EMPOWERING SPACECRAFT WITH REINFORCEMENT LEARNING NAVIGATION

Minor project-II report submitted
in partial fulfillment of the requirement for award of the degree of

Bachelor of Technology
in
Computer Science & Engineering

By

MAKKENA BHARGAVI	(21UECS0352)	**(VTU19258)**
KALLE BHAVYA SREE	(21UECS0719)	**(VTU20697)**
PASALA RAKESH NARAYANA	(21UECS0451)	**(VTU19318)**

Under the guidance of
Dr.D.RAJESH, M.E., Ph.D.
PROFESSOR

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
SCHOOL OF COMPUTING

VEL TECH RANGARAJAN Dr. SAGUNTHALA R&D INSTITUTE OF SCIENCE & TECHNOLOGY

(Deemed to be University Estd u/s 3 of UGC Act, 1956)
Accredited by NAAC with A++ Grade
CHENNAI 600 062, TAMILNADU, INDIA

May, 2024

EMPOWERING SPACECRAFT WITH REINFORCEMENT LEARNING NAVIGATION

Minor project-II report submitted
in partial fulfillment of the requirement for award of the degree of

Bachelor of Technology
in
Computer Science & Engineering

By

MAKKENA BHARGAVI	(21UECS0352)	**(VTU19258)**
KALLE BHAVYA SREE	(21UECS0719)	**(VTU20697)**
PASALA RAKESH NARAYANA	(21UECS0451)	**(VTU19318)**

Under the guidance of
Dr.D.RAJESH, M.E., Ph.D.
PROFESSOR

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
SCHOOL OF COMPUTING

VEL TECH RANGARAJAN Dr. SAGUNTHALA R&D INSTITUTE OF SCIENCE & TECHNOLOGY

(Deemed to be University Estd u/s 3 of UGC Act, 1956)
Accredited by NAAC with A++ Grade
CHENNAI 600 062, TAMILNADU, INDIA

May, 2024

CERTIFICATE

It is certified that the work contained in the project report titled ”EMPOWERING SPACECRAFT WITH REINFORCEMENT LEARNING NAVIGATION” by MAKKENA BHARGAVI (21UECS0352), KALLE BHAVYA SREE (21UECS0719), PASALA RAKESH NARAYANA (21UECS0451) has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

Signature of Supervisor
Computer Science & Engineering
School of Computing
Vel Tech Rangarajan Dr. Sagunthala R&D
Institute of Science & Technology
May, 2024

Signature of Professor In-charge
Computer Science & Engineering
School of Computing
Vel Tech Rangarajan Dr. Sagunthala R&D
Institute of Science & Technology
May, 2024

DECLARATION

We declare that this written submission represents our ideas in our own words and where others' ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

MAKKENA BHARGAVI

Date: / /

KALLE BHAVYA SREE

Date: / /

PASALA RAKESH NARAYANA

Date: / /

APPROVAL SHEET

This project report entitled ”EMPOWERING SPACECRAFT WITH REINFORCEMENT LEARNING NAVIGATION” by MAKKENA BHARGAVI (21UECS0352), KALLE BHAVYA SREE (21UECS0719), PASALA RAKESH NARAYANA (21UECS0451) is approved for the degree of B.Tech in Computer Science & Engineering.

Examiners	**Supervisor**
	Dr.D.Rajesh, M.E., Ph.D., Professor

Date: / /
Place:

ACKNOWLEDGEMENT

We express our deepest gratitude to our respected **Founder Chancellor and President Col. Prof. Dr. R. RANGARAJAN B.E. (EEE), B.E. (MECH), M.S (AUTO),D.Sc., Foundress President Dr. R. SAGUNTHALA RANGARAJAN M.B.B.S.** Chairperson Managing Trustee and Vice President.

We are very much grateful to our beloved **Vice Chancellor Prof. S. SALIVAHANAN,** for providing us with an environment to complete our project successfully.

We record indebtedness to our **Professor & Dean, Department of Computer Science & Engineering, School of Computing, Dr.V. SRINIVASA RAO, M.Tech., Ph.D.,** for immense care and encouragement towards us throughout the course of this project.

We are thankful to our **Head, Department of Computer Science & Engineering, Dr.M.S.MURALI DHAR, M.E., Ph.D.,** for providing immense support in all our endeavors.

We also take this opportunity to express a deep sense of gratitude to our **Dr.D.Rajesh, M.E., Ph.D., Professor,** for his cordial support, valuable information and guidance, he helped us in completing this project through various stages.

A special thanks to our **Project Coordinators Mr.V.ASHOK KUMAR, M.Tech., Mrs.U. HEMAVATHI, M.E., Mrs.C. SHYAMALA KUMARI, M.E.,** for their valuable guidance and support throughout the course of the project.

We thank our department faculty, supporting staff and friends for their help and guidance to complete this project.

MAKKENA BHARGAVI	**(21UECS0352)**
KALLE BHAVYA SREE	**(21UECS0719)**
PASALA RAKESH NARAYANA	**(21UECS0451)**

ABSTRACT

Spacecraft navigation is a critical aspect of many space exploration missions, requiring precise control and decision-making to ensure safe and efficient movement through complex environments.Traditional navigation methods often rely on pre-programmed algorithms or human intervention, which may not be optimal for dynamically changing conditions.Empowering spacecraft navigation with reinforcement learning (RL) models, which have shown promise in learning complex control policies from experience.Exploring the effectiveness of various RL techniques, includes the Agentless Lunar Lander, Random Agent, Reflex Agent, and Deep Q Network (DQN) Agent models, in controlling spacecraft movement. The Agentless Lunar Lander model serves as a baseline for comparison, representing a simplistic approach to spacecraft navigation.We then introduce the Random Agent and Reflex Agent models, which leverage RL to explore and exploit different control strategies based on environmental feedback.Finally, the DQN Agent model, utilizes deep neural networks to approximate optimal action-value functions and make informed navigation decisions.

With extensive training, evaluation, and comparison of these RL models, assess their performance in spacecraft navigation tasks.Performance metrics such as average reward, episode length, and collision avoidance rates will be analyzed to determine the most effective RL model for spacecraft navigation.By showcasing the potential of RL in optimizing spacecraft navigation, this contributes to advancing autonomous control systems for future space exploration missions.The development of robust and adaptive navigation systems capable of navigating spacecraft through challenging environments with minimal human intervention.

Keywords:

Spacecraft navigation, Reinforcement learning, Autonomous control, Space exploration, Agent-based models, Deep Q Network (DQN), Control policies, Decision-making, Collision avoidance, Space missions.

LIST OF FIGURES

LIST OF ACRONYMS AND ABBREVIATIONS

AI	Artificial Intelligence
CLEP	China’s Lunar Exploration Program
CTX	Context Camera
DQN	Deep Q- Network
GPU	Graphical Processing Unit
HiRISE	High Resolution Imaging Science Experiment
LOLA	Lunar Orbiter Laser Altimeter
LRO	Lunar Reconnaissance Orbiter
ML	Machine Learning
MRO	Mars Reconnaissance Orbiter
NASA	National Aeronautics and Space Administration
RL	Reinforcement Learning
TPU	Tensor Processing Unit

TABLE OF CONTENTS

Chapter 1

INTRODUCTION

1.1 Introduction

Space exploration has always been a frontier of human ambition, with missions venturing into the vast unknown of outer space. A critical aspect of these endeavors is spacecraft navigation, which involves safely guiding vehicles through complex environments to reach their destinations. Traditionally, spacecraft navigation has relied on pre-programmed trajectories and manual intervention from ground control. However, with advancements in artificial intelligence (AI) and machine learning (ML), there is growing interest in leveraging autonomous navigation systems powered by reinforcement learning (RL) algorithms.

Reinforcement learning offers a promising approach to spacecraft navigation by enabling autonomous decision-making based on real-time feedback from the environment. RL algorithms learn optimal control policies through trial and error, interacting with the environment to maximize cumulative rewards. This paradigm shift towards autonomous navigation has the potential to revolutionize space exploration by enhancing operational efficiency, reducing reliance on ground control, and enabling spacecraft to adapt to dynamic and unforeseen circumstances.

This mainly focuses on the implementation and evaluation of RL models for spacecraft navigation. Specifically, exploring the efficacy of different RL algorithms, including Agentless Lunar Lander, Random Agent, Reflex Agent, and Deep Q Network (DQN) Agent, in controlling spacecraft movement. The Agentless Lunar Lander serves as a baseline for comparison, while the Random and Reflex Agents provide insights into the benefits of RL for navigation tasks. The DQN Agent, a sophisticated RL model, is employed to showcase the potential for advanced algorithms to optimize navigation performance.

Through extensive experimentation and analysis, aims to assess the capabilities and limitations of each RL model in spacecraft navigation scenarios. By evaluating factors such as control accuracy, fuel efficiency, collision avoidance, and adaptability to varying conditions, seeks to identify the most effective RL approach for autonomous spacecraft navigation. Ultimately, this research contributes to advancing the field of space exploration by demonstrating the feasibility and benefits of RL-based navigation systems for future missions.

1.2 Aim of the project

The aim of the project is to contribute to the advancement of space exploration by demonstrating the potential of RL-based navigation systems to enhance operational efficiency, reduce reliance on ground control, and enable adaptive spacecraft behavior in dynamic environments.

To investigate the application of reinforcement learning (RL) algorithms for spacecraft navigation.Specifically, evaluating the effectiveness of different RL models, including Agentless Lunar Lander, Random Agent, Reflex Agent, and Deep Q Network (DQN) Agent, in controlling spacecraft movement.Through experimentation and analysis, we determine the most efficient and reliable RL approach for autonomous navigation tasks.

1.3 Project Domain

The project domain encompasses spacecraft navigation, a critical aspect of space exploration involving guiding spacecraft through various maneuvers, orbital transfers, and landing procedures. Spacecraft navigation is essential for mission success, ensuring accurate trajectory planning, collision avoidance, and optimal fuel consumption. Reinforcement learning (RL) techniques offer promising solutions for spacecraft navigation by enabling autonomous decision-making based on past experiences and environmental feedback. By applying RL algorithms, spacecraft can adapt to dynamic and uncertain environments, enhance operational efficiency, and reduce reliance on ground control systems. This project explores the application of RL models, including Deep Q Networks (DQN), in spacecraft navigation tasks, aiming to evaluate their performance, scalability, and feasibility for real-world space missions. The insights gained from this research contribute to advancing autonomous spacecraft navigation capabilities and expanding the frontiers of space exploration.

1.4 Scope of the Project

The project scope for simulating lunar landing using reinforcement learning with multiple agents—specifically a random agent, reflex agent, and Q-network agent—entails a comprehensive exploration of varying approaches to autonomous lunar descent. Each agent will play a distinct role in this simulation, highlighting different levels of sophistication and learning capability within the framework of reinforcement learning.The random agent will serve as a baseline, representing the most basic form of decision-making. This agent will make landing decisions purely at random, offering a stark contrast to more advanced strategies. By incorporating the random agent, we can assess the effectiveness of more complex agents relative to a completely uninformed approach.

The reflex agent will introduce a more structured approach to decision-making. This agent will rely on predefined rules or heuristics to navigate the lunar landing scenario. These rules might involve simple if-then statements based on the agent's sensory inputs (such as altitude, velocity, and terrain characteristics). The reflex agent's performance will shed light on the benefits and limitations of rule-based strategies in this context.The Q-network agent represents the cutting-edge of the project, leveraging deep reinforcement learning techniques. This agent will be trained using a Q-learning algorithm with a neural network as a function approximator. The Q-network agent learns to map state-action pairs to expected rewards, enabling it to make informed decisions based on past experiences. The training process will involve simulating lunar landing scenarios repeatedly, allowing the agent to refine its decision-making skills over time through trial and error.

The project's objectives will include assessing each agent's performance based on key metrics such as landing accuracy, fuel efficiency, and computational efficiency. Additionally, the scalability and adaptability of each approach will be evaluated to determine the feasibility of deploying such agents in real-world lunar missions. Insights gained from this project will contribute to the broader field of autonomous space exploration, showcasing the potential of reinforcement learning in addressing complex decision-making tasks.

Chapter 2

LITERATURE REVIEW

[1] C. Wang et al., [2019] proposed a detailed new type of lander with magnetorheological fluid dampers implemented as the primary struts, which can absorb the impact energy from landing. Its damping force is controlled by semi-active controlled currents based on the derived hydrodynamics of magnetorheological fluid dampers and can adjust to practical landing conditions. The simulation models of landers with the aluminum honeycomb and semi-active control are constructed in MSC Adams separately to compare their landing performances. Their simulation results show that this new type of lander can effectively reduce the largest acceleration by 26.18 percent under the largest acceleration response condition and the largest compression by 11.77 percent under the largest compression of primary struts condition. Its performance for more inclined and both smoother and rougher landing surfaces is better than that of traditional passive landers. It is believed that the proposed new type of lander is capable of landing on more complex and unexplored terrains for future lunar explorations.

[2] D. E. Smith et al., [2015] has developed The Lunar Orbiter Laser Altimeter (LOLA), an instrument on the payload of NASA's Lunar Reconnaissance Orbiter spacecraft (LRO) (Chin et al., in Space Sci. Rev. 129:391–419, 2007). The instrument is designed to measure the shape of the Moon by measuring precisely the range from the spacecraft to the lunar surface, and incorporating precision orbit determination of LRO, referencing surface ranges to the Moon's center of mass. LOLA has 5 beams and operates at 28 Hz, with a nominal accuracy of 10 cm. Its primary objective is to produce a global geodetic grid for the Moon to which all other observations can be precisely referenced.

[3] J. Liu et al., [2021] has developed the principles, process, method and result of landing site selection of China's Lunar Exploration Program (CLEP), and then analyzed the support of the selected landing sites to the corresponding lunar research and also pointed out the outcomes that could possibly contribute to the key lunar questions on the basis of the selected landing sites of CE-4 and CE-5. Finally, this

approach analyzed the development trend of China's follow-up lunar landing missions, and suggested that the South Pole Region of the Moon could be the landing site of high priority for the future CE missions.

[4] K. Iiyama et al., [2021] has proposed a new integrated framework for identifying safe landing locations and planning in-flight divert maneuvers. The state-of-the-art algorithms for landing zone selection utilize local terrain features such as slopes and roughness to judge the safety and priority of the landing point. However, when there are additional chances of observation and diverting in the future, these algorithms are not able to evaluate the safety of the decision itself to target the selected landing point considering the overall descent trajectory. In response to this challenge, we propose a reinforcement learning framework that optimizes a landing site selection strategy concurrently with a guidance and control strategy to the target landing site. The trained agent could evaluate and select landing sites with explicit consideration of the terrain features, quality of future observations, and control to achieve a safe and efficient landing trajectory at a system-level. The proposed framework was able to achieve 94.8 percentage of successful landing in highly challenging landing sites where over 80 percentage of the area around the initial target lading point is hazardous, by effectively updating the target landing site and feedback control gain during descent.

[5] K. Sacksteder et.al., [2017] has developed In-Situ Resource Utilization is the collection, processing, storing and use of materials encountered in the course of human or robotic space exploration that replace materials that would otherwise be brought from Earth to accomplish a mission critical need at reduced overall cost and risk. For the first time, a mainstream ISRU technology development program has been established by NASA to incorporate this approach into extended duration exploration of the moon and Mars. The program includes the development of capabilities to perform excavation and collection of lunar regolith, including imbedded volatile materials, and the production of oxygen to supply propulsion and life support consumables during the extended human missions currently planned to begin in the early 2020's.

[6] L. Ma, Z. Shao et al., [2016] presented a direct trajectory optimization framework for powered descent and landing of reusable rockets with restartable engines. The rocket engine is throttleable, but the number of engine restart is limited. Given the engine characteristics, a unified problem formulation with a multi-phase structure

is established. The finite-element collocation approach is utilized to transcribe the established problem into a nonlinear programming problem solved by the interior-point optimizer. Simulation results illustrate that the presented framework exhibits unification and adaptability to effectively deal with trajectory optimization problems of reusable rockets with limitedly restartable engines.

[7] M. Golombek et al., [2017] The selection of the Discovery Program InSight landing site took over four years from initial identification of possible areas that met engineering constraints, to down selection via targeted data from orbiters (especially Mars Reconnaissance Orbiter (MRO) Context Camera (CTX) and High-Resolution Imaging Science Experiment (HiRISE) images), to selection and certification via sophisticated entry, descent and landing (EDL) simulations. Constraints on elevation latitude, ellipse size and a load bearing surface without thick deposits of dust, severely limited acceptable areas to western Elysium Planitia.

