A Simulation Program that Controls Rockets using AI Trained with Machine Learning

Zhenrui Guo

Santa Margarita Catholic High School , 22062 Antonio Pkwy,
Rancho Santa Margarita, CA 92688

ABSTRACT

Launching rockets is a costly and complex task that is prone to a high rate of failure. To help address these challenges, the need for simulations to help emulate reality when developing rockets is crucial to making sure the cost overhead stays as low as possible. Our rocket AI seeks to tackle this challenge through the use of machine learning and a series of emulated phases to demonstrate an AI's ability to properly operate a rocket. The primary challenge throughout the development of the rocket's AI is the need to introduce and train for unexpected situations, which is done by training for a generalized variant of the situations in question.

KEYWORDS

Unity ,Machine Learning, Rockets, Simulation

1. Introduction

With the low rate of success in the Rocket launching industry, trying to make rocket traveling and landing more convenient and "smart" is an important factor to consider.[2] The issue this creates is that there is a considerable amount of time and money to properly run. While modern rocketry is getting cheaper as "SpaceX has estimated a Falcon 9 launch cost of $2,700 per kilogram versus $20,000 per kilogram," [3] there is still a significant cost overhead that must be considered when trying to minimize the cost of development. Therefore it is important to be able to continue running tests even via a simulated environment to ensure testing continues at a steady pace while also giving opportunities to identify issues between each of the real-life tests.[5] It also saves time to install a controller on the rocket, instead a pre-made AI system will be installed on the rocket to save money and time.[10] The AI system will grow with the Rocket, to develop real-time usage of the AI system.

1.1. Method Proposal

To properly address this problem, we will want to put together a proof of concept showing how machine learning can be leveraged to better guide the activities of rockets during their flight. This is done to further contribute to the creation of new novel methods of implementing control schemes that will provide rockets with better guidance. The solution itself is broken down into three separate phases. The first phase covers the landing aspect of the rocket and tackles the question of how AI can optimize landing behavior so that the rocket will be able to land softly enough to avoid significant damage.[14] The second has to do with properly orienting the rocket to face the direction it needs to at a given time. The third phase tackles the traveling aspect where the rocket must reach a specific goal once in the air from liftoff. This separation of concerns allows us to hone in on different aspects of the AI system so that each can be sufficiently accurate in the final ensemble.

2. CHALLENGES

2.1. Challenge A

The most notable challenge, in general, has to be figuring out in what manner the AI should be trained in the first place. It was hard to train the AI, because if we train the AI so strictly, it can never achieve success, if we train too loosely, the AI often doesn't meet our requirements or goals. Part of this difficulty also stems from the overall design of the environment the agent is operating under as it can also impact the reliability of the AI and how it goes about succeeding.

2.2. Challenge B

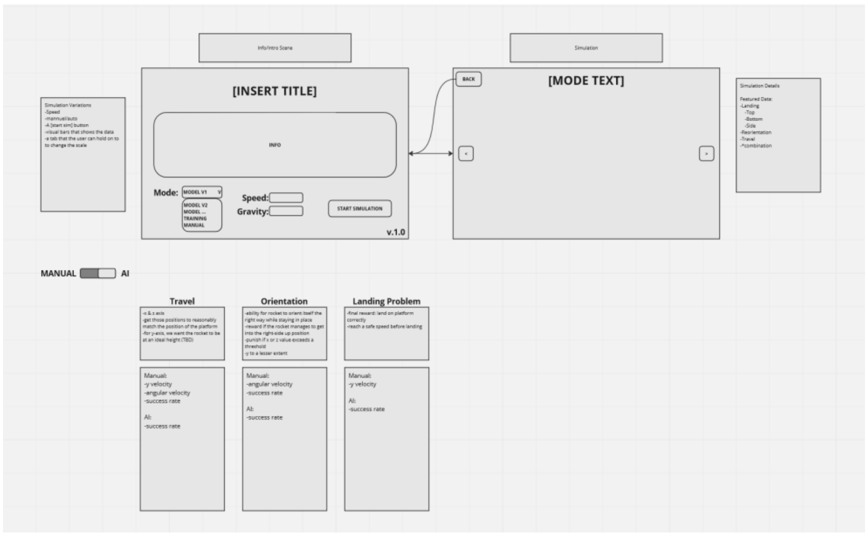
Another notable challenge is figuring out how to best measure and record relevant metrics. This ties into the previous challenge as there was also a lot of uncertainty over how much of a reward or punishment should be given out by the various actions the agent attempts. Set a variable that best fits or best according to real life. Making sure that the environmental behaviors and restrictions are aligned with what to expect from the real world by introducing helper reference objects helps to make sure that the eventual behavior that emerges applies even to real life.

2.3. Challenge C

The application itself must also be presented in a way that is accessible for the general users to emphasize the importance of AI in streamlining training processes. One notable way we can seek out this improvement is to add a separate mode when the human can control their rocket under similar bounds the AI was subjected to better compare how they match up with the AI in question. The AI must also be easily viewable to see how the AI is performing at the given task.

3. METHOD ANALYSIS

3.1. Diagram



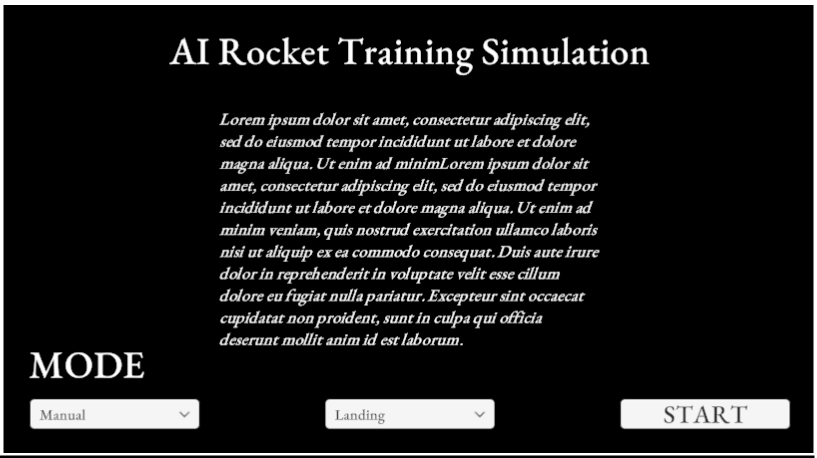
3.1.1. System Overview

The program itself starts on the title screen where the user can make various configurations to get to the specific scenes that they would want to be at. From the title screen, we are then able to progress to the landing, traveling, or orienting scenes to either view or control agents from within each phase. The project can be broken down into the User Scene Selection, AI Agent, and User Interaction components. Each one of these sets out to showcase the result of the machine learning techniques used and how it compared to the control of a regular person. To properly achieve this vision, a combination of Unity and Machine Learning packages was used to implement the necessary changes.

