The Astro Spider

ULTIMATE LANDER
VIP EDITION
DELUXE 4.2
FREE

AI Of The Year Award

Introduction

A small game about a spider who desperately wants to explore the space. His mission is simple: he has to visit 5 planets before he achieves his desire.

Although the task seems simple, navigating through space and landing on planets is not a simple task. It requires some very good manouvering skills.

Now comes your part! Of course a spider cannot manouver a rocket, so you'll have to do it. DO IT FOR THE SPIDER!

The rocket has a laser gun attached, that helps the spider destroy the asteroids and the enemy spaceships that it encounters.

Game controls

- **WASD / Left Thumb Stick** – turn the rocket (use the secondary thrusters)
- **Space / Right Trigger** – use the main thruster
- **Mouse / Right Thumb Stick** – rotate the camera
- **Left Mouse Button / Right Button** – shoot lasers

Technologies

The game was made entirely using the Unity game engine.

The packages we used are:

- the **new Input System** for Unity
- **Universal Rendering Pipeline (URP)**

Everything else was entirely written from scratch.

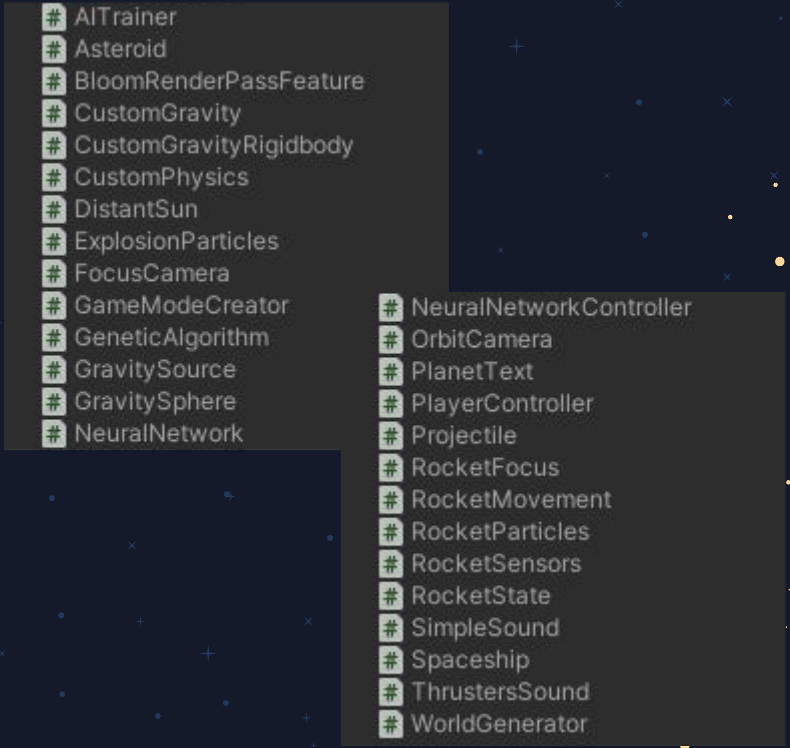
Architecture

We implemented a framework that allow us to directly apply custom physics to some game objects.

To use the custom physics, the only thing that must be done is attach the **CustomGravityRigidbody** script to the desired object.

The rocket prefab has been designed to be used in any game mode (player-controlled or AI-controlled) without the need of modifying the behavior scripts.

The **GeneticAlgorithm** and **NeuralNetwork** scripts have been designed to be standalone classes. They can be used in any C# project.


