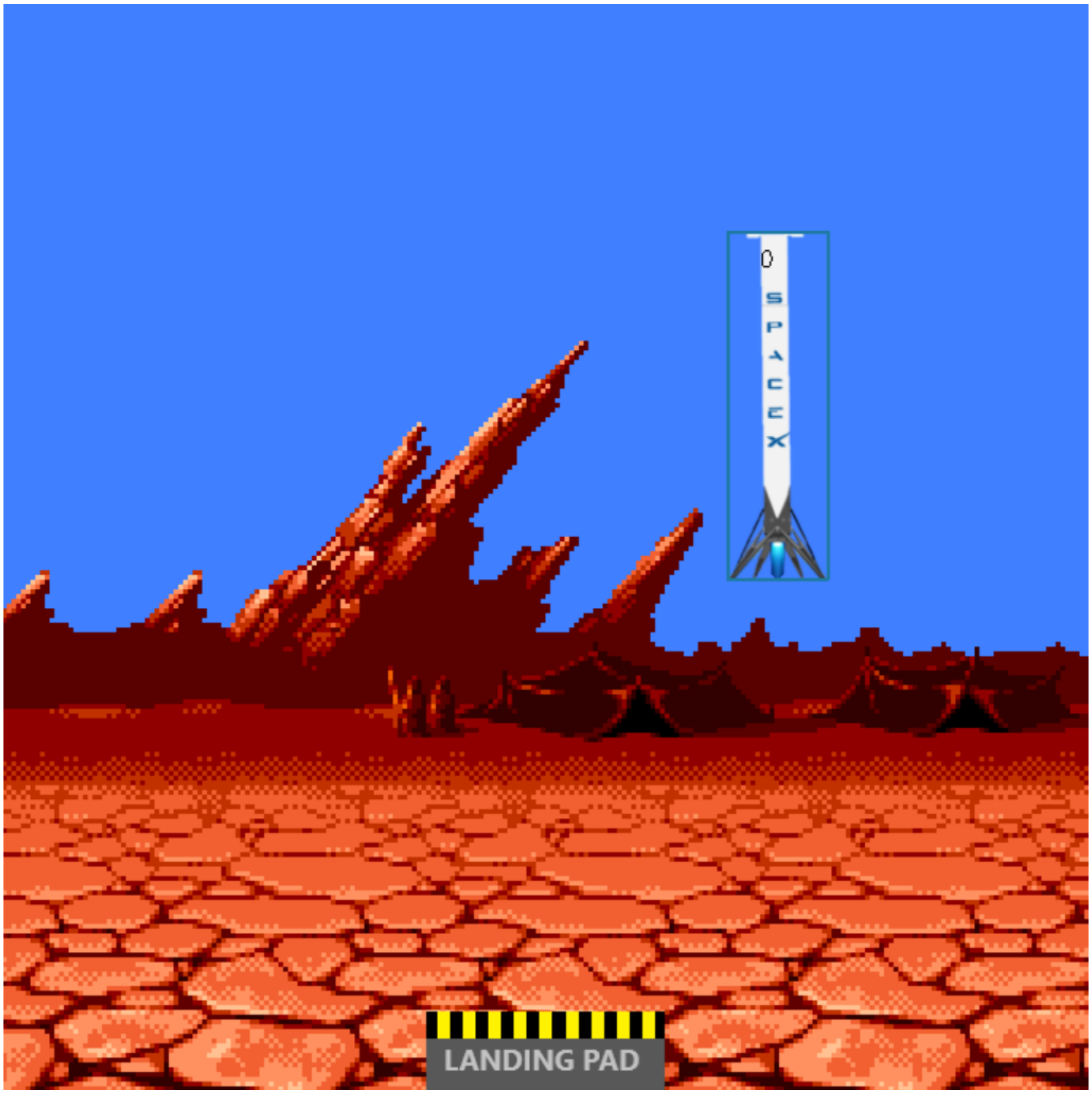
Rocket Booster Agent



Attempting to land Elon's SpaceX rocket booster on Mars

Harth Majeed
3547 Intelligent Agents & Reinforcement Learning

Contents

Motivation

The inspiration for this project started when I saw a YouTube video about an ant simulator. Little ants would leave their nest looking for food, and as they walked they would draw a path. Once they find food they will walk that path back to their nest.

My interest peaked, how could such a program be written? How does it work? What is this field called? I then discovered another video where a set of 3 limbs learned to jump over a ball. Then soon enough a video about a drone learning to fly to a target point. This fascinated me and decided to make a version of it as my project.