3.2. Component Analysis A

The first component we will be highlighting is the User Scene Selection component. This particular element is important in that it determines what information should be presented to the user from the beginning to make it clear what the user can access and see. The implementation for this part was done through the initial outlining within Miro before moving it to Unity and its UI library.

3.2.1. UI Screenshot



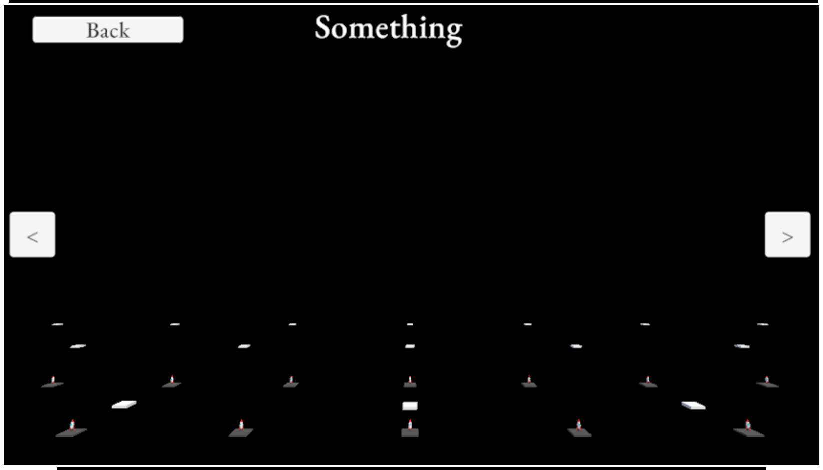
AI Rocket Training Simulation

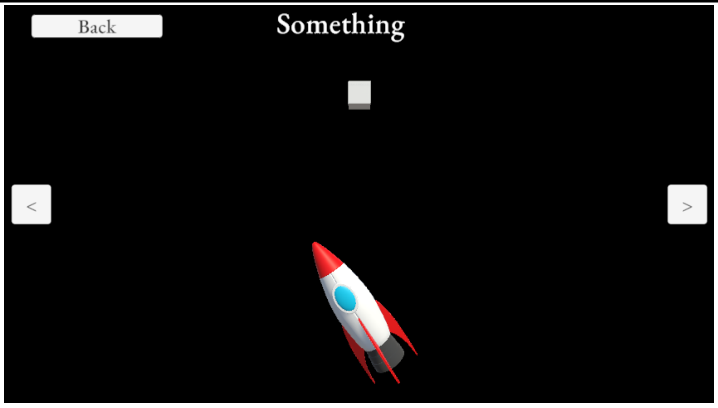
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minimLorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

