A SpaceQ Rocket explores Deep Space with Deep Q-Networks

Frederick Shpilevskiy
fshpil@students.cs.ubc.ca

Mauricio Soroco
m.soroco@alumni.ubc.ca

Joel Vandervalk
vjoel1@student.ubc.ca

Abstract

Reinforcement learning (RL) is a successful machine learning approach to training an agent through its interactions with a dynamic environment. We use the RL framework to train a spaceship agent to navigate a dynamic star system. We present an exploration of the challenges and ideas that we encountered throughout our investigation of the problem.

1 Introduction

Reinforcement learning (RL) is an area of machine learning (ML) that investigates how an agent learns from interacting with a dynamic environment [1]. In a typical RL model, an agent is able to observe its environment and take actions. The agent receives an input as the current state of the environment and makes a decision about which action to perform. The action will lead to a new state for which there may be an associated reward or penalty. The reward value is used to determine the value of an agent's decision. RL involves learning an optimal strategy for making sequential decisions where both immediate and future consequences of actions are considered. This learning emerges from a trade-off between exploring the environment's state space and exploiting the information about the environment that has already been gathered. Typically, an agent will start by exploring its environment by making random decisions; as it learns more, it will take actions based on decisions informed by earlier exploration. Finally, an agent can generalize their knowledge of the environment to un-visited states to make informed decisions [2].

We use Deep Q-Networks (DQN) [3] to train a spaceship to navigate a simulated two-dimensional star system by travelling to checkpoints within the system. Both the bodies within the star system and the spaceship are affected by gravitational forces. The spaceship is able to observe its surroundings and change its trajectory by moving at constant velocity. The agent does not have access to any hidden variables that might reveal the rules of the dynamics of the system. It must stay within the star system, within a defined boundary. It is terminated if it gets too close to a body or exits the boundary.

Our goal is to learn whether the agent is able to understand the logic underlying the dynamics of the environment and use this understanding to navigate the changing system. Our contribution involves employing an existing reinforcement learning method, to train an agent to navigate a dynamic environment, the logic of which the agent is not aware. We investigate the agent's ability to navigate physical systems and learn to leverage the laws that govern them to achieve its objective.

2 Related Work

Numerous games have successfully used reinforcement learning. In this section, we talk about some of the related work that has influenced our project.

Preprint. Under review.

2.1 Reinforcement Learning in Games

AlphaGo [4]: Although reinforcement learning was successfully used by AlphaGo to solve the challenging Go game, the problem domain and environment are distinct from our spaceship navigation challenge. Unlike the environment in our problem, where the spaceship must navigate around moving celestial bodies subject to gravitational forces, the dynamics of the game of Go are deterministic and do not include continuous and changing environments.

DQN in Atari 2600 games: In a study by Mnih et al. [3], DQN's ability to learn from raw visual input was demonstrated. Though, the Atari games environment is simpler and less dynamic than our challenge. Because the games do not have the same level of complexity and physics-based interactions, the approaches they used need to be modified to tackle the specific challenges of our problem, such as understanding the logic of the environment and applying it to traverse the changing system.

Mirowski et al. [5] used deep reinforcement learning for 3D maze navigation. This research is closely related to our problem because it addresses how to navigate in a dynamic environment. A 3D maze is navigated by the agent in this research, which may not require as many physics-based interactions as our star system problem. The environment in this work may not change as quickly as our problem, and the agent does not need to consider gravitational forces. As a result, the approach outlined in this work may need modifications to handle the unique challenges associated with our problem.

3 Background

3.1 Deep Q-Networks

The agent's objective is to maximize its cumulative reward from future states, $\sum_{t \leq T} \gamma^t r_t$, where T is the terminal time step and each subsequent reward r_t is discounted by a factor of γ. We train the agent episodically, where each episode is a new simulation of T timesteps. The function that determines the action that is taken given a state is the policy. In Q-learning, an agent learns an action-value function $Q(s, a)$ that estimates the expected cumulative reward for taking action a in state s. The action-value function can be updated iteratively by method of temporal differences [6] according to,

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha \left(r + \gamma \max_{a'} Q(s', a') \right), \tag{1}$$

where α is the learning rate, r is the immediate reward, γ is the discount factor, and s' is the next state and a' is an action taken from s'.

