Interim Demo

Doomsday: Underground Uprise

Technische Universität München

Winter Semester 2023/2024
Games Laboratory

- Matija Jajcinovic
- Hongbo Chen
- Haorui Tan

Overview and Team Cooperation

We assigned the coding tasks in such a way that we had low codependency between each other, extending the idea of the layered timeline (like writing the Tower Defense Logic without needing a concrete Tower Model).
Matija was designing the Camera Logic, Game State Logic, 3D Asset Generation and MLagents Agent Code Design;
Haorui was working on logical design and animation of defense towers,enemies and battle robot logic.
Hongbo did the

Implemented Features

We successfully implemented numerous features, although we had to defer a few due to an unforeseen increase in workload during the implementation process.

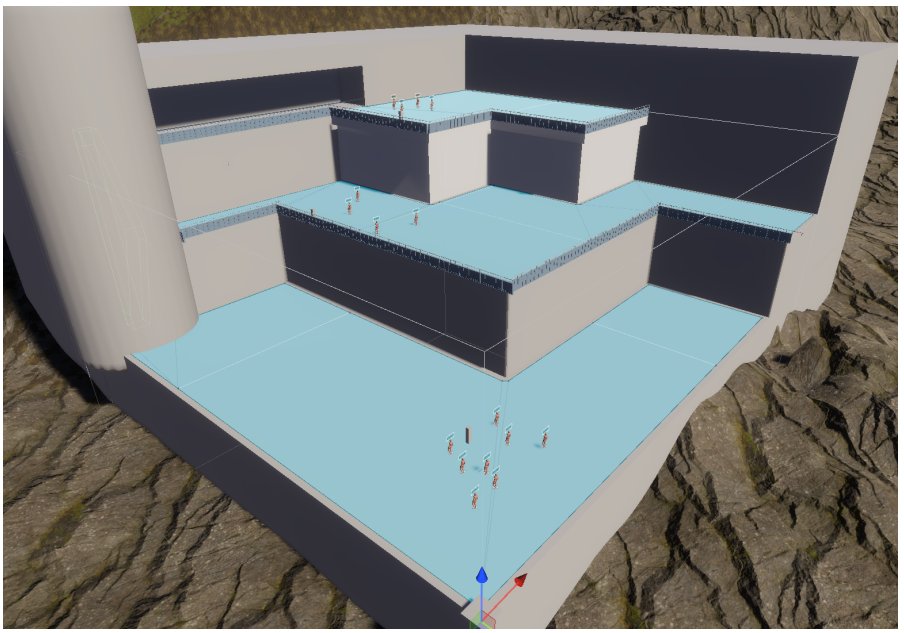
Hongbo

What I have implemented:

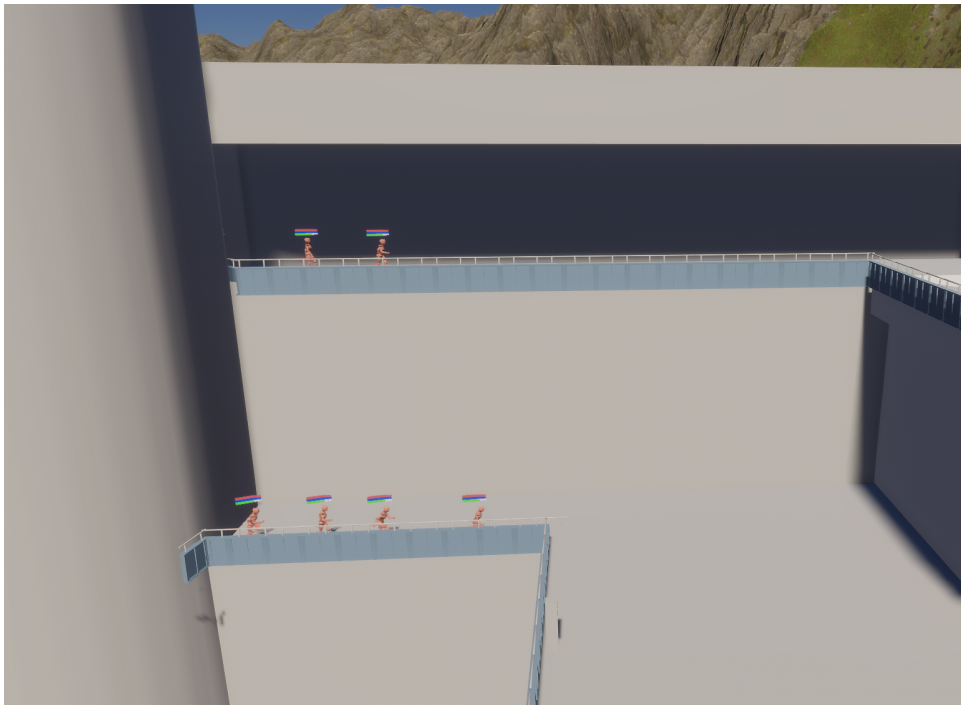
1. Underground citizens' navigation system with elevator logic:

Completeness: **layer3:** completed all the needed logic and the people in the underground basement can navigate to different layers correctly.

