Reinforcement Learning for Spacecraft Orbits

Christopher Figueroa, Sai Shreya Jillella
Damini Singh Thakur, Aishwarya Uyyala

***Abstract*—** This project introduces an innovative 2D Python orbital mechanics game powered by a simple and efficient reinforcement learning algorithm. Our program showcases a spacecraft's autonomous mastery of orbit dynamics around procedurally generated planets. The state and reward system, based on crucial parameters like gravity, speed, apoapsis, periapsis, and eccentricity, make up the learning process, with rewards maximizing upon successful orbit achievement. A combination of fuzzy logic and neural networks dictate the steps our 2D spacecraft requires to reach orbit around any planet.

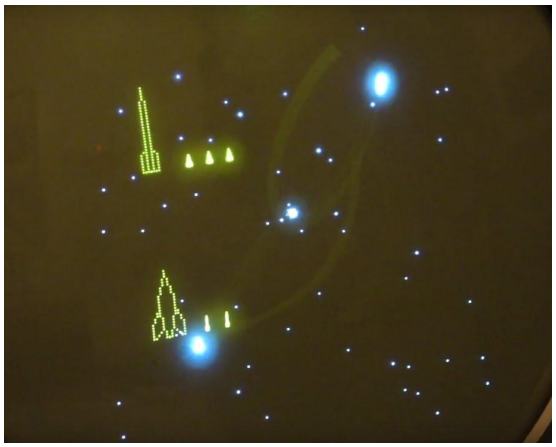
***Index Terms*—** HAL/S, Spacewar, Reinforcement Learning, Apoapsis, Periapsis, Eccentricity, Pygame, Torch, Anaconda, Dynamic Regressor Network, Mean Absolute Error (MAE), Sigmoid, Fuzzy Logic, Fuzzy Rules, Loss, Hidden Layers, Torch Net, Semi Major Axis, Specific Orbital Energy, Crisp Values, Non-Dynamic Regressor Network, Godot

Index Terms:

I. Introduction

In May 5th 1961, the United States launched its first manned rocket through the conception of NASA's Project Mercury. The Freedom7 spacecraft, the first American rocket to successfully orbit the Earth, utilized a flight computer to pilot the vessel into a stable orbit. This early flight computer utilized the high-level language HAL/S to apply a series of hard-coded instructions to the rocket. Every variable and function in the program had to be meticulously tested before the launch. Even one typo or miscalculation of a physics variable would have resulted in the death of the astronauts on board the spacecraft [1].

Spacewar Game [2]:



Only one year after the launch of the first American rocket, MIT developed Spacewar, the first space flight video game in 1962. This 2D physics simulation game featured no planets, but instead two opposing spacecraft which had the ability to fire at one another. Unlike many future 2D space games, Spacewar featured the use of Newtonian physics to simulate the

momentum of the space crafts. To change direction, one had to fire the rocket engines in the direction in which one wanted to travel. The spacecraft's momentum could only be altered if one continuously throttled in the desired direction [2].

II. Motivation

Our motivation in developing this game arises from a desire to provide an efficient and accessible platform for a video game integrated machine-learning learning program. By employing a simplified and game-optimized physics engine, we have successfully crafted a program that stands apart from others in terms of computational efficiency. The choice to utilize a small number of relevant parameters. Each of the randomized planets are characterized by their own unique radius, mass, gravity, and orbital velocity. Our use of 4 parameters and procedurally generated planets distinguishes our approach [5].

Physics Constants:

- G: gravitational constant
- G-value: $6.67430 * 10^{-11}$

The 4 Planet Parameters [5]:

- R: planet radius
- M: planet mass
- Gravity: $G * (M / D^2)$
- Orbital velocity: $\sqrt{((M*G) / R)}$

In this program, we leverage the power of reinforcement learning to enable a spacecraft to autonomously learn the correct sequence of steps needed to orbit a planet. The challenge lies in the fact that each planet is characterized by a randomized radius and mass, adding an element of unpredictability to the gravitational interactions. The spacecraft, equipped with the ability to adapt and learn, navigates the complexities of orbit by employing five fundamental physics equations.

