Landing LunarLander-v2 with Deep-Q Learning

Carl Joseph Cepe
PUP, BSCS 3-4
Antipolo City, Philippines
09205090839
carlcepe909@gmail.com

Jim Schandler Flordeliza
PUP, BSCS 3-4
Binangonan, Philippines
09213914335
flordelizajs77@yahoo.com

Aaron Bryan Parrilla
PUP, BSCS 3-4
Taguig City, Philippines
09771176229
aaronbryanparrila@gmail.com

Ray Jay Arevalo
PUP, BSCS 3-4
Taytay, Philippines
09483117997
rayjay.arevalo08@gmail.com

ABSTRACT
Machine learning, a prominent field in the field of computer science. It is a branch of artificial intelligence that focuses on the idea that systems can learn from data much like humans learn from experience. One method of machine learning is reinforcement learning. This project aims to develop a model that can predict an action for the LunarLander-v2 environment and achieve the winning condition of 200 points through Reinforcement learning using the DQN algorithm with a standard Multilayer Perceptron is used. The best model is trained with 1.6 million time-steps and gets an average score of 168 points near 200 points.

CCS Concepts
• Computing methodologies~Machine learning~Learning paradigms~Reinforcement learning~Sequential decision making•Computing methodologies~Modeling and simulation~Model development and analysis~Model verification and validation•Computing methodologies~Artificial intelligence

Keywords
Reinforcement Learning; DQN; MLP; Gym; LunarLander-v2; OpenAi.

1. INTRODUCTION
In the world of technology, software, and programming artificial intelligence has been a crucial and prominent field, in it is machine learning which is a method of data analysis that automates analytical model building. It is a branch of artificial intelligence based on the idea that systems can learn from data, identify patterns and make decisions with minimal human intervention [2]. Under the many methods of machine learning is reinforcement learning, it is the training of machine learning models to make a sequence of decisions. In it, the A.I undergoes a game-like situation and the computer employs trial and error scenarios for the A.I to learn solutions to the problem, the programmers then employ a reward system to award or penalize the A.I for each action it performs [7]. By doing this project we are garnering valuable experiences in the field of A.I and will surely help us in future endeavors regarding this specific field. This project applies A.I to drone maneuvering using LunarLanderv2 simulation environment from OpenAI Gym. With that, we can understand and dissect the process of A.I implementation and reinforcement learning in specific environments, in this case in the Lunar Lander simulation from the Box2d environment set, with which we can build upon for other potential projects that may come. This kind of project, application of AI in simulations, are a great way to demonstrate and test AI techniques and algorithms and will allow us to test our understanding of Machine learning and reinforcement learning.

1.1 BACKGROUND
1.1.1 Reinforcement Learning
Reinforcement Learning is a machine learning training method which focuses on learning optimal behavior in each environment with rewards and penalties. The agent learns through interactions and observations with the environment and will learn what actions yields to rewards and the ways how to maximize reward, a similar way to trial-and-error search. The value of the actions that the agent takes is not measured by an instant reward they return, but also the delayed reward they might yield, such as the agents can learn the actions they need to take and will lead to success in an unseen environment without a supervising entity. Reinforcement learning took the Markov Decision Processes' (MPD) problem of optimal control [8].

Reinforcement learning involves an agent exploring and learning on an unknown environment to achieve success. RL is based on the hypothesis that all goals are portrayed by the maximization of expected cumulative reward.

The Main Elements of an RL System are:

1. The agent or the learner.

2. The environment the agent interacts with.

3. The policy that the agent follows to take actions.

4. The reward signal that the agent observes upon taking actions

The goal of an RL algorithm is to find the action policy that maximizes the average value that it can get from every state of the environment. RL algorithms can be categorized in two ways: model-free and model-based. Model-free algorithms do not build models of the environment, as they are more like trial-and-error algorithms which run experiments with the environment where the agent is and using potential actions and obtain the optimal policy from it directly. These types of algorithms are either value-based or policy-based.