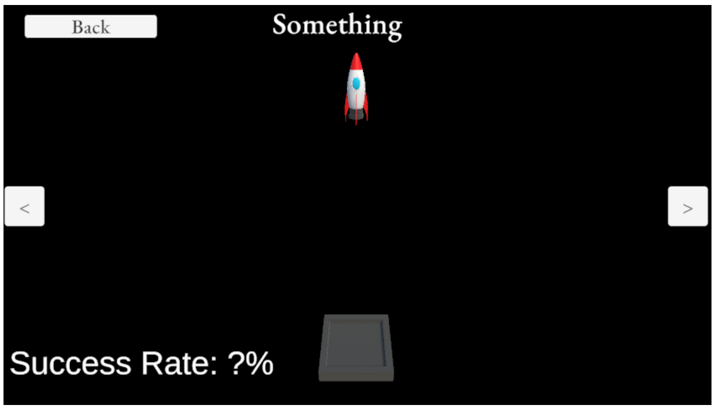
MODE

Manual

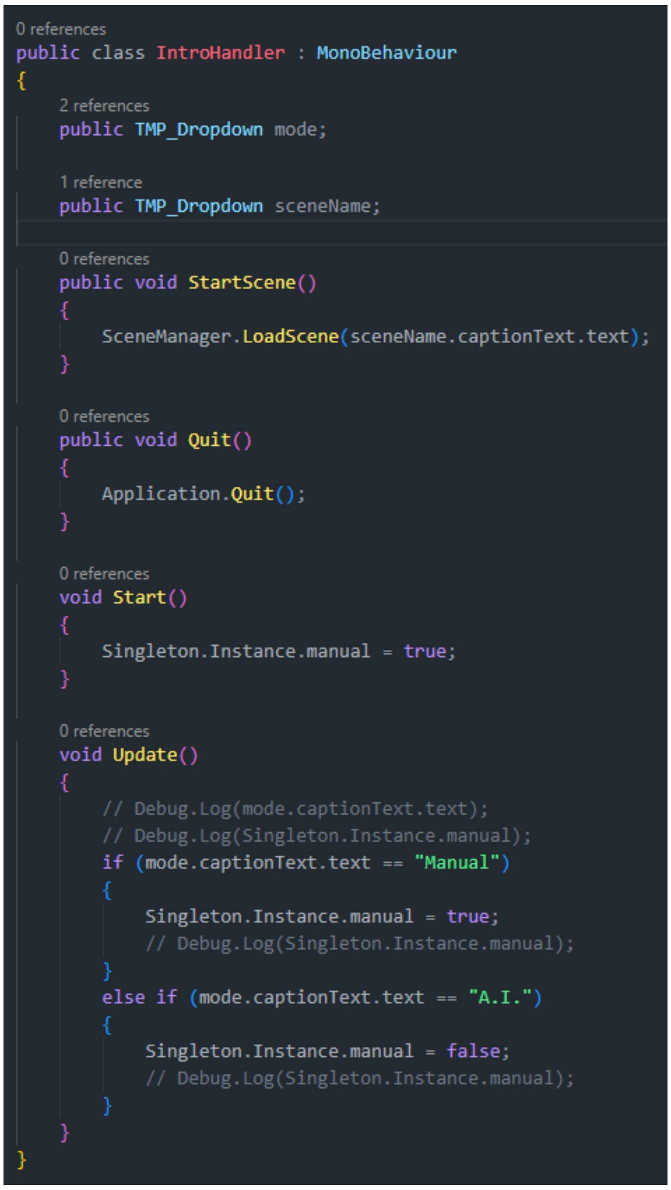
Landing







3.2.2. Code Sample



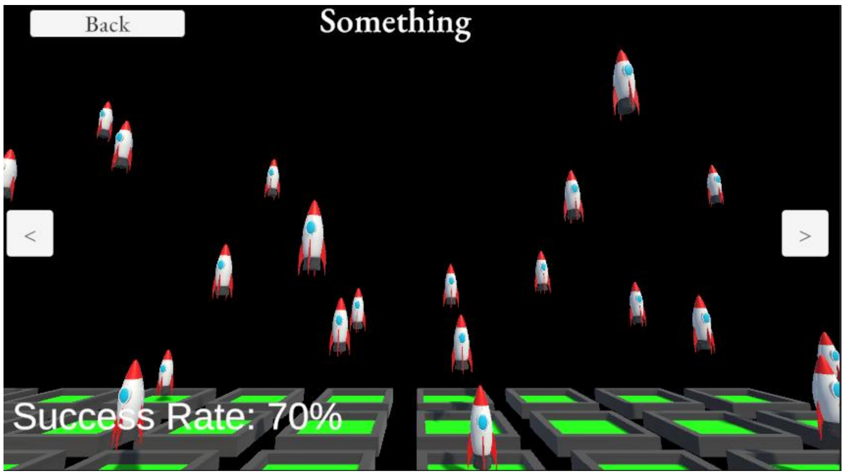
```
0 references
public class IntroHandler : MonoBehaviour
{
    2 references
    public TMP_Dropdown mode;

    1 reference
    public TMP_Dropdown sceneName;

    0 references
    public void StartScene()
    {
        SceneManager.LoadScene(sceneName.captionText.text);
    }

    0 references
    public void Quit()
    {
        Application.Quit();
    }

    0 references
    void Start()
    {
        Singleton.Instance.manual = true;
    }

    0 references
    void Update()
    {
        // Debug.Log(mode.captionText.text);
        // Debug.Log(Singleton.Instance.manual);
        if (mode.captionText.text == "Manual")
        {
            Singleton.Instance.manual = true;
            // Debug.Log(Singleton.Instance.manual);
        }
        else if (mode.captionText.text == "A.I.")
        {
            Singleton.Instance.manual = false;
            // Debug.Log(Singleton.Instance.manual);
        }
    }
}
```

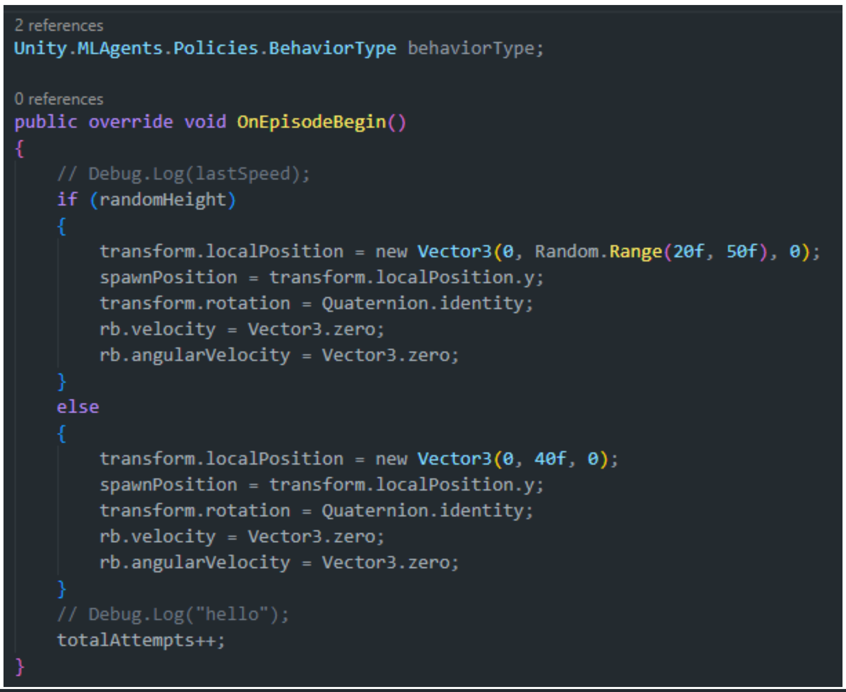
3.2.3. Code Explanation

The code is used to detect the mode in which the scene should be set up, to see if the user put A.I. or Manual mode, and give a function to buttons in the scene to make them terminals to the final scene the user is trying to achieve. Through the pressing of buttons, it also sets different training or manual scene up and switches them as needed. The preferences that are set by the user are saved in a singleton that is running in the background to make sure the choice is persistent between scenes while also being able to neatly reset upon reentering the selection screen where the singleton is located.