The equations include gravity, described by Newton's law, governs the force between the spacecraft and the planet, while the orbital velocity equation determines the necessary speed for a stable orbit. Additionally, the apoapsis and periapsis equations identify the highest and lowest points in the orbit, respectively, contributing to the spacecraft's trajectory optimization. The program further incorporates the eccentricity equation, quantifying the ellipticity of the orbit. Together, these physics principles form the foundation for an autonomous spacecraft to master the art of orbiting any randomly generated planet, showcasing the potential of reinforcement learning in celestial navigation [6].

Physics Constants:

- G: gravitational constant
- D: distance from planet

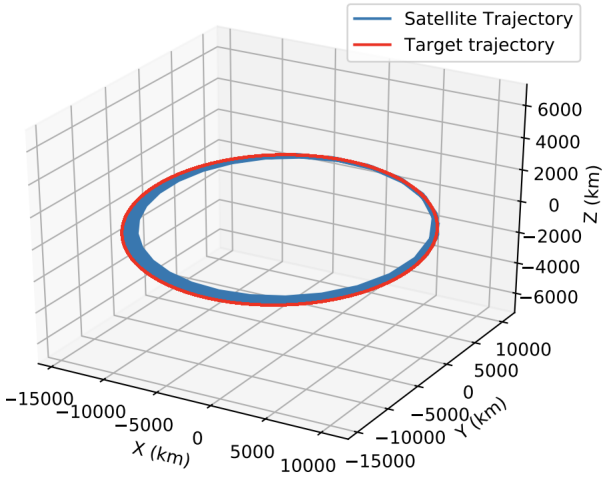
The 5 Spacecraft Parameters [6]:

- Gravity: $G * (M / D^2)$
- Velocity: rocket movement in meters/sec
- (A) Apoapsis: highest point in orbit
- (P) Periapsis: lowest point in orbit
- Eccentricity: (A-P) / (A+P)

III. Related Research

In our quest to understand and integrate the latest advancements in reinforcement learning, we delved into research conducted by Western Michigan University and Carleton University. At Western Michigan University, the focus lies in the exploration of highly complex and intricate physics calculations within the realm of reinforcement learning.

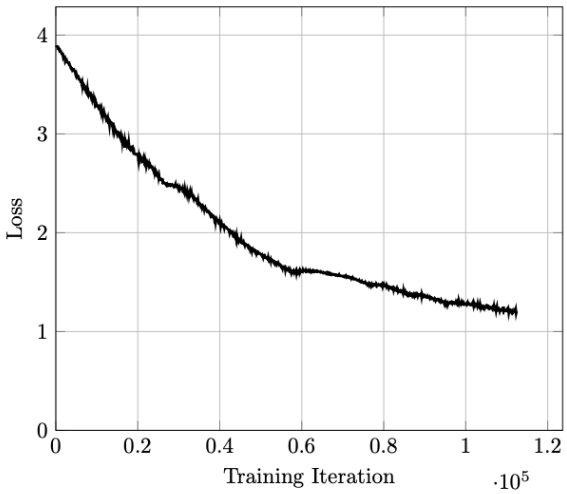
Western Michigan University Earth Orbit Diagram [3]:



Their research, set on a real-life scale of an orbit around Earth, incorporates a plethora of variables to accurately model the dynamics of celestial bodies. The employment of advanced learning models, including Markov decision processes, Q-Learning, policy gradient methods, reward functions, and experience replay, underscores the comprehensive nature of their approach. The wealth of variables and the intricacy of the models contribute to a high-fidelity representation of orbital dynamics but may introduce computational challenges for applications seeking efficiency and simplicity, such as our game [3].

On the other hand, research from Carleton University adopts a more streamlined approach compared to the extensive variable set used by Western Michigan University. While still utilizing dozens of variables, the research at Carleton University demonstrates a middle ground, avoiding unnecessary complexity. The learning models employed, including the D4PG algorithm, deep guidance, ideal controller, dynamics, and delay functions, showcase a commitment to realistic physics calculations without overwhelming the system with an excessive number of parameters. This nuanced balance offers insights and inspirations for our project, guiding us to strike an optimal equilibrium between computational efficiency and realistic physics simulations in the context of our randomized planetary orbits [4].