[8] M.Pontani et al., [2010] The particle swarm optimization technique is a population-based stochastic method developed in recent years and successfully applied in several fields of research. It represents a very intuitive (and easy to program) methodology for global optimization, inspired by the behavior of bird flocks while searching for food. The particle swarm optimization technique attempts to take advantage of the mechanism of information sharing that affects the overall behavior of a swarm, with the intent of determining the optimal values of the unknown parameters of the problem under consideration. In this research the method is applied to a variety of space trajectory optimization problems, i.e., the determination of periodic orbits in the context of the circular restricted three-body problem, and the optimization of (impulsive and finite thrust) orbital transfers. Despite its simplicity and intuitiveness, the particle swarm algorithm proves to be quite effective in finding the optimal solution to all of the applications considered in the paper, with great numerical accuracy.

[9] N. Remesh et al., [2016] analyzed the problem of soft landing on Moon is formulated and solved using different solution schemes. The solution scheme consists of one of the two approaches i) direct ii) indirect and one of different optimization techniques (viz. gradient based and gradient free techniques). Gradient free (Particle Swarm optimization and Differential Evolution) and gradient based optimization techniques are explored to solve the problem for direct and indirect approaches. In indirect approach, the optimal control problem is transformed into a two point bound-

ary value problem (TPBVP) using Pontryagin's minimum principle and the appropriate initial costates are selected using different optimization techniques. The performances of these optimization techniques are evaluated for the indirect approach for soft landing trajectory design problem. In direct approach, it is transformed into a non linear programming (NLP) problem and is solved by using an optimizer. The performance of the direct schemes is compared with indirect schemes. A scheme based on the indirect approach and differential evolution technique is evaluated superior to other options for the soft landing trajectory problem. Also the trajectory design is carried out with different constant thrust levels and their sensitivities are analyzed.

[10] N. Remesh et al., [2017] researched for safe and precise lunar landing it is required to guide the landing module to a pre-specified target location within specified accuracy limits by bringing down the landing velocity to near zero level. To achieve this, the thrust vector needs to be steered in an optimum way to minimize the fuel consumption. In this paper, a set of lunar centered 3 dimensional cartesian coordinates are used as state variables to represent the landing module translational dynamics and to ensure the soft landing at a specified target location. The precise landing trajectory design problem is formulated as an optimal control problem with constant thrust level and it is transformed into a two point boundary value problem(TPBVP) using the indirect approach based on Pontryagin's principle. The solution procedure of this TPBVP uses Differential Evolution, an evolutionary optimization technique. For real-time trajectory planning, few guidance laws are evaluated against the optimal control law obtained. The soft landing trajectory is propagated with control angles computed by the guidance algorithm to achieve the target conditions. The optimum fuel requirements with constant thrust to achieve different specified target conditions using the optimal control law are compared with fuel requirements obtained from simulated trajectory with guidance algorithm. Also the trajectory design is carried out with different constant thrust levels and the performance of the guidance laws is analyzed.

[11]R. Cortes-Martinez et al., [2021]addressed the problem of powered descent control of a lunar lander during its final landing phase, where soft and precise landing at the target cite is required. In the literature, this problem has been solved considering a fully actuated lunar lander. Recently, landers are designed as complex systems with several onboard payloads and subsystems for completing demanding

mission tasks. There are increasing constraints on mass budget and space availability onboard a lander. In order to meet these requirements, a novel controller using a single thruster is proposed. The proposed controller has three control inputs (thrust and two gimbaled angles) to control the six degrees of motion of a lander; these control inputs are non-affine. The proposed controller is designed based on variable structure control technique augmented with a high order filter, an immersion and invariance-based mass estimator and a sliding mode angular velocity observer. The stability analysis using Lyapunov theory and the results of the numerical simulations along with Monte Carlo simulations of the system show that the precise landing of the lunar lander using a single thruster is feasible.

[12] S. Chen et al., [2021] has developed the first deep learning method to identify multiple lunar features simultaneously for potential energy source discovery. Based on the state-of-the-art deep learning model, High Resolution Net, this model can efficiently extract semantic information and high-resolution spatial information from the input images, which ensures the performance for recognizing the lunar features. With a novel framework, this method can recognize multiple lunar features, such as craters and riles, at the same time. They also used transfer learning to handle the data deficiency issue. With comprehensive experiments on three datasets to show the effectiveness of the proposed method.

[13] S. Kumar Choudhary et al., [2019] investigate the optimal strategy for trajectory design to ensure the soft landing of the lander from the Lunar parking orbit to the lunar surface with minimum consumption of fuel. The trajectory design of lunar lander is studied via two cases by formulating the optimal control problems, where specific requirements of this soft landing problem are all incorporated in the problem formulation. To analyse the proposed optimal strategies for a soft landing the paper briefly illustrates the numerical simulation results and it shows that the required velocity for the soft landing is achieved with minimum fuel consumption. The investigated computational methods for the optimal solutions of the Moon-Lander problem in both two cases are conceptually simple and efficient.

[14] W. Li, X. Wang, et al., [2020] has designed soft landing trajectory, from lunar parking orbit to the surface of Moon. To ensure vertical landing using minimum fuel, the landing trajectory design is done in four phases: Hohmann transfer phase, power descend phase, attitude maneuver and vertical descend phase. The fuel optimization is done in the power descend phase. Then the attitude of the probe is adjusted and the

probe is made vertical during the attitude maneuver. In final vertical descend phase the steering angle is kept as 90 degree to land the probe vertically. Optimal soft landing trajectory is designed using indirect approach of optimization. The indirect approach uses, Pontryagin's minimum principle to transfer optimal control problem to Two Point Boundary Value (TPBV) problem, and to express the control variable as a function of costate variables. Then the state and costate equations are integrated at optimum control angle, with the initial costate found by the different optimization techniques, to get the optimal trajectory.

[15] X.-L. Liu et al., [2018] designed the optimal control law to ensure the soft landing of the lander with the least fuel consumption. It is formulated as a constrained optimal control problem, where specific requirements of this soft landing problem are all incorporated in the problem formulation. Then, by utilizing the specific features of the problem, this optimal soft landing problem is transformed into an equivalent standard optimal control problem subject to continuous state inequality constraints. A computational method is developed, based on the control parameterization in conjunction with a time scaling transform and the constraint transcription method, to design the optimal controller for this new constrained optimal control problem. Numerical results are presented for illustration.

Chapter 3

PROJECT DESCRIPTION

3.1 Existing System

The NASA-sponsored ”Moonjs” platform is a notable system designed for simulating lunar lander missions. Developed as an open-source, web-based simulation, Moonjs offers users an interactive experience of navigating and landing a spacecraft on the Moon. This platform is particularly valuable for educational purposes, providing students, researchers, and enthusiasts with a hands-on understanding of the complexities involved in lunar landings.

Moonjs incorporates realistic physics simulations, including lunar gravity, thrust dynamics, and terrain interactions, to create an immersive and educational experience. Users can experiment with different landing strategies, understand the challenges faced by actual lunar missions, and gain insights into spacecraft control and navigation.

One of the key features of Moonjs is its accessibility. Being web-based, the platform can be accessed from any standard web browser without the need for specialized software or hardware. This accessibility promotes widespread use and engagement, making it an effective tool for STEM education and outreach initiatives.

Furthermore, Moonjs serves as a practical testing ground for developing and refining landing algorithms and strategies for future lunar missions. By simulating various scenarios and conditions, researchers can evaluate and optimize spacecraft performance and mission success criteria.

Advantages of the Existing system

1. Moonjs provides a highly realistic simulation environment, accurately depicting the challenges and dynamics of landing a spacecraft on the Moon

2. Moonjs is easily accessible to a wide audience. Users can access the simulation through a web browser without the need for specialized software or hardware.

3. Moonjs encourages collaboration and innovation within the space exploration community. Developers can contribute to the platform's codebase, improving its functionality.
4. It provides a platform for testing and refining landing algorithms and autonomous landing systems.

3.2 Proposed System

The economic feasibility of a lunar lander reinforcement learning project depends on several factors. Initially, the costs associated with hardware, including lunar lander prototypes and simulation environments, must be considered. These expenses could include development kits, computational resources, and specialized components. Additionally, personnel costs for skilled engineers and researchers proficient in reinforcement learning and robotics are significant.

The project's economic viability also depends on potential benefits and applications. If successful, this research could lead to advancements in autonomous navigation systems for space missions or commercial applications such as asteroid mining or lunar exploration.

Moreover, attracting funding or partnerships from space agencies, private companies, or investors interested in space technology can be crucial for sustaining the project. Balancing these costs against potential benefits and funding opportunities will determine the economic feasibility of pursuing a lunar lander reinforcement learning initiative.

The proposed system involves training various agents to navigate a lunar lander simulation using different approaches comparing the performance of four agents: an agentless lunar lander, a random agent, a simple reflex agent, and a deep Q-network (DQN) agent.The agentless lunar lander serves as the baseline, allowing observation of the lunar lander's behavior without any intelligent control. This scenario helps understand the complexity and challenges of the task. The random agent operates purely on chance, making random decisions without learning from previous experiences. This agent demonstrates the minimal level of performance achievable without any strategy.The simple reflex agent utilizes predefined rules to make decisions based on the current state of the lunar lander. This approach reflects a basic form of intelligence by reacting to immediate environmental cues.The deep Q-network (DQN) agent is a more sophisticated model trained using reinforcement learning. It learns to make decisions by maximizing cumulative rewards through trial and error.

The DQN agent uses a neural network to approximate the optimal action-value function, enabling it to navigate the lunar lander more effectively over time. Each agent's performance is evaluated based on criteria such as successful landings, fuel efficiency, and speed of convergence to optimal behavior. By comparing these agents, researchers gain insights into the effectiveness of different approaches to lunar lander control. The DQN agent typically outperforms the other agents due to its ability to learn complex strategies and adapt its behavior based on feedback from the environment.

Advantages of the proposed system

1. This approach serves as a baseline, relying solely on physics simulations without intelligent control. It demonstrates the necessity of agents for complex tasks like lunar landing.

2. Despite its simplicity, a random agent provides insights into the baseline performance, highlighting the significant improvement achieved with intelligent agents. Its advantage lies in exposing the necessity of deliberate decision-making over random actions.

3. Simple Reflex Agent agent responds to the environment with predefined rules, offering a step toward intelligence with programmed behaviors. Its advantage is clarity; its limitations, however, are evident in complex scenarios.

4. DQN Agent advantage lies in adaptability, allowing it to handle intricate situations by learning from experience.

3.3 Feasibility Study

3.3.1 Economic Feasibility

The economic feasibility involves several considerations. The cost factors include hardware (computational resources, sensors), software development (algorithms, simulations), and personnel (engineers, researchers). The potential benefits are assessed, such as advancements in AI for autonomous space missions, which can reduce human intervention and operational costs in future lunar programs and the project's timeline and milestones need careful planning to align with available funding and projected outcomes. Leveraging open-source technologies and partnerships with academic institutions or space agencies can mitigate costs and accelerate progress. Overall, while the initial investment may be significant, the long-term economic value lies in developing sophisticated AI for space exploration, potentially leading to cost-efficient and sustainable lunar missions.

3.3.2 Technical Feasibility

The technical feasibility of a Lunar Lander reinforcement learning project involving multiple agents, can be assessed based on various factors. Simulating an agentless lunar lander, essentially a baseline without decision-making, is straightforward as it involves basic physics calculations. Secondly, a random agent, which makes decisions randomly, is also technically feasible and serves as a basic control. Moving to more complex agents, implementing a simple reflex agent that responds to specific states with predetermined actions is feasible within reasonable computational resources. However, a DQN agent, which learns a policy through deep neural networks, presents greater challenges. The feasibility depends on computing power and data requirements, as training a DQN demands substantial computational resources and may necessitate access to high-performance computing or cloud services.

3.3.3 Social Feasibility

Undertaking this project involving various agents and presents several social feasibility considerations.Promotes public engagement with cutting-edge technologies by showcasing advancements in AI and robotics. It can captivate public interest in space exploration and AI applications, potentially fostering STEM interest among students and enthusiasts.The project's success could inspire future collaborations and research initiatives aimed at solving complex challenges in space exploration. Overall, while this project offers exciting prospects for AI development, addressing societal concerns and fostering responsible innovation will be pivotal for its social feasibility and impact.

3.4 System Specification

3.4.1 Hardware Specification

• System :Intel i3 or Above

• Hard Disk : 40 GB

• Monitor : 14’ Colour Monitor

• GPU: NVIDIA GTX 1060

• Ram : 16GB

• Storage : 500 GB SSD

3.4.2 Software Specification

• Python Version : Python 3.7

• Reinforcement Learning Libraries : PyTorch, TensorFlow, NumPy

• Simulation Environment : OpenAI Gym

3.4.3 Standards and Policies

Anaconda Prompt
Anaconda prompt is a type of command line interface which explicitly deals with the ML(MachineLearning) modules.And navigator is available in all the Windows,Linux and MacOS.The anaconda prompt has many number of IDE’s which make the coding easier. The UI can also be implemented in python.
Standard Used: ISO/IEC 27001

Google Colaboratory
Colab is a hosted Jupyter Notebook service that requires no setup to use and provides free access to computing resources, including GPUs and TPUs. Colab is especially well suited to machine learning, data science, and education.
Standard Used: ISO/IEC 27001

Chapter 4

METHODOLOGY

4.1 Reinforcement Learning Architecture for Lunar Lander

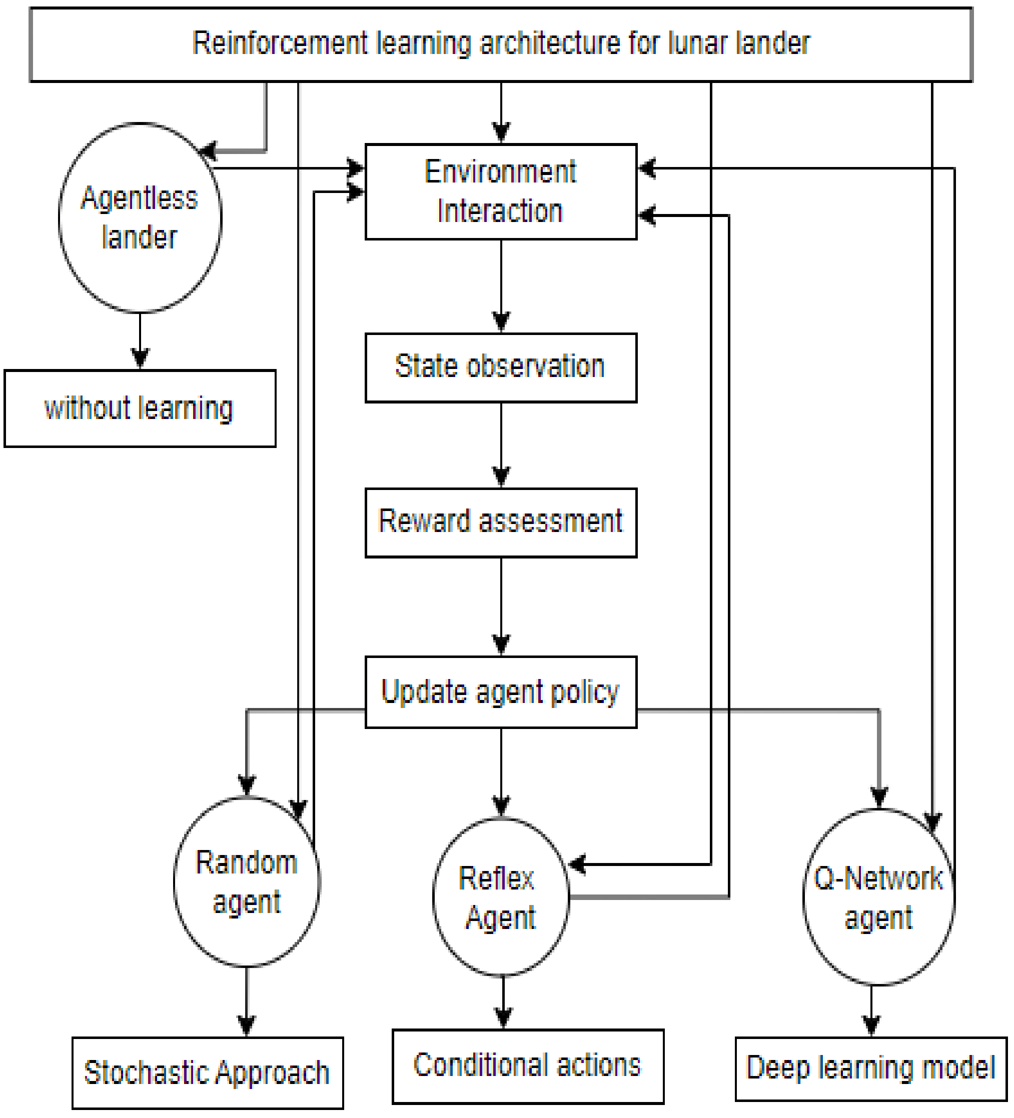


Figure 4.1: **Lunar Lander Architecture**

The above Figure 4.1 represnts the architecture diagram that depicts a reinforcement learning architecture for a lunar landing using a Q-network. The agent in this case is the lunar lander itself. It receives observations about its environment (position, velocity, etc.) from the environment and takes actions (firing thrusters, etc.) to influence its movement. The environment represents the lunar surface and the physical laws that govern the lander's motion. It provides the agent with observations about its state and a reward signal based on the outcome of its actions. In a successful landing scenario, the reward would be high, while a crash would result in a negative reward. The Q-network is the core of the reinforcement learning system. The Q-value represents the expected future reward the agent can get by taking a specific action in a particular state. In this case, it would likely consist of past experiences of the agent interacting with the environment, including the observations, actions, and corresponding rewards. The experience replay buffer is a storage unit that holds the agent's past experiences. By randomly sampling experiences from this buffer during training, the Q-network can learn from a more diverse set of scenarios and improve its generalization capability. The relay buffer acts as an intermediary between the agent and the training data. It most likely stores the most recent experiences of the agent and feeds them into the experience replay buffer.

