Q-Learning algorithm1 , testing accumulated learning. In this project, we tested how iterative retraining of a Q-Learning model on its own navigation attempts might impact its percent improvement over a random navigation procedure. This might allow successful learning with limited data given sufficient time. In our results, we are looking for improvement over random results and any new navigation methods that the computer can develop such as a slingshot method. Possible applications of this AI technology could be guiding unmanned rockets in asteroid mining3. Results suggest that many repeated iterations of training a Q learning model may have a slight negative effect on performance. Additional testing suggests of the amount of random simulations ran to train a model had a slight negative correlation to performance, and the location of planets play an incredibly significant role in Q-Learning performance. Learning models also display trajectories suggesting that Q-Learning develop different methods to effectively arrive at the reward.

**INTRODUCTION**

This paper focuses on the practicality of retraining Q-learning with the goal of navigating a plane in space. Literature suggests that, with the right reward, reinforcement learning can help an agent navigate a predefined space1. Exploring this topic will be incredibly useful for current and future navigation in outer space. By letting a computer determine the optimal path through a field of obstacles, space navigation will become much simpler. Some applications of this research are for unmanned vehicles which can be programmed to travel a certain way based on the results of the reinforcement learning algorithm. Additionally, it can be useful for navigating dangerous and unknown environments, as the computer can develop different strategies to effectively travel through the plane without the risk of loss of life or resources from a real-life simulation. In the case of computationally complex simulations, retraining iteratively on initial simulations and the resulting navigation data from Q-learning could add necessary efficiency to reinforcement learning. We hypothesize that iterative Q-Learning retraining on random simulation data and the resulting learned navigation data will correlate with higher improvements over random navigation strategies.

This mode of creating simulations is also very easy to do. This project modeled reinforcement learning in Google Colab, a python script with low ram and computing power. To get around these computing limitations, we used a Q-learning model1. This simple algorithm lets the users access reasonable and accurate results even with low computing power.

In this project, researchers also wonder if the computer is able to develop new strategies for traveling through a plane. One such strategy could be the slingshot strategy3 (where the “rocket ship” will be “slingshotted” around a planet using its gravitational field to arrive at its target faster or more efficiently) which is used by real rocket ships and satellites. One such path is graphed out in Figure 1. By looking at and graphing the data, we will be able to determine how the computer model developed or “learned.”

**RESULTS**

Figure 3 is the trend line for percent improvement as the number of *qruns* increased. From the graph, we can infer that there is a slightly negative linear relationship between percent improvement and *qruns*. This means that lower values for *qruns* yields higher percent improvement results. There was nearly a one percent point decrease in improvement over random navigation when iteratively training one additional time.

Figure 4 is the trend line for percent improvement as the number of simulations ran increased. From this trendline, we can conclude that there is a slight negative linear relationship between percent improvement and *NumSims*. This means that there is a higher percent improvement when less simulations are run. Almost every model trained with larger *NumSims* underperformed compared to models trained on less data, overall, the difference in percent improvement between the model with the least data to the model with the most data was over 6 percentage points.

Figure 2 is the plot of different combinations of planets we tested. The darker the data point, the most successful the test runs were, as measured by percent improvement. This suggests that there is a slight benefit of having planets closer to the initial starting point of the particle rather than closer to the reward. Additionally, the computer did better when the planets were in between the starting position and the reward zone. Depending on planet locations, some models did as well as over 70% better than random simulations, and some models did nearly 20% worse than random simulations.

**DISCUSSION**

Overall, the data does not support our hypothesis that the number of *qruns*, and thus more iterative training on simulation data and resulting model navigation, will improve model performance.Larger amounts of simulation data, *NumSims,* also seemed to make the Q-Learning less effective at reaching the reward. However, the evaluated Q-learning models were still better than random simulations, meaning that the computer model was able to properly calculate more optimal actions to take to reach the reward.

Additionally, we found that the location of the planets played an incredibly significant role in how effective machine learning was. The fact that there was a slight benefit of having planets closer to the initial starting point of the rocket rather than near the reward can be attributed to the smaller ‘gravitational’ acceleration near the reward, making actions chosen by the computer have a greater effect. Additionally, the computer tended to use a slingshot strategy (employing the gravitational force of the planets to optimize the path to the reward) when the planets were in between the starting position and the reward zone. Since there tends to be a higher percent improvement in this scenario, we can conclude that a slingshot strategy can be beneficial for travel in a 2-D space with varying gravitational fields.

The way gravity was calculated could be a source of error due to its lack of accuracy. In a next trial run, it would be best to use interpolation, finding the gravitational force based on the ship’s relative location to the four points surrounding it. This would make the gravitational field continuous and could reduce inaccuracies. However, our data demonstrates that our machine learning algorithm was effective and useful.

One limitation was the computing power available for this project. Since we only had access to one computer, we had to choose a low-computing power program to run our code and we weren’t able to test extremes. To complete the project in a reasonable amount of time, we could only run a maximum of 10,000 simulations, 7 *qruns* and 20 different planet pair locations. With more computing power, we could increase these values to test extremes and run more tests to generate more accurate results. Parallel computing would be a good option for this.