Carleton University Training Iteration Loss Graph [4]:



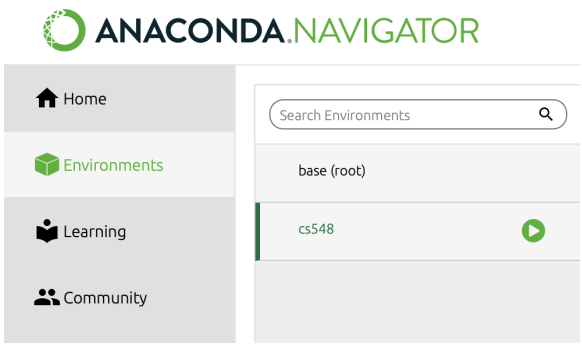
(b) Loss

Unlike many existing training algorithms burdened with numerous complex parameters that can impede performance, our program streamlines the learning process by focusing on key elements crucial for mastering orbital mechanics. This deliberate simplicity ensures a smoother and less computationally intense experience, making our game an ideal tool for both education and entertainment. Through these considerations, we aim to foster a user-friendly environment that encourages exploration and understanding of orbital mechanics, in contrast to more intricate research-oriented models, such as those conducted by Western Michigan University.

IV. Technical Code Overview

Our 2D Python orbital mechanics game combines various libraries to create a dynamic and engaging simulation. The program leverages the Pygame library for graphics rendering, the Math and Numpy libraries for numerical computations, and the Torch library for reinforcement learning, the program operates within the Anaconda environment and employs Jupyter notebooks for interactive development and analysis.

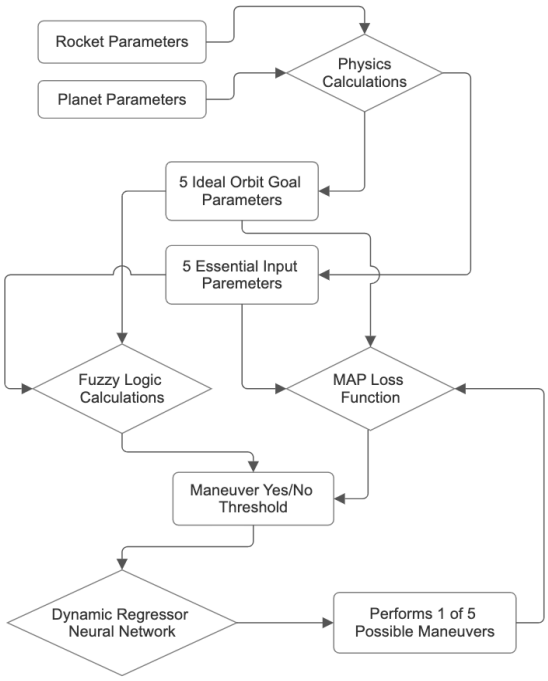
Anaconda Navigator Mac OS:



The heart of the reinforcement learning aspect lies in the integration of the dynamic regressor network, a neural network architecture allowing for adaptable network sizes. The learning process is guided by the mean absolute error (MAE) loss function from the Torch library, emphasizing the precision of predictions. The sigmoid activation function is strategically employed to confine the action variable to a range of 0 to 5, aligning with the five possible spacecraft moves.

The state and reward system plays a pivotal role in shaping the learning process. The state is characterized by key parameters, including current gravity, speed, apoapsis, periapsis, and eccentricity, encapsulating the crucial elements for determining the spacecraft's position and motion. The reward system is intricately tied to the proximity of the current state to the ideal orbital conditions.

Functions Diagram:



When the spacecraft successfully achieves orbit, the reward is maximized, providing positive reinforcement for desirable behavior. The game offers five possible moves—Rotate right, Rotate left, Throttle up, Throttle down, and Apply throttle—each influencing the spacecraft's trajectory, and the reward system dynamically responds to these actions, driving the learning model towards optimal orbital maneuvers. This overall setup ensures a reliable environment for the reinforcement learning algorithm to evolve and master the complexities of orbital mechanics in the 2D space simulation.

V. Physics Parameters

The physics equations underlying the 2D orbital mechanics game provide a foundation for calculating key parameters influencing the spacecraft's motion and orbit around a planet. These parameters also play a role in accurately portraying the forces

planets of varying mass and radii would have on the rocket.