4.2 Design Phase

4.2.1 Data Flow Diagram

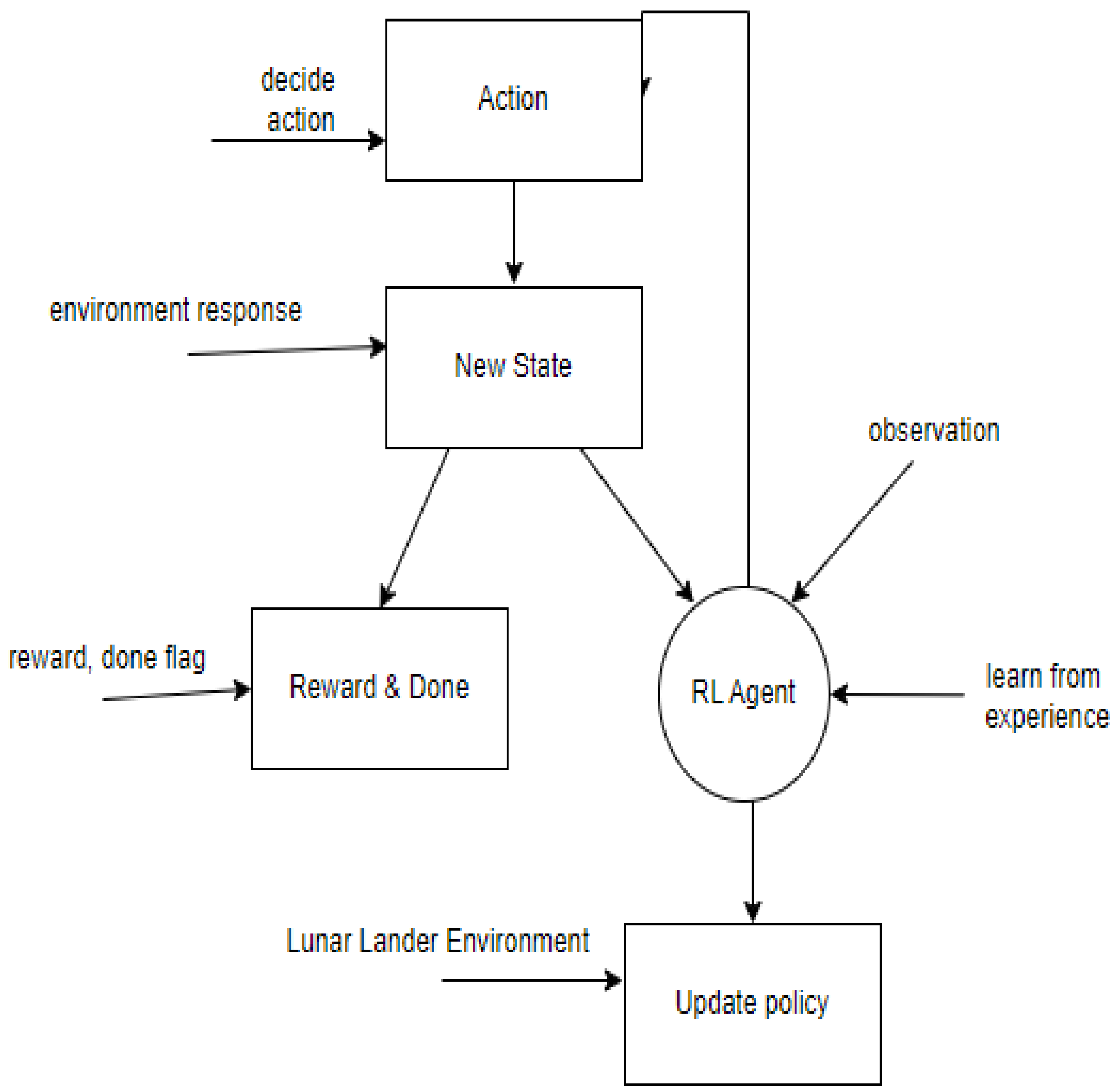


Figure 4.2: **Data Flow Diagram of Lunar Lander**

The above Figure 4.2 represents the data flow diagram given involves multiple agents collaborating for successful descent.Through iterative learning, the system refines its decision-making process to achieve safe and precise lunar landing. Feedback loops between agents and the environment ensure adaptive behavior,

4.2.2 Use Case Diagram

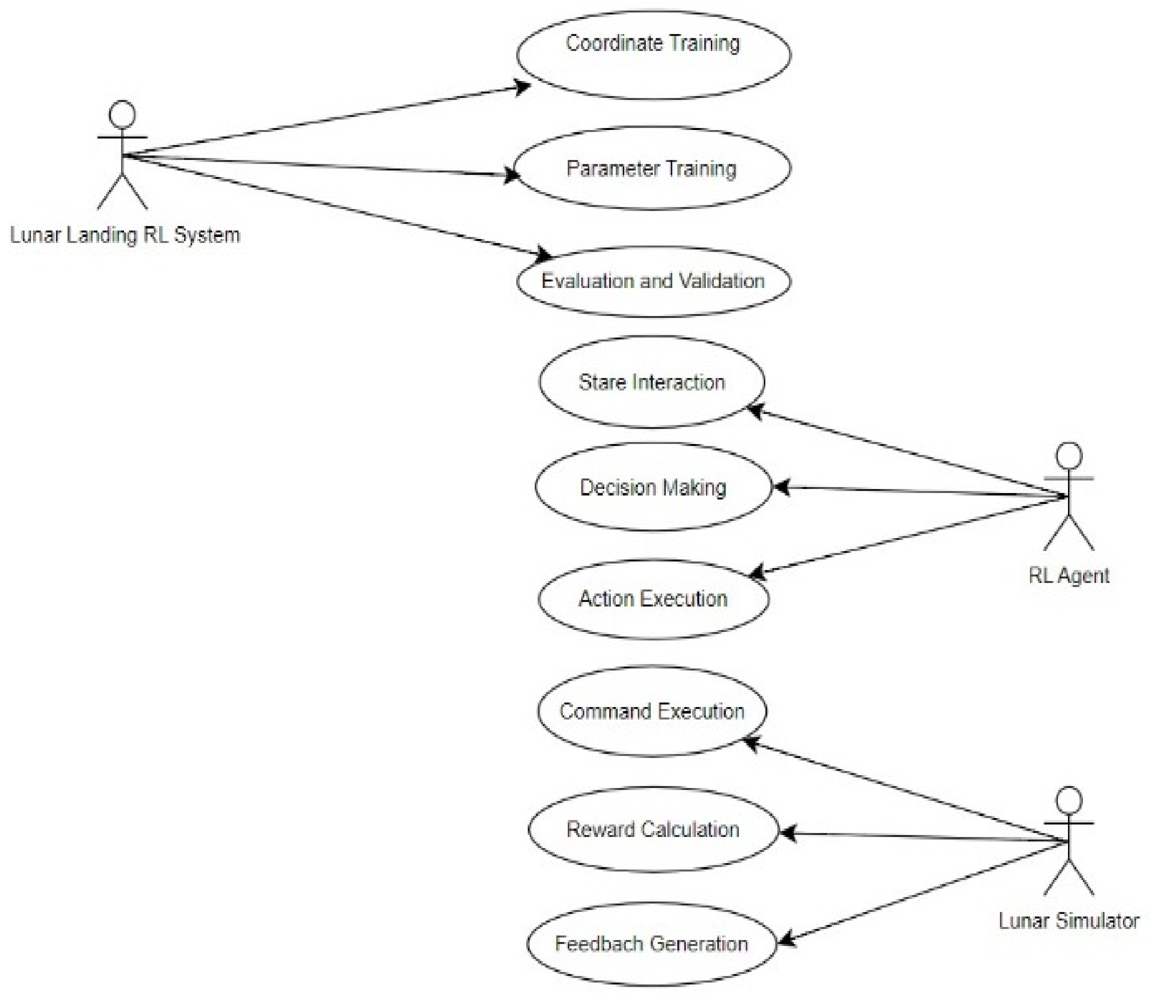


Figure 4.3: **Use Case Diagram of Lunar Lander**

The above Figure 4.3 represents the use case diagram illustrates the process of lunar landing by a spacecraft utilizing deep reinforcement learning (DRL) with various agents.DRL agents including perception, decision-making, and control modules enable the spacecraft to autonomously navigate, adjust trajectory, and land safely on the lunar surface. Through continuous interaction and learning, the spacecraft optimizes landing strategies based on environmental feedback, ensuring successful missions while minimizing risks.

4.2.3 Class Diagram

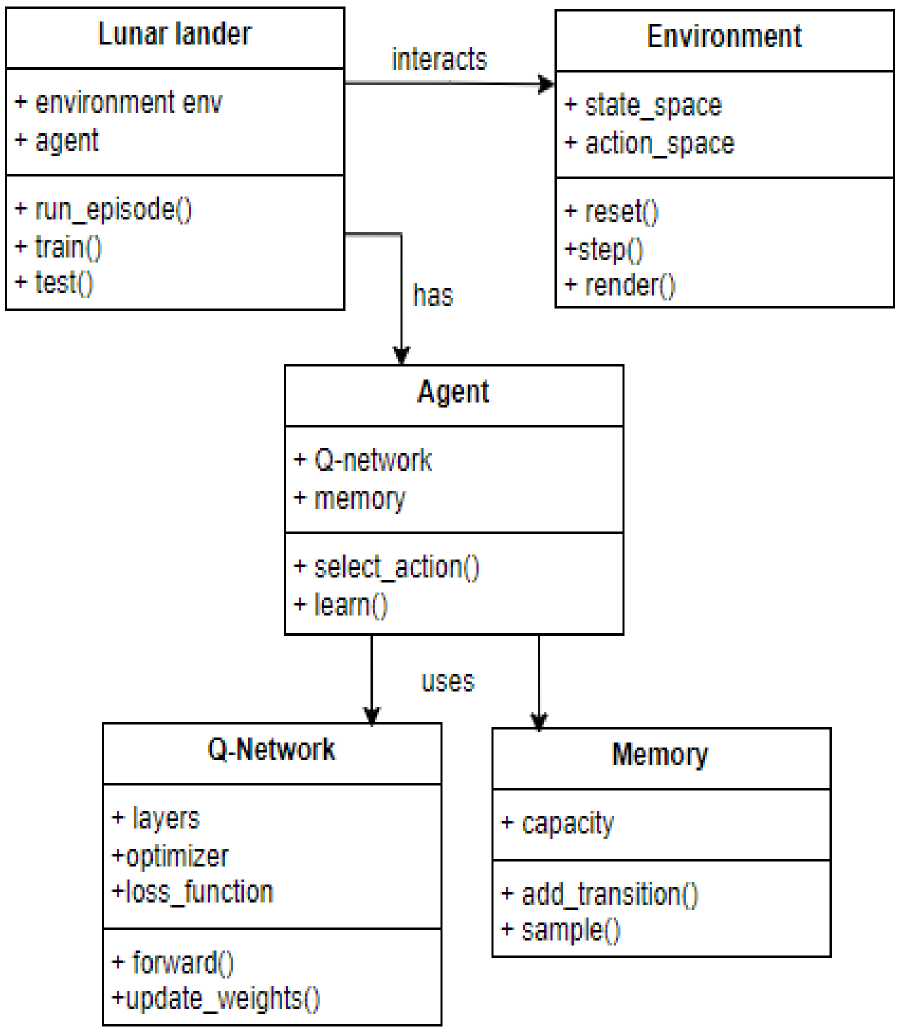


Figure 4.4: **Class Diagram of Lunar Lander**

The above Figure 4.4 represents the class diagram that depicts a reinforcement learning (RL) system for a lunar lander. The core components are the environment and agent.The environment defines the state space (possible states the lander can be in) and the action space (actions the agent can take). The agent interacts with the environment through these methods and a step() method that takes an action and returns the resulting state and reward. The agent also has methods to train and test its policy, and potentially a way to visualize the environment. Internally, the agent uses a Q-Network to learn a policy for selecting actions. The Q-Network utilizes a memory to store past experiences and a loss function to measure the difference between predicted and actual rewards.

4.2.4 Sequence Diagram

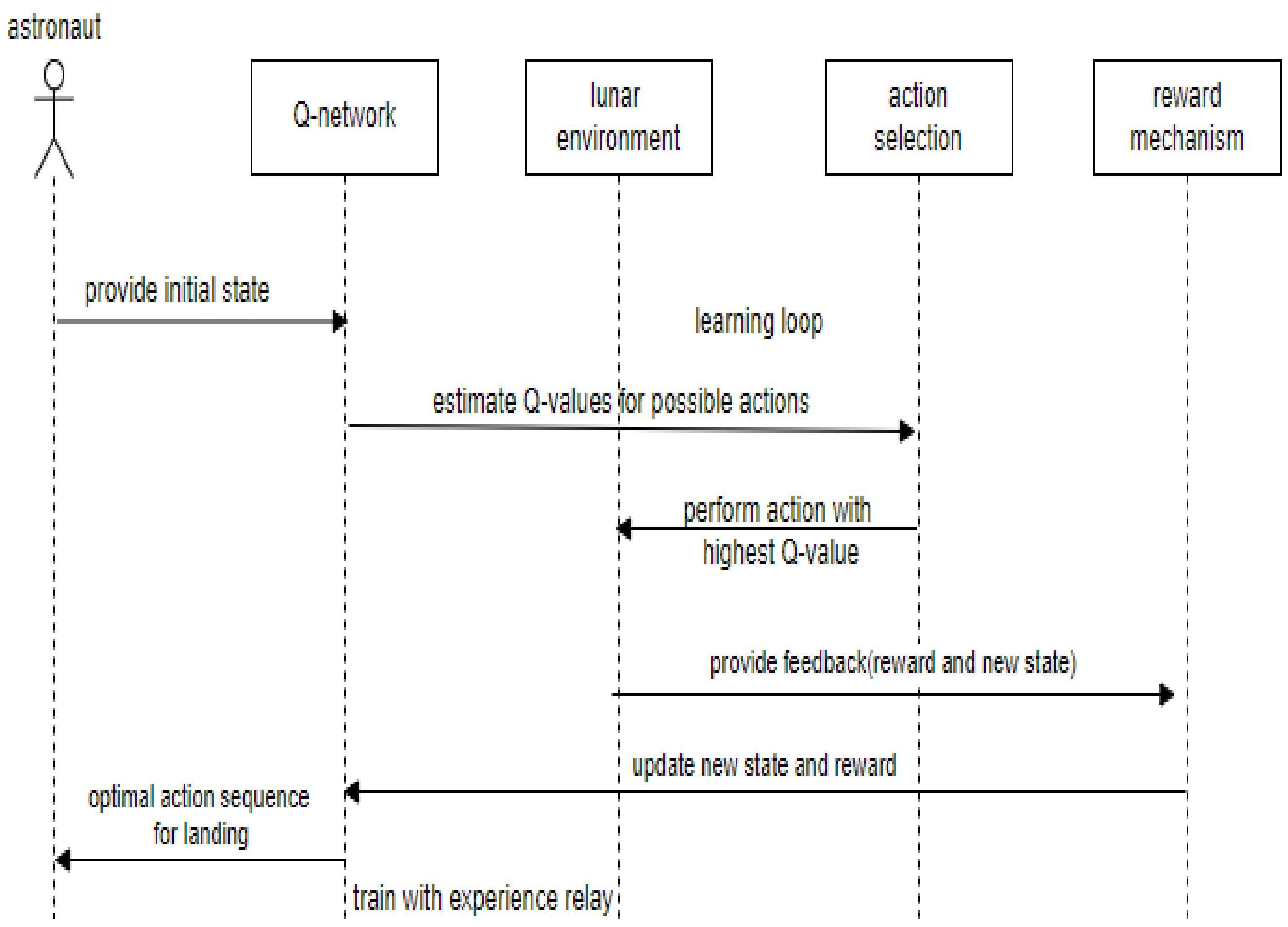

astronaut
Q-network
lunar environment
action selection
reward mechanism
provide initial state
learning loop
estimate Q-values for possible actions
perform action with highest Q-value
provide feedback(reward and new state)
update new state and reward
optimal action sequence for landing
train with experience relay

Figure 4.5: **Sequence Diagram of Lunar Lander**

The above Figure 4.5 represents sequence diagram that outlines the iterative process of lunar landing using reinforcement learning with a Q-network. The astronaut provides the initial state of the lunar lander, which is then used by the system to estimate Q-values for available actions. Based on these Q-values, an action is selected, and the environment simulator applies it, updating the lander's state and generating a new observation and reward. The system then uses this information to update the Q-network, incorporating past experiences stored in the experience replay buffer. This loop continues iteratively, with the system continuously learning and improving its landing strategy based on observed outcomes and rewards.