Collision mesh(blue layers) used for nav mesh:



elevator effects:

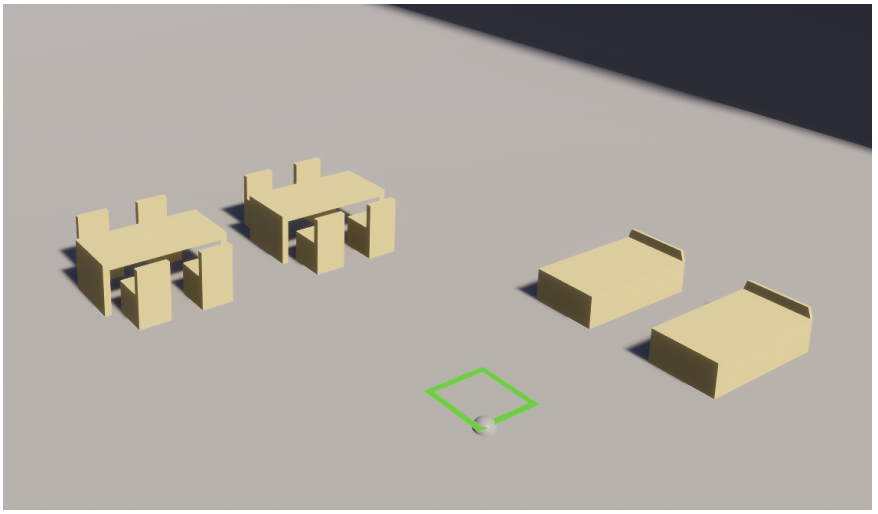


2. underground basement building system:

Completeness:Layer 3: The underground basement building system has been completed, providing the fundamental building functions.

Further works: However, it still requires the implementation of border constraint and collision detection logic to ensure proper placement and prevent overlapping of buildings. Furthermore, future iterations of the system will require an expanded list of building options, which will involve collaboration with the team responsible for building modeling.

Construction System:



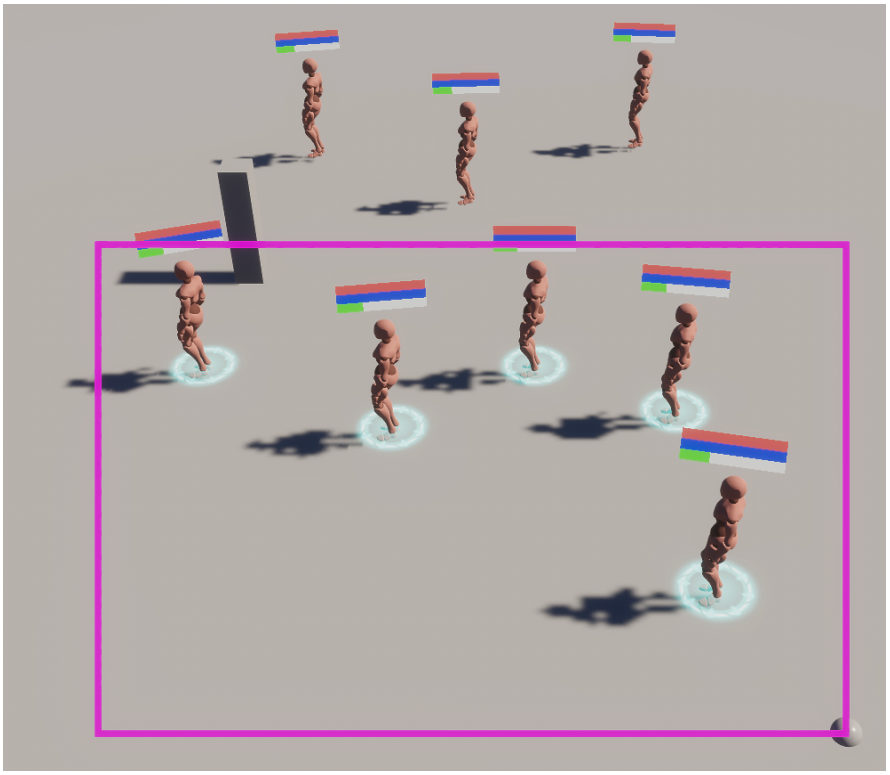
3. People's selection box and deployment logic:

Completeness:Layer 2: The current method of detecting the selection box, which utilizes a mesh trigger established by the vertices from the camera position to the vertices hit by the raycast function in Unity.

