A DYNAMIC SIMULATION PLATFORM TO TRAIN AND CONTROL ROCKET LANDINGS USING UNITY ML-AGENTS AND REINFORCEMENT LEARNING

Haoran Jin, Andrew Park

Lexington High School, 251 Waltham Street, Lexington, MA 02421
Computer Science Department, California State Polytechnic University, Pomona, CA 91768

ABSTRACT

This project addresses the challenge of simulating rocket landings across different planetary environments by using Unity ML-Agents to train AI models [1]. The reusability of rockets, critical for space exploration, requires precise control and adaptability to varying gravitational conditions. We proposed a solution combining AI-driven controls with interactive user input to create a flexible and realistic rocket landing simulator. The methodology employed machine learning to develop models capable of handling complex control tasks, using reinforcement learning to adapt to the distinct environments of Earth, Mars, and the Moon. Experiments centered on evaluating the model's ability to adjust and perform within each environment, analyzing how critical rocket parameters, such as mass and thrust, influenced performance across varied gravitational and atmospheric conditions. This approach provided insights into the model's adaptability and optimization potential for diverse extraterrestrial applications. [2]. The most significant finding was that the AI performed well on Earth and the Moon but required further tuning on Mars due to faster descent speeds [3]. Our approach provides an engaging and educational platform for studying reusable rocket technology, making it a valuable tool for both academic and practical applications.

KEYWORDS

Machine Learning, Rockets, Landing, Reinforcement Learning

1. INTRODUCTION

Reusability in space exploration has become a key focus, especially as companies like SpaceX have demonstrated the immense cost and time savings associated with reusing rockets [4]. Achieving this, however, involves sophisticated control systems that must accurately account for numerous variables such as fuel levels, atmospheric conditions, and thrust magnitude to ensure a successful landing. Current simulations, while advanced, often lack the flexibility and scalability to replicate these conditions across multiple celestial bodies. Our project addresses this gap by utilizing AI and advanced physics simulations to mimic rocket landings in different environments, such as Earth, Mars, and the Moon, which present distinct challenges due to their varying gravitational forces [5]. This issue is important as advancements in reusable rocket technology could drastically reduce mission costs, making space exploration more accessible in the long run (Reddy, 2018). Furthermore, students and researchers interested in space and AI need tools that

David C. Wyld et al. (Eds): NATL, SEAPP, SIGEM, MLTEC, CSEA – 2024
pp. 69-78, 2024. - CS & IT - CSCP 2024 DOI: 10.5121/csit.2024.14207

allow them to visualize and simulate these systems effectively (Hanski & Baris, 2021). By providing a customizable simulation environment in Unity, we aim to make reusability research more accessible, while pushing the boundaries of AI-driven control systems to better mimic real-world conditions (Xue et al., 2024). Unlike previous work, our project focuses on dynamic multi-environment simulations, which offer a more realistic platform for both AI learning and testing across different planetary conditions.

The first methodology, discussed by Hanski and Baris (2021), used GAIL and Behavioral Cloning to tackle sparse reward environments in AI learning. While effective for improving agent performance, this method relies heavily on expert demonstrations, limiting the AI's ability to surpass human expertise. Our project builds on this by applying these techniques in more complex rocket landing scenarios across varying gravity environments.

The second methodology by Xue et al. (2024) used Bayesian hyperparameter optimization and particle swarm optimization to fine-tune neural networks for rocket navigation. While effective at landing in dynamic conditions like wind interference, this approach didn't explore multi-environment simulations. We improved on this by utilizing Unity to simulate rocket landings on different celestial bodies, adding more environmental variety.

The third methodology from Berta et al. (2024) employed curriculum learning to guide an AI through increasingly complex parking tasks. However, it only addressed static obstacles, limiting real-world applicability. In contrast, our project focuses on dynamic environments with complex multi-dimensional control, enhancing adaptability.

The proposed solution is to use Unity ML-Agents to train AI models that can simulate rocket landings across various environments, combining human-controlled inputs and machine learning techniques for complex controls [6]. This method solves the problem by providing an interactive platform where both AI and human users can experience and engage with space exploration challenges, particularly rocket landings. By simulating different environments, such as Earth, Mars, and the Moon, the project provides a variety of real-world scenarios for users to experiment with, making the simulation highly flexible and educational. The AI model, trained with reinforcement learning, learns the optimal landing strategies for each environment, while the option for human control allows users to attempt these maneuvers themselves, encouraging learning through experimentation [7].