**MATERIALS AND METHODS**

For this project, we decided to analyze the effect of the number of simulations run, the amount of Q-learning iterations*,* and the location of the planets. The number of simulations run (*NumSims)* is the number of random simulations that are run to train an initial model. This is the same amount of simulations that the Q-learning algorithm is run on. The number of iterations that the Q-learning algorithm is run (*qruns)* concerns the main hypothesis. The higher the value of *qruns*, the more times the algorithm applied Q-learning to the data and the more “rigorous” the Q-learning is. Finally, we generated random locations for two planets in the 16 by 16 grid, running simulations on a gravity field based on those locations.

The iterative training, with *qruns* iterations, worked as follows: initial training always runs on random, simulated navigation data. *NumSims* simulations, often in the thousands, inevitably navigate a gravitational field successfully to earn a reward at least a few times. This data is fed into the Q-Learning algorithm. Having trained, the Q-Learning algorithm is now evaluated *NumSims* times. This is training with *qruns* = 1. The navigation data can then be fed into the Q-learning algorithm. The resulting model now has *qruns* = 2. Larger values of *qruns* simply meant training further on the latest model’s navigation data, generating a new ‘latest’ model, and repeating the procedure as desired.

For the computer to “learn,” we needed to define a reward. The research investigates whether the computer model can find a path to end up in a reward area. This reward area was set to a circular area in the top right of the plane. This made sure that the computer model could know whether it had succeeded in a given iteration, and its success could be quantified.

To give the computer some test data to learn from, we ran hundreds of random simulations. We placed the rocket at random locations in the bottom left quadrant and initiated it with a positive vector velocity (where both entries in the vector were positive). Additionally, although the planet locations were random, we limited the area that they could be to only in the center regions of the grid (4 to 12 in both x and y directions). The planet locations were uniformly generated in this region to make planet coordinates never land on lattice points where gravity was calculated. During the simulation, the rocket had 30 different opportunities to choose and perform a random action (one of increasing velocity up, left, down, right, or staying still) that would move it around in the grid. After 30 opportunities, if the rocket has not reached the reward, the simulation is terminated. This way, we would eliminate processing time while letting the rocket still have the possibility of ending up at the reward. Another thing to note is that the computer model could reach the reward solely based on motion from the force of gravity4, from its actions, or a combination of both. This represents a real-life scenario where a ship will be affected by gravity, but will also be able to adjust course.Then, the computer would run the Q-learning algorithm, calculating whether the action taken at a specific point was the “right” one, or the action that got it closer to the reward2. After running the Q-learning algorithm on all the simulation data, the computer now has estimates for which action is best at each location. Finally, we reran the simulations using the same amount of *NumSims* as the random simulations. In this instance of testing, the computer has the best actions at each location in the grid and we recorded whether the resulting simulations were better than the initial random simulations or worse.

To determine how much better a machine learning algorithm was at navigating a plane than random simulations, a separate simulation code evaluated it. This evaluation measures how effective random simulations were at getting to the reward, how effective a machine learning algorithm was at getting to the reward and calculated the percent difference2.

One thing to note about our experiment is the way we calculated gravity at different locations. To decrease the computing power required to run simulations, we only calculated gravity at the 0.1 intervals on our coordinate grid. However, we changed the location of the ship by decimal increments, meaning that the ship could be between any four lattice points. In this case, the location was rounded to the nearest integer when calculating gravity. The gravitational field and thus acceleration affecting the ship at a point was rounded to the nearest computed location. Given that the planets were point masses, this discretization of gravity also decreases the chance of gravity being computed exactly at a planet’s location, resulting in division by zero.

All code for the research is available on the github5.

**REFERENCES**

1. Kochenderfer, Mykel J, et al. *Algorithms for Decision Making*. Cambridge Massachusetts Institute Of Technology, 2022.
2. Dayal, Aveen, et al. “Reward Criteria Impact on the Performance of Reinforcement Learning Agent for Autonomous Navigation.” *Applied Soft Computing*, vol. 126, Sept. 2022, p. 109241, https://doi.org/10.1016/j.asoc.2022.109241.
3. Mark Joseph Sonter. *The Technical and Economic Feasibility of Mining the Near-Earth Asteroids*. 1996.
4. Randall Dewey Knight. *Physics for Scientists and Engineers : A Strategic Approach*. Boston, Pearson, 2017.
5. Rousseau, T. et al. *Extra-Planetary Reinforcement Learning.* Github. https://github.com/ThePapayaInstitute/ExtraPlanetaryReinforcementLearning/tree/main

A graph with dots and circles

Description automatically generated  
**Figure 1. Sample path of agent employing a slingshot technique.** This plot shows a sample path of the particle (blue dots) around the plane with two planets (red dots). The green region is where the agent earns a reward. Given this trajectory, beginning in the lower left, the total reward would have been 20. This is over 500% higher than the reward of a random path, which was 3.2.

A green bar with black text

Description automatically generatedA graph of green dots

Description automatically generated

**Figure 2. Planetary locations and learning algorithm performance.** This graphic visualizes how the position of planets made it possible for the algorithm to achieve a higher reward. The darker the color the greater the percent improvement as a decimal. The largest percent improvement was 73.3%.

A graph with blue dots

Description automatically generated

**Figure 3. Trend in Q-learning’s improvement as *qruns* vary.** This graph shows that as the number of *qruns* was increased, the average improvement decreased. It decreased by about 0.925% per integer increase in *qruns*.

A graph with blue dots

Description automatically generated

**Figure 4.** This graph shows that as the number of *NumSims* increased, the average improvement also decreased. The average improvement decreased by about -8.86 x 10-3 % per additional 1000 simulations.