Deep Q-Networks (DQN), first introduced by Mnih et al. [3], is a reinforcement learning algorithm that combines Q-learning with deep neural networks to learn from high-dimensional states, such as images or large state spaces, by approximating the optimal action-value function. Through environmental interactions, the agent accumulates experiences given by (s, a, r, s') where s is the state in which the agent performed action a yielding reward r and state s'. These experiences are stored in a set called the replay memory, $\mathcal{D}$ [3].

The neural network with parameters θ approximating the action-value function, called the policy network, takes in a state and outputs the estimated Q-value for each possible action. To train the policy network we feed an experience sample from $\mathcal{D}$, and use the output Q-values to minimize the loss given by,

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s') \sim \mathcal{D}} \left[\left(Q(s, a; \theta) - \left(r + \gamma \max_{a'} Q(s', a'; \theta^-) \right) \right)^2 \right], \tag{2}$$

where θ^- are the parameters of a target network that is used to stabilize learning. The parameters of the target network are updated periodically with the parameters of the policy network.

The DQN algorithm, inspired by the work of Mnih et al. [3], is provided in Appendix A.

4 Problem Statement

We formally define our problem as a Markov Decision Process (MDP) with the following components:

- **Action space** ($\mathcal{A}$): The action space consists of a set of discrete actions that the agent can perform at each time step, such as adjusting the thrust and direction of the spaceship.
- **Transition function** ($\mathcal{P}$): The transition function determines the next state s' given the current state s and the action a taken by the agent. The transition function captures the dynamics of the environment, including the gravitational forces acting on the spaceship and the motion of celestial bodies.
- **Reward function** ($\mathcal{R}$): The reward function maps a state-action pair (s, a) to a scalar reward value r. The agent receives a positive reward for reaching a checkpoint and a negative reward for colliding with an obstacle. The reward function may also include penalties for excessive fuel consumption or time spent.
- **Discount factor** (γ): The discount factor determines the importance of future rewards in the agent's decision-making process. A value of γ close to 1 indicates that the agent values future rewards highly, while a lower value places more emphasis on immediate rewards.
- **Policy** (π): The policy is a mapping from states to actions, representing the agent's decision-making process. In the context of DQN, the policy is determined by the action-value function $Q(s, a; \theta)$, and actions are selected using an ϵ-greedy exploration strategy.

4.1 Action space and Transition function

In our problem, the action space is fixed and our transition function is a function of the physics of the simulation. To solve the problem, we are permitted to modify the form of the states, the reward function, discount factor, and policy function. Our action space is defined by a set of five actions: $\mathcal{A} = \{\text{UP}, \text{DOWN}, \text{LEFT}, \text{RIGHT}, \text{STAY}\}$. Each action thrusts the spaceship in a direction at constant velocity. For instance, UP action will translate the spaceship upwards by a predefined speed. The STAY action will result in no additional thrust – the spaceship's movement will only depend on the physics of the system. Each step, only one action is taken.

The dynamics of the system are given by a simple N-body simulation. Given a set of bodies $\mathcal{B}$, where each body has a series of vectors for its position $\mathbf{x}$, velocity $\mathbf{v}$, acceleration $\mathbf{a}$, and a scalar mass M, we update $\mathbf{x}$ and $\mathbf{v}$ and compute the acceleration due to gravity of each body in $\mathcal{B}$. The acceleration due to gravity of body A by body B in the direction of B is given by,

$$\mathbf{a}_A = \frac{GM_B(\mathbf{x}_B - \mathbf{x}_A)}{\|\mathbf{x}_B - \mathbf{x}_A\|_2^3} \tag{3}$$

where G is the gravitational constant.