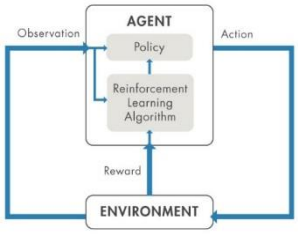

Figure 1. Reinforcement learning overview diagram [5].

1.1.2 Deep Q-Learning

Q- learning is a reinforcement learning algorithm which specializes in finding the action that yields the most reward, eventually learning a policy that maximizes the total reward. In Q-learning, we use a Q-table to keep track of that state, actions, and their respective rewards. The agent will try and explore different actions at different states through trial and error, then after learning each state and action, the agent will update the Q-Table with new Q-values. With this understanding, our agent will be able to take the sequence of actions that will yield the most reward [9].

Learning has its limitations though, imagining an environment with 10,000 states and 1,000 actions per state. This would result in the generation of 10 million cells, which can be too much. Thus, in Deep Q-learning, rather than using a Q-table, we use a neural network to estimate the Q-values function. The state is considered as the input and the Q-value of all possible actions from that state is generated as the output [3].

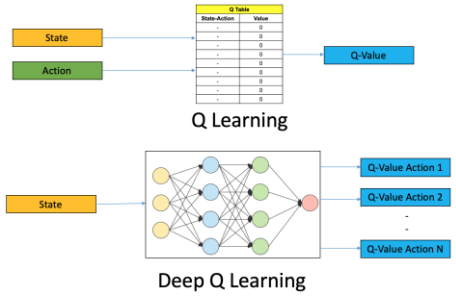

Figure 2. Comparison of Q learning and Deep Q learning [3].

1.1.3 Gym

Gym is a toolkit from OpenAi to be used for comparing and developing reinforcement learning algorithms. This gym is compatible with numerical computational library such as TensorFlow, Theano, and Stable baselines3. The gym library consists of sets of environment or test problems which can be used for trying and working out our reinforcement learning algorithm. These environments accept general algorithms since each environment shares interface [6].

1.1.4 LunarLander-v2

LunarLander-V2 is one of the test problems environments in the OpenAi gym toolkit. This environment aims to simulates the proper landing of a lunar rocket in 2-dimensional space. LunarLander-v2 has been included in the Box2D environments in which control task is continuous. The goal is to land properly in the landing pad and achieve 200 points by maneuvering the rocket by engaging the thrusters. For LunarLander-v2, it has 4 possible distinct action spaces for the agent: fire the main engine; fire the left engine; fire the right engine; and to do nothing. These actions are the ways to maneuver the rocket in the environment. LunarLander-v2 also has a box observable space where it displays the possible position of the agent in the environment. Both action and observation space are the attributes of type Space, in which they define the format of the valid actions and observations [6]. The rocket starts at the top of the screen.

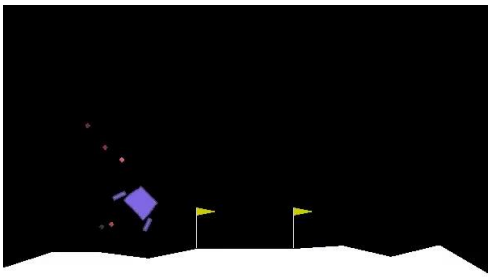

Figure 3. LunarLander-v2 [6].

1.2 RELATED STUDIES

1.2.1 Reinforcement Learning: State-of-the-Art (Adaptation, Learning, and Optimization)

There have been many progressions in the field of both A.I and the game development. Many would agree that A.I implementation is one of the components that greatly affect how gameplay, game experience, and game progression due to the various contributions it can add to table. Reinforcement learning and games have a long and mutually beneficial common history. From one side, games are rich and challenging domains for testing reinforcement learning algorithms. From the other side, in several games the best computer players use reinforcement learning. From a different point of view, games are an excellent testbed for RL research. Games are designed to entertain, amuse and challenge humans, so, by studying games, we can (hopefully) learn about human intelligence, and the challenges that human intelligence needs to solve. At the same time, games are challenging domains for RL algorithms as well, probably for the same reason they are so for humans: they are designed to involve interesting decisions. The types of challenges move on a wide scale, and we aim at presenting a representative selection of these challenges, together with RL approaches to tackle them [10].