4.2.5 Activity Diagram

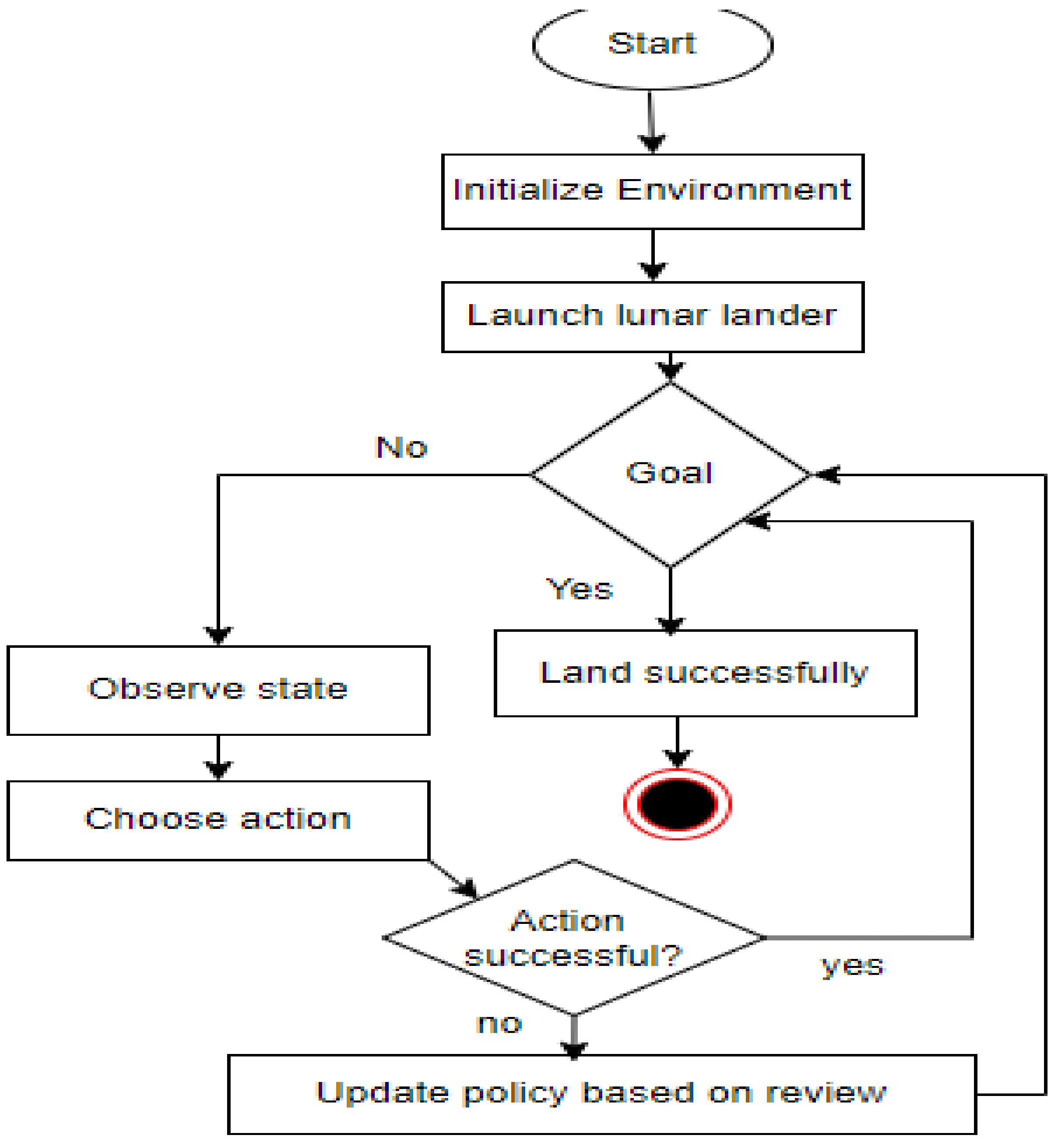


Figure 4.6: **Activity Diagram of Lunar Lander**

The above Figure 4.6 represents the activity diagram for lunar landing using deep reinforcement learning illustrates the sequential steps involved in the spacecraft's descent. It begins with initialization, followed by sensor data collection, state estimation, and decision-making using deep reinforcement learning algorithms. The spacecraft adjusts its thrust and orientation based on learned policies to achieve a safe landing.

4.3 Algorithm & Pseudo Code

4.3.1 Enhanced Reinforcement learning

Environment Setup:

• Define the Lunar Lander environment with its state space, action space, and dynamics (e.g., gravity, fuel consumption, etc.).

• Determine the reward system (e.g., positive reward for safe landing, penalties for crashes, fuel usage).

Agent Types:

• Agentless Lunar Lander: This serves as a baseline where the lander has no intelligence and moves purely based on physics simulation.

• Random Agent: Makes random actions regardless of the state.

• Simple Reflex Agent: Acts based on predefined rules (e.g., thrust if altitude is decreasing rapidly).

• Deep Q-Network (DQN) Agent: Trains a neural network to approximate the Q-values for state-action pairs.

1. Initialization

Initialize the RL environment, including the lunar lander state and action spaces. Create instances for each type of agent (agentless, random, simple reflex, DQN).

2. Training Loop (for DQN Agent)

a. Initialization:

• Initialize the DQN model with a neural network architecture suitable for the task.

• Define the replay buffer to store experiences (state, action, reward, next state) for training.

b. Exploration vs. Exploitation:

• Implement an greedy strategy to balance exploration (random actions) and exploitation (actions based on learned Q-values).

c. Interaction with the Environment:

For each episode:

• Reset the environment to start a new episode.

• Observe the initial state of the lander.

• Choose an action using the -greedy policy based on the current Q-values.

• Execute the action in the environment and observe the next state and reward.

• Store the experience (state, action, reward, next state) in the replay buffer.

d. Training the DQN Model:

• Sample a batch of experiences from the replay buffer.

• Compute the target Q-values using the Bellman equation.

• Train the DQN model to minimize the loss between predicted Q-values and target Q-values.

e. Updating Policy:

• Update the -greedy policy to gradually shift from exploration to exploitation as training progresses.

3. Evaluation:

After training, evaluate each agent's performance: Run simulations with each agent over multiple episodes. Measure and compare metrics such as average rewards, successful landings, etc.

4. Analysis:

Analyze the performance of different agents: Compare learning curves, convergence speed, and robustness. Identify strengths and weaknesses of each agent type.

4.3.2 Pseudo Code

```
Initialize:
    Define the Lunar Lander environment (state space, action space)
    Initialize the Q-table with random values or use a neural network as a Q-function approximator

Define Parameters:
    Learning rate (alpha)
    Discount factor (gamma)
    Exploration rate (epsilon)
    Number of episodes
    Maximum steps per episode

Training Loop:
    for episode in range(number_of_episodes):
        Initialize the environment
        Observe initial state (s)
        total_reward = 0

        for step in range(max_steps_per_episode):
            // Choose action using epsilon-greedy policy
            if random_uniform() < epsilon:
                action = random_action()
            else:
                action = argmax(Q(s, a))  // Choose action with highest Q-value

            // Take action and observe new state and reward
            new_state, reward, done = environment.step(action)

            // Update Q-value using Q-learning equation
            current_q_value = Q(s, action)
            max_future_q_value = max(Q(new_state, all_actions))
            new_q_value = current_q_value + alpha * (reward + gamma * max_future_q_value -
                current_q_value)
           Q(s, action) = new_q_value

            // Update state and total reward
            s = new_state
            total_reward += reward

            // Break if episode is done
            if done:
                break

        // Decay exploration rate
        epsilon = decay_function(epsilon)

        // Print progress
        if episode % print_interval == 0:
            print("Episode:", episode, "Total Reward:", total_reward)

Testing Loop:
    // Use the learned policy (Q-values) to test the agent
    for episode in range(number_of_test_episodes):
        Initialize the environment
        Observe initial state (s)
        total_reward = 0

        while not done:
            // Choose action greedily
            action = argmax(Q(s, a))

            // Take action and observe new state and reward
            new_state, reward, done = environment.step(action)

            // Update state and total reward
            s = new_state
            total_reward += reward

        // Print total reward obtained in the test episode
        print("Test Episode:", episode, "Total Reward:", total_reward)
```

4.4 Module Description

4.4.1 The Lunar Lander Environment

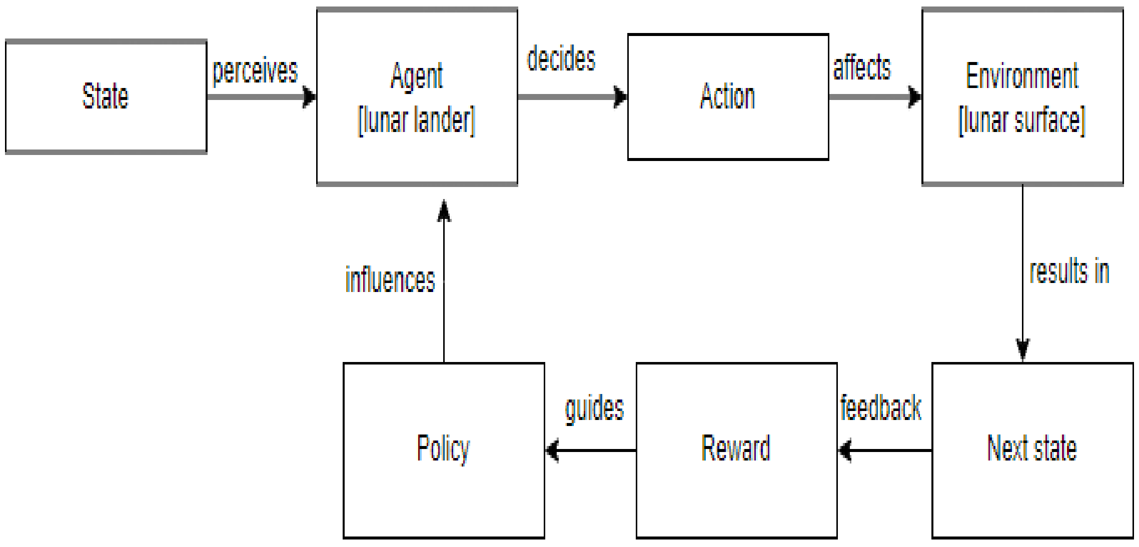


Figure 4.7: **Lunar Lander Environment**

The agent has four discrete actions available: Do nothing, Fire right engine, Fire main engine, Fire left engine.The agent's observation space consists of a state vector with 8 variables:Its (x,y) coordinates. The landing pad is always at coordinates (0,0).Its linear velocities (x',y'). Its angle .Its angular velocity . Two booleans, l and r, that represent whether each leg is in contact with the ground or not. After every step, a reward is granted. The total reward of an episode is the sum of the rewards for all the steps within that episode. For each step, the reward: is increased/decreased the closer/further the lander is to the landing pad, is increased/decreased the slower/faster the lander is moving. is decreased the more the lander is tilted (angle not horizontal),is increased by 10 points for each leg that is in contact with the ground,is decreased by 0.03 points each frame a side engine is firing,is decreased by 0.3 points each frame the main engine is firing. The episode receives an additional reward of

-100 or +100 points for crashing or landing safely respectively. An episode ends (i.e) the environment enters a terminal state if, the lunar lander crashes (i.e) if the body of the lunar lander comes in contact with the surface of the moon. The absolute value of the lander's x-coordinate is greater than 1 i.e. it goes beyond the left or right border.

4.4.2 Agentless Lunar Lander

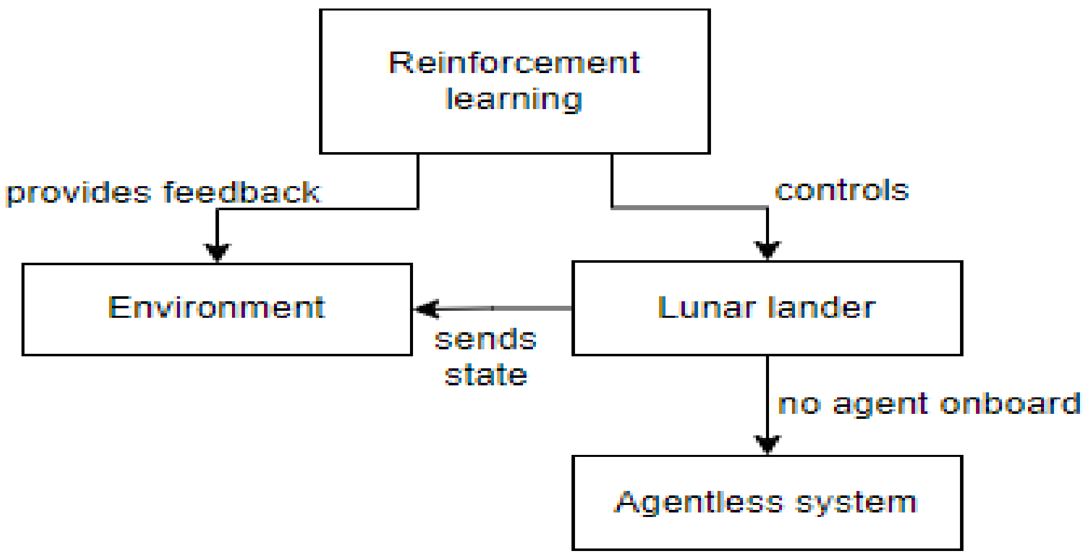
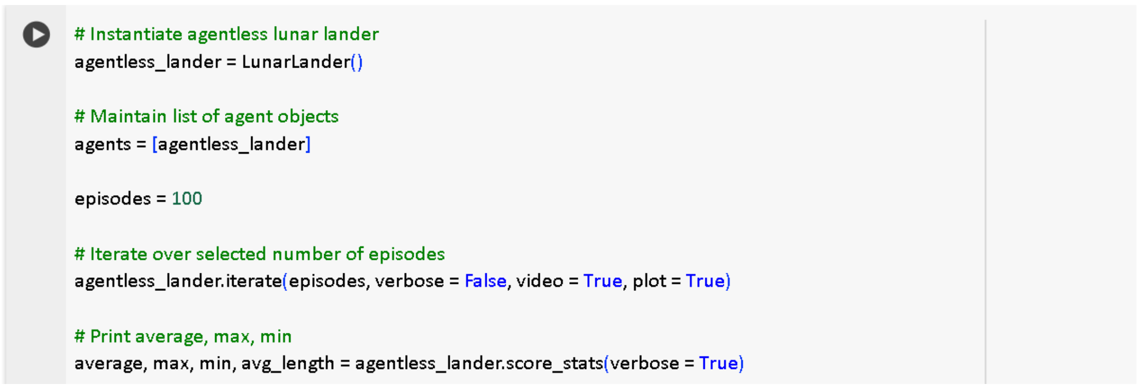


Figure 4.8: **Agentless Lunar Lander**

The Agentless Lunar Lander model is a basic model for spacecraft navigation that doesn't use any RL techniques. It simply follows the default physics engine of the environment without taking any actions. The model doesn't learn from the environment or adapt its behavior based on the state of the environment. The Agentless Lunar Lander model is useful for comparing the performance of other RL models with a basic model that only relies on the physics engine of the environment. Despite its simplicity, the Agentless Lunar Lander model can still achieve some level of success in navigating the spacecraft to the desired position, depending on the complexity of the environment. It can help identify the minimum requirements for successful spacecraft navigation in the environment. The Agentless Lunar Lander model can be useful for debugging and troubleshooting issues with the environment or the RL models.



```
# Instantiate agentless lunar lander
agentless_lander = LunarLander()

# Maintain list of agent objects
agents = [agentless_lander]

episodes = 100

# Iterate over selected number of episodes
agentless_lander.iterate(episodes, verbose = False, video = True, plot = True)

# Print average, max, min
average, max, min, avg_length = agentless_lander.score_stats(verbose = True)
```