AITrainer
Asteroid
BloomRenderPassFeature
CustomGravity
CustomGravityRigidbody
CustomPhysics
DistantSun
ExplosionParticles
FocusCamera
GameModeCreator
GeneticAlgorithm
GravitySource
GravitySphere
NeuralNetwork
NeuralNetworkController
OrbitCamera
PlanetText
PlayerController
Projectile
RocketFocus
RocketMovement
RocketParticles
RocketSensors
RocketState
SimpleSound
Spaceship
ThrustersSound
WorldGenerator

Features

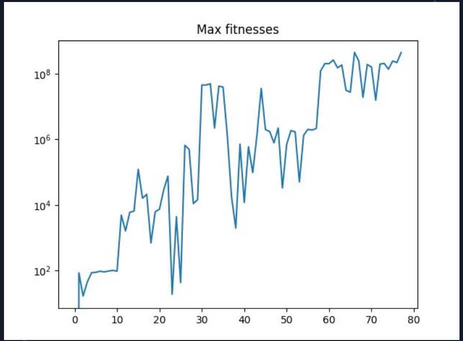
AI

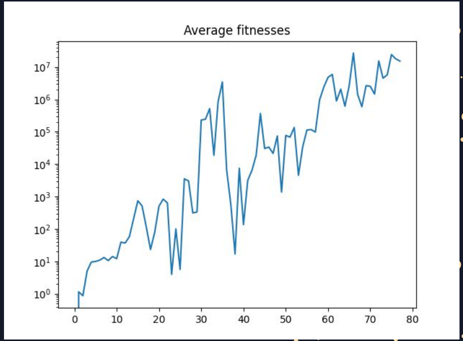
- **Planet lander**

We used machine learning to train an agent to learn how to land on 5 planets. The goal of the agent was to learn how to land on planets with very little impact.

In order to train the AI, we implemented the Neuroevolution of Augmenting Topologies algorithm (NEAT). We didn't use any existing implementation for the algorithm, it was written from scratch.

Even though the game is 3D, the AI for Planet Landing operates in 2D, as training it for 3D is much more complicated. We also removed the obstacles in the world while training the AI.



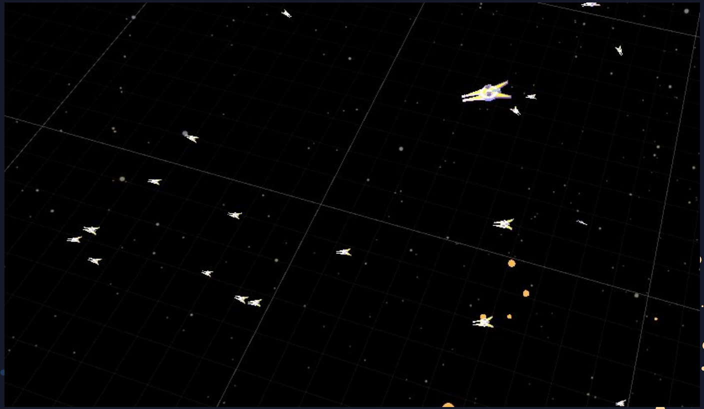


AI

- **Boids**

Besides this planet lander AI, we also implemented boids for the enemy spaceships in the game mode where the rocket is being controlled by a human player.

The spaceships work by the **standard rules of boids**.



Physical Simulation

- **Rocket movement**

The Rocket contains a component called **RocketMovement**. This component is responsible for handling how the rocket reacts when the thrusters are being "accelerated".

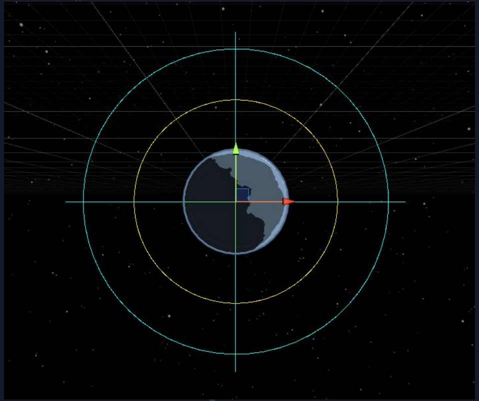
The rocket had 4 weak side thrusters and 1 big main thruster. The side thrusters are mainly used to tilt the rocket, while the big thruster is for going forward.