1.2.2 Playing Atari with Deep Reinforcement Learning

The study presents the first deep learning model that successfully learned control policies directly from high-dimensional sensory input through reinforcement learning. It is true that performance of systems with hand-crafted features combined with linear value functions or policy representations heavily rely on the quality of the feature representation as mentioned in the paper. Hence, the researchers demonstrated in the paper a convolutional neural network (CNN) that can overcome such issues and learn successful control policies from raw video data in complex reinforcement learning environments, they trained it with a variant of Q-Learning algorithm with stochastic gradient descent that updates the weights. They applied their method to seven Atari 2600 games from the Arcade Learning Environment with no adjustments to the architecture or learning algorithms. The seven games they tested on were Beam Rider, Breakout, Enduro, Pong, Q*bert, Seaquest, and lastly Space Invaders. The model they proposed was able to produce outstanding results with 6 out of 7 of the games, greater

than those of previous iterations, and with no adjustments made. This paper showcases the immense potential reinforcement learning algorithms possess, especially when paired with models that have similar potential like CNN. Their implementation of reinforcement learning is proof that reinforcement learning can completely dominate simple environments and are more than capable of being used in more complex ones as training for greater purposes beyond just A.I in games [1].

1.2.3 Landing the Lunar Lander with Reinforcement Learning

In the Lunar Lander problem, the researchers were task with designing an algorithm that can successfully learn to land a rocket onto a landing pad by firing one of its 3 boosters (bottom, left, right). The rocket starts someplace above the landing pad at some known position and rotation at the start of each episode. To maximize the reward, the algorithm should fire three boosters in such a way that the rocket lands safely on the landing pad. To overcome the challenge, the researcher devised a Policy Gradient Algorithm. The gradient method's purpose is to discover neural network weights that characterize the optimum course of action for the rocket in each condition. The neural network takes in data about the current state (where the rocket is in space), processes it through a single hidden layer, and then calculates the likelihood that the rocket should fire. As the algorithm sees various states through each episode, it maps reward to the action taken each of those states and develops a policy that maximizes reward. On a regular basis, the weights of the neural network describing the policy are updated using the gradient between actual reward and expected reward (using the back-propagation algorithm). The research shown a successful landing using the policy gradient algorithm after over 3000 training episodes. This sort of method can be used to solve any multi-step problem with a clear end objective and several inputs at each phase [4].

2. OBJECTIVES

The general objective of this project is to develop and train a model with reinforcement learning using Deep Q-Learning algorithm that can land the vehicle on the landing pad and achieve the 200 points winning condition.

2.1 Instruments

The following hardware and software are used in the development of the project. Each are considered as instruments that allow the accomplishment of the project.

Table 1. Sony Vaio F Specifications

CPU	Intel(R) core (TM) i7-2720QM CPU @ 2.20GHz (8 CPUs)
RAM	8GB ram
DirectX version	Direct X 11
Operating system	Windows 7 Ultimate 64 bits
GPU	NVIDIA GeForce GT540M
Driver version	391.35
CUDA version	9.1.84 driver
CUDA cores	96
Memory available	16.8 GB free of 445GB

Table 2. Software instruments

ANACONDA	commercial edition
Test pad	Jupyter notebook
Web Browser	Microsoft Edge
Visual Studio Build Tools	2019
Programming language	Python
Library	Stable-Baseline3

2.1.1 Dependencies

Table 3 shows the software installed and imported that are required by the project program.

Table 3. Dependencies

box2d gym	Install	!pip install gym[box2d] pyglet
Stable-Baselines3	Install	!pip install stable-baselines3[extra]
Gym	Import	import gym
OS	Import	import os
DQN	Import	from stable_baselines3 import DQN
DummyVecEnv	Import	from stable_baselines3.common.vec_env import DummyVecEnv
Evaluate policy	Import	from stable_baselines3.common.evaluation import evaluate_policy

2.2 Model

The model trained will be used to predict the next action based on the current state of the agent in the environment. For this project, 4 models are trained with different time steps.