Figure 4.9: **Agentless System**

4.4.3 Random Agent

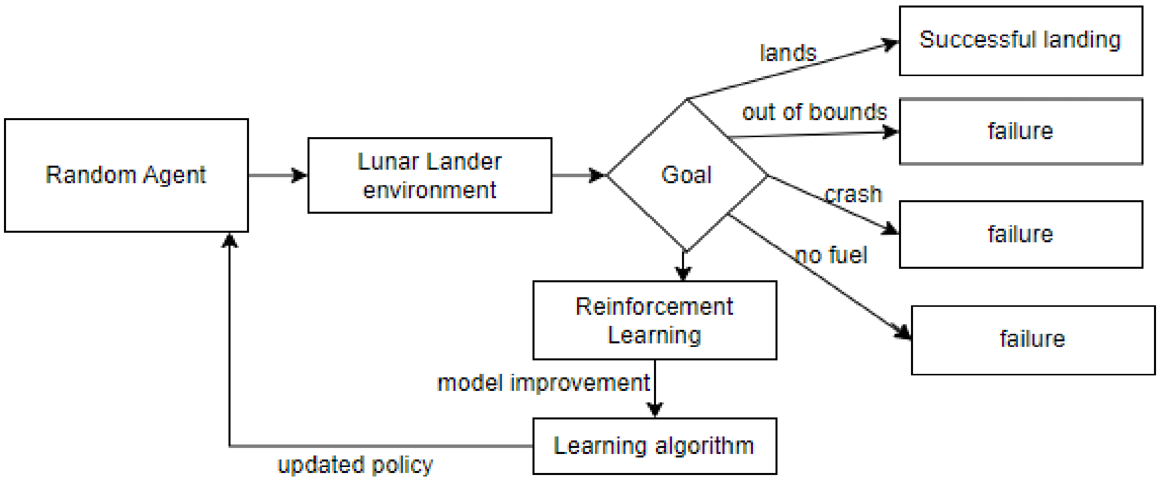
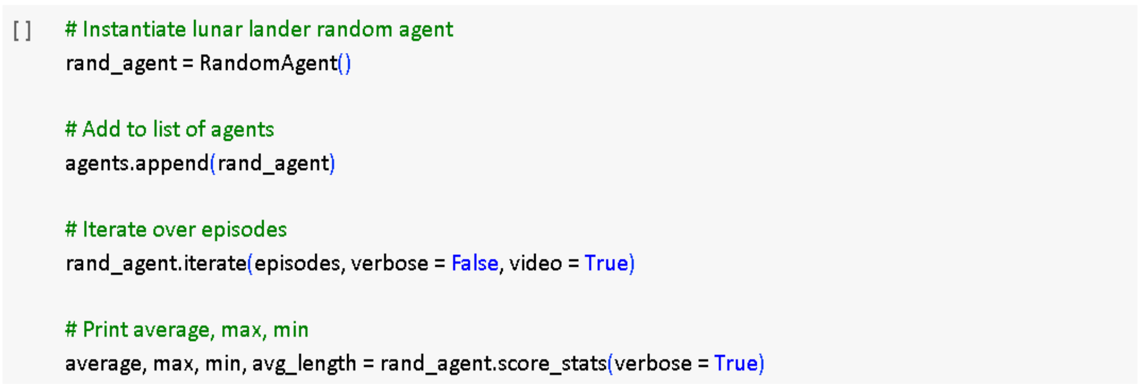


Figure 4.10: **Random Agent Lunar Lander**

The Random Agent model is a simple RL model that selects actions randomly without considering the current state of the environment. It does not learn from the environment or adapt its behavior over time. The Random Agent model is useful for evaluating the difficulty of the environment and the reward function. It provides a baseline performance that can be used to compare the performance of other RL models. The Random Agent model is typically used as a reference model for evaluating the performance of more sophisticated RL models. In some cases, random agents can still provide useful insights into the structure of the environment and the effectiveness of different actions. Random agents can be improved by incorporating a simple model of the environment or by using a more informed random strategy, such as selecting actions with a probability that is proportional to their expected utility.



```
[ ]  # Instantiate lunar lander random agent
     rand_agent = RandomAgent()

     # Add to list of agents
     agents.append(rand_agent)

     # Iterate over episodes
     rand_agent.iterate(episodes, verbose = False, video = True)

     # Print average, max, min
     average, max, min, avg_length = rand_agent.score_stats(verbose = True)
```

Figure 4.11: **Random Agent**

4.4.4 Reflex Agent

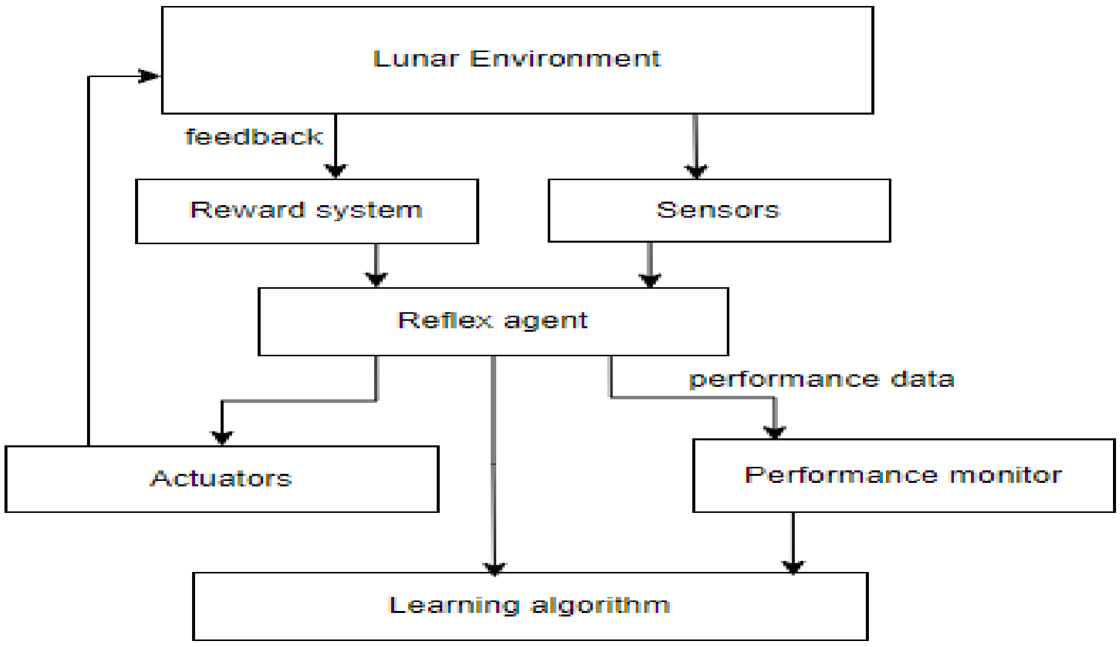
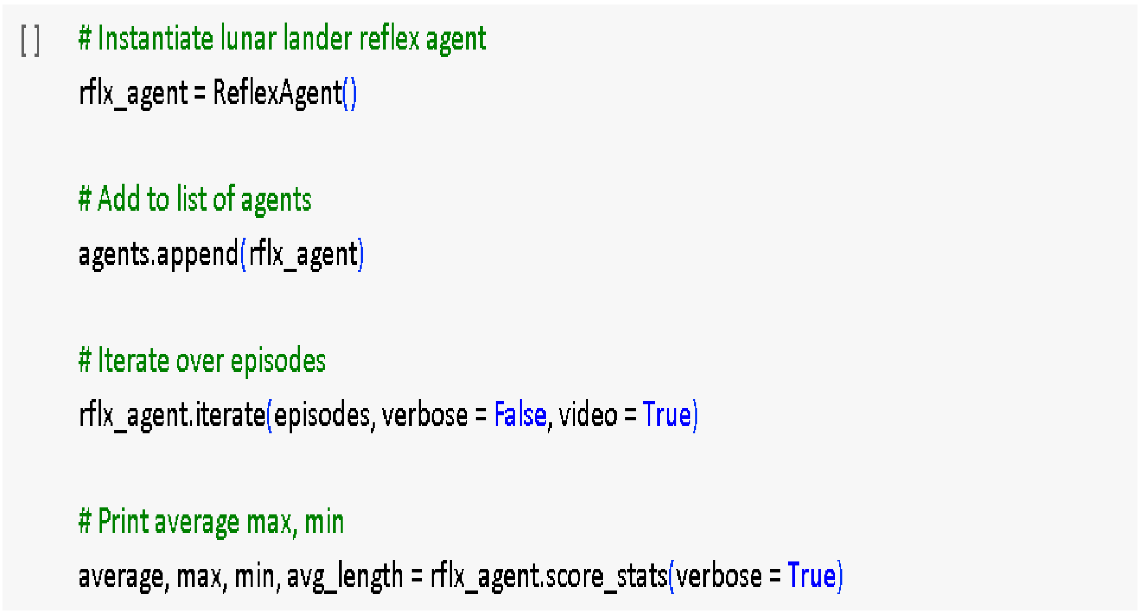


Figure 4.12: **Reflex Agent Lunar Lander**

Reflex agents are the simplest type of agents that select actions based on the current percept, ignoring the rest of the percept history. They use condition-action rules to determine the action to take based on the current state of the environment. Reflex agents are useful in environments where the state changes slowly and the agent can react quickly to changes in the environment. However, reflex agents can get stuck in infinite loops in environments with delayed effects, as they only consider the current percept and not the history of percepts. To avoid this issue, reflex agents can incorporate a simple model of the environment to anticipate the effects of their actions.

These model-based reflex agents can handle partially observable environments by maintaining an internal state that represents the unobserved aspects of the environment. Reflex agents are a fundamental building block in the design of more complex agents, and provide a useful starting point for developing intelligent systems.



```
[ ]  # Instantiate lunar lander reflex agent
     rflx_agent = ReflexAgent()

     # Add to list of agents
     agents.append(rflx_agent)

     # Iterate over episodes
     rflx_agent.iterate(episodes, verbose = False, video = True)

     # Print average max, min
     average, max, min, avg_length = rflx_agent.score_stats(verbose = True)
```

Figure 4.13: **Reflex Agent**

4.4.5 Deep Q Network gent

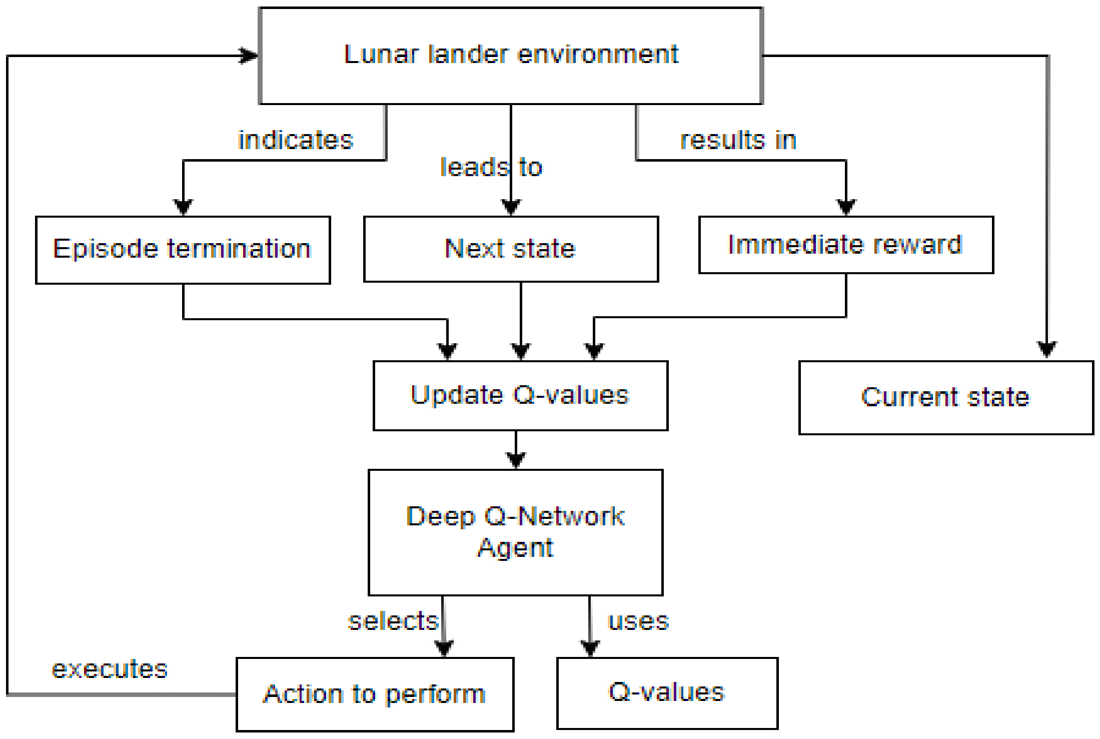
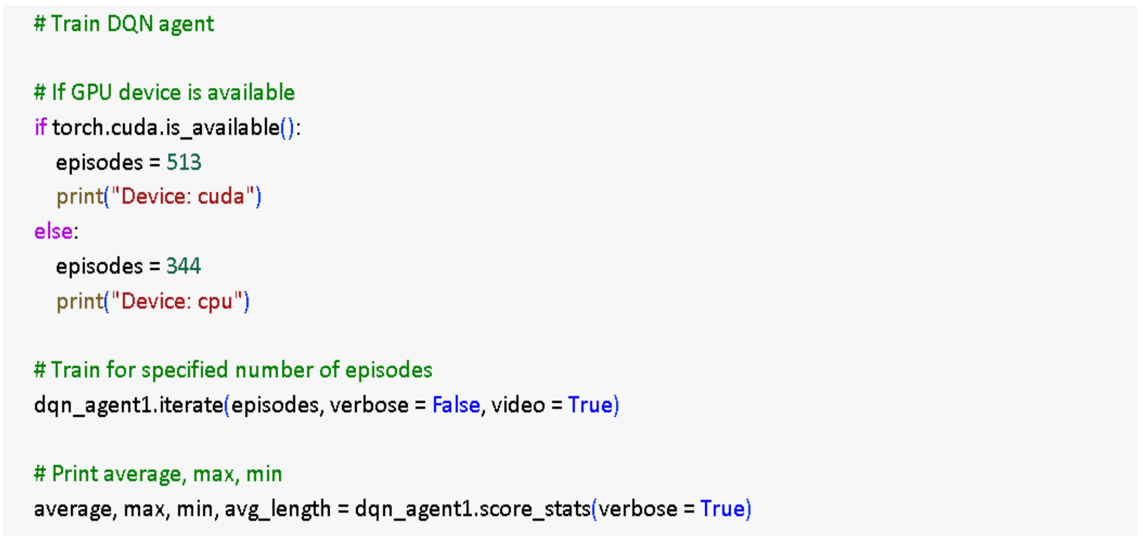


Figure 4.14: **DQN Agent Lunar Lander**

A DQN agent is a type of reinforcement learning agent that uses a neural network to approximate the Q-value function of a Markov decision process (MDP). The Q-value function estimates the expected discounted cumulative reward of taking a particular action in a given state and following a policy thereafter. Deep Q Network Agent combines the power of deep neural networks with Q-learning to learn the optimal action-value function. The DQN agent uses an experience replay buffer to store and reuse past experiences, which helps to reduce the correlation between consecutive samples and improves learning efficiency. The DQN agent also uses a target network that is updated periodically to stabilize the training process and improve convergence. The DQN model is implemented using a deep neural network with multiple hidden layers, and the output layer provides the Q-values for all possible actions given a state. The DQN model is trained using a loss function that measures the difference between the predicted Q-values and the target Q-values, and the model parameters are updated using an optimization algorithm such as Adam. The DQN agent continues to learn and improve its policy as it interacts with the environment, and it can adapt to changing dynamics and new situations by fine-tuning its parameters or retraining from scratch.



```
# Train DQN agent

# If GPU device is available
if torch.cuda.is_available():
    episodes = 513
    print("Device: cuda")
else:
    episodes = 344
    print("Device: cpu")

# Train for specified number of episodes
dqn_agent1.iterate(episodes, verbose = False, video = True)

# Print average, max, min
average, max, min, avg_length = dqn_agent1.score_stats(verbose = True)
```