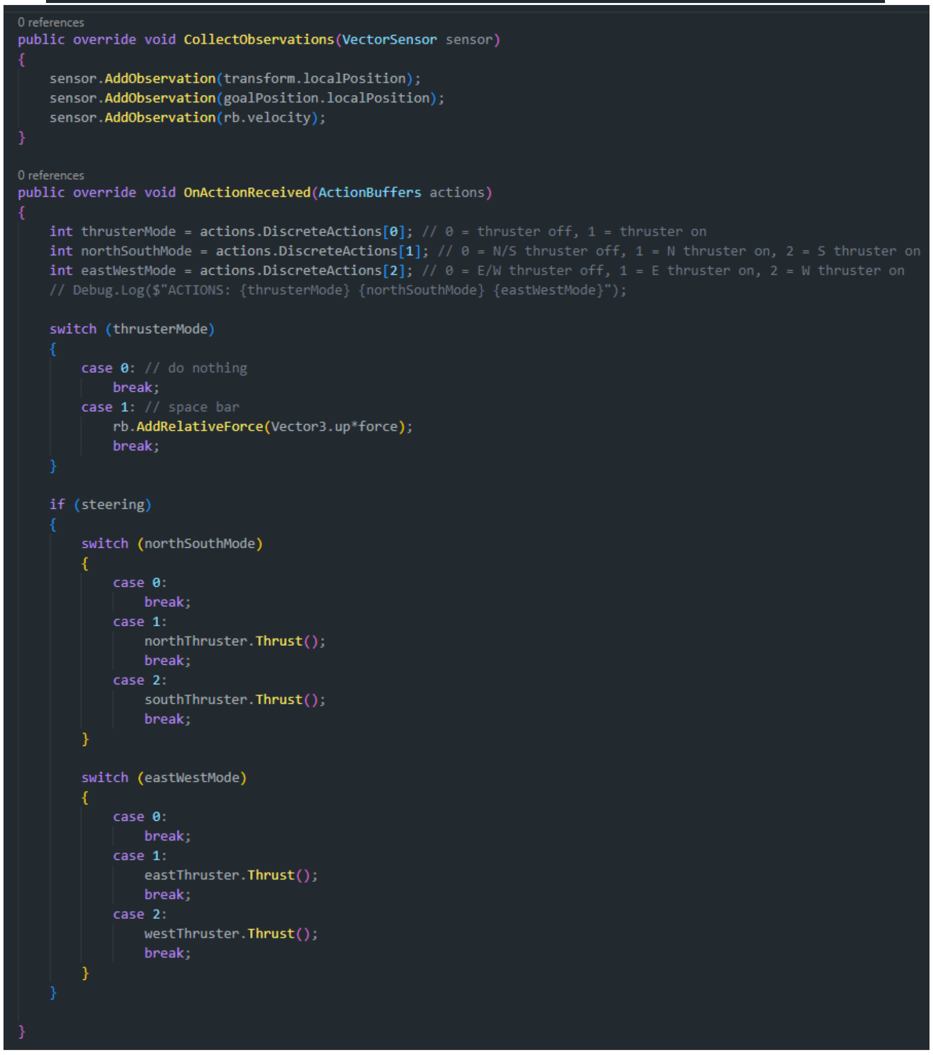
3.3. Component B

The central component tying together the whole project is the AI agents that were trained throughout this project. Each AI was trained through machine learning to create neural networks that are specialized for specific tasks. The training for these models was done in Unity using the UnityML package. The aforementioned tasks themselves were divided into landing, orienting, and traveling phases. This separation was done to ensure the AI will not have too many things it is trying to accomplish at once which could ultimately slow down the results we would be training for.





```
2 references
Unity.MLAgents.Policies.BehaviorType behaviorType;

0 references
public override void OnEpisodeBegin()
{
    // Debug.Log(lastSpeed);
    if (randomHeight)
    {
        transform.localPosition = new Vector3(0, Random.Range(20f, 50f), 0);
        spawnPosition = transform.localPosition.y;
        transform.rotation = Quaternion.identity;
        rb.velocity = Vector3.zero;
        rb.angularVelocity = Vector3.zero;
    }
    else
    {
        transform.localPosition = new Vector3(0, 40f, 0);
        spawnPosition = transform.localPosition.y;
        transform.rotation = Quaternion.identity;
        rb.velocity = Vector3.zero;
        rb.angularVelocity = Vector3.zero;
    }
    // Debug.Log("hello");
    totalAttempts++;
}
```



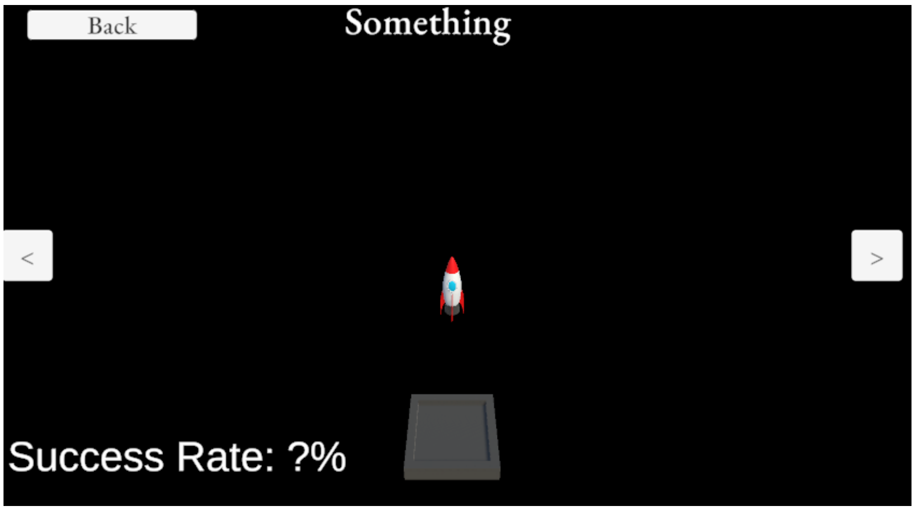
```
0 references
public override void CollectObservations(VectorSensor sensor)
{
    sensor.AddObservation(transform.localPosition);
    sensor.AddObservation(goalPosition.localPosition);
    sensor.AddObservation(rb.velocity);
}

0 references
public override void OnActionReceived(ActionBuffers actions)
{
    int thrusterMode = actions.DiscreteActions[0]; // 0 = thruster off, 1 = thruster on
    int northSouthMode = actions.DiscreteActions[1]; // 0 = N/S thruster off, 1 = N thruster on, 2 = S thruster on
    int eastWestMode = actions.DiscreteActions[2]; // 0 = E/W thruster off, 1 = E thruster on, 2 = W thruster on
    // Debug.Log($"ACTIONS: {thrusterMode} {northSouthMode} {eastWestMode}");

    switch (thrusterMode)
    {
        case 0: // do nothing
            break;
        case 1: // space bar
            rb.AddRelativeForce(Vector3.up*force);
            break;
    }

    if (steering)
    {
        switch (northSouthMode)
        {
            case 0:
                break;
            case 1:
                northThruster.Thrust();
                break;
            case 2:
                southThruster.Thrust();
                break;
        }

        switch (eastWestMode)
        {
            case 0:
                break;
            case 1:
                eastThruster.Thrust();
                break;
            case 2:
                westThruster.Thrust();
                break;
        }
    }

}
```