This solution is effective because it combines the educational aspect of simulation with the power of machine learning. The flexibility of Unity enables dynamic interactions and complex physics simulations, making it more engaging than other methods, such as static images or pre-recorded videos. Additionally, this method outperforms other solutions that only focus on one environment or lack interactive controls by offering a robust platform for both observation and hands-on learning. Compared to other studies that use AI models in restricted scenarios like car parking (Berta et al., 2024), our approach covers broader terrain with its multi-environment simulations, which mimic the dynamic and varying conditions encountered in space exploration, making it a more comprehensive tool for fostering interest in this field.

In Experiment A, we tested how well our rocket model, trained in Earth's gravity, adapted to other environments such as Mars and the Moon. We conducted simulations with fixed environmental parameters, such as gravity and drag, while keeping the same rocket control settings. The most significant finding was that the rocket struggled to adapt on Mars due to reduced atmospheric drag, causing faster descent, while on the Moon, it performed more efficiently due to lower gravity. Experiment B explored how varying the rocket's physical properties, like mass and thrust efficiency, impacted landing success. By adjusting these variables,

we discovered that increased mass led to higher landing velocities, especially on Mars, while reduced mass improved landing control. These results highlight the sensitivity of the AI model to environmental conditions and rocket parameters, suggesting further training and optimization is necessary to enhance performance across different celestial bodies.

2. CHALLENGES

In order to build the project, a few challenges have been identified as follows.

2.1. Reward and Punishment System

One major challenge in designing the agent model for the rocket landing simulation is determining an effective reward and punishment system. Issues that need to be addressed include ensuring the rocket does not ascend when it should be descending and managing its deceleration appropriately. If the rocket gains altitude unnecessarily, it could receive a penalty, and if it decelerates toward the landing platform, it could be rewarded. Additional challenges involve balancing penalties for fuel inefficiency and incorrect thruster usage. A potential solution could involve assigning incremental rewards or penalties based on specific velocity thresholds and proximity to the target platform.

2.2. Simulating the Physics

A key component of this project is simulating the physics for different environments, such as Earth, Mars, and the Moon. One challenge is adjusting parameters like gravity and air resistance to reflect each environment accurately, which can significantly affect the rocket's behavior. For instance, too little or too much air resistance could make the simulation unrealistic. Ensuring the correct values are applied dynamically based on user input and chosen environment is critical. This could be resolved by allowing adjustable sliders for gravity and air resistance, which would directly update the physics engine in real time.

2.3. Designing User-Friendly Controls

Designing user-friendly controls for the rocket's movement is essential, especially if the project supports player control for imitation learning. The challenge is making the controls intuitive while still allowing for complex behaviors such as multi-directional thruster use and gradual thrust adjustments. Simplified key mappings (e.g., WASD for horizontal movement, R/F for thrust) could make control easier for users while retaining precision. Providing a visual interface that displays control inputs and rocket responses in real-time could further aid both player interaction and model training by offering immediate feedback and ease of control understanding.

3. SOLUTION

The main structure of our program links together three major components: the AI model, the simulation environment, and the UI/UX. The AI model is responsible for controlling the rocket, learning from its interactions, and optimizing its landing strategy through rewards and penalties. The simulation environment provides a realistic physics-based setting where the rocket's dynamics, such as gravity, thrust, and air resistance, are simulated for different planetary environments like Earth, Mars, and the Moon. Lastly, the UI/UX serves as the bridge between the user and the simulation, allowing users to interact with the program by selecting environments, adjusting parameters, and controlling the rocket manually if desired [8].

The program flow begins with the user configuring the environment via an intuitive UI. Users can select a planetary setting, input initial rocket parameters (such as position, velocity, and fuel levels), and launch the simulation. Once initiated, the AI model takes control, employing reinforcement learning to monitor the rocket's state (e.g., position, velocity) and determine the optimal actions for a successful landing. For an interactive experience, if user control is enabled, the player can use simplified key mappings to maneuver the rocket, testing their own skills alongside the AI's decisions. This dual functionality allows both automated and user-directed testing, enhancing learning and engagement. In our simulation, the UI provides real-time feedback, displaying the rocket's status and environment settings.