2.2.1 Framework

The environment for this project is the LunarLander-v2 from the box2d collection in the OpenAi gym toolkit. Gym toolkit allows comparing and training of different reinforcement learning algorithms. OpenAi environments are represented by Spaces.

2.2.2 Action Space

The action space for the LunarLander-v2 is discrete. Having 4 possible values corresponding to different actions. The line "env.action_space" can be used to check the type of action the agent can take.

Table 4. Discrete values that the agent can take and their corresponding actions

Actions
0: Do nothing
1: Fire left engine
2: Fire down engine
3: Fire right engine

2.2.3 Observation Space

The environment has a box observation space. The line "env.observation_space" can be used to check the type and properties of the observation space. Whenever the agent takes an action inside the environment. The environment returns the observations space in a Box(-inf, inf, (8,), float32) meaning an array with 8 values raging from -inf to inf with float32 format. Each value corresponds to the state and location of the agent.

Table 5. Contents of the array returned by the environment

horizontal position	vertical position	horizontal velocity	vertical velocity	angle velocity	angular velocity	left leg contact	right leg contact

2.2.4 Reward

The reward threshold for the LunarLander-v2 is 200. The landing pad is always at coordinates (0,0) and the rocket is always respawning at the top. Moving from the top of the screen to landing pad zero speed is about 100 to 140 points. If the lander moves away from landing pad it loses reward back. Firing the main engine adds -0.3 points each frame while each leg contact adds +10 points. The episodes finish if the lander crashes or comes to rest receiving additional -100 or +100 points. The goal of the lander is to land on the landing pad as faster as possible while using the engine minimally to position itself properly.

2.3 Training

4 models are trained for the project. The 1st model is trained with 10,000 time-steps. The 2nd model is trained with 100,000 time-steps. The 3rd is trained with 1,000,000 time-steps. The 4th model is trained with a stopping function. In every 10,000 episodes the model will be evaluated and when the mean score reaches 200, it will then stop the training. The general rule of thumb is that the longer the model is trained, the better the results. Which is not necessarily true and is still dependent on the problem. The Default hyperparameters of the DQN of stable_baselines3 are used.

Table 6. Training for each model

Model Name	Time steps	Duration
RL_model_10ksteps	10000	5-10 mins
RL_model_100ksteps	100,000 (99849)	20 mins
RL_model_1milsteps	1,000,000 (999659)	1 hr
best_model	1,608,111	2 hrs

2.3.1 Saving the model

The models are saved with the following file name conventions "RL_model_number of time steps". In total, 4 models are created and saved. The logs for each training are also saved having 4 training logs in total.

3. SYSTEM ARCHITECTURE

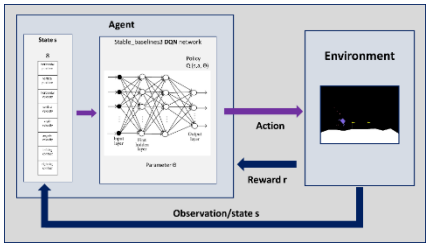

Figure 4. System Architecture.

The observation space, which is an array consisting of 8 values ranging from -inf to inf in float32 format, are used as inputs for the DQN. Each value in the array corresponds to the state of the agent in the environment. This serves as the eye for the agent. The DQN network uses the standard feed forward network which is a multilayer perceptron (MLP). The stable_baselines3 MlpPolicy is an alias for the "stable_baselines3.dqn.policies.DQNPolicy". This policy implements actor critic. The network consists of 8 inputs, 2 hidden layers, and an output of 4 nodes. The output is the optimize policy and from this, an action is made. Based from the agent's action a reward and observation space are returned by the environment. Figure 5 shows the parameters for the MlpPolicy that is used.

```
def __init__(
    self,
    observation_space: gym.spaces.Space,
    action_space: gym.spaces.Space,
    lr_schedule: Schedule,
    net_arch: Optional[List[int]] = None,
    activation_fn: Type[nn.Module] = nn.ReLU,
    features_extractor_class: Type[BaseFeaturesExtractor] = FlattenExtractor,
    features_extractor_kwargs: Optional[Dict[str, Any]] = None,
    normalize_images: bool = True,
    optimizer_class: Type[th.optim.Optimizer] = th.optim.Adam,
    optimizer_kwargs: Optional[Dict[str, Any]] = None,
):
```

Figure 5. Parameters of the stable_baselines3 MLP policy.