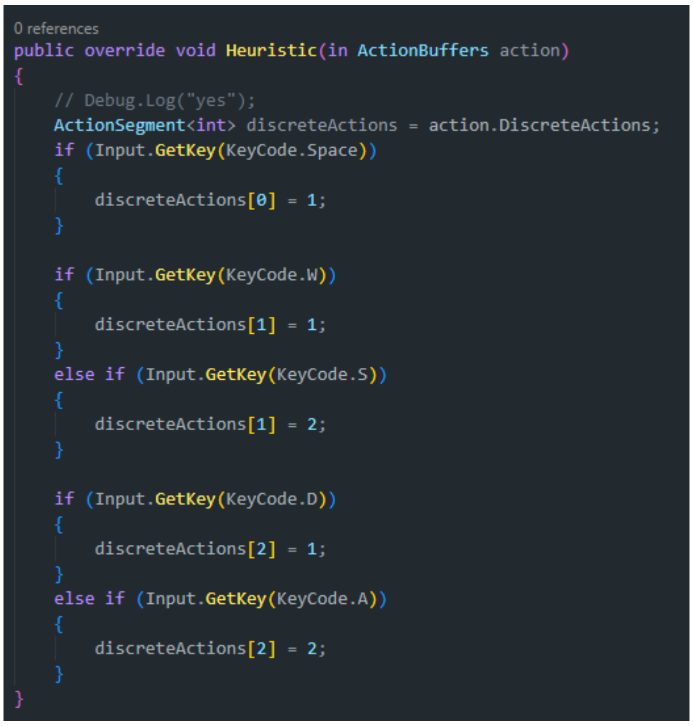
When the AI is being trained, there are a series of functions and steps that need to be taken for the system to work. The machine learning cycle could be summarized as the following: Observing, Acting, and Evaluating. The first part of this cycle starts with the "OnEpisodeBegin," which is where the initial conditions for that specific AI's attempt are set. This is then followed by a "CollectObservations" function which takes in relevant information about the world around it. The Action step of the cycle is then executed by the "OnActionReceived" function which determines what inputs and actions the rocket should do. In the case of our agents, we are taking discrete actions which apply to which inputs we are pressing rather than continuous actions since we are not performing direct manipulations beyond input. Finally, the evaluation step is taken in the "OnTriggerEnter" function where the rocket in the case touches the ground and we can determine if the episode was a failure or not. If the rocket succeeds, we give a reward and end the current episode. If the rocket fails, then we punish it with a negative reward and end the episode there. In each case, the overall neural network is notified of the result during training to adjust the weight of future episodes accordingly.

3.4. Component C

User interaction

The use of simulation environments in understanding humans and A.I. interactions provides a safe and controlled platform for users to experiment and learn.[15] By immersing themselves in these simulated environments, users can gain valuable experience and insights into the differences between human and A.I. behavior.[12] Additionally, the simulation environment can provide feedback on the performance of the user, allowing them to improve their interaction strategies.





```
0 references
public override void Heuristic(in ActionBuffers action)
{
    // Debug.Log("yes");
    ActionSegment<int> discreteActions = action.DiscreteActions;
    if (Input.GetKey(KeyCode.Space))
    {
        discreteActions[0] = 1;
    }

    if (Input.GetKey(KeyCode.W))
    {
        discreteActions[1] = 1;
    }
    else if (Input.GetKey(KeyCode.S))
    {
        discreteActions[1] = 2;
    }

    if (Input.GetKey(KeyCode.D))
    {
        discreteActions[2] = 1;
    }
    else if (Input.GetKey(KeyCode.A))
    {
        discreteActions[2] = 2;
    }
}
```

The “Heuristic” function highlighted above is part of the script that determines where actions should go in the absence of any AI. The reason for this is that Unity ML agents are programmed to automatically check if they already have a neural network attached to them or if a training session is currently ongoing. If neither is true, then it will assume that the user intends to manually control the agent and will switch over to the heuristic mode. In this mode, we can directly feed the action buffers based on keypresses to have our agents react as usual without us having to implement a completely separate control scheme for user control. In this case, the client is listening for keypresses and feeds the information to the appropriate action buffer so that the agent’s “OnActionReceived” function knows how to translate the input into the expected action.

4. EXPERIMENTS

4.1. Experiment A

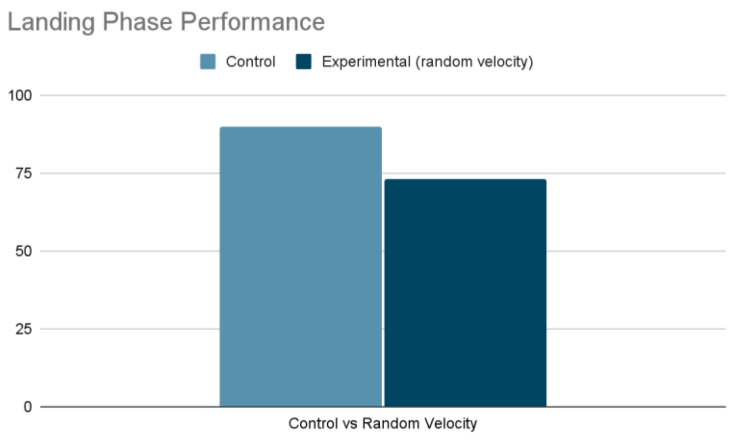
The main potential blind spot we will want to address is the overall accuracy of the rocket’s AI. The reason this is important is because the efficacy of the AI hinges on it being able to perform tasks as close to perfection as possible. While it has yet to reach that degree of performance, tracking its current abilities will shine a light on existing weaknesses.