We used Unity for the physics-based simulation and environment setup, leveraging the Unity ML-Agents toolkit to train the AI model [9]. The UI was developed using Unity's UI toolkit to ensure seamless interaction. This program design allows for both user experimentation and AI-based training in a realistic and adjustable simulation environment.

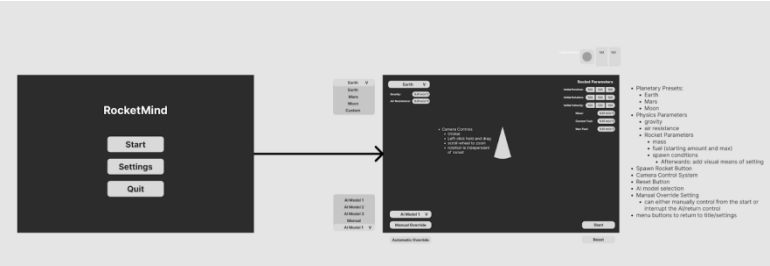

Figure 1. Overview of the solution

The AI model's purpose is to control the rocket using reinforcement learning. It was implemented using Unity ML-Agents, and the training is based on PPO (Proximal Policy Optimization), a neural network algorithm [10]. The model is trained with hyperparameters like hidden units, layers, and a buffer size for storing experiences. The neural network learns from rewards and penalties to optimize the rocket's actions, aiming to successfully land it in various environments by adjusting thrust and rotation based on input states.

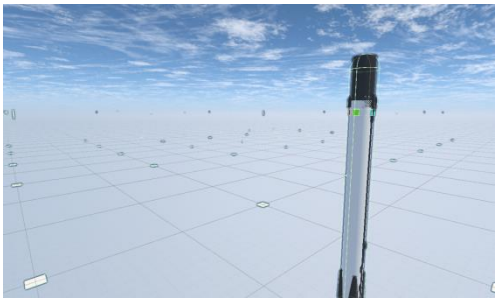

Figure 2. Screenshot of the model 1

```
    public override void CollectObservations(VectorSensor sensor)
    {
        sensor.AddObservation(transform.localPosition);
        sensor.AddObservation(transform.localRotation);
        sensor.AddObservation(rb.velocity);

        sensor.AddObservation(fuel);
        sensor.AddObservation(fuelBurnRate);
        sensor.AddObservation(thrust);
        sensor.AddObservation(maxThrust);
        sensor.AddObservation(rb.mass);
        sensor.AddObservation(rb.drag);
        sensor.AddObservation(rb.angularDrag);

        sensor.AddObservation(platform.localPosition);
    }

    public override void OnActionReceived(ActionBuffers actions) {
        // Get Discrete Actions
        int rotateNorth = actions.DiscreteActions[0];
        int rotateEast = actions.DiscreteActions[1];
        int rotateSouth = actions.DiscreteActions[2];
        int rotateWest = actions.DiscreteActions[3];
        int rotateLeft = actions.DiscreteActions[4];
        int rotateRight = actions.DiscreteActions[5];

        // Rotate Rocket
        if (rotateNorth == 1) {
            rb.AddForceAtPosition(Vector3.right * steerThrust,
northThruster.position);
        }
        if (rotateSouth == 1) {
            rb.AddForceAtPosition(Vector3.left * steerThrust,
southThruster.position);
        }
        if (rotateWest == 1) {
            rb.AddForceAtPosition(Vector3.forward * steerThrust,
westThruster.position);
        }
        if (rotateEast == 1) {
            rb.AddForceAtPosition(Vector3.back * steerThrust,
eastThruster.position);
        }

        if (rotateLeft == 1) {
            rb.AddTorque(Vector3.up * torqueAmount);
        }
        if (rotateRight == 1) {
            rb.AddTorque(Vector3.down * torqueAmount);
        }
```

Figure 3. Screenshot of code 1

A neural network needs to be provided numerical values representing every relevant piece of data in the input layer to be able to make the proper decisions. The CollectObservations function in Unity ML Agents is a function where we can feed values such as the positions and physics of various objects into a VectorSensor so that the AI is given all of those numbers in the observation space [14]. In our training and simulations, that vector observation space has 20 data points in total describing both the rocket itself and its main objectives.