Further works:

1. This approach lacks robustness when the ground terrain is not a simple plane. Therefore, the logic for establishing the trigger mesh needs to be enhanced to accommodate complex terrains and ensure accurate selection box detection.
2. the deployment logic could be more wise, which means the player don't have to click the workers one by one rather just click one type of the buildings then the workers know to evenly approach to the buildings' positions

people's selection box:

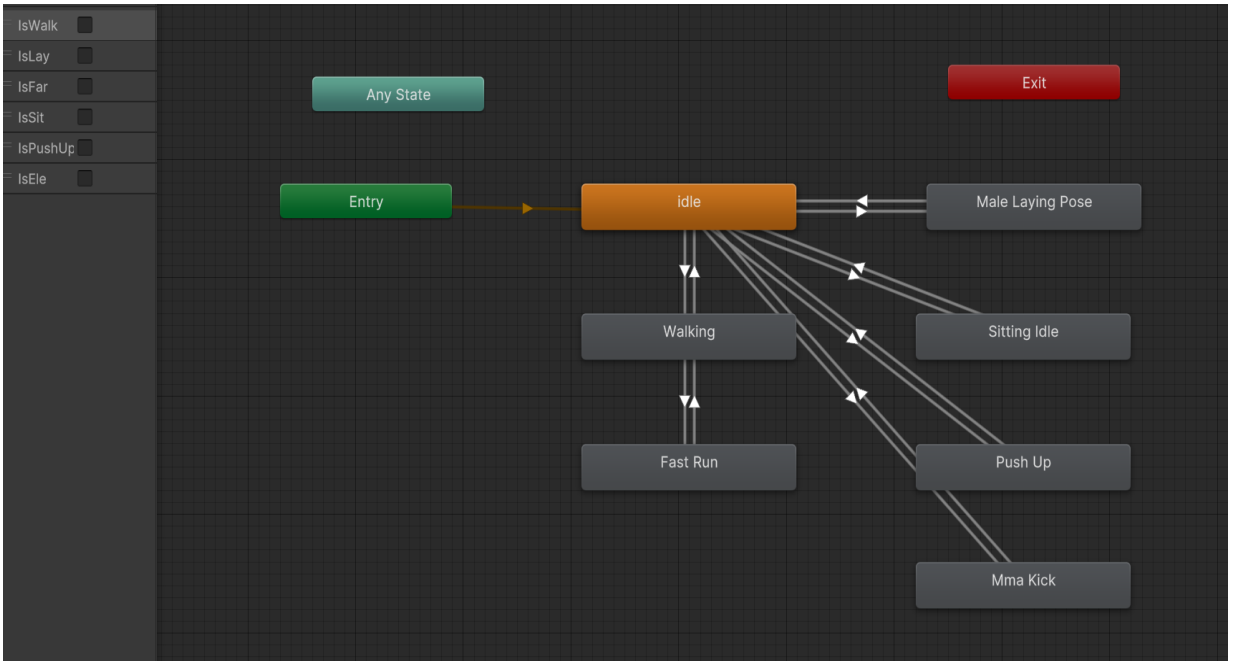


4. Gathering animations of people and achieving animation control logic:

Completeness: Layer 3: The fundamental logic for animations controls of people has been completed.

Further works: However, to add more variations, further work is needed. This includes expanding the available animations and refining the animation control logic to provide a wider range of movements and behaviors for the underground citizens.

animation control logic:

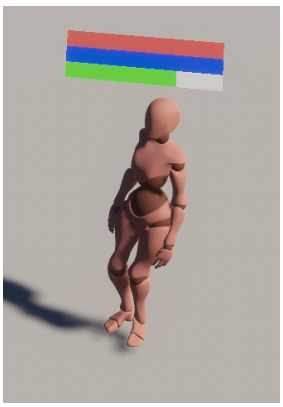


5. People's status bar:

Completeness: Layer 2: The current implementation of the people's status bar can display all the necessary information regarding an underground citizen's status.

Further works: There is still room for improvement to make the status display more intuitive and easier to interact with. Enhancements may include visual cues, intuitive icons, and streamlined user interactions to provide a more user-friendly experience.