DQN is used for the LunarLander-v2 because the environment has a discrete action space and a box observation space. DQN works with environments with discrete actions and box, MultiDiscrete, MultiBinary, and Dict observation space. The DQN is compatible to the LunarLander-v2's spaces.

Table 4. Action and observation space for DQN

Space	Action	Observation
Discrete	✓	✓
Box	✗	✓
MultiDiscrete	✗	✓
MultiBinary	✗	✓
Dict	✗	✓

DQN in stable-baselines3 only implement vanilla Deep Q-Learning and has no extensions such as Double-DQN, Dueling-DQN and Prioritized Experience Replay. DQN has several available parameters but in this project only few are used.

```
Init signature:
DQN(
    policy: Union[str, Type[stable_baselines3.dqn.policies.DQNPolicy]],
    env: Union[gym.core.Env, stable_baselines3.common.vec_env.base_vec_env.VecEnv, str],
    learning_rate: Union[float, Callable[[float], float]] = 0.0001,
    buffer_size: int = 1000000,
    learning_starts: int = 50000,
    batch_size: Union[int, NoneType] = 32,
    tau: float = 1.0,
    gamma: float = 0.99,
    train_freq: Union[int, Tuple[int, str]] = 4,
    gradient_steps: int = 1,
    optimize_memory_usage: bool = False,
    target_update_interval: int = 10000,
    exploration_fraction: float = 0.1,
    exploration_initial_eps: float = 1.0,
    exploration_final_eps: float = 0.05,
    max_grad_norm: float = 10,
    tensorboard_log: Union[str, NoneType] = None,
    create_eval_env: bool = False,
    policy_kwargs: Union[Dict[str, Any], NoneType] = None,
    verbose: int = 0,
    seed: Union[int, NoneType] = None,
    device: Union[torch.device, str] = 'auto',
    _init_setup_model: bool = True,
)
```

Figure 6. Hyperparameters of DQN.

3.1 Markov Decision Process paradigm

Assume that the current state at any given time step is representative of all time-steps that have come before.

Symbols

S = all possible states (observation space).

s = individual state.

A = all possible actions (action space).

a = individual action.

R = reward distribution given (s,a).

P = transition probability distribution of St+1 given (s,a)

Given the last state and action taken, the transition probability is the distribution of the states that will get next.

γ = discount factor

How much we want the agent discount future reward. Agent tries to take action that maximizes its reward. The more the agent predicts future rewards, the less likely the reward is. Expected reward in the next time step is valued more than the next 2 to n time steps.

Objective: $\pi^* = \max(\sum_{t>0} \gamma^t r^t)$

Maximum expected cumulative reward. Maximizing reward and discounting future rewards. Policy that maps the states to the actions. Overall optimal policy. Overall policy of what kinds of actions the agent should be taking in those states.

Value function: V π (s)

Expected cumulative reward. Expected reward from being in a particular state.

Q-value function: Q π (s,a)

Consider state and action. Tells what is the maximum expected reward that we can get from the situation in being in a particular state taking a particular action. Then following the policy afterward.

Q*(s,a) ≈ Q (s,a, Θ)

Optimal Q value function given a particular state and action. The most expected reward we can get given a particular state and action and then following a policy. DQN to approximate/estimate the Q*(s,a). Θ = represents all weights in the function we are using to approximate the optimal Q. Could be any kinds of function like linear regression model. A Deep Neural network. Simple dense layer consists of weights and biases.