I decided to build an AI that could land a rocket, much like SpaceX's rocket boosters that can land on their own. The twist here is that it will be on Mars rather than earth. Python will be used for this project.

Objectives

The objectives for this project will be:

- Build an animated rocket booster that can land with basic physics
- Make the program take input for testing
- Convert the program to make it AI ready
- Implement the AI technique

Tools

In order to realize the objectives there are a couple of libraries that must be learned from python. The first library is pygame, it is a set of python modules that are designed for writing video games. Pygame supports 2D animations by drawing pictures on the canvas (screen), within a while loop, you keep re-drawing the images as you update their (x,y) positions, thus creating an animation. The physics will need to be handled by writing custom code based on physics equations.

The second library is neat-python. NEAT stands for NeuroEvolution of Augmenting Topologies, it is an evolutionary algorithm that creates artificial neural networks. The algorithm created a population of genomes (agents/rockets), each genome containing a node (neuron) with connections genes (a single connection between neurons). Each genome contains two sets of genes that describe the artificial neural network.

The way the algorithm works is each agent/rocket is assigned a score, the higher the better. Once all the agents have landed/exploded, the top performing agents are then selected to create the new generation. In this process they are mutated and crossed over. This process is continued over a specified number of generations or until a specific score is achieved.

Pygame Implementation

The pygame library is written well and the class and function calls are intuitive and easy to understand. However time is still required to understand how pygame works overall, how to get up your classes, how to set up the animation, etc. The examples online for pygame are limited, when it comes to finding a similar topic, but enough to get you started on a project.

Difficulties and Challenges

There are several problems encountered with pygame. The first issue would be the loop structure. Pygame requires a while loop in order to function, for every loop you run functions that change the object's positions, then call the draw on screen function to update the screen buffer, as you loop it looks like an animation. Understanding the ordering of functions, the scope in which you call the draw functions, all matter and make a difference.

The physics were another problem, when first creating gravity the rocket would disappear from the frame instantly, because the acceleration values were overshooting. Setting an acceleration value of roughly 0.000001 did the trick. It was later understood that it is encouraged to use the Clock class within pygame. The clock class contains a function called tick(), you pass in the frame per second desired and this fixes the physics and acceleration, the acceleration was set to 0.02 and this was much more suitable.

The boundaries, landing pad, and crashing outside the landing pad were other problems that needed to be broken down to pieces and incrementally developed and tested to cover all the edge cases.

Overall developing the game was an incremental process that needed to be tested on every change, thus resulting in several days of development.