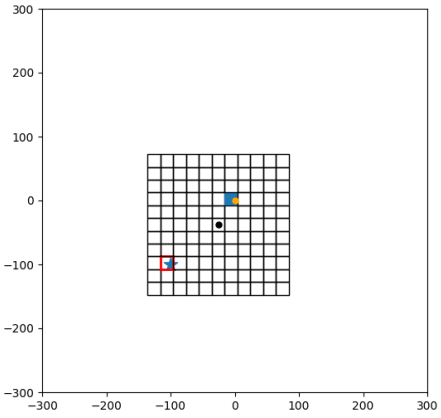

Figure 1: The agent between the checkpoint and body. The star is the checkpoint, the agent is the black dot, and the body is the yellow dot. The grid represents the state.

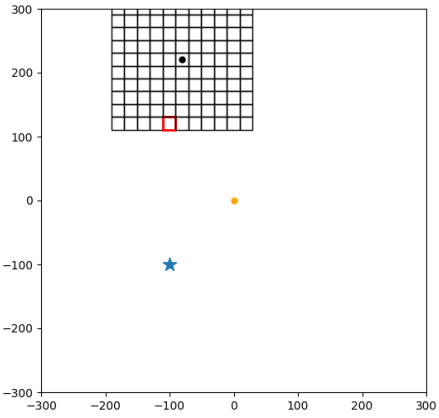

Figure 2: The agent is aware of the direction of the checkpoint (red highlight in neighbourhood) but is not aware of any bodies until they enter its "field of vision".

The pseudocode of the simulation is provided in Appendix A.

4.2 State Space

We design the state to be invariant to the absolute position of the checkpoint and obstacles. We use a square grid to reflect the agent's "field of vision", a diagram of the grid is given in Figure 1. The grid has two channels: one for the checkpoint (the red outlined cell), and one for the obstacles (the blue filled cell). If an obstacle or the checkpoint is within a cell, its respective channel is filled with a value of 1. If the checkpoint is not within the grid, we fill the cell nearest to it (Figure 2). To represent the dynamics of the system, we append a fixed number of grids from previous time-steps to the state as additional channels (one additional grid results in a total of 4 channels). If no previous timesteps exist, such as at the start of the simulation, we use a blank grid (all zeros). The agent is also provided with a series of neighbourhood grids from past time-steps at regular intervals. The intervals of these time steps and how many such frames it remembers are both hyperparameters of the simulation. Thus the current frame depicting the agent's neighbourhood and the past frames are the features constituting the state. We initially believed past information was necessary so that the agent can perceive the effects of the forces and its own movement. This problem has discrete actions and a large state space, therefore Deep Q-Networks (DQN) are an excellent fit for this problem [2].

4.3 Reward Function and Policy

Each timestep, the agent was given a small penalty proportional to the distance between it and the checkpoint. This is meant to incentivize it to get to the checkpoint as soon as possible. It is given a positive reward upon reaching the checkpoint and a penalty if it is terminated as a result of getting too close to a body or exiting the boundary.

$$r = \begin{cases} p & \text{checkpoint reached} \\ -p & \text{agent terminated} \\ \xi(1 - \frac{\zeta}{\mathbf{x}_{\text{checkpoint}} - \mathbf{x}_{\text{agent}}}) & \text{otherwise} \end{cases} \tag{4}$$

where p, ξ, ζ are all positive constants.

We use a convolutional neural network (CNN) as our policy network. We experiment with different number of convolutional layers, kernel sizes, and fully connected layers.

5 Results and Discussion

We investigate a simple version of the problem. There is a single body at the centre of the environment and the checkpoint is placed in the lower left corner (see 1,2).

5.1 Random starts

We initially ran the simulation with a fixed position for the agent. We observed that exploring the environment by taking random actions was difficult as the force of gravity would eventually pull the agent into the body. After 10 thousand episodes, the agent had not reached the checkpoint once. As a result, the agent had never received a reward, which led to it learning a suboptimal policy. To force exploration of the environment, each episode, we intialized the agent in random positions. As a result, the agent experienced more states and reached the checkpoint – and experienced a reward finally. This led the agent to learn alternative policies and improved its performance.