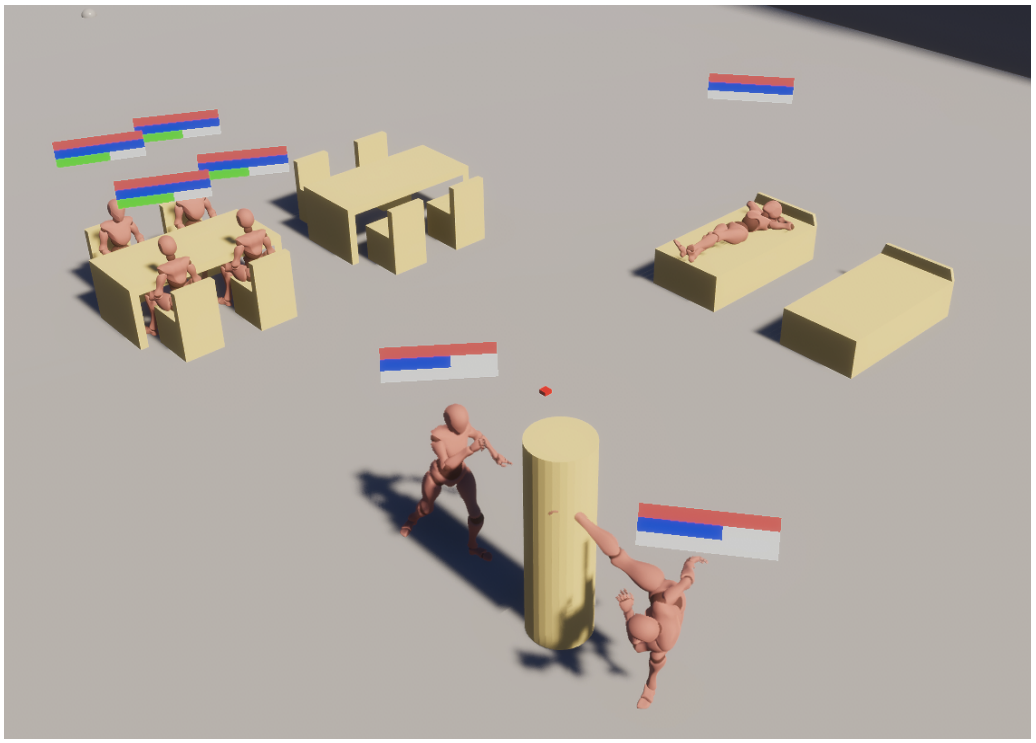
people’s status bar, red bar denotes health value; blue bar denotes energy value which has a relation to the work efficiency and green bar denotes hunger value which measures how much time remains until the people need to eat::



6. Interaction logic between people and buildings:

Completeness: **Layer 3:** All important logics related to the interaction between people and buildings have been finished. This includes animation switching, people finding correct seats within the buildings, effects generated by people in relation to the buildings, and effects generated by the buildings in relation to the people. These interactions create a dynamic and immersive environment for the underground citizens.

People can locate to the right position and do the right movement and buildings can bring influence to the people's status value:



7. Electricity voltage system:

Completeness: **Layer 3:** The important logics for the electricity voltage system have been completed. This includes handling the effects caused by the voltage system

Further works: integrating them with the surface combat. The electricity voltage system adds another layer of complexity to the underground environment, impacting various aspects of the simulation.

Electricity Voltage Logic:

```
if
    demanded EV <= possess EV
then
    all facilities can work in 100% efficiency
if
    demanded EV > possess EV
then
    all facilities work in (possess EV)/(demanded EV)*100% efficiency
```

Assume EF denotes efficiency, CE denotes consumed electricity per building, GE denotes generated electricity per building:

$$EF = (\frac{\sum CE}{\sum GE})^2$$

8. Demo display of voltage and character status and construction menu:

Completeness: **Layer 3:** The demo display, which showcases the voltage system, character status, and construction menu.

demo display of voltage and construction menu:

What challenges have I encountered:

1. **Problem:** How to make the right user interaction logic when the player use mouse button to click different objects
    a. **Solutions:** define different collision detect layers in Unity, and each layer is used in a certain task
2. **Problem:** When working on the elevator logic, I used a trigger for detecting collision meshes of the people, Which firstly didn't function as I expected.
    a. **Solutions:** The trigger in Unity only works when the object is a rigid body
3. **Problem:** When working on the people deployment logic, there are a lot of places to use the ray casting function provided in Unity, different raycastings are expected to detect certain different layers to achieve the logic that people can react