4. EVALUATION AND TESTING

Each model is tested in 10 episodes. The scores achieved by the model in each episode are added and the average is computed. The average will be the point of comparison for each model.

The block of code below will be used to test the model in 10 episodes. Instead of using random to take action in the environment, the trained model will be used to predict the action in the environment.

```
episodes = 10
for episode in range(1, episodes+1):
  obs = env.reset()
  done = False
  score = 0
  while not done:
    env.render()
    action, _ = model.predict(obs)
    obs, reward, done, info = env.step(action)
    score+=reward
  print('Episode:{} Score:{}'.format(episode, score))
env.close()
```

The code block below is used to evaluate the model and extract the average points of the model in 10 episodes.

```
evaluate_policy(model, env, n_eval_episodes=10, render=True)
env.close()
```

Below are the screenshots of tables from the saved logs of each training for the models. Tensorboard is used to generate the tables.

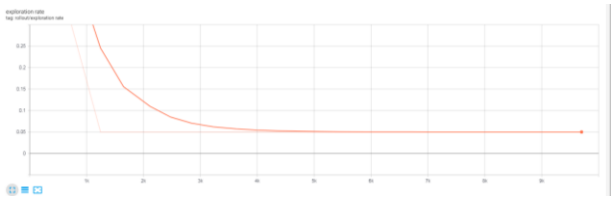

Figure 7. Exploration rate of model 1.

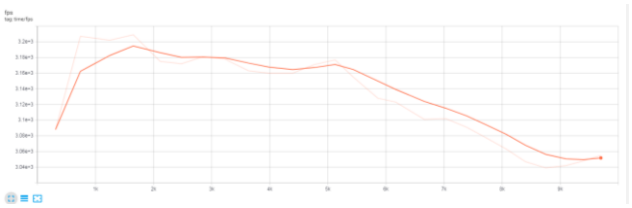

Figure 8 Fps of model 1.

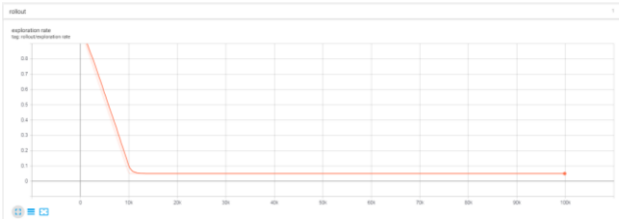

Figure 9. Exploration rate of model 2.

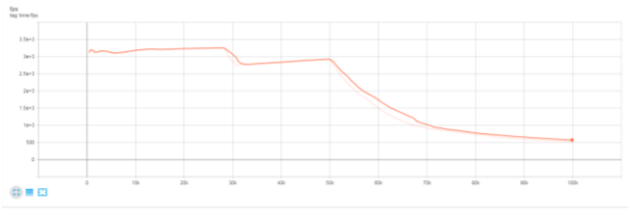

Figure 10. Fps of model 2.

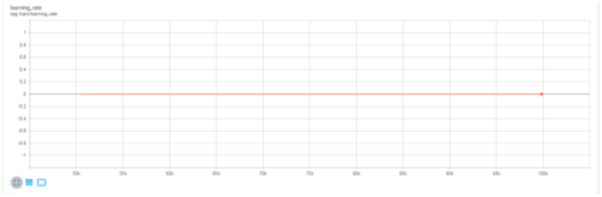

Figure 11. Learning rate of model 2.

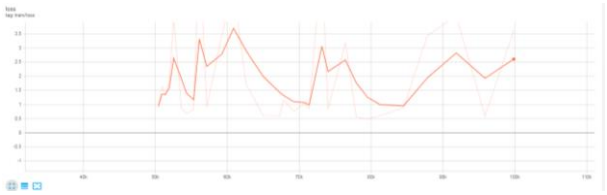

Figure 12. Loss of model 2.