5.2 Death or Misery

As the agent was learning, we experimented with a positive reward system that increased as the agent approached the checkpoint. The failure of this was that the agent was incentivised to accumulate positive rewards by avoiding both bodies and the checkpoint. We could have given a one time reward for reaching unexplored states, but it was difficult to determine which states should be considered unexplored. Having changed to the reward system of Equation 4, we observed a decrease in the overall survival time of the agent. Further, we found that the agent would aim for the body. This is because the agent preferred to terminate rather than survive and accumulate penalties (and then

terminate anyway). We attributed this to a lack of training time for the agent to discover the rewards in the environment – in part due to the difficulty of exploration. We later found that with our simulation hyperparameters, a large proportion of the simulations were bound to fail irrespective of the agents decisions (see section 5.3) so we conclude that the agent was learning and was simply miserable.

5.3 Agent Speed

The agent's speed is very important to the simulation. If it is too high, the problem is fairly trivial as the agent can overcome the gravitational forces and avoid interacting with the physics of the simulation. If the speed is too low, the agent may not be able to be able to get to the checkpoint and will instead learn that termination is the optimal policy. In some experiments with lower speeds, we observed that the agent will at first attempt to get to the checkpoint; but once it realizes that it does not have enough speed to get to the checkpoint, it will turn and go straight into the body as fast as possible.

5.4 Pausing Replay Memory

At the start of training, the agent chooses actions randomly with a probability of 0.95. As training progresses, the probability of choosing random actions decays exponentially. Our replay memory buffer has a fixed size; when it is full, the older experiences are replaced by newer ones. As training progresses, the agent takes less random actions, which reduces its exploration. We observed that as the number of training episodes increased, the replay buffer would be filled with experiences that reinforce the current policy – even if it was a suboptimal one. To resolve this problem, we increased the amount of exploration at later episodes by reducing the decay and introduced an offline learning phase in which the agent would not add new experiences to the replay buffer. The offline learning phase resulted in an extreme increase in the loss (Equation 2); in the future, we plan to explore this result.

5.5 Q-scores

5.5.1 Visualization

To be able to see what the policy network was learning, we visualized the Q-scores in a heatmap. To generate the heatmap, we sampled the state at each position in the environment (as if the agent was there). The intervals of the samples are given by the unit length of a cell in the agent's state grid.

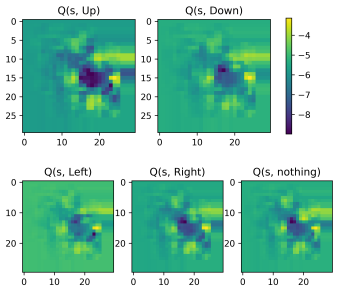

Figure 3: A heatmap representing the Q-scores at each position of the environment. The true position of the checkpoint is the top left. Episode 10 thousand of a CNN policy network without random initial agent placement. There are no positive Q-scores.

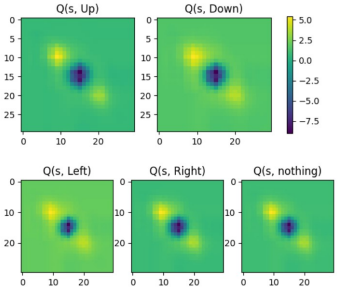

Figure 4: The yellow circle in the top left corresponds to the true position of the checkpoint. Episode 60 of a CNN policy network with random initial agent placement.

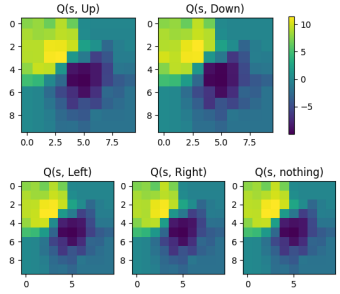

Figure 5: The yellow circle in the top left corresponds to the true position of the checkpoint. Episode 6 thousand of a CNN policy network with coarse grids and random initial agent placement.

Figures 3, 4, 5 present the resulting Q-score visualizations of a sample of the experiments that we conducted. All of the Q-scores for experiments with random initial agent positions (Figures 4, 5) have a positive value near the agent's true checkpoint position and a negative value around the body.