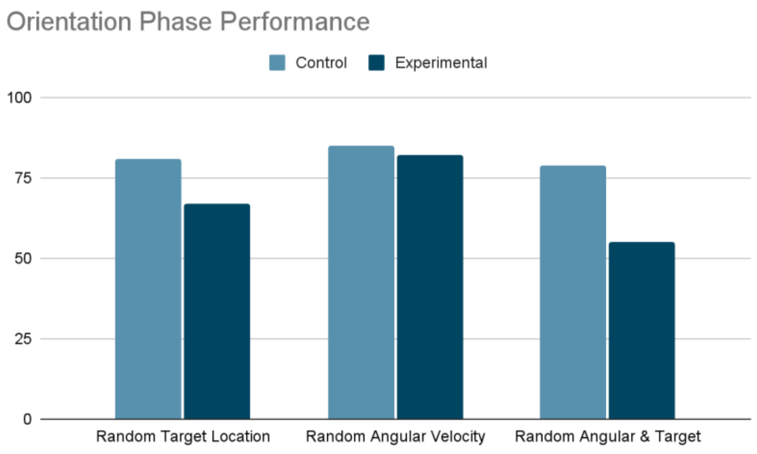
4.1.1. Design

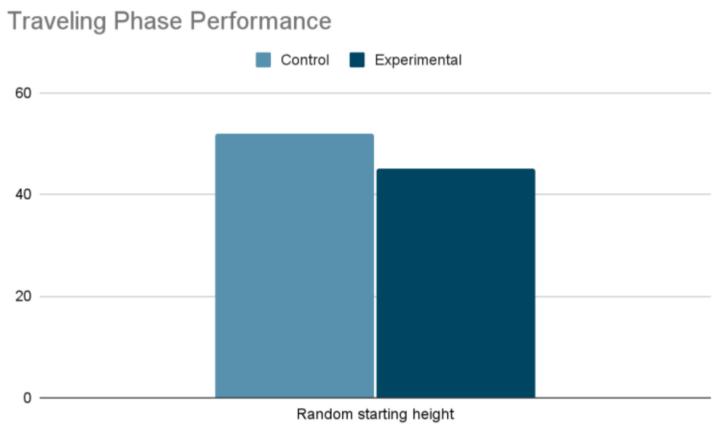
To test the AI to identify weaknesses, we will be recording 100 episodes of the AI trying to perform the task and calculating its success rate relative to the number of episodes tested. We will start with a control group that consists of the current AI iteration and the environment they were trained in. There will then be a series of experimental groups that each change one part of their respective environment to see how the AI fares. The changes made will depend on the phase being tested and are outlined like so:

Landing:
-random starting velocity
Orientation:
-random target location
-random angular velocity
Traveling:
-spawn the rocket in a position higher than the platform (random starting velocities if applicable)

4.1.2. Data and Visualization







4.1.3. Analysis

When we were testing the landing phase's performance, there was a noticeable 17% dip in the success rate when introducing random starting velocities. This can be attributed to the fact that this factor was not trained for in the creation of the model. With that being said, both agent configurations performed relatively well at 90% and 73% respectively; the reason that the random velocity variant didn't have a bigger loss in performance is that the model was trained for random starting heights, which is related enough to random starting velocity to have an impact on how the rocket reacts as it nears the platform.

When testing for the orientation phase, the performance of the experimental variants was lower than that of the control. The introduction of a random variable that it was not trained for introduced new difficulties that the agent was not expecting. The biggest drop in performance relative to the control occurred when both random targets and random angular velocity were combined. The reason for this is that in situations where the target is already near the orientation of the rocket, but has much too high of an angular velocity, the agent does not necessarily have enough time to adjust and avoid penalties. When the variables are isolated, an important note is that the random target location has a bigger impact given that the model was only trained assuming targets were above it.

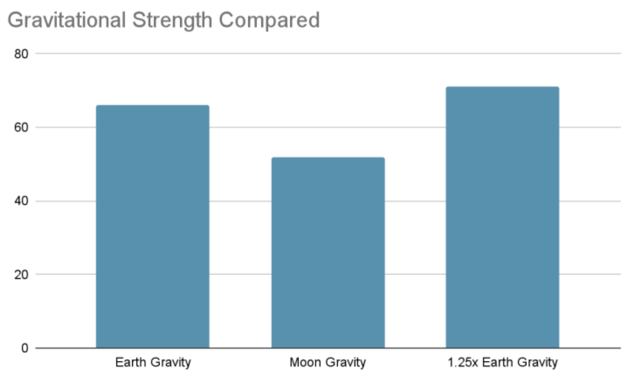
As for the traveling phase, the data exhibits the same issue as the landing phase in the experimental runs suffered compared to the control. This is because the higher starting positions of the agents in the variant gave the rockets less time for horizontal adjustments needed to correctly reach the goal. The result of this shortcoming is that rockets that start closer are prone to overshooting the goal and flying way above track as they tried to overcorrect themselves.

4.2. Experiment 2

Another more extended blind spot worth exploring is the agent's ability to travel between more than one goal. This is mainly because space travel can have a wide array of differing conditions, especially when it comes to gravity. Therefore it is important to observe how the rocket would fare under such conditions.

To effectively test the agent's ability to navigate between multiple goals, we will be testing the AI's success rate across different environment configurations to identify what layouts the rocket is good at handling and what environments are most difficult for the AI. To maintain consistency, we will be keeping the same goal positions and starting points so that they do not influence results. The environment specifications are as follows:

Environment A: Earth gravity
Environment B: moon gravity
Environment C: ”heavier than Earth” gravity (1.25x)
The rest of the environments themselves remain the same to isolate the gravity variable. In each run, the rocket starts above the same platform and is tasked to travel towards a goal that is also the same across every environment for a given run. Once they reach the goal, they must then orient themselves upward before landing safely on the platform below.