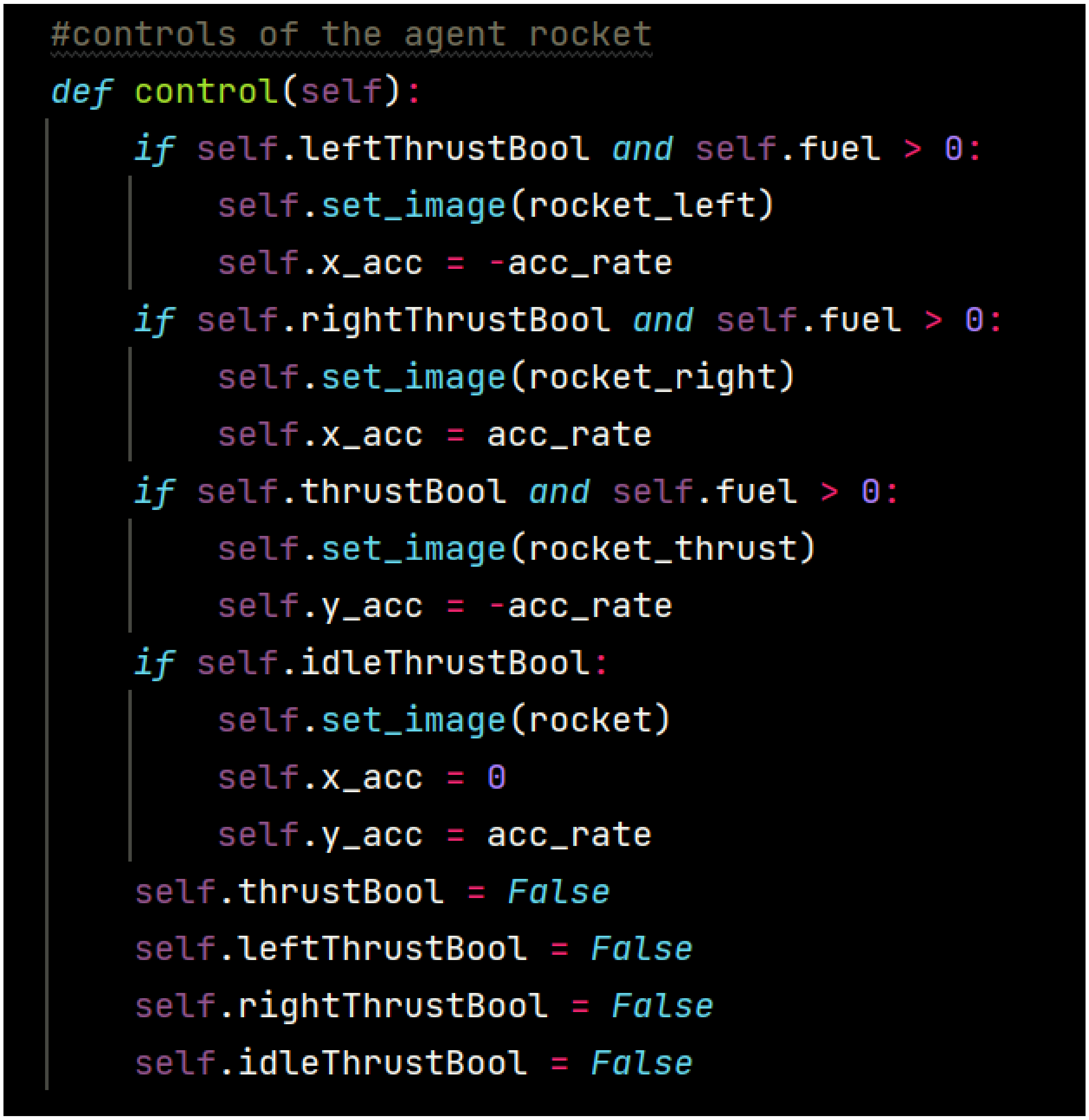
NEAT Implementation

The neat library is relatively easier to use and learn than pygame. It uses a configuration text file for fine tuning the settings, similar to hyper parameter tuning. Values that affect the population size, max fitness score criteria, number of inputs, number of outputs, and any hidden layers.

Difficulties and Challenges

There are several examples online that help you understand the basics of implementing the neat algorithm to your project solution. However there are conditions that cannot be foreseen if using this library for the first time.

The structure of the program must be suitable for the algorithm to run. All the controls must be called by function. For example, in order to engage the rocket to thrust to the right a boolean value will need to be set “thrustRightBool”, then a function be executed given the condition that “thrustRightBool” is true. The below is a snippet of the code:



```
#controls of the agent rocket
def control(self):
    if self.leftThrustBool and self.fuel > 0:
        self.set_image(rocket_left)
        self.x_acc = -acc_rate
    if self.rightThrustBool and self.fuel > 0:
        self.set_image(rocket_right)
        self.x_acc = acc_rate
    if self.thrustBool and self.fuel > 0:
        self.set_image(rocket_thrust)
        self.y_acc = -acc_rate
    if self.idleThrustBool:
        self.set_image(rocket)
        self.x_acc = 0
        self.y_acc = acc_rate
    self.thrustBool = False
    self.leftThrustBool = False
    self.rightThrustBool = False
    self.idleThrustBool = False
```

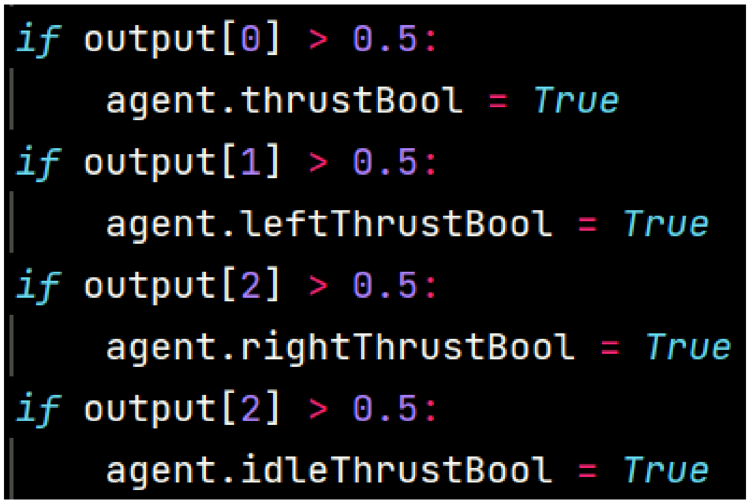
The rocket class has to be self-contained as much as possible, it cannot depend on other classes or variables and must be self-sufficient to function as much as possible. The reason for this requirement is that NEAT will create a list that contains the population, the agents/rockets. Given this requirement each agent will be looped and executed every while loop iteration.