The Q-scores around the edges of the environment are positive. We expect that when an agent is near the edge of the environment, there should be a reduced Q-score as the agent is close to exiting the boundary and incurring a penalty. In fact, the Q-scores near the boundary are lesser than the area around the checkpoint but greater than the area around the body. This may indicate that the agent is aware of the gravitational effect of the star and considers the area around the star to be more "dangerous" than the edges because of this fact (see Appendix for further discussion).

5.5.2 Q-score Similarity

The heatmaps are fairly similar across actions. This is not necessarily expected as, for example, going UP on either side of the body will result in an orbit, but going RIGHT or LEFT may result in the agent getting to close to the body and being terminated. One explanation for the similarly in Q-scores across the actions is that the next state after taking an action may be the same. If our neighbourhood grids are too coarse, after any action is taken, the bodies may still remain in the same cell, only moving within it – which the agent does not perceive. Similarly, if there are no bodies remotely close to the agent, then the grids will be identical regardless of the agent's action. We increased the stride between past grids so that the agent is better able to perceive dynamics. While all of the heatmaps were similar (see Figure 5), we found that a larger proportion of runs achieved the checkpoint (our runs are shown in Appendix B). This adjustment gave the largest improvement aside from the random intial agent placement. We also decreased the coarseness of the grids. A higher resolution grid was better able to capture the change in states as a result of the agent's movement.

5.6 Best Performing Model

We combined the intuitions we gathered to develop our best performing model. We used a fine-grain grid with large stride between past grids, trained with random initial agent positions, and reduced the decay in the probability of choosing random actions. Our hyperparameters are included in Appendix B in Table 1. The model is available in our repository in `./models/big_stride_not_coarse_hard_updates`. Interestingly, unlike Figure 5, this model has a slightly lower Q-score in the very corner of the top-left quadrant and a higher Q-score along the adjacent edges. We include a more in-depth discussion in Appendix C.

6 Conclusion

We implemented DQN to train a spaceship agent to navigate a dynamic star system. Our investigation of numerous models and environment heuristics, such as random starts, agent speed adjustments, and changes to grid coarseness and stride, provided light on the difficulties encountered in this learning task and why an agent may fail to achieve its objective. The modifications that provided the most significant improvement in reaching the checkpoint included enhanced grid resolution, increased stride and a lower learning rate. We visualised the model's Q-scores in a two-dimensional density plot. Future works are needed to investigate the similarity of these Q-scores across actions, as well as to determine the climb in loss during offline learning. Similarly more hyperparameter tuning would allow the agent to learn a more optimal policy and perhaps extrapolate to a multiple body environment with more complex gravitational forces. We hypothesize that training the agent with a higher margin of error within which it must reach the checkpoint, and decreasing it over time would allow the agent to experience greater rewards earlier to encouraging it to reach the checkpoint.

References

[1] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," 1996.

[2] M. van de Panne. [Online]. Available: https://www.cs.ubc.ca/~van/cpsc533V/index.html?

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski et al., "Human-level control through deep reinforcement learning," Nature, vol. 518, no. 7540, pp. 529–533, 2015. [Online]. Available: https://www.nature.com/articles/nature14236

[4] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. V. D. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot et al., "Mastering the game of go with deep neural networks and tree search," Nature, vol. 529, no. 7587, pp. 484–489, 2016. [Online]. Available: https://www.nature.com/articles/nature16961

[5] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. J. Ballard, A. Banino, M. Denil, R. Goroshin, P. Sprechmann, Y. W. Teh et al., "Learning to navigate in complex environments," in 4th International Conference on Learning Representations, ICLR 2016, 2016. [Online]. Available: https://arxiv.org/abs/1611.03673

[6] M. Gormley. [Online]. Available: https://www.cs.cmu.edu/~mgormley/courses/10601-s17/slides/lecture26-ri.pdf