All three of the environments that we simulated had a generally decent performance considering the tasks that they were assigned, with each achieving a greater than 50% success rate. With that being said, the moon gravity scenarios performed the worst in comparison to the others. This may be in large part due to the lower gravity resulting in a shorter travel time as less force is required to reach higher velocities. This in turn means that there is less time for the rocket to make adjustments and so increases the risk of it overshooting the goal and causing a failed run. To support this notion, the rocket exhibited the highest success rate in the 1.25x Earth Gravity scenario which has the largest gravitational force needed to be overcome during our testing. With that being said, the relative success of the agents, in general, shows that it allows the user to simulate not just an environment that is similar to planet Earth but also can fit into other environments that contain a planet or celestial body that has a different mass or radius from Earth. To improve the performance in lower gravity environments, one method we could pursue down the line is to train with a proper deceleration mechanism as right now too much of the rocket's movement is impacted by the gravity of the environment it is present in. Giving finer control both for users and the AI will allow for more granular control when traveling through space.

5. METHODOLOGY

5.1. Methodology A

Ashish Gupta discusses the use of reinforcement learning to "solve the "Lunar Lander" Environment in OpenAI gym by training a Deep Q-Network(DQN) agent". [4] It includes experience replay and epsilon-greedy exploration, and specific hyperparameters were selected for model training. After 600 training episodes, the agent became fully trained and landed the lander successfully every time. The article also includes a reward values figure per experience during training, where blue lines denote the reward for each training episode and the orange line represents the rolling mean of the last 100 episodes. In terms of limitation, the most notable one is the fact that the environment is two-dimensional rather than three-dimensional, which our solution seeks to tackle.

5.2. Methodology B

Cathy Wu and her students are addressing a somewhat different problem as they are trying to improve algorithms that are meant to solve the last-mile delivery problem. The main point of overlap with our subject is the focus on reaching a final destination combined with the fact that their “solver algorithms work by breaking up the problem of delivery into smaller subproblems to solve — say, 200 subproblems for routing vehicles between 2,000 cities.” [6] This approach of breaking down the overall problem is similar to how we needed to break down the various points of a rocket’s journey into separate phases to make training easier to conduct. One particular aspect of their AI approach that is very effective is how despite “a neural network trained on the ‘medium-quality’ subproblem solutions available as the input data ‘would typically give medium-quality results… [they] were able to leverage the medium-quality solutions to achieve high-quality results’” [6] at a rate much faster than prior methods.

5.3. Methodology C

Yann Berthelot leveraged machine learning to train an AI to properly get a plane to take off. To achieve this, he took a wide variety of variables to represent thrust, lift/drag coefficients, drag, lift, and fuel consumption to more accurately portray the environment state.[7] This scenario is created and run using Python/TensorForce where agents are rewarded or punished based on how they perform in the environment in terms of objective and time. Their solution was very effective and outperformed even actual pilots within the simulation boundaries provided. This solution was achieved by tuning a variety of hyperparameters, among which was the exploration parameter, which “is the probability that a random action will be taken instead of the action decided by the agent’s policy.”[8] Their approach was very thorough and provided us with more ideas and information to further improve our approaches. There are, however, some limitations with the solution that was provided, the principal of which is the fact that there might be some variable like air resistance in the real world which may not be accounted for.[11] This is a challenge that even our project faces which introduces further complications and complexity to the project. Another limitation present is that the emulated scenario is in two dimensions, something that we go beyond in our three-dimensional environments.[13]

6. CONCLUSIONS

6.1. Limitations and Improvements

Limitations: -It does not account for every edge case

For the landing and orienting part of the project, the movement is very limited to linear movement, and rotating around the same point. This falls into the issue of scenarios that are “too ideal” to be useful for real-life scenarios as it does not account for enough environmental factors. After that, if it is in a real environment, in case of emergency, the manual mode is quite difficult for humans to take exact control of. This is mainly a limitation of the medium as a regular computer keyboard does not make a sufficient substitute for actual rocket controls.

-Not enough variance in some of the scenarios

For example, there is a lack of air resistance, Earth-accurate gravity, collision with other celestial bodies, and a lack of distance between the rocket and the goal. All of these factors lead to inaccuracies when compared to the real world, so to account for this we will need to update the scale of distance followed by more AI training.

-The distance the rockets are traveling within is quite small

The distance is short because the environment is very easy for the AI to find the goal, and it also limits the AI to discover in a longer distance and to cover a larger area. This was done under the current stage of the project to allow for visual clarity as working in a larger scale environment can make things harder to see.

To Fix: - Overall improvement in accuracy
We also just generally need to further improve the accuracy of our AI, which can be accomplished with more training periods, hyperparameter adjustments, and an increasingly randomized environment.

- Improvements to the camera system
Creates a camera that follows the rocket and creates pov for the rockets, more cameras in general to limit blind spots around the rockets. Improvements in this part of the project will also unlock our ability to use larger scale environments as the issue of visual clarity is not as much of an issue should we improve on this front.

6.2. Concluding Remarks

To further improve on the baseline project, retraining the AI in a more dynamic 3D environment with more refined processes will be necessary to get closer to a proper autonomous rocket navigation system. We will want to train for additional uncertainties such as drag, variable gravities, and the presence of other celestial bodies aside from planets.

7. Summaries

7.1. Experiment Recap

In the first experiment, the main focus was on introducing randomness similar to a real environment. When it comes to simulating real scenarios, the ability to control and react to unexpected changes in the environment becomes significant, such as crashes, external forces, or a mistake while operating. With that being said, if the rocket is trained for a certain scenario, it also creates a better general performance for the rocket in all environments, increasing its ability to do more difficult tasks in the future. Therefore the focus is not so much on if the success rate of the experimental groups is higher than their controls, but rather to what degree the agents can minimize the loss in performance.[9] The experiment was run with 100 episodes per configuration to determine respective success rates. For example, in the orientation phase tests, there was a bigger dip when introducing a random target location as opposed to a random starting velocity, which exposes the need to train the agents with more varied target locations to patch the weakness.