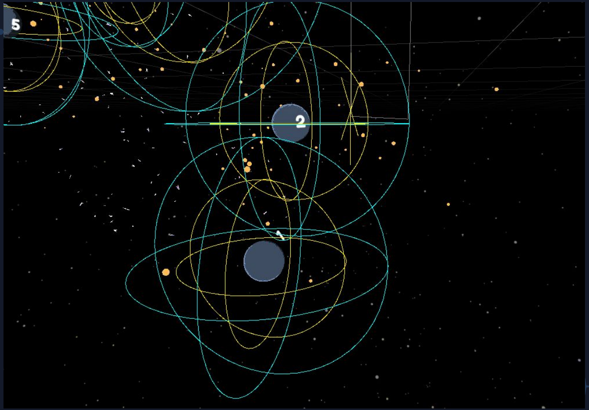
Physical Simulation

- **Gravity**

We also implemented custom gravity. Each planet has a gravity field. The gravity fields of each planet can collide with each other and they consist of two regions:

- **yellow region** – where the gravity force is a constant value
- **blue region** – where the gravity force start so fade away until it reaches 0





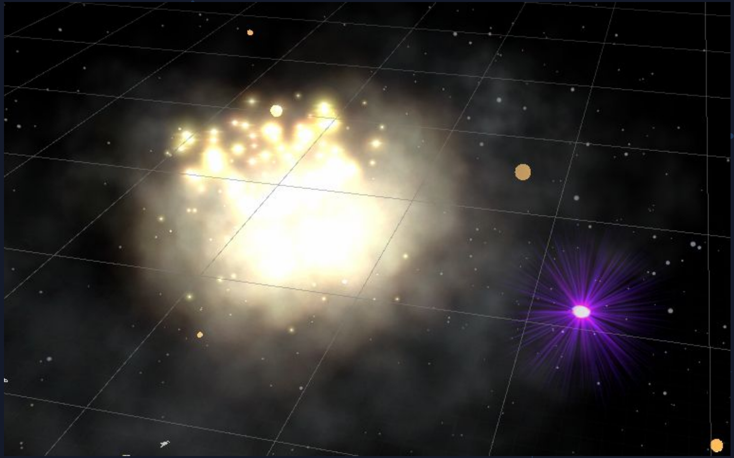
Animations

In addition to all of those physics and collisions systems, we are playing an animation whenever the spider on top of the rocket shoots the gun.

Particle systems

We used particle systems for:

- the thrusters
- the rocket projectiles
- exploding asteroids and spaceships



Custom shaders

- **Cartoon effect shader** – it makes the light fall onto surfaces with an equal intensity (Fig. 1)
- **Black outline** – behind the objects being rendered (Fig. 2)
- **Planet surface shader** – just displace the normals a little bit. This makes the surface seem a bit rough because of the lighting, even though the surface is flat (Fig. 3)