Gravity Equation:

$$F_g = \left(G \times (\frac{M}{D^2}) \right)$$

The gravity equation represents the acceleration in meters per second squared (m/s^2) a spacecraft experiences at a given distance from a planet. The variable D represents the distance from the planet's center. As the spacecraft's altitude changes, the gravitational force dynamically changes, affecting the force the planet exerts on the rocket.

Specific Orbital Energy:

$$E = \left(\left(\frac{v^2}{2} \right) - \left(\frac{G \times M}{r} \right) \right)$$

The specific orbital energy is the difference between the sum of the kinetic energy and the potential energy of a given object in (MJ/kg). The value of this parameter is used to determine the semi major axis. The variable V represents the current velocity of the spacecraft. G defines the gravitational constant, M is the planet's mass, and R is the radius of the planet [8].

Semi Major Axis:

$$S = \left(\frac{-(G \times M)}{(2 \times E)} \right)$$

The semi major axis represents the average overall distance a spacecraft is from a planet while in orbit. The calculation of this parameter is necessary for the determination of the apoapsis and periapsis when the eccentricity is unknown. The G represents the gravitational constant, M defines the mass of the planet, and R is the planet's radius [7].

Apoapsis Equation:

$$Ra = S \times \sqrt{\left(1 - \left(\frac{(E \times S)}{(G \times M)} \right) \right)}$$

Apoapsis represents the highest point in the spacecraft's orbit. It is the point farthest from the planet. When the eccentricity is known the equation 'Ra = a(1 + e)' can be used to find the apoapsis. However when the eccentricity is not known, the semi major axis (S) and the specific orbital energy (E) must be applied to the formula above to obtain the apoapsis.

Periapsis Equation:

$$Rp = S \times \sqrt{\left(1 + \left(\frac{(E \times S)}{(G \times M)} \right) \right)}$$

Periapsis signifies the lowest point in the spacecraft's orbit, which is the closest point to the planet. Like Apoapsis, the periapsis can be obtained using a variant of the apoapsis equation that utilizes the specific orbital energy (E) and the semi major axis (S). When the eccentricity is known, the formula Rp = a(1 - e)' is applicable [8].

Eccentricity Equation:

$$e = \left(\frac{\left(R_a - R_p\right)}{\left(R_a + R_p\right)} \right)$$

Eccentricity measures the deviation of the spacecraft's orbit from a perfect circle. It is calculated as the absolute difference between Apoapsis (Ra) and Periapsis (Rp) divided by their sum. The closer the value is to zero, the closer the eccentricity is to a perfectly circular orbit.

The ideal values for these physics parameters represent conditions when the spacecraft is in a stable circular orbit. The parameters of the current state of the spacecraft correspond to the values below when the rocket has reached the ideal conditions. These ideal values serve as benchmarks for the spacecraft's state, providing a reference for the fuzzy logic system to evaluate and optimize the spacecraft's actions to achieve and maintain a stable orbit [5].

The ideal state conditions:

- Gravity: The force of gravity that the planet exerts on the spacecraft should correspond to the pull at the ideal orbital altitude. In the case of this game, the ideal orbit distance from the planet's core is equal to the planet's radius + 50 meters.
- Velocity: The velocity at orbit is determined by the square root of the expression $(M \cdot G) / R+50$. The radius + 50 value corresponds to the ideal orbit distance.
- Apoapsis and Periapsis: The apoapsis and periapsis should be approximately equal, indicating a balanced and symmetric orbit.
- Eccentricity: The eccentricity should be close to 0, signifying a nearly circular orbit.

VI. Fuzzy Logic Crisp Values

Before the reinforcement learning algorithm can determine what steps the spacecraft needs to achieve a stable orbit, the program first needs a system to compare the ideal state values with the current state values. The current conditions of the spacecraft must be evaluated using a logic system that categorizes each of the 5 spacecraft physics parameters, and evaluates how close they are to the ideal state conditions.

In our current project, the fuzzy logic system serves our state evaluation mechanism. Within this 2D orbital mechanics game, crisp values, which represent the range of possible values for each parameter, serve as pivotal indicators for various physics parameters influencing the spacecraft's motion. The five parameters—gravity, velocity, apoapsis, periapsis, and eccentricity—are each categorized into Low, Medium, and High ranges, defining the possible states of these variables. These crisp values are then strategically assigned to rules within the fuzzy logic system to determine whether the spacecraft should perform a maneuver or not.