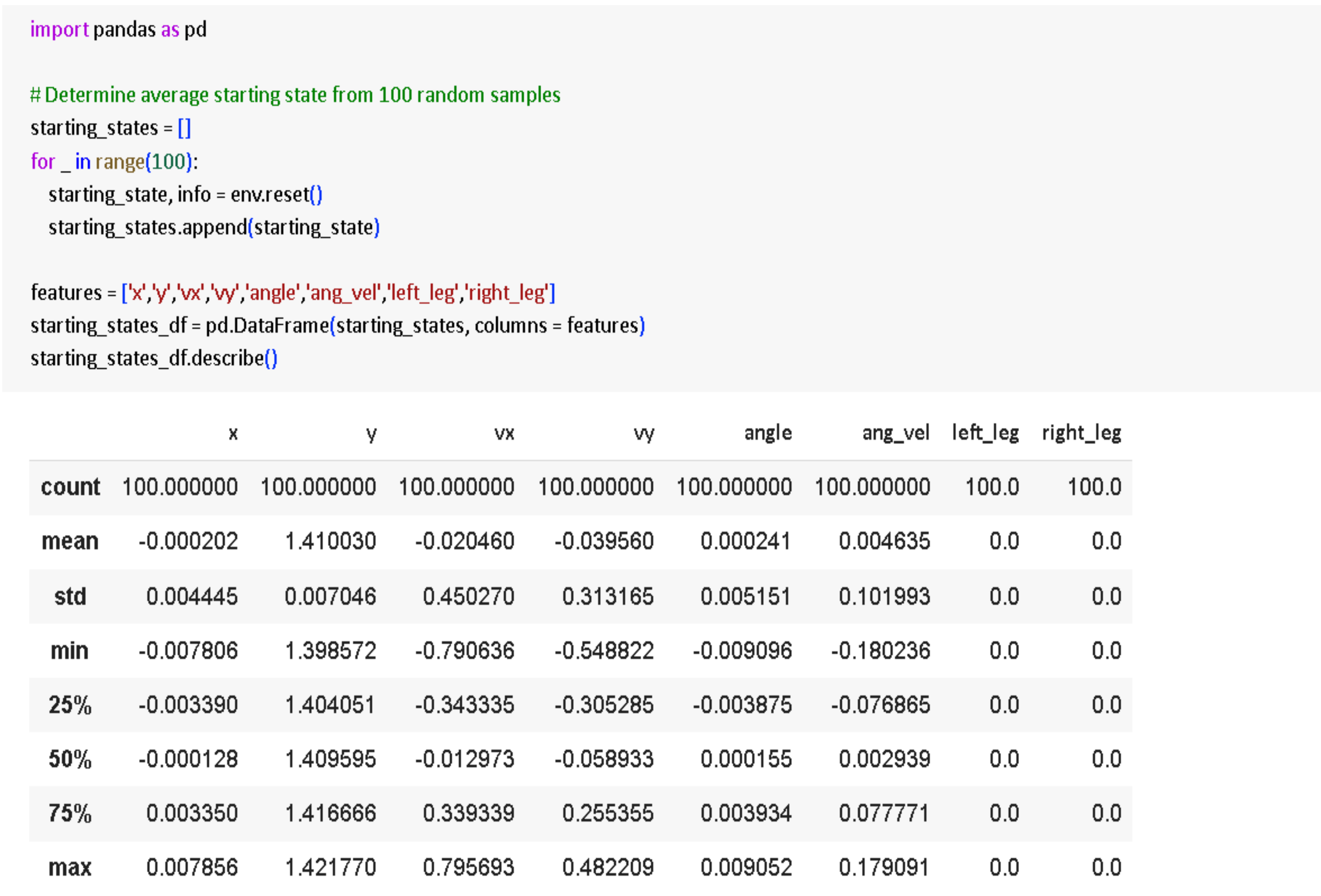
Figure 4.15: **Deep Q Network Agent**

Chapter 5

IMPLEMENTATION AND TESTING

5.1 Input and Output

5.1.1 Input Design



```
import pandas as pd

# Determine average starting state from 100 random samples
starting_states = []
for _ in range(100):
   starting_state, info = env.reset()
   starting_states.append(starting_state)

features = ['x','y','vx','vy','angle','ang_vel','left_leg','right_leg']
starting_states_df = pd.DataFrame(starting_states, columns = features)
starting_states_df.describe()
```

```
                x           y          vx          vy       angle     ang_vel  left_leg  right_leg
count  100.000000  100.000000  100.000000  100.000000  100.000000  100.000000     100.0      100.0
mean    -0.000202    1.410030   -0.020460   -0.039560    0.000241    0.004635       0.0        0.0
std      0.004445    0.007046    0.450270    0.313165    0.005151    0.101993       0.0        0.0
min     -0.007806    1.398572   -0.790636   -0.548822   -0.009096   -0.180236       0.0        0.0
25%     -0.003390    1.404051   -0.343335   -0.305285   -0.003875   -0.076865       0.0        0.0
50%     -0.000128    1.409595   -0.012973   -0.058933    0.000155    0.002939       0.0        0.0
75%      0.003350    1.416666    0.339339    0.255355    0.003934    0.077771       0.0        0.0
max      0.007856    1.421770    0.795693    0.482209    0.009052    0.179091       0.0        0.0
```

Figure 5.1: **Input to the Agents**

From the Figure 5.1 we can identify that the average starting states, features, samples and required parameters has been given as the input to the DQN Agents.

5.1.2 Output Design

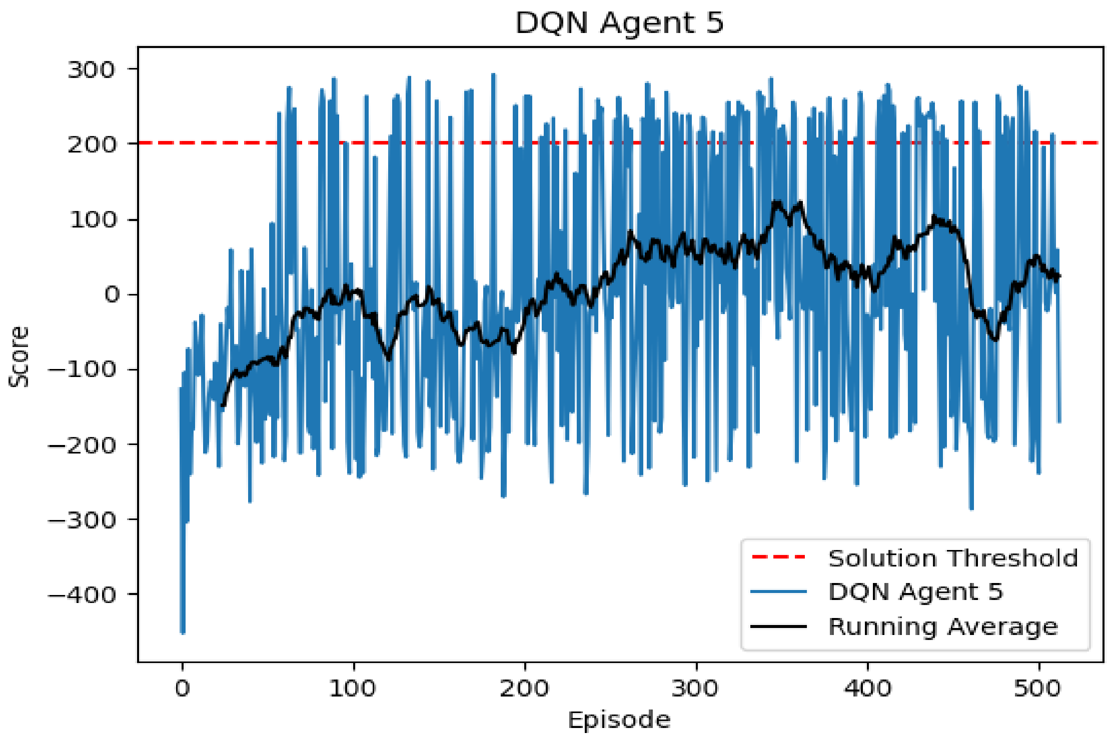


Figure 5.2: **DQN Agent Output**

From the Figure 5.2 we can identify that the average has increased compared to Agent 4 and the average episode length has decreased and the agent lands more successfully without any hovering.

5.2 Testing

5.3 Types of Testing

5.3.1 Unit testing

Input

```
# Instantiate agentless lunar lander
agentless_lander = LunarLander()

# Maintain list of agent objects
agents = [agentless_lander]

episodes = 100

```

```
# Iterate over selected number of episodes
agentless_lander.iterate(episodes, verbose = False, video = True, plot = True)

# Print average, max, min
average, max, min, avg_length = agentless_lander.score_stats(verbose = True)
```

Test result

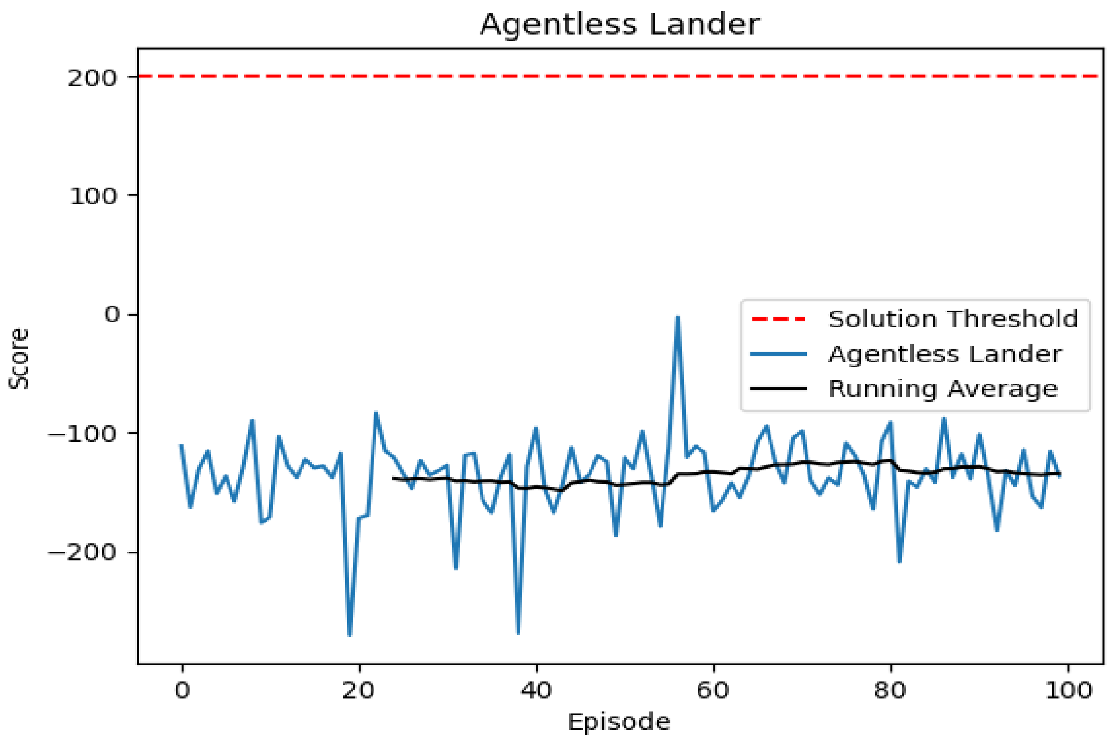


Figure 5.3: **Unit Testing**

From the Figure 5.3 we can depict that the lunar lander simply falls in a slightly curved trajectory toward the ground.

5.3.2 Integration testing

Input

```
def plot_comparison(agents):
    '''
    Plot scores of multiple agent objects
    '''
    # Initialize plot
    plt.figure(1)
    plt.ylim([-600,300])

    # Plot solution threshold
    plt.axhline(y = 200, color = "r", linestyle = "dashed", label = "Solution Threshold")

    # Initialize list of agent names
    names = []

    # Iterate over agents list
    for agent in agents:
        # Plot agent's scores
        plt.plot(agent.scores, label = agent.name)
        names.append(agent.name)

    plt.title(", ".join(names))
    plt.xlabel("Episode")
    plt.ylabel("Score")
    plt.legend()
    plt.show();
# Instantiate lunar lander reflex agent
rflx_agent = ReflexAgent()

# Add to list of agents
agents.append(rflx_agent)

# Iterate over episodes
rflx_agent.iterate(episodes, verbose = False, video = True)

# Print average max, min
average, max, min, avg_length = rflx_agent.score_stats(verbose = True)
```

Test result

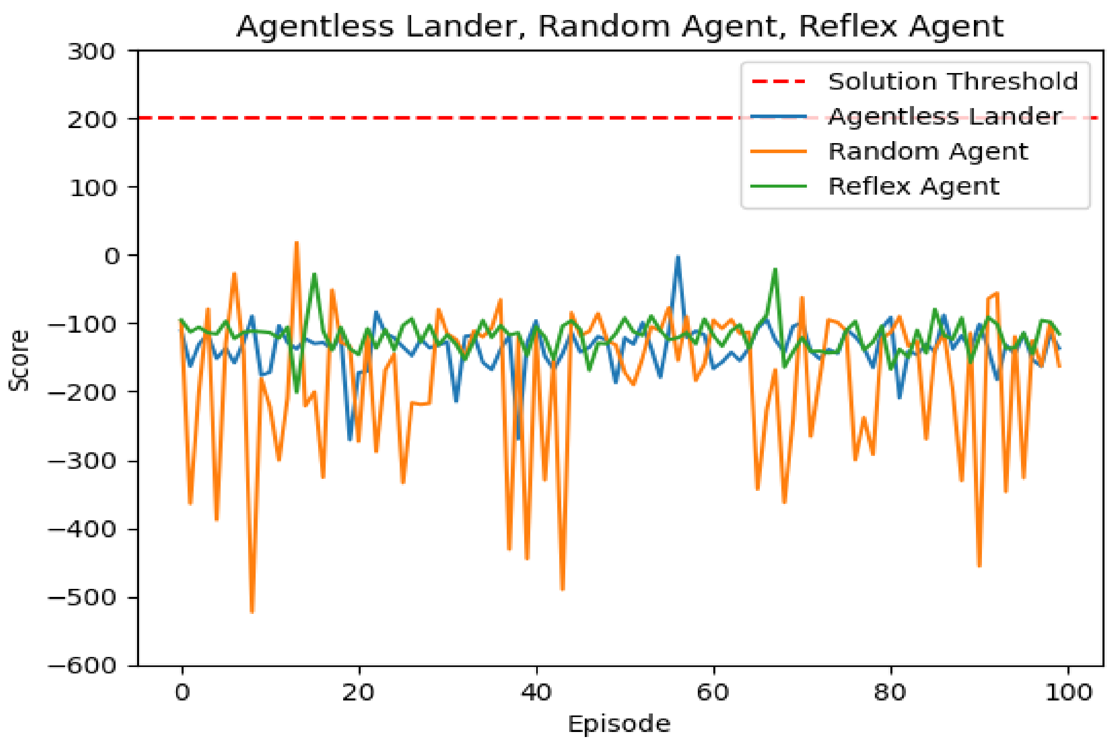


Figure 5.4: **Integration Testing**

The above Figure 5.4 shows that the agentless lunar lander is relatively more consistent and successful compared to the random agent and also has a slightly higher average score.

5.3.3 System testing

Input

```
# Reset the second lunar lander DQN agent
dqn_agent5 = DQNAgent(name, alpha, epsilon, gamma, tau, lr_period, lr_decay, batch_size, buffer_size
    )

# If GPU device is available
if torch.cuda.is_available():
    episodes = 730
    print("Device: cuda")
else:
    episodes = 513
    print("Device: cpu")

# Train for specified number of episodes
dqn_agent5.iterate(episodes, verbose = False, video = True)
```

```

# Print average, max, min
average, max, min, avg_length = dqn_agent5.score_stats(verbose = True)
```

Test Result

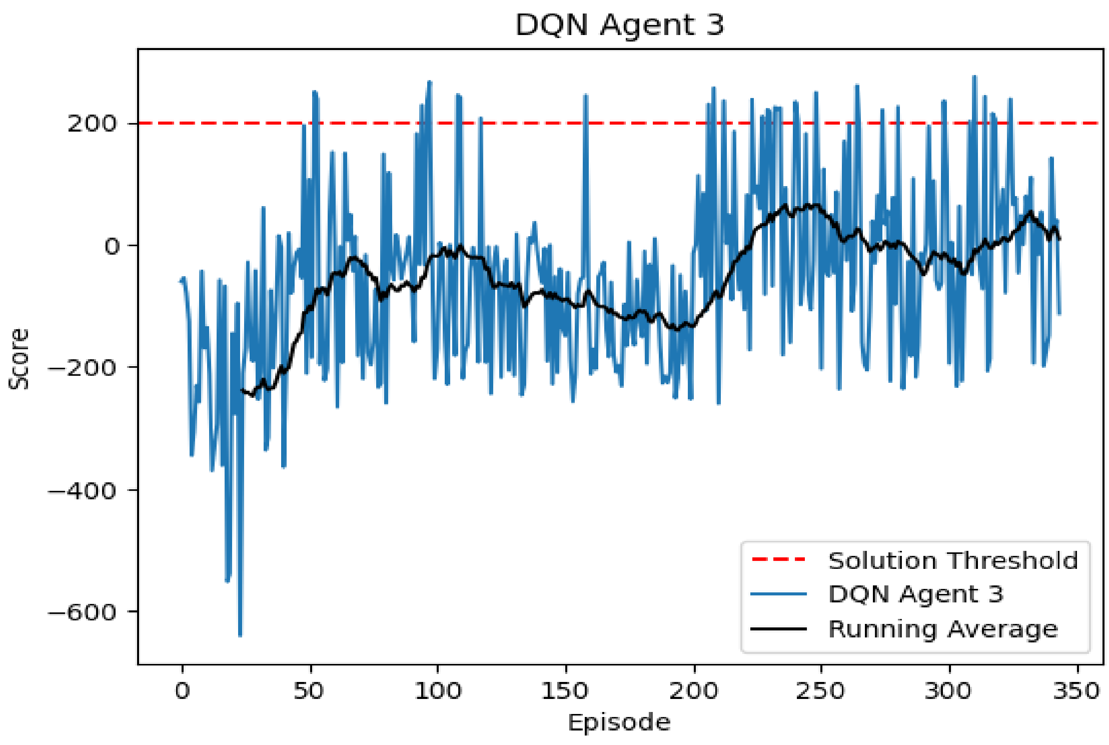


Figure 5.5: **System Testing**

From the Figure 5.5 we can see that the average score was lower this time and the agent is experiencing more high scores because the state space is so large, it is having difficulty transfering the optimal actions for those transitions to other transitions in the environment.