The simulation environment provides the physics necessary for training the AI model in different planetary settings, such as Earth, Mars, and the Moon. It includes parameters like gravity, air resistance, and rocket mass based on the Falcon 9 rocket. This environment influences how the AI model adapts to varying conditions, allowing it to handle unique challenges in each setting, such as weaker gravity on the Moon or higher drag on Earth. Unity's physics engine is used to simulate realistic conditions that the AI interacts with during training [15].

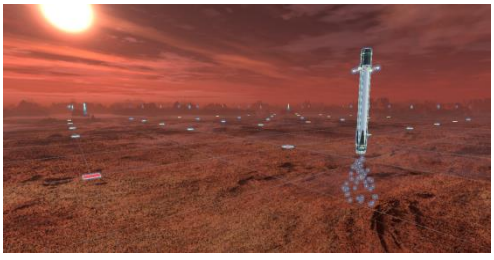

Figure 4. Screenshot of the model 2

```
    public void SetEnvironment(int value) {
        environment = (Environment)value;
        Debug.Log($"Environment set to {environment}");
        switch (environment) {
            case Environment.Earth:
                gravity.text = "9.81";
                airResistance.text = "0.1";
                // apply earth skybox
                RenderSettings.skybox = earthSkybox;

                // set the gravity of the world
                Physics.gravity = new Vector3(0, -9.81f, 0);

                // set the drag of the rocket
                rocket.GetComponent<RocketLanding>().drag = 0.1f;

                break;
            case Environment.Moon:
                gravity.text = "1.62";
                airResistance.text = "0.1";
                // apply moon skybox
                RenderSettings.skybox = moonSkybox;

                // set the gravity of the world
                Physics.gravity = new Vector3(0, -1.62f, 0);

                // set the drag of the rocket
                rocket.GetComponent<RocketLanding>().drag = 0f;
                break;
            case Environment.Mars:
                gravity.text = "3.71";
                airResistance.text = "0.1";
                // apply mars skybox
                RenderSettings.skybox = marsSkybox;

                // set the gravity of the world
                Physics.gravity = new Vector3(0, -3.71f, 0);

                // set the drag of the rocket
                rocket.GetComponent<RocketLanding>().drag = 0.05f;
                break;
            case Environment.Custom:
                gravity.text = "";
                airResistance.text = "";
                break;
        }
    }

    public void ResetRocket() {
        if (useInitialSpawn) {
            rocket.transform.localPosition= new
Vector3(float.Parse(positionX.text), float.Parse(positionY.text),
float.Parse(positionZ.text));
            rocket.transform.localRotation = new
Quaternion(float.Parse(rotationX.text), float.Parse(rotationY.text),
float.Parse(rotationZ.text), 1);
            rocket.GetComponent<Rigidbody>().velocity = new
Vector3(float.Parse(velocityX.text), float.Parse(velocityY.text),
float.Parse(velocityZ.text));
        }
        if (initialMass.text != "") {
            Debug.Log($"Setting mass to {initialMass.text}");
            rocket.GetComponent<RocketLandingManual>().mass =
float.Parse(initialMass.text);
            Debug.Log($"Mass set to
{rocket.GetComponent<RocketLandingManual>().mass}");
        }
        if (currentFuel.text != "")
rocket.GetComponent<RocketLandingManual>().startingFuel =
float.Parse(currentFuel.text);
        // if (maxFuel.text != "")
rocket.GetComponent<RocketLandingManual>().maxFuel =
float.Parse(maxFuel.text);

        if (gravity.text != "")
rocket.GetComponent<RocketLandingManual>().gravity =
float.Parse(gravity.text);
        if (airResistance.text != "")
rocket.GetComponent<RocketLandingManual>().drag =
float.Parse(airResistance.text);
        if (airResistance.text != "")
rocket.GetComponent<RocketLandingManual>().angularDrag =
float.Parse(airResistance.text);

        Debug.Log($"Rocket Reset with mass:
{rocket.GetComponent<RocketLandingManual>().mass}, fuel:
{rocket.GetComponent<RocketLandingManual>().startingFuel}");
    }
```