A Algorithms

Algorithm 1 N-body simulation

Input: $\mathcal{B}$ is a set of bodies where each body B has position $\mathbf{x}_B$, velocity $\mathbf{v}_B$, acceleration $\mathbf{a}_B$, and mass M_B, T is the number of steps, α is the stepsize

for $t \in T$ **do**
  for $A \in \mathcal{B}$ **do**
    Update $\mathbf{x}_A \leftarrow \mathbf{x}_A + \alpha \mathbf{v}_A$
    Update $\mathbf{v}_A \leftarrow \mathbf{v}_A + \alpha \mathbf{a}_A$
  end for
  for $A \in \mathcal{B}$ **do**
    Set $\mathbf{a}_A \leftarrow 0$
    for $B \in \mathcal{B}$ **do**
      Update $\mathbf{a}_A \leftarrow \mathbf{a}_A + \frac{GM_B(\mathbf{x}_B - \mathbf{x}_A)}{\|\mathbf{x}_B - \mathbf{x}_A\|_2^3}$
    end for
  end for
end for

Algorithm 2 Deep Q-Network (DQN) Algorithm

Input: ϵ-greedy exploration strategy, learning rate α, discount factor γ
Output: Trained Q-network

Initialize Q-network $Q(s, a; \theta)$ with random weights
Initialize target network $Q(s, a; \theta^-)$ with weights $\theta^- = \theta$
Initialize replay buffer $\mathcal{D}$
for each episode **do**
  Initialize state s
  Initialize TERMINATED to `false`
  while ¬TERMINATED **do**
    Select action a using ϵ-greedy exploration strategy based on $Q(s, a; \theta)$
    Execute action a, observe reward r and next state s'
    Update TERMINATED to `true`, if s' is a terminal state
    Store transition (s, a, r, s') in replay buffer $\mathcal{D}$
    Sample a random minibatch of transitions from $\mathcal{D}$
    Update $Q(s, a; \theta)$ using gradient descent on loss function $\mathcal{L}(\theta)$
    Update target network $Q(s, a; \theta^-)$ periodically
    Update state $s \leftarrow s'$
  end while
end for

B Reproducibility

Our models and experiment framework are available in our repository.

Our experimental results are available on Weights and Biases.

Table 1: Hyperparameters for best performing model

Number of convolutional layers	Kernel size	Pooling size	Number of linear layers	Learning rate	Training episodes	Decay	Grid width (in cells)	Cell width	Number of past grids	Past grid stride length
1	5	0	2	1e-5	6000	6000	11	20	1	16

C Supplementary Material

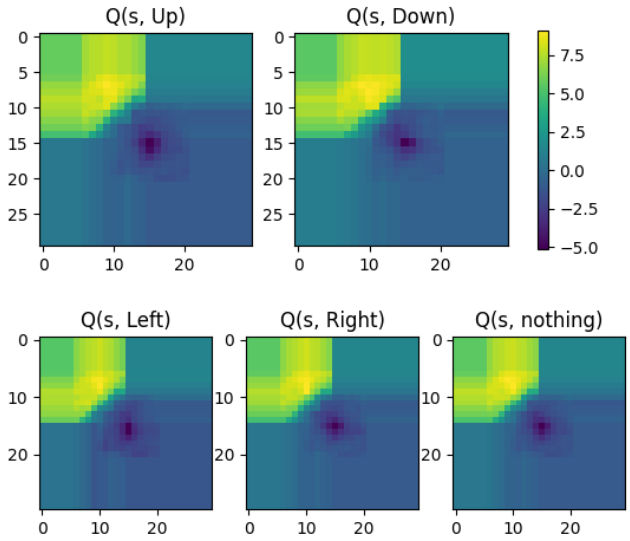

Figure 6: A heatmap representing the Q-scores at each position of the environment. The yellow circle in the top left corresponds to the true position of the checkpoint. Episode 6k of a CNN policy network with coarse grids and larger stride. The Q-score is relatively low in the corner due to the lower degrees of freedom caused by the boundaries of the environment. This indicates the agent is learning that it cannot surpass the boundaries and that being in the corner is sub-optimal due to the larger restriction on its movements.