5 Crisp Parameter Ranges:

- Gravity: 0 to 1.5 (surface g factor)
- Velocity: 0 to 2.0 (orbital v factor)
- Apoapsis: 0 to 300 (meters)
- Periapsis: 0 to 300 (meters)
- Eccentricity: 0.0 to 1.0

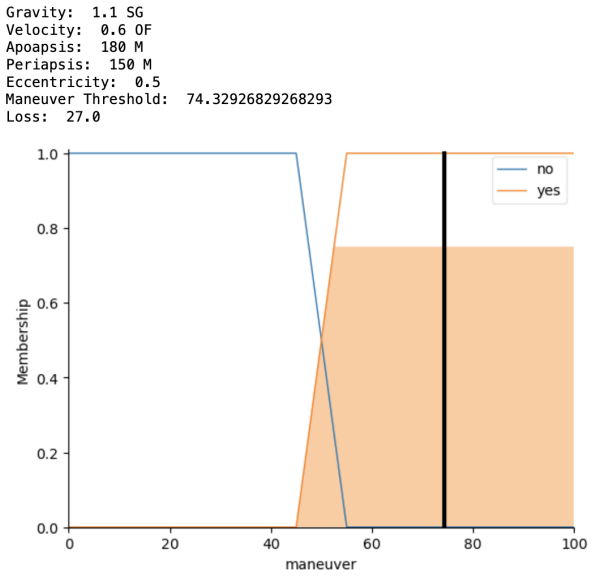
3 Crisp Categories:

- Low: lower end of ranges
- Medium: ranges within orbit
- High: higher end of ranges

The fuzzy logic system operates based on two fundamental rules, each tailored to handle specific scenarios. The first rule associates Low and High crisp values with upper and lower extremes, indicating situations where the spacecraft's parameters are far from the ideal orbit conditions. Conversely, Medium crisp values signify that the spacecraft has reached a state closely resembling orbit, serving as a threshold indicating no immediate need for maneuvers.

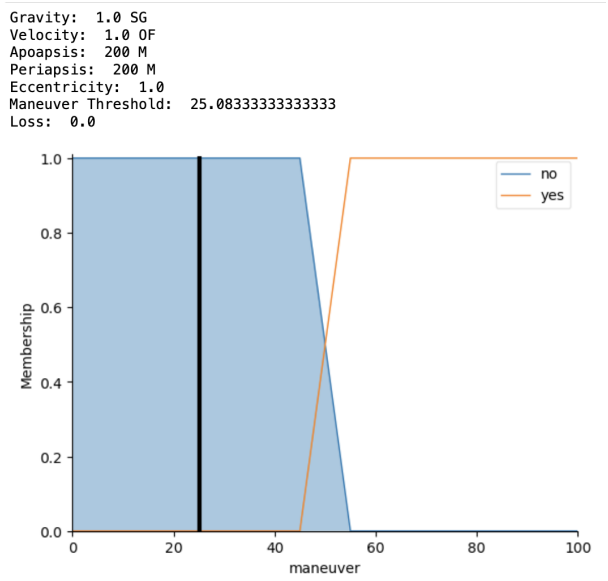
The maneuver threshold score becomes a critical parameter in deciding the spacecraft's actions. This threshold is derived from the fuzzy logic output graph, where a score between 0 and 100 is assigned based on the crisp values of the parameters. A lower score suggests that the spacecraft is closer to the ideal state corresponding to an orbit, while a higher score indicates the need for maneuvers to adjust its trajectory.

Maneuver Needed Graph:



For instance, in the case of the upper graph displaying the fuzzy logic output indicating that the spacecraft needs to maneuver, the loss is measured at 27, and the maneuver threshold score is determined to be 74. Since the maneuver threshold score exceeds 50, the spacecraft is prompted to perform maneuvers to get closer to the ideal orbit conditions.