The work required to make NEAT ready for the program, assuming the application has been overhauled, is the main function. Two additional inputs are required: genomes and configuration (comes from the configuration file).

Testing the new implementation, the population of agents created was 5, once all 5 agents crashed the next generation should start with another 5 agents. Beyond this step is tuning the NEAT algorithm with the configuration file.

Results and Analysis

The results of this project were poor and there are many factors and reasons that lead to this to be discussed below. The training for the rockets is proving to be ineffective, even after 500 generations there are no good results.

- The input is a tuple of the values being fed into the network. Various inputs were tested but did not succeed.
    - 3 inputs, agent (x, y) and distance from the landing pad
    - 4 inputs, agent (x, y) and landing pad (x, y)
    - 7 inputs, agent (x, y), agent x_speed, agent y_speed, fuel, land pad (x, y)
    - 10 inputs, agent (x, y), agent x_speed, agent y_speed, fuel, land pad (x, y), distance from landing pad, agent x acceleration, agent y acceleration
- The outputs return a list, see the code snippet below:
    - 

```
if output[0] > 0.5:
    agent.thrustBool = True
if output[1] > 0.5:
    agent.leftThrustBool = True
if output[2] > 0.5:
    agent.rightThrustBool = True
if output[2] > 0.5:
    agent.idleThrustBool = True
```

    - An attempt was made at 3 outputs, removing the idleThrustBool, but this did not change the results very much. The output however should be 4, to represent the proper controls for the agent, therefore the issue cannot entirely lie here.

- There were various hidden layers tested, however the recommended number by the algorithm was zero, since the algorithm modifies the hidden layer(s) of the neural network every generation.
  - 0, 1, 2, 4
  - 4 hidden layers increased the behaviour at the start of the generation, rather than thrusting upward there was more horizontal movement.
  - A radical attempt was made, 10 hidden layers, no improvement was visible.
- The scoring system may be the culprit. The fitness score is calculated by how much fuel is left, the distance to the landing pad, whether the agent crashed or landed, and if the agent was greater or lesser than 2.0 m/s when landing. The scoring system is very important as it determines the fitness score, it determines which genes live and which ones die.
- The rocket class itself, may or may not be the issue, this is a difficult possibility to examine. Analyzing the class code itself, there are 2 functions responsible for moving the agent, the move() function updates the position of the agent (x, y) based on the x_speed and y_speed. The second function is called control(), it sets the x_acceleration and y_acceleration values from either negative to positive, thus changing the agent's direction. Perhaps the issue lies here, the calculations determine the direction of the agent and its momentum, if they are inaccurate it will affect the performance of the algorithm because it cannot evolve the neural nets because almost every rocket results in poor performance by crashing. The question here then is how to validate if the physics model implemented is correct?

As discussed above in the analysis, a rework of the scoring system and a validation of the physics model implemented are very likely to be the root cause; a rework of these 2 modules may bear fruit to better results.

Conclusion

In conclusion the project was not successful in landing Elon's SpaceX rocket booster on Mars. The rockets mainly just fall to the floor or try to get back into orbit. A rework of the scoring system and the physics model should prove useful in producing better results.

Even though the project was not successful there is much knowledge that was obtained and new perspectives discovered. Perhaps the goals of this project were too ambitious for an amaetur and a beginner with little experience in this field.

Future Works

This project will be reimplemented again in the future. However before that, like building the rocket class, the knowledge of this field must be incrementally built upon. The next project in line is to start with a simple idea, for example jumping over obstacles, shooting non-moving objects, driving and avoiding the edges of the road, etc.

Starting from small project ideas and understanding the nuances will better prepare for the next project, and eventually the agent will be able to land on Mars.

References

- NEAT library
  - Was used to learn more about NEAT
  - https://neat-python.readthedocs.io/en/latest/neat_overview.html
- Pygame
  - Was used to learn more about pygame functions
  - https://www.pygame.org/wiki/about
- Neural Network NEAT
  - Self driving car using NEAT, how I learned about NEAr
  - https://github.com/Hilicot/Neural_Network_NEAT
- Lunar Lander
  - The game Lunar Lander all in one python file
  - I was able to reference and learn the physics, collision handling, and the loop structure of pygame
  - https://github.com/phildecroos/Lunar-Lander
- NEAT Chrome Dinosaur
  - This gave a very simple example of how to implement NEAT in your code
  - https://github.com/codewmax/NEAT-ChromeDinosaur

Thank you for reading my report and thank you for the wonderful class! I learned a lot!