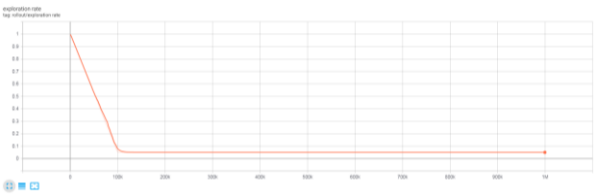

Figure 13. Exploration rate of model 3.

Figure 14. Learning rate of model 3.

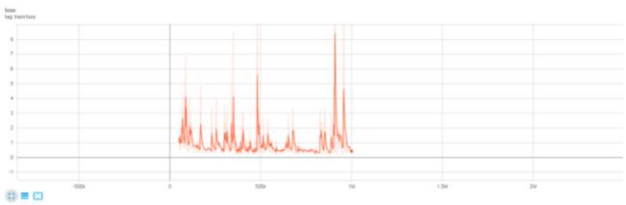

Figure 15. Loss of model 3.

Figure 16. Learning rate of model 4.

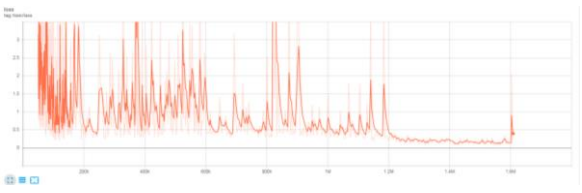

Figure 17. Loss of model 4.

Figure 18. Exploration rate of model 4.

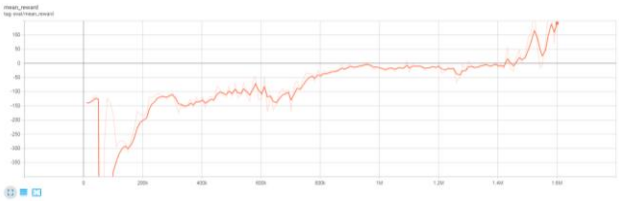

Figure 19. Mean reward of model 4.

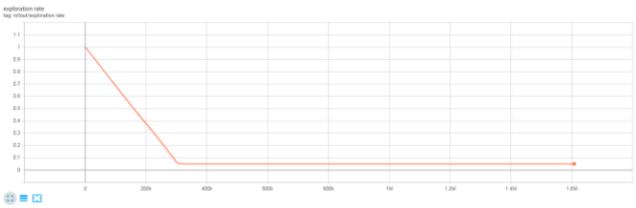

Figure 20. Mean episode length of model 4.

5. RESULTS

Table 5. Average performance of each model

Model number	Model Name	Total Time steps	Average Reward in 10 episodes
1	RL_model_10ksteps	10000	-551.6681
2	RL_model_100ksteps	100,000 (99849)	-128.91245
3	RL_model_1milsteps	1,000,000 (999659)	92.043365
4	best_model	1,608,111	168.22249

Table 6. Model 1 scores in 10 episodes

Episode 1	-776.68427
Episode 2	-507.07166
Episode 3	-325.51135
Episode 4	-842.6614
Episode 5	-774.11597
Episode 6	-470.26276
Episode 7	-357.2369
Episode 8	-484.80716
Episode 9	-441.19678
Episode 10	-467.47052

Table 7. Model 2 scores in 10 episodes

Episode 1	-134.2113
Episode 2	-87.697754
Episode 3	-136.74174
Episode 4	-131.59702
Episode 5	-127.45332
Episode 6	-143.19261
Episode 7	-177.23793
Episode 8	19.251215
Episode 9	-133.15
Episode 10	-125.24803

Table 8. Model 3 scores in 10 episodes

Episode 1	224.93268
Episode 2	37.89814
Episode 3	147.40558
Episode 4	-72.56583
Episode 5	173.10825
Episode 6	126.6047
Episode 7	175.01701
Episode 8	71.71902
Episode 9	187.38043
Episode 10	171.3242

Table 9. Model 4 scores in 10 episodes

Episode 1	207.50687
Episode 2	226.33224
Episode 3	243.96944
Episode 4	262.40552
Episode 5	280.97784
Episode 6	273.49655
Episode 7	228.87918
Episode 8	110.44519
Episode 9	249.19621
Episode 10	218.24292