No Maneuver Needed Graph:



On the other hand, the lower graph, depicting a spacecraft that has already reached orbit, showcases a loss of 0 and a maneuver threshold score of 25. In this scenario, the score falling below 50 signifies that the spacecraft no longer needs to maneuver, as it is in a state close to the ideal orbit conditions. The lower the loss, the closer the spacecraft is to achieving the desired orbit, emphasizing the effectiveness of the fuzzy logic system in guiding the spacecraft's actions based on crisp parameter values [9].

VII. Neural Network

After the fuzzy logic system evaluates the spacecraft's current state, the mean absolute error (MAE) serves as the next essential step in the reinforcement learning process. The MAE function plays a critical role in assessing the performance of the Dynamic Regressor Neural Network within the 2D

orbital mechanics game. This function takes as input a list of current state values and their corresponding ideal values, effectively quantifying the absolute error between the two sets of values.

MAE Function:

```
# initializes Mean Absolute Percentage Error loss function
def get_mape(invals, ideals):
    diff_grav = (abs(ideals[0]-invals[0]) / ideals[0]) * 100
    diff_vel = (abs(ideals[1]-invals[1]) / ideals[1]) * 100
    diff_ap = (abs(ideals[2]-invals[2]) / ideals[2]) * 100
    diff_per = (abs(ideals[3]-invals[3]) / ideals[3]) * 100
    diff_ecc = (abs(ideals[4]-invals[4]) / ideals[4]) * 100
    mape = (diff_grav+diff_vel+diff_ap+diff_per+diff_ecc) / 5
    return mape
```

The resulting error values are then used to determine the loss of the current state, providing a measure of how closely the spacecraft aligns with the ideal orbit conditions. In essence, the MAE function serves as a crucial metric to gauge the spacecraft's proximity to achieving the desired orbital state, with a higher loss indicating a greater deviation from the ideal values [10].

Dynamic Regressor Network:

```
# initializes the Dymanic regressor net which allows for flexibility of size
class DynamicRegressorNet(torch.nn.Module):
    def __init__(self, n_hidden, n_output):
        super(DynamicRegressorNet, self).__init__()
        self.fc1 = torch.nn.Linear(in_features=5, out_features=n_hidden)
        self.fc2 = torch.nn.Linear(in_features=n_hidden, out_features=n_output)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

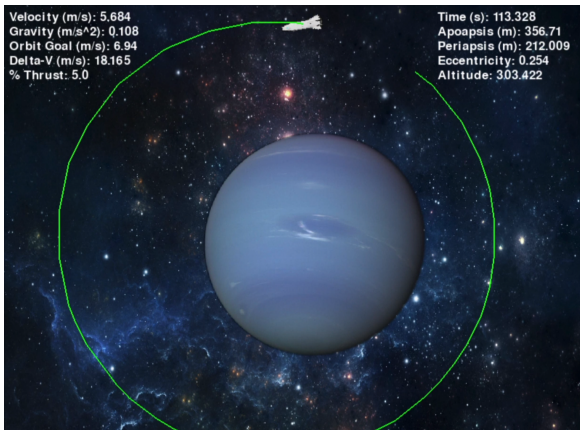
The dynamic regressor neural network further enhances the adaptability and flexibility of the learning model. This neural network architecture allows for the use of networks with varying sizes and shapes, providing a dynamic framework that can accommodate different complexities of orbital mechanics. In contrast to a non-dynamic regressor, which yielded incorrect shape errors during implementation, the dynamic regressor proves to be a more robust solution. The challenge of determining the correct shape and size of the network is mitigated by the dynamic regressor.

The features integrated into the neural network include five input nodes, each corresponding to one of the physics parameters—gravity, speed, apoapsis, periapsis, and eccentricity. The network encompasses five hidden layers, contributing to the model's capacity to capture intricate relationships within the data. The output layer comprises a single node used to predict the action variable, guiding the spacecraft's maneuvers. This architecture, combined with the MAE loss function, ensures that the neural network effectively learns and adapts to the complexities of orbital mechanics, optimizing the spacecraft's actions based on the current state and ideal values [11].