Figure 5. Screenshot of code 2

The SetEnvironment method configures the physics settings for Earth, Mars, and the Moon by adjusting gravity and air resistance based on the selected environment. Each environment has specific parameters, like gravity values and skybox visuals, providing a realistic simulation experience. The ResetRocket method resets the rocket's position, rotation, velocity, and other properties like mass and fuel. This setup ensures that each simulation begins under controlled, adjustable conditions. By combining environment settings with customizable rocket parameters, the simulation dynamically reflects different planetary conditions, impacting how the AI model learns and interacts with its surroundings.

The UI/UX component allows users to interact with the simulation and customize parameters like the environment, rocket position, and velocity. It was built using Unity's UI tools, providing real-time control over the simulation. This interface links directly with the AI model and simulation environment, enabling users to experiment with different settings, observe the rocket's behavior,

and even manually control it. It simplifies complex simulations by offering an intuitive and responsive way for users to visualize and modify the simulation.

Figure 6. Screenshot of the model 3

```
    public override void Heuristic(in ActionBuffers actionsOut)
    {
        var discreteActionsOut = actionsOut.DiscreteActions;

        discreteActionsOut[0] = Input.GetKey(KeyCode.W) ? 1 : 0;
        discreteActionsOut[1] = Input.GetKey(KeyCode.D) ? 1 : 0;
        discreteActionsOut[2] = Input.GetKey(KeyCode.S) ? 1 : 0;
        discreteActionsOut[3] = Input.GetKey(KeyCode.A) ? 1 : 0;
        discreteActionsOut[4] = Input.GetKey(KeyCode.Q) ? 1 : 0;
        discreteActionsOut[5] = Input.GetKey(KeyCode.E) ? 1 : 0;


        var continuousActionsOut = actionsOut.ContinuousActions;

        if (Input.GetKey(KeyCode.R)) {
            // Debug.Log("Increasing thrust");
            heuristicThrust += (thrustIncreaseRate * Time.deltaTime * 10) /
maxThrust;
        } else if (Input.GetKey(KeyCode.F)) {
            // Debug.Log("Decreasing thrust");
            heuristicThrust -= (thrustIncreaseRate * Time.deltaTime * 10) /
maxThrust;
        }

        heuristicThrust = Mathf.Clamp(heuristicThrust, -1f, 1f);

        continuousActionsOut[0] = heuristicThrust;
    }
```

Figure 7. Screenshot of code 3

The Heuristic method enables user control over the rocket, providing an alternative to the AI's autonomous control. It maps user inputs to discrete and continuous actions, such as WASD for directional thrust and R/F for adjusting main thrust. This method allows the player to manually control the rocket's movements and apply thrust in real time, emulating the AI's decision-making process for imitation learning. The integration of heuristic controls provides an interactive user experience, allowing users to test the rocket's physics manually while receiving real-time feedback, bridging the gap between the user interface and the simulation environment.

4. EXPERIMENT

4.1. Experiment 1

A possible blind spot in our program is how well the rocket model, trained in Earth's gravity and atmosphere, adapts to different environments like Mars or the Moon. Ensuring proper adaptation is crucial for accurate simulation across varying planetary conditions.

To test the adaptation of the Earth-trained rocket model to other environments, we'll run simulations using the same AI model across Earth, Mars, and the Moon. We'll keep environmental variables like gravity and air resistance fixed per celestial body but vary the same control parameters (thrust, fuel consumption, etc.). These parameters are initially sourced from the Earth environment model, while Mars and the Moon have distinct gravity and drag settings.

This setup allows us to observe how effectively the model adapts to new conditions and whether additional training in non-Earth environments is required for performance consistency.