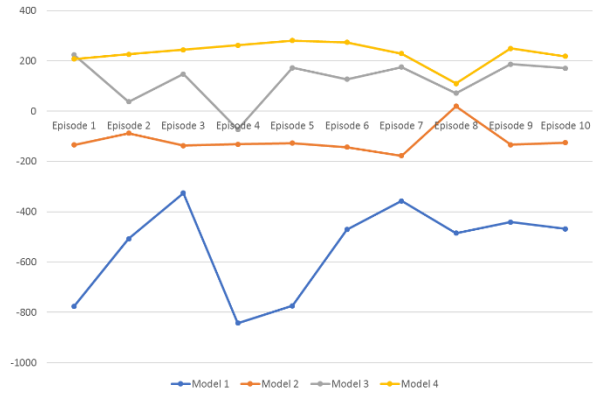

Figure 21. Comparison of scores of each model in 10 episodes.

The result shows that the higher the number of time-steps it took during training, the better the performance is. But since the target threshold is 200, a stopping function must be placed to stop the model on reaching higher values than the target 200. Model 4 with a stopping function turns to perform better than the others, however its score are reaching above 200 meaning the model is trying to maximize the reward and reach more than 200. This can be solved by adjusting the frequency it takes before calling the callback function during training.

6. CONCLUSION AND RECOMMENDATION

Based from the results, model 4 is the best model. Model 4 reach an average reward of 168 which is the closest to 200 among the other 3 models. This shows that DQN with a simple feed forward neural network works for LunarLander-v2 however it would take a million time-steps for it produce near 200 points result.

Based from the procedure and result of the project, it can be said that the performance of the model can still be improved. The following are recommended in order to improve the performance of the model.

- Train the model longer.
- Experiment on the number of episodes before evaluation.
- Tune and explore the hyperparameters.
- Try different algorithms or policies.
- Tune the number of frequencies in the callback.

7. REFERENCES

[1] Antonoglou, I., Graves, A., Kavukcuoglu, K., Mnih, V., Riedmiller, M., & Wierstra, D. 2013. Playing Atari with Deep Reinforcement Learning. https://arxiv.org/pdf/1312.5602.pdf.

[2] Applications of Machine Learning - Javatpoint. Www.Javatpoint.Com. Retrieved July 6, 2021, from https://www.javatpoint.com/applications-of-machine-learning.

[3] Choudhary, A. 2020. A Hands-On Introduction to Deep Q-Learning using OpenAI Gym in Python. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/.

[4] Lefkovitz, M. Project 2: Landing the Lunar Lander with Reinforcement Learning. Webflow. https://uploads-

ssl.webflow.com/5ca243f8aa6259298225af48/5d6603835e626f68cf2623c8_Lunar%20Lander%20Project%20Report.pdf

[5] Mathworks. Reinforcement Learning: A Brief Guide. MATLAB & Simulink. Retrieved July 6, 2021, from https://www.mathworks.com/company/newsletters/articles/reinforcement-learning-a-brief-guide.html.

[6] openai. Gym: A toolkit for developing and comparing reinforcement learning algorithms. Retrieved July 6, 2021, from https://gym.openai.com/docs/.

[7] Osiński, B., & Budek, K. 2018. What is reinforcement learning? The complete guide. Deepsense.Ai. https://deepsense.ai/what-is-reinforcement-learning-the-complete-guide/.

[8] Verma, P., & Diamantidis, S. 2021. What is Reinforcement Learning? Synopsys. https://www.synopsys.com/ai/what-is-reinforcement-learning.html.

[9] Wang, M. 2021. Deep Q-Learning Tutorial: minDQN - Towards Data Science. Medium. https://towardsdatascience.com/deep-q-learning-tutorial-mindqn-2a4c855abffc.

[10] Wiering, M., & Otterlo, V. M. 2012. Reinforcement Learning: State-of-the-Art (Adaptation, Learning, and Optimization, 12) (2012th ed., Vol. 12). Springer.