IIX. Reinforcement Learning

The reinforcement learning loop within the 2D orbital mechanics game involves a dynamic interplay between the current state of the spacecraft, the Torch Net function, and the Mean Absolute Error (MAE) loss calculation, orchestrating a continuous cycle of learning and adaptation. At the core of this loop is the determination of actions, which are pivotal for guiding the spacecraft's maneuvers in response to the dynamically changing orbital environment.

Spacecraft in Stable Orbit:



The current state of the spacecraft, characterized by the five physics parameters—gravity, speed, apoapsis, periapsis, and eccentricity—serves as input to the Torch Net function. This function, powered by the Dynamic Regressor Neural Network, predicts the action for the spacecraft based on the learned relationships between the input parameters and optimal maneuvers. The Sigmoid activation function is strategically employed to confine the action output to a float within the range of 0 to 1. This result is then multiplied by 5, expanding the range to 0 to 5. The resulting action number effectively determines the specific maneuver to be executed, whether it involves rotation or acceleration of the spacecraft.

5 Possible Spacecraft Actions:

- Rotate right
- Rotate left
- Throttle up
- Throttle down
- Apply throttle

The reinforcement learning loop hinges on the calculation of the reward, a pivotal feedback mechanism that influences the learning process. The reward is computed using the MAE function, which quantifies the loss between the current state values and the ideal values. The higher the loss, the greater the deviation from the optimal orbit conditions, consequently influencing the learner's decisions.

The loss, acting as a critical signal, determines the learner's next move, guiding the adjustment of the action value to optimize the spacecraft's trajectory. This continuous loop of action, reward, and learning ensures that the spacecraft progressively refines its maneuvers, adapting to the complexities of orbital mechanics and inching closer to the ideal state of orbit.

IX. Current Progress

The current state of our 2D orbital mechanics game showcases considerable success in several key components, with the Fuzzy Logic and State comparison functions functioning flawlessly and consistently producing the expected results. The Mean Absolute Error functions and the Dynamic Regressor Network, vital components of the reinforcement learning process, also demonstrate reliability, contributing to accurate loss calculations and appropriately scaled learner steps. The integration of physics calculations has proven effective for the most part, with the majority of parameters being accurately computed.

However, it's noted that the periapsis value tends to fall below the anticipated threshold, suggesting a minor inconsistency in the calculation of this specific parameter. While this issue may warrant further investigation, the overall performance of the physics calculations remains robust.

Action Stuck Error Print:

```
Action:  3.9144935607910156
throttle up
Action:  3.939772605895996
throttle up
Action:  3.96461820602417
throttle up
Action:  3.9890313148498535
apply throttle
Action:  4.013011932373047
apply throttle
Action:  4.036561012268066
apply throttle
Action:  4.059679985046387
apply throttle
Action:  4.082370281219482
apply throttle
Action:  4.104633808135986
apply throttle
Action:  4.1264729499816895
apply throttle
Action:  4.147890090942383
apply throttle
```

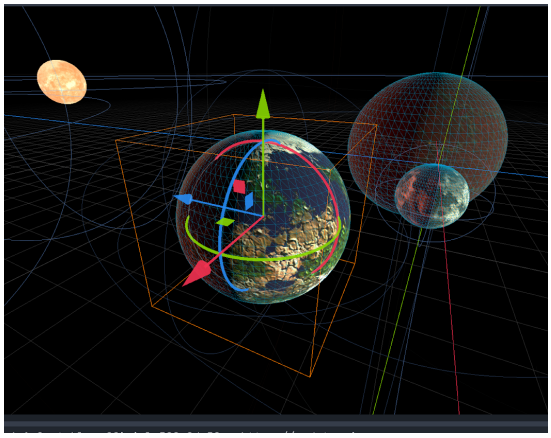
The most notable challenge lies within the Reinforcement Learning progress, where occasional hitches in action selection are observed. Specifically, the program encounters occasional instances where the

action selection gets stuck in the 5th action ‘apply throttle’, indicating a potential bottleneck or undesired behavior in the learning process. Further analysis and debugging efforts may be required to pinpoint the root cause of this issue and ensure a more seamless and dynamic progression in action selection. Despite this challenge, the foundational components of the program are solid, providing a strong basis for further refinement and optimization to enhance the overall performance of the 2D orbital mechanics game.

X. Future Work

In our ongoing pursuit of advancing spacecraft training simulations, we are excited to announce our ambitious plans to expand our project into a three-dimensional (3D) environment. This evolution aims to elevate the realism and complexity of our simulations by introducing a z-axis to capture the dynamic interactions of spacecraft within a 3D space. The inclusion of a third dimension will allow for the development of a more versatile program, providing 3D game users with a reliable means to train rockets to navigate in real-scale planetary environments.

Godot 3D Solar System:



To bring this vision to fruition, we have strategically chosen the Godot game engine as the foundation for our 3D spacecraft training simulation. Godot's versatility, user-friendly interface, and robust capabilities make it an ideal platform for developing complex simulations. Leveraging Godot's features, we aim to seamlessly integrate additional physics parameters, accommodating the nuances of 3D space, while maintaining the adaptability and efficiency that characterize our existing 2D project.

To facilitate the expansion into the 3D realm, we plan to incorporate key libraries, including AI Godot Gym, Godot-Python, and GD Extension. These libraries will enhance our project's capabilities, enabling artificial intelligence training, seamless Python integration, and extended functionality within the Godot game engine. By harnessing the power of these libraries and the capabilities of Godot, we anticipate that our 3D spacecraft training simulation will not only refine the learning experience but also open new possibilities for training and research in the field of orbital mechanics within the context of real-scale planets and a 3D environment [12].

XI. Conclusion

In conclusion, our research and development efforts in the realm of 2D orbital mechanics have yielded a robust and efficient program tailored for spacecraft learning in video games. The adaptability of our reinforcement learning models allows the spacecraft to seamlessly navigate around procedurally generated planets with varying sizes and mass. The random parameters of these planets, including radius, mass, gravity, and orbital velocity, are effectively handled by the spacecraft, requiring only five essential physics calculations.

Our choice of reinforcement learning models, incorporating fuzzy logic, mean absolute error (MAE) loss, and the dynamic regressor network (DRN), empowers the spacecraft to adapt and learn in a computationally efficient manner. Unlike more complex spacecraft learning models utilized by research institutions such as Western Michigan University and Carleton University, our program excels in efficiency while maintaining efficacy. The focused set of physics calculations and the streamlined use of reinforcement learning models make our program particularly well-suited for 2D video games, providing a powerful tool for developers seeking to implement realistic and adaptable orbital mechanics in their gaming environments.

In essence, our research and development efforts signify a step forward in creating accessible and efficient solutions for spacecraft learning in 2D video games. The adaptability to varied planetary parameters, coupled with the computational efficiency of our chosen models, positions our program as a valuable asset for developers aiming to incorporate realistic orbital dynamics seamlessly into their gaming experiences.

References

[1] J. Raney. et al, “Computers in Spaceflight”, *NasaHistory.gov*, p.5-8, 1984.

[2] N. Wardrip-Fruin, “Spacewar, the First Video Game”, *Analogue.co*, p.1-4, 2023.

[3] D. Kolosa, “A Reinforcement Learning Approach to Spacecraft Trajectory Optimization”, *Western Michigan University*, p.19-36, 2019.

[4] K. Hovell and Steve Ulrich, “On Deep Reinforcement Learning for Spacecraft Guidance”, *Carleton University*, p.4-14, 2020.

[5] A. Kitsantonis, “Chapter 1 Orbital Mechanics”, *Oer.pressbooks.pub*, p.4-19, 2021.

[6] O. Stax, “Kepler’s Laws of Planetary Motion”, *Courses.lumen.learning*, p.1-5, 2022.

[7] J. Duetsch, “Energy of Orbits”, *Physics.ucsc.edu*, p.1-2, 2003.

[8] A. Vanhamme, “Orbital Mechanics, Orbit Equation and Kepler I”, *Faculty.fui.edu*, p.3-9, 2018.

[9] S. Priy and A. Rajput, “Fuzzy Logic Introduction”, *Geeksforgeeks.org*, p.2-5, 2023.

[10]: S. Kotz. et al, “Absolute Error & Mean Absolute Error (MAE)”, *Statisticshowto.com*, p.1-4, 2023.

[11]: R. Taylor, “Dynamic Linear Regression Models”, *Pyflux.readthedocs.io*, p.2-6, 2016.

[12] M. Ivan, “Godot RL Agents”, *Github.com*, p.2-4, 2023.