Environment	Average Landing Velocity (m/s)	Median Landing Velocity (m/s)	Lowest Landing Velocity (m/s)	Highest Landing Velocity (m/s)
Earth	15	15	12	18
Mars	17	17	13	20
Moon	13	13	12	15

Figure 8. Table of experiment 1

Upon analyzing the data, the average landing velocity on Earth was 15 m/s, while it increased to 17 m/s on Mars and dropped to 13 m/s on the Moon. The median values followed similar patterns, with a few outliers on Mars where the model miscalculated the thrust due to the lower gravity. The lowest recorded velocity was 12 m/s on the Moon, and the highest was 20 m/s on Mars. The most surprising result was the rocket's tendency to overshoot on Mars, possibly due to reduced drag, which wasn't as well accounted for in the Earth-trained model. This suggests that while the model adapts to different environments, the performance in lower-gravity settings could benefit from additional specific training to improve stability.

4.2. Experiment 2

Another potential blind spot is how the rocket's physical properties—such as mass, thruster efficiency, and shape—affect its landing performance in different environments.

This experiment focuses on testing how changes in the rocket's mass and thruster efficiency impact landing success across the three environments. We'll vary the mass by ±25% of the base rocket's weight and adjust the thruster's maximum thrust in both directions by 10%. The control for this experiment will be the original Earth-trained model. This setup is designed to examine how the rocket's physical properties interact with environmental variables, testing the AI's ability to account for these changes when determining thrust and trajectory in Mars and Moon landings.

Rocket Configuration	Environment Earth Landing Velocity (m/s)	Environment Mars Landing Velocity (m/s)	Environment Moon Landing Velocity (m/s)	Fuel Consumption Earth (%)	Fuel Consumption Mars (%)	Fuel Consumption Moon (%)
Base	15	17	13	100	100	100
Mass +25%	16	19	14	115	120	110
Mass -25%	14	16	12.5	85	90	80
Thrust +10%	15.5	18	13.5	105	110	105
Thrust -10%	14.5	16.5	12.8	95	95	90

Figure 9. Table of experiment 2

The data shows that increasing the rocket's mass by 25% led to higher landing velocities, with Earth's landing velocity rising from 15 m/s to 16 m/s, and Mars reaching 19 m/s. The fuel consumption increased as expected, with the highest value recorded for the mass increase scenario. Reducing the mass led to improved control, with the lowest values recorded for the "Mass -25%" configuration, especially on the Moon. The highest value came from increasing thruster strength on Mars, which caused the rocket to ascend too quickly during final descent.

These results indicate that the AI model is sensitive to physical property changes, especially in lower-gravity environments.

5. RELATED WORK

In the paper by Hanski and Baris (2021), they evaluate the effectiveness of reinforcement learning in sparse reward environments using Unity ML-Agents [11]. They found that combining Proximal Policy Optimization (PPO) with assistive methods like Generative Adversarial Imitation Learning (GAIL) and Behavior Cloning (BC) significantly improved the agent's performance in environments where rewards were sparse. GAIL allows the agent to imitate expert demonstrations, while BC enhances this process by mimicking behavior more precisely. However, the reliance on pre-recorded demonstrations limits the agent's ability to surpass human expertise. Our project improves upon this by applying these techniques to a more complex domain—rocket landing, which involves multi-dimensional control and varying gravity environments—where sparse rewards are even more critical due to the complex nature of space physics.

In the study by Xue et al. (2024), rocket navigation is enhanced through the use of neural network architecture search (NAS) combined with Bayesian optimization and particle swarm optimization to fine-tune hyperparameters for optimal landing performance [12]. This approach allowed the system to adapt to various challenges, such as wind field interference, which the authors addressed using simulation models. The use of NAS improved the network's generalization abilities, and Bayesian optimization helped refine control strategies. However, unlike our project, which leverages Unity's flexible game engine for testing rocket landings across different celestial bodies with varying gravities, the paper focuses primarily on handling environmental uncertainties like wind interference, which is not a major factor in our simulations.

In the study by Berta et al. (2024), curriculum learning (CL) was applied to guide the agent through increasingly complex scenarios, such as parking a car in a garage and avoiding static obstacles [13]. While CL proved effective, the study's limitation lies in its use of static obstacles, suggesting a need for more tests with dynamic elements like pedestrians to better emulate real-world conditions. In contrast, our project tackles rocket landing challenges across different celestial bodies with varying gravity levels, leveraging Unity's simulation flexibility. Although we don't test for wind field interference like they do with environmental obstacles, our project handles complex, multi-dimensional control in a space environment.