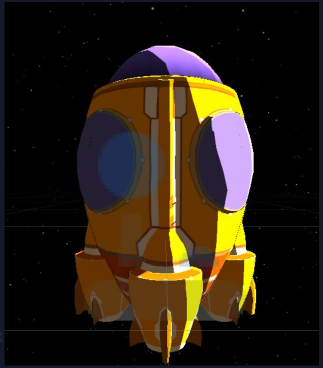


Fig. 1

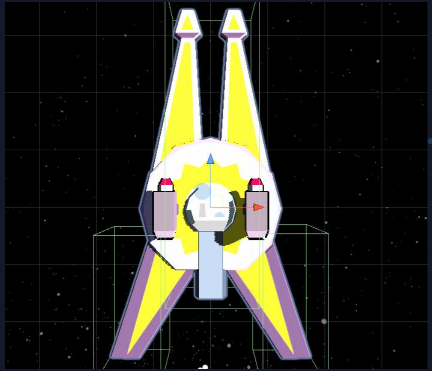


Fig. 2

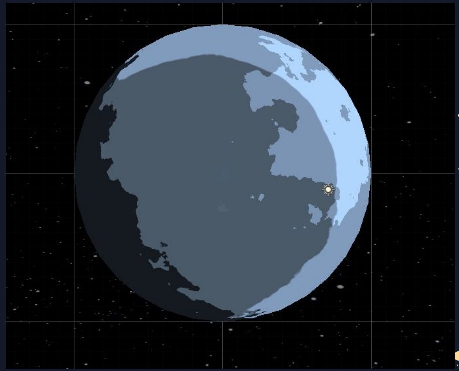
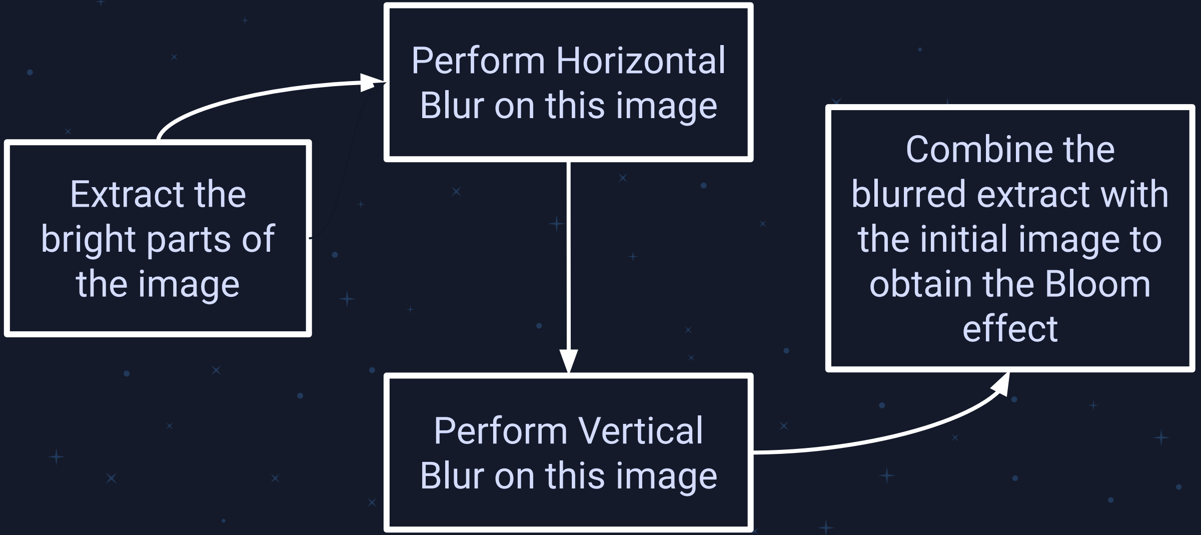


Fig. 3

Post-processing shaders

- We've also added a bloom post-processing effect. The effect was implemented from scratch using render features (the only way to do post-processing while using the Scriptable Rendering Pipeline in Unity)

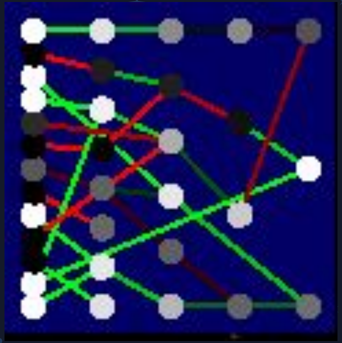

Extract the bright parts of the image
Perform Horizontal Blur on this image
Perform Vertical Blur on this image
Combine the blurred extract with the initial image to obtain the Bloom effect

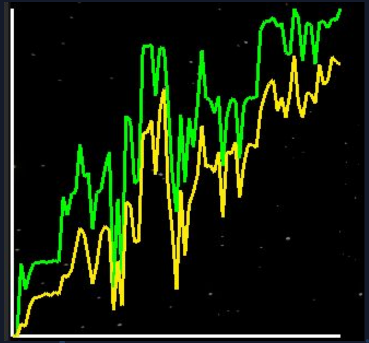
Custom UI elements

Another minor thing we implemented with regards to graphics programming are some of the UI elements:

- the visualizer for the neural network
- the fitness evolution chart

The only way to create custom UI elements is by defining the geometry manually, so the vertices and indices for these elements are created and updated manually.





Thanks!

Bara Tudor	Nanu Alexandra
Blahovici Andrei	Popa Bogdan-Ioan
Buhai Darius	Preda Mihai-Dragoș
Dabu Alexandru-Florentin	Stăncioiu Silviu
Dumitrescu Delia	Vasiliu Diana-Elena

For more information, check out the Github repository.