5.3.4 Test Result

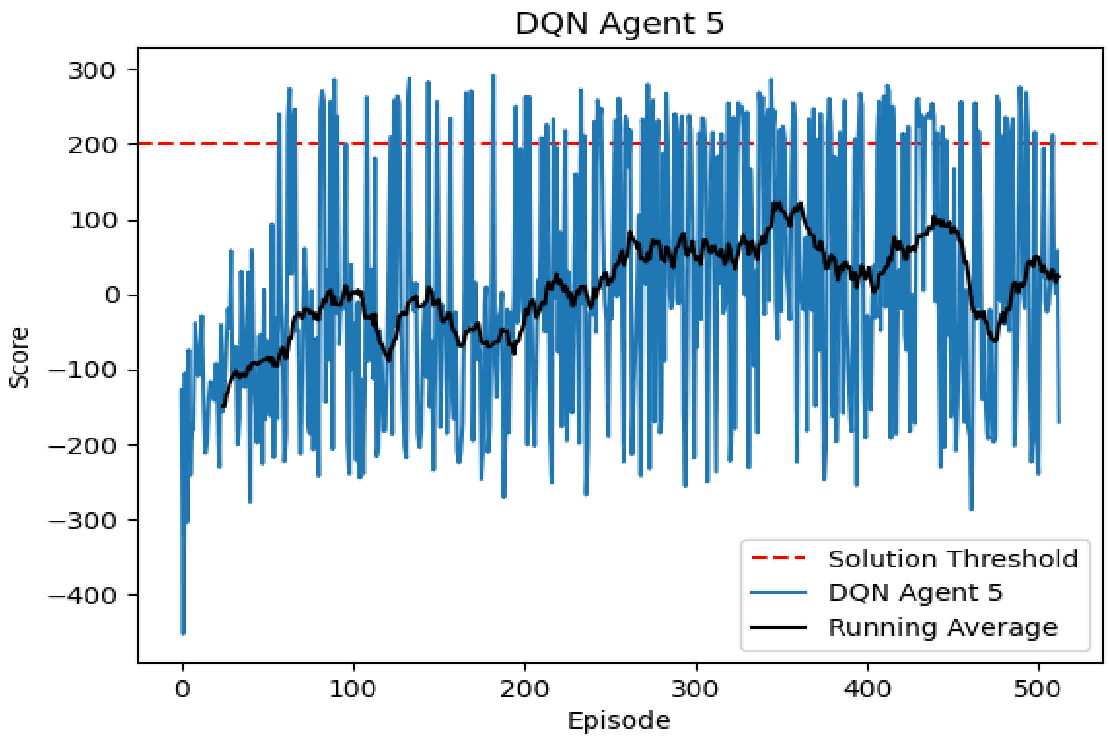


Figure 5.6: **Landing Of Spacecraft**

From the Figure 5.6 we can identify that the average has increased compared to Agent 4 and the average episode length has decreased and the agent lands more successfully without any hovering.

Chapter 6

RESULTS AND DISCUSSIONS

6.1 Efficiency of the Proposed System

The efficiency of lunar landing can be significantly enhanced through the application of Q-network reinforcement learning. In this approach, a Q-network is trained to approximate the optimal action-value function, which determines the best action to take in a given state to maximize long-term rewards. This technique is particularly effective for lunar landing due to its ability to adapt to complex and uncertain environments.

Q-networks can learn to navigate the challenging dynamics of lunar descent by continuously evaluating the consequences of different actions in varying lunar conditions. By training the network on simulated lunar landscapes and gravitational forces, it can develop robust strategies for safe and efficient landing.

Q-networks enable adaptive decision-making during the landing process. The network learns from both successful and unsuccessful landings, iteratively refining its policy to achieve higher success rates and minimize resource expenditure, such as fuel usage.

Furthermore, Q-networks offer scalability and versatility, capable of handling diverse scenarios and evolving environments without the need for manual intervention. This adaptability is crucial for lunar missions where conditions can change rapidly.

6.2 Comparison of Existing and Proposed System

Existing system:(NASA-sponsored ”Moonjs” Platform for Lunar Landing)

Moonjs provides a web-based simulator that allows users to manually pilot a lunar lander spacecraft in a simulated environment.The simulator incorporates realistic physics, including lunar gravity, terrain dynamics, and propulsion systems.It is primarily used for educational purposes, enabling students and enthusiasts to understand the challenges and complexities of lunar landings. It also serves as a testing ground for developing and refining landing strategies and algorithms.As a web-based platform, Moonjs is easily accessible to a wide audience, promoting interest and engagement in lunar exploration and space science.

Proposed system:(Lunar Lander Q Network Reinforcement Learning)

In this approach, a Q-learning network is trained using reinforcement learning algorithms to enable an AI agent to learn how to land a lunar lander autonomously. The Q network learns from interactions with the simulated environment by receiving rewards or penalties based on its actions, aiming to maximize long-term rewards (i.e., successful landings).This method involves complex training processes where the AI agent iteratively learns to make decisions (such as throttle control) based on the state of the lander and the environment. The goal is to develop a policy that guides the agent to perform successful landings under various conditions.It is valuable for developing autonomous landing systems for real-world lunar missions. It explores the capabilities of AI in navigating and controlling spacecraft in dynamic environments like the Moon.

6.3 Sample Code

```
# Instantiate agentless lunar lander
agentless_lander = LunarLander()

# Maintain list of agent objects
agents = [agentless_lander]

episodes = 100

# Iterate over selected number of episodes
agentless_lander.iterate(episodes, verbose = False, video = True, plot = True)

# Print average, max, min
average, max, min, avg_length = agentless_lander.score_stats(verbose = True)
# Instantiate lunar lander random agent
rand_agent = RandomAgent()

# Add to list of agents
agents.append(rand_agent)

# Iterate over episodes
```

```
rand_agent.iterate(episodes, verbose = False, video = True)

# Print average, max, min
average, max, min, avg_length = rand_agent.score_stats(verbose = True)
# Instantiate lunar lander reflex agent
rflx_agent = ReflexAgent()

# Add to list of agents
agents.append(rflx_agent)

# Iterate over episodes
rflx_agent.iterate(episodes, verbose = False, video = True)

# Print average max, min
average, max, min, avg_length = rflx_agent.score_stats(verbose = True)
# Train DQN agent

# If GPU device is available
if torch.cuda.is_available():
    episodes = 513
    print("Device: cuda")
else:
    episodes = 344
    print("Device: cpu")



# Train for specified number of episodes
dqn_agent5.iterate(episodes, verbose = False, video = True)

# Print average, max, min
average, max, min, avg_length = dqn_agent5.score_stats(verbose = True)
```

Output

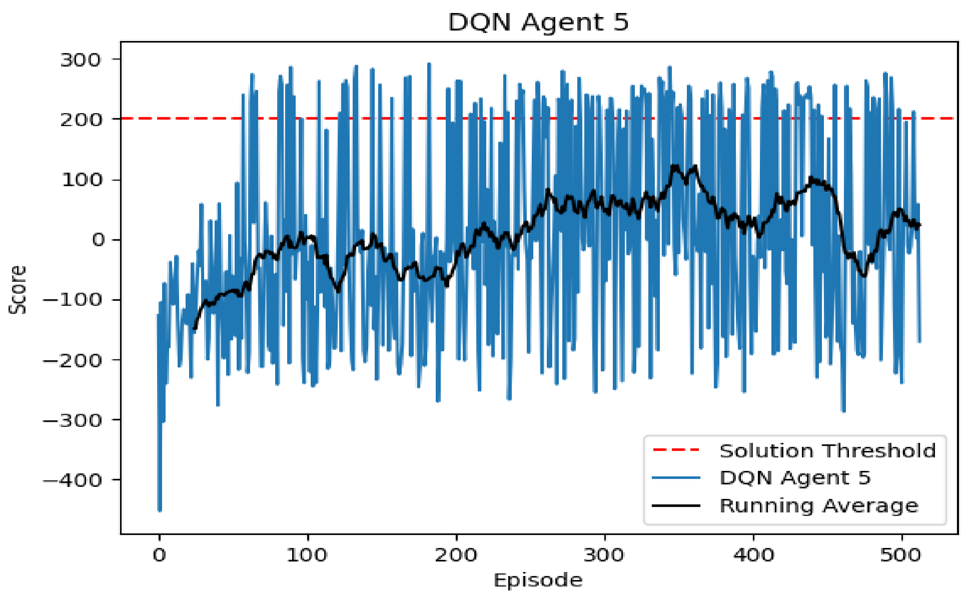


Figure 6.1: **Output of DQN Agent**

From the Figure 6.1 we can identify that the average has increased compared to Agent 4 and the average episode length has decreased.

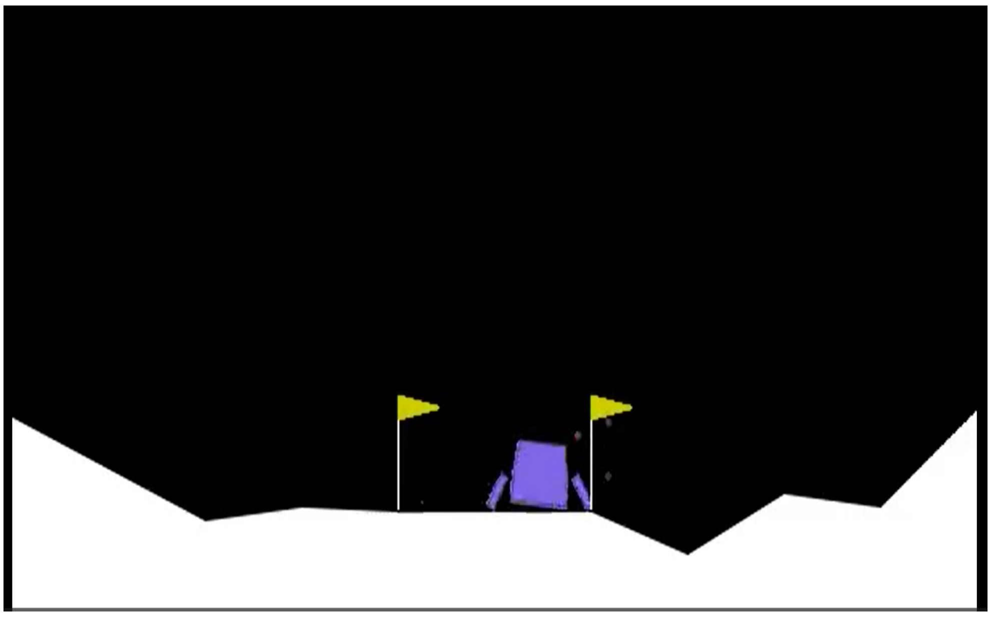


Figure 6.2: **Successful Landing of Spacecraft**

From the Figure 6.2 we can identify that the spacecraft landed successfully in the target position without any hovering.

Chapter 7

CONCLUSION AND FUTURE ENHANCEMENTS

7.1 Conclusion

The Lunar Lander Reinforcement Learning simulation project presented a comprehensive comparison of four distinct artificial agents within a lunar landing scenario. The study confirmed the intuitive expectation that a deep reinforcement learning (RL) agent would outperform non-learning agents.

The agents compared included an agentless lunar lander (which didn't make active decisions), a random agent (acting without strategy), a simple reflex agent (making decisions based on immediate inputs), and a deep Q-network (DQN) agent utilizing advanced RL techniques. Among the non-learning agents, the simple reflex agent demonstrated marginal improvement over the agentless lunar lander, while the random agent proved to be the least effective strategy.

The standout performer was the deep Q-network agent, employing a neural network architecture consisting of three fully-connected layers. This DQN agent leveraged reinforcement learning principles to learn and refine its landing strategies over time, adapting its decisions based on feedback received from the simulation environment.

project highlighted the efficacy of deep reinforcement learning in complex decision-making tasks like lunar landing. The DQN agent's ability to learn from experience and optimize its approach demonstrated significant advantages over static and random strategies. This investigation underscores the potential of RL techniques in solving dynamic and challenging real-world problems.

7.2 Future Enhancements

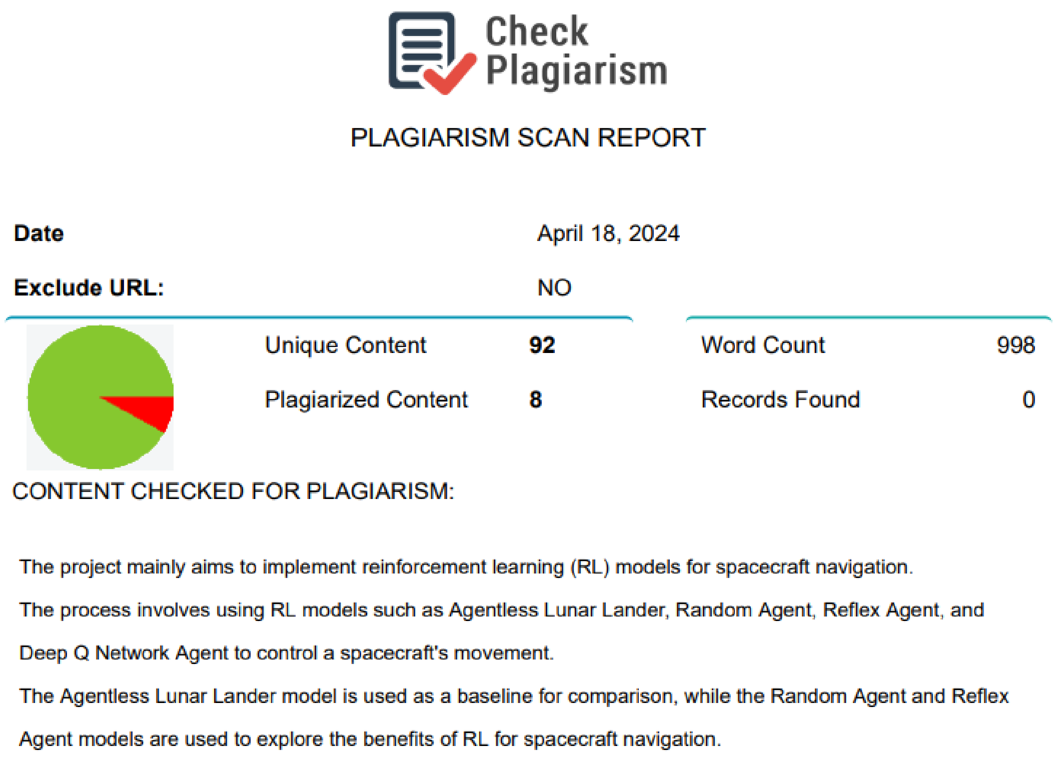
One key enhancement could involve implementing cooperative or competitive multi-agent strategies. Cooperative agents could work together to achieve a common goal, such as efficiently landing on the moon's surface while minimizing fuel consumption collectively. This could involve developing communication protocols between agents or allowing them to share learned strategies and experiences.

Additionally, introducing competitive elements could foster strategic decision-making among agents. For example, agents could compete to land closest to a designated target or achieve the fastest landing time, introducing a competitive reinforcement learning dynamic. Implementing these features would require adapting the existing simulation to accommodate multiple agents simultaneously, including adjusting reward structures and learning algorithms to account for interactions between agents.

Moreover, exploring the concept of heterogeneous agents with varying capabilities and objectives could provide further depth to the simulation. For instance, agents with different landing constraints (e.g., limited fuel or varying landing precision) could collaborate or compete within the same environment, leading to more complex learning scenarios. Overall, integrating multiple agents into the Lunar Lander simulation opens up a rich space for exploring cooperative and competitive reinforcement learning strategies, enhancing the project's realism and complexity.