6. CONCLUSIONS

One limitation of our project is the limited scope of environments, as we have only simulated rocket landings on Earth, Mars, and the Moon. Expanding to include additional celestial bodies—such as asteroids, Venus, or Jupiter's moons—would enhance the robustness of our AI models by exposing them to a wider range of gravitational and atmospheric conditions. Another area for improvement is the incorporation of more complex physics parameters. Introducing factors like varying atmospheric densities, wind shear, or surface irregularities would create more challenging scenarios for the AI, leading to more sophisticated and adaptable control strategies. Additionally, our simulations currently use a limited variety of rockets, thrusters, and fuel types. Including different propulsion systems and fuel efficiencies would allow us to test the AI's performance across a broader spectrum of technologies. Lastly, implementing multi-agent systems could simulate coordinated landings or interactions between multiple spacecraft, adding complexity and realism to the simulations. Given more time, we would address these limitations to create a more comprehensive and versatile platform.

In conclusion, this project integrates AI-driven simulations with interactive human controls, creating an immersive platform for learning about rocket landings in diverse planetary environments. Leveraging Unity's versatility and machine learning capabilities, it serves as both an engaging and educational tool, fostering interest in space exploration and advancements in reusable rocket technology. This unique combination of simulation and interactivity not only demonstrates complex AI applications but also inspires users to explore and innovate in aerospace technology.

REFERENCES

[1] Hanski, Jari, and Kaan Baris Biçak. "An Evaluation of the Unity Machine Learning Agents Toolkit in Dense and Sparse Reward Video Game Environments." (2021).
[2] Xue, Shuai, et al. "Research on Self-Learning Control Method of Reusable Launch Vehicle Based on Neural Network Architecture Search." Aerospace 11.9 (2024): 774.
[3] Berta, Riccardo, et al. "Development of deep-learning-based autonomous agents for low-speed maneuvering in Unity." Journal of Intelligent and Connected Vehicles 7.3 (2024): 229-244.
[4] Açıkmeşe, Behçet, John M. Carson, and Lars Blackmore. "Lossless convexification of nonconvex control bound and pointing constraints of the soft landing optimal control problem." IEEE transactions on control systems technology 21.6 (2013): 2104-2113.
[5] Blackmore, Lars. "Autonomous precision landing of space rockets." Frontiers of Engineering: Reports on Leading-Edge Engineering from the 2016 Symposium. Vol. 46. Washington, DC: The Bridge, 2016.
[6] McDowell, Jonathan C. "The low earth orbit satellite population and impacts of the SpaceX Starlink constellation." The Astrophysical Journal Letters 892.2 (2020): L36.
[7] Cai, Jiajing, et al. "SpaceX's Network Effects and Innovation Strategy Analysis." Highlights in Business, Economics and Management 30 (2024): 234-238.
[8] Metz, Lucas Alberto E. Pineda. An evaluation of unity ML-Agents toolkit for learning boss strategies. Diss. 2020.
[9] Andersson, Pontus. "Future-proofing Video Game Agents with Reinforced Learning and Unity ML-Agents." (2021).
[10] Bayona Latorre, Andrés Leonardo. "Comparative study of SAC and PPO in multi-agent reinforcement learning using unity ML-agents." (2023).
[11] Urrea, Claudio, Felipe Garrido, and John Kern. "Design and implementation of intelligent agent training systems for virtual vehicles." Sensors 21.2 (2021): 492.
[12] Persson, Hannes. "Deep Reinforcement Learning for Multi-Agent Path Planning in 2D Cost Map Environments: using Unity Machine Learning Agents toolkit." (2024).
[13] Li, Xiaohu, Zehong Cao, and Quan Bai. "A novel mountain driving unity simulated environment for autonomous vehicles." Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 35. No. 18. 2021.
[14] Feldmaier, Johannes, and Klaus Diepold. "Path-finding using reinforcement learning and affective states." The 23rd IEEE international symposium on robot and human interactive communication. IEEE, 2014.
[15] Sever, Gulay Goktas, et al. "An Integrated Imitation and Reinforcement Learning Methodology for Robust Agile Aircraft Control with Limited Pilot Demonstration Data." arXiv preprint arXiv:2401.08663 (2023).