For the second experiment, the main focus was on the agent's ability to navigate between multiple goals in a set course, with one variable being the strength of gravity in the given scenario. The environment's gravities are separated as follows: Earth gravity, moon gravity, and "heavier than Earth" gravity. The rocket starts from the same position and is tasked with each goal before finally landing on a platform. While each environment yielded a success rate greater than 50%, the moon gravity scenario performed noticeably worse due to shorter travel time allowing for fewer adjustments. Meanwhile, the opposite was true in the case of the "1.25" variant. Overall, the experiment demonstrates that the agent can simulate different environments, but suggests improvement through the implementation of a deceleration mechanism for finer control.

7.2. Methodology Comparison

When exploring the different solutions and implementations created by others to solve the challenge of AI piloting vehicles, the methodologies we explored accomplished various important aspects.[1] Ashish Gupta's implementation of the lunar lander's landing sequence took the general workflow for machine learning and created an agent that was able to perform a successful landing given the bounds of the provided environment. Ashish was able to achieve nearly perfect results even when injecting chaotic starting conditions as the lander had different rotations and velocities at the start of every episode.[4] That being said, the main shortcoming of Ashish's implementation is that it is working under a two-dimensional environment, which we are seeking to improve upon by working under a three-dimensional environment.

The next methodology we explored was Cathy Wu's investigation and work on improving algorithm performance for solving the last-mile delivery problem.[6] Cathy's method of breaking down the overall problem into a series of easier-to-solve subproblems allowed for the solver algorithms to not only find results faster but to also generate high-quality results from medium-quality inputs. While the results are very promising, the main limitation is that there wasn't a tangible product that we could directly compare within the context of rocketry, but we were nonetheless able to benefit from the lessons provided.

The final methodology we explored was Yann Berthelot's implementation of an AI that can successfully pilot a plane off of the runway.[8] In much the same way as Ashish, a specific environment was created to facilitate testing, but does so with a lot of attention given to relevant factors in physics that may affect results. He also optimized a variety of hyperparameters such as the exploration parameter to reach the eventual result.[4] Nevertheless, there still was the limitation of two dimensions and an absence of other environmental factors such as weather and drag. Our project seeks to better match adversarial conditions by simulating our scenarios in a three-dimensional environment.

REFERENCES

[1] Örs, A. O. (2020, December 3). The Role of Machine Learning in Autonomous Vehicles. Electronic Design. https://www.electronicdesign.com/markets/automotive/article/21147200/nxp-semiconductors-the-role-of-machine-learning-in-autonomous-vehicles

[2] Harwood, W. (2023, April 20). SpaceX Starship rocket launch ends in midair explosion minutes after liftoff. CBS News. Retrieved from https://www.cbsnews.com/texas/news/spacex-starship-launch-explosion-video/

[3] Moynihan, P., & Ustinov, E. (2022). Bending the cost curve. Aerospace America. Retrieved from https://aerospaceamerica.aiaa.org/features/bending-the-cost-curve/
[4] Ashish, G. (2020, August 28). AI learning to land a rocket: Reinforcement learning. Towards Data Science. https://towardsdatascience.com/ai-learning-to-land-a-rocket-reinforcement-learning-84d61f97d055
[5] Günther Waxenegger-Wilfing, Kai Dresia, Jan Deeken, Michael Oschwald (2021). Machine Learning Methods for the Design and Operation of Liquid Rocket Engines -- Research Activities at the DLR Institute of Space Propulsion. arXiv. Advance online publication. https://arxiv.org/abs/2102.07109
[6] Ham, B. (2021, December 10). Machine learning speeds up vehicle routing. MIT News. https://news.mit.edu/2021/machine-learning-speeds-vehicle-routing-1210
[7] Berthelot, Y. (2020, April 26). How I taught a plane to fly using RL. Towards Data Science. https://towardsdatascience.com/how-i-taught-a-plane-to-fly-using-rl-c170a152b771
[8] Berthelot, Y. (2020, August 25). AI learns to fly: Part 2 — Create your custom RL environment and train an agent. Towards Data Science. https://medium.com/towards-data-science/ai-learns-to-fly-part-2-create-your-custom-rl-environment-and-train-an-agent-b56bbd334c76
[9] Kuutti, S., Fallah, S., Barber, P., Jin, Y., & Bowden, R. (2019, December 23). A Survey of Deep Learning Applications to Autonomous Vehicle Control. arXiv. https://arxiv.org/pdf/1912.10773.pdf
[10] Author links open overlay panelAndreas Theissler a, a, b, c, d, Highlights•Machine learning subfields overview relevant for automotive predictive maintenance•The aim is to make the field accessible to maintenance or machine learning experts•Machine learning-enabled predictive maintenance for automotive systems paper s, & AbstractRecent developments in maintenance modeling fuelled by data-based approaches such as machine learning (ML). (2021, June 24). Predictive maintenance enabled by machine learning: Use cases and challenges in the automotive industry. Reliability Engineering & System Safety. https://www.sciencedirect.com/science/article/pii/S0951832021003835
[11] Schwabacher , M. (n.d.). Machine Learning for Rocket Propulsion Health Monitoring - ResearchGate. https://www.researchgate.net/publication/246740095_Machine_Learning_for_Rocket_Propulsion_Health_Monitoring
[12] Shao, K., Tang, Z., Zhu, Y., Li, N., & Zhao, D. (2019, December 26). A survey of deep reinforcement learning in video games - arxiv.org. arXiv. https://arxiv.org/pdf/1912.10944.pdf
[13] Salameh, H., Wu, M., & Ulrich, C. M. (2020, May 15). (PDF) training a game AI with machine learning - researchgate. https://www.researchgate.net/publication/341655155_Training_a_Game_AI_with_Machine_Learning
[14] Reid, E. (2023, May 10). Supercharging search with Generative AI. Google. https://blog.google/products/search/generative-ai-search/
[15] Zheng, Y. (2019, September 10). Reinforcement learning and video games. arXiv.org. https://arxiv.org/abs/1909.04751