Chapter 8

PLAGIARISM REPORT



Check Plagiarism

PLAGIARISM SCAN REPORT

Date	April 18, 2024
Exclude URL:	NO

Unique Content	**92**	Word Count	998
Plagiarized Content	**8**	Records Found	0

CONTENT CHECKED FOR PLAGIARISM:

The project mainly aims to implement reinforcement learning (RL) models for spacecraft navigation.

The process involves using RL models such as Agentless Lunar Lander, Random Agent, Reflex Agent, and Deep Q Network Agent to control a spacecraft's movement.

The Agentless Lunar Lander model is used as a baseline for comparison, while the Random Agent and Reflex Agent models are used to explore the benefits of RL for spacecraft navigation.

Figure 8.1: **Plagarism Report**

Chapter 9

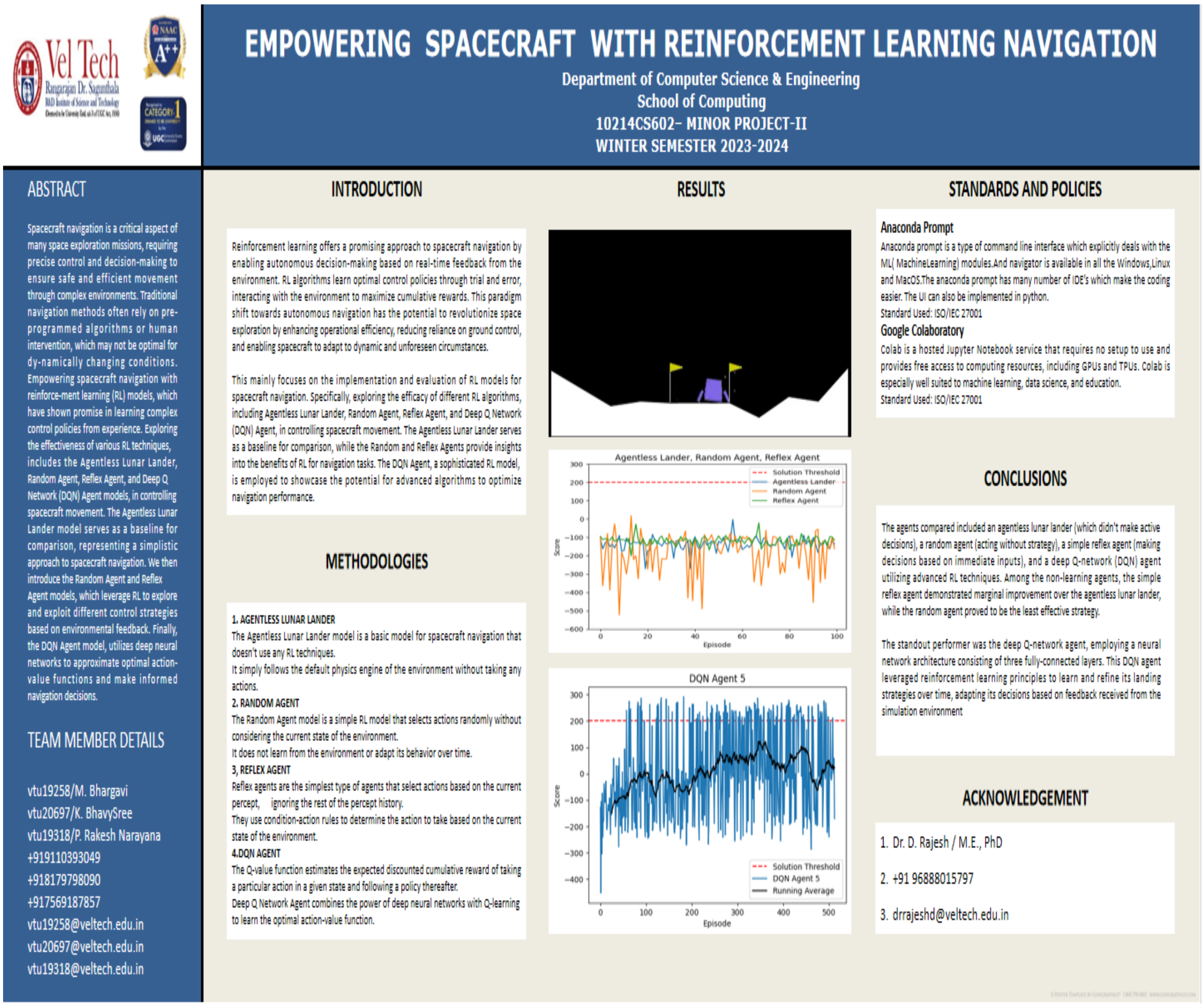
SOURCE CODE & POSTER PRESENTATION

9.1 Source Code

```

# Instantiate agentless lunar lander
agentless_lander = LunarLander()

# Maintain list of agent objects
agents = [agentless_lander]

episodes = 100

# Iterate over selected number of episodes
agentless_lander.iterate(episodes, verbose = False, video = True, plot = True)

# Print average, max, min
average, max, min, avg_length = agentless_lander.score_stats(verbose = True)
# Instantiate lunar lander random agent
rand_agent = RandomAgent()

# Add to list of agents
agents.append(rand_agent)

# Iterate over episodes
rand_agent.iterate(episodes, verbose = False, video = True)

# Print average, max, min
average, max, min, avg_length = rand_agent.score_stats(verbose = True)
# Instantiate lunar lander reflex agent
rflx_agent = ReflexAgent()

# Add to list of agents
agents.append(rflx_agent)

# Iterate over episodes
rflx_agent.iterate(episodes, verbose = False, video = True)

# Print average max, min
```

```
average, max, min, avg_length = rflx_agent.score_stats(verbose = True)
# Train DQN agent

# If GPU device is available
if torch.cuda.is_available():
    episodes = 513
    print("Device: cuda")
else:
    episodes = 344
    print("Device: cpu")

# Train for specified number of episodes
dqn_agent1.iterate(episodes, verbose = False, video = True)

# Print average, max, min
average, max, min, avg_length = dqn_agent1.score_stats(verbose = True)
# Instantiate second lunar lander DQN agent
dqn_agent2 = DQNAgent(name, alpha, epsilon, gamma, tau, lr_period, lr_decay, batch_size, buffer_size
    )

# Train for specified number of episodes
dqn_agent2.iterate(episodes, verbose = False, video = True)

# Print average, max, min
average, max, min, avg_length = dqn_agent2.score_stats(verbose = True)
# Instantiate lunar lander DQN agent
dqn_agent3 = DQNAgent(name, alpha, epsilon, gamma, tau, lr_period, lr_decay, batch_size, buffer_size
    )

# Train for specified number of episodes
dqn_agent3.iterate(episodes, verbose = False, video = True)

# Print average, max, min
average, max, min, avg_length = dqn_agent3.score_stats(verbose = True)
# Instantiate lunar lander DQN agent
dqn_agent4 = DQNAgent(name, alpha, epsilon, gamma, tau, lr_period, lr_decay, batch_size, buffer_size
    )

# Train for specified number of episodes
dqn_agent4.iterate(episodes, verbose = False, video = True)

# Print average, max, min
average, max, min, avg_length = dqn_agent4.score_stats(verbose = True)
# Reset the second lunar lander DQN agent
dqn_agent5 = DQNAgent(name, alpha, epsilon, gamma, tau, lr_period, lr_decay, batch_size, buffer_size
    )

# If GPU device is available
if torch.cuda.is_available():
    episodes = 730
    print("De\large{\paragraph{} \Large{\paragraph{} vice: cuda")
else:
    episodes = 513
    print("Device: cpu")

# Train for specified number of episodes
dqn_agent5.iterate(episodes, verbose = False, video = True)

# Print average, max, min
average, max, min, avg_length = dqn_agent5.score_stats(verbose = True)
```

9.2 Poster Presentation



EMPOWERING SPACECRAFT WITH REINFORCEMENT LEARNING NAVIGATION

Department of Computer Science & Engineering
School of Computing
10214CS602– MINOR PROJECT-II
WINTER SEMESTER 2023-2024

ABSTRACT

Spacecraft navigation is a critical aspect of many space exploration missions, requiring precise control and decision-making to ensure safe and efficient movement through complex environments. Traditional navigation methods often rely on pre-programmed algorithms or human intervention, which may not be optimal for dy-namically changing conditions. Empowering spacecraft navigation with reinforce-ment learning (RL) models, which have shown promise in learning complex control policies from experience. Exploring the effectiveness of various RL techniques, includes the Agentless Lunar Lander, Random Agent, Reflex Agent, and Deep Q Network (DQN) Agent models, in controlling spacecraft movement. The Agentless Lunar Lander model serves as a baseline for comparison, representing a simplistic approach to spacecraft navigation. We then introduce the Random Agent and Reflex Agent models, which leverage RL to explore and exploit different control strategies based on environmental feedback. Finally, the DQN Agent model, utilizes deep neural networks to approximate optimal action-value functions and make informed navigation decisions.

TEAM MEMBER DETAILS

vtu19258/M. Bhargavi
vtu20697/K. BhavySree
vtu19318/P. Rakesh Narayana
+919110393049
+918179798090
+917569187857
vtu19258@veltech.edu.in
vtu20697@veltech.edu.in
vtu19318@veltech.edu.in

INTRODUCTION

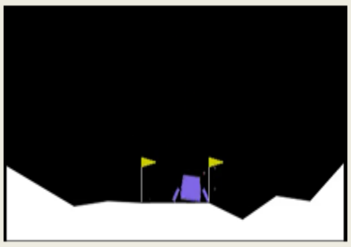
Reinforcement learning offers a promising approach to spacecraft navigation by enabling autonomous decision-making based on real-time feedback from the environment. RL algorithms learn optimal control policies through trial and error, interacting with the environment to maximize cumulative rewards. This paradigm shift towards autonomous navigation has the potential to revolutionize space exploration by enhancing operational efficiency, reducing reliance on ground control, and enabling spacecraft to adapt to dynamic and unforeseen circumstances.

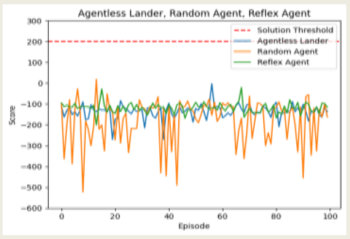
This mainly focuses on the implementation and evaluation of RL models for spacecraft navigation. Specifically, exploring the efficacy of different RL algorithms, including Agentless Lunar Lander, Random Agent, Reflex Agent, and Deep Q Network (DQN) Agent, in controlling spacecraft movement. The Agentless Lunar Lander serves as a baseline for comparison, while the Random and Reflex Agents provide insights into the benefits of RL for navigation tasks. The DQN Agent, a sophisticated RL model, is employed to showcase the potential for advanced algorithms to optimize navigation performance.

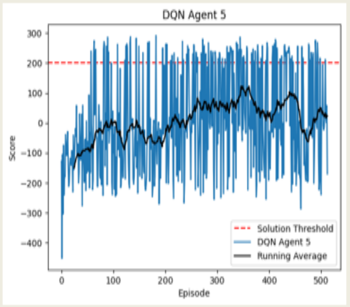
METHODOLOGIES

1. AGENTLESS LUNAR LANDER
The Agentless Lunar Lander model is a basic model for spacecraft navigation that doesn't use any RL techniques.
It simply follows the default physics engine of the environment without taking any actions.
2. RANDOM AGENT
The Random Agent model is a simple RL model that selects actions randomly without considering the current state of the environment.
It does not learn from the environment or adapt its behavior over time.
3, REFLEX AGENT
Reflex agents are the simplest type of agents that select actions based on the current percept, ignoring the rest of the percept history.
They use condition-action rules to determine the action to take based on the current state of the environment.
4.DQN AGENT
The Q-value function estimates the expected discounted cumulative reward of taking a particular action in a given state and following a policy thereafter.
Deep Q Network Agent combines the power of deep neural networks with Q-learning to learn the optimal action-value function.

RESULTS







STANDARDS AND POLICIES

Anaconda Prompt
Anaconda prompt is a type of command line interface which explicitly deals with the ML(MachineLearning) modules.And navigator is available in all the Windows,Linux and MacOS.The anaconda prompt has many number of IDE's which make the coding easier. The UI can also be implemented in python.
Standard Used: ISO/IEC 27001
Google Colaboratory
Colab is a hosted Jupyter Notebook service that requires no setup to use and provides free access to computing resources, including GPUs and TPUs. Colab is especially well suited to machine learning, data science, and education.
Standard Used: ISO/IEC 27001

CONCLUSIONS

The agents compared included an agentless lunar lander (which didn't make active decisions), a random agent (acting without strategy), a simple reflex agent (making decisions based on immediate inputs), and a deep Q-network (DQN) agent utilizing advanced RL techniques. Among the non-learning agents, the simple reflex agent demonstrated marginal improvement over the agentless lunar lander, while the random agent proved to be the least effective strategy.

The standout performer was the deep Q-network agent, employing a neural network architecture consisting of three fully-connected layers. This DQN agent leveraged reinforcement learning principles to learn and refine its landing strategies over time, adapting its decisions based on feedback received from the simulation environment

ACKNOWLEDGEMENT

1. Dr. D. Rajesh / M.E., PhD
2. +91 96888015797
3. drrajeshd@veltech.edu.in

Figure 9.1: **Poster Presentation**

References

[1] C. Wang, H. Nie, J. Chen and H. P. Lee, ”The design and dynamic analysis of a lunar lander with semi-active control”, Acta Astronautica, vol. 157, pp. 145-156, 2019.

[2] D. E. Smith, M. T. Zuber, G. B. Jackson, J. F. Cavanaugh, G. A. Neumann, H. Riris, X. Sun, R. S. Zellar, C. Coltharp, J. Connelly et al., ”The lunar orbiter laser altimeter investigation on the lunar reconnaissance orbiter mission”, Space science reviews, vol. 150, no. 1, pp. 209-241, 2015.

[3] J. Liu, X. Zeng, C. Li, X. Ren, W. Yan, X. Tan, X. Zhang, W. Chen, W. Zuo, Y. Liu et al., ”Landing site selection and overview of china’s lunar landing missions”, Space Science Reviews, vol. 217, no. 1, pp. 1-25, 2021.

[4] K. Iiyama, K. Tomita, B. A. Jagatia, T. Nakagawa and K. Ho, ”Deep reinforcement learning for safe landing site selection with concurrent consideration of divert maneuvers”, arXiv preprint arXiv:2102.12432, 2021.

[5] K. Sacksteder and G. Sanders, ”In-situ resource utilization for lunar and mars exploration”, 45th AIAA Aerospace Sciences Meeting and Exhibit, pp. 345, 2017.

[6] L. Ma, Z. Shao, W. Chen and Z. Song, ”Trajectory optimization for lunar soft landing with a hamiltonian-based adaptive mesh refinement strategy”, Advances in Engineering Software, vol. 100, pp. 266-276, 2016.

[7] M. Golombek, D. Kipp, N. Warner, I. J. Daubar, R. Fergason, R. L. Kirk, R. Beyer, A. Huertas, S. Piqueux, N. Putzig et al., ”Selection of the insight landing site”, Space Science Reviews, vol. 211, no. 1, pp. 5-95, 2017.

[8] M. Pontani and B. A. Conway, ”Particle swarm optimization applied to space trajectories”, Journal of Guidance Control and Dynamics, vol. 33, no. 5, pp. 1429-1441, 2019.

[9] N. Remesh, R. Ramanan and V. Lalithambika, ”Fuel optimum lunar soft landing trajectory design using different solution schemes”, International Review of Aerospace Engineering (IREASE), vol. 9, no. 5, pp. 131-143, 2016.

[10] N. Remesh and R. Ramanan, ”Optimal 3d lunar soft landing trajectory design and performance evaluation of explicit guidance laws”, 2017 First International

Conference on Recent Advances in Aerospace Engineering (ICRAAE), pp. 1-6, 2017.

[11] R. Cortes-Martinez, K. D. Kumar and H. Rodriguez-Cortes, ”Precise power descent control of a lunar lander using a single thruster”, Acta Astronautica, vol. 186, pp. 473-485, 2021.

[12] S. Chen, Y. Li, T. Zhang, X. Zhu, S. Sun and X. Gao, ”Lunar features detection for energy discovery via deep learning”, Applied Energy, vol. 296, pp. 117085, 2021.

[13] S. Kumar Choudhary, K. Raj and V. Muthukumar, ”Optimal trajectory design and analysis for soft landing on the moon from lunar parking orbits”, Int. J. Space Science and Engineering, vol. 5, no. 4, 2019.

[14] W. Li and X. Wang, ”Optimization of Lunar Landing Trajectories Using Deep Reinforcement Learning,” Aerospace Science and Technology, vol. 106, 2020.

[15] X.-L. Liu, G.-R. Duan and K.-L. Teo, ”Optimal soft landing control for moon lander”, Automatica, vol. 44, no. 4, pp. 1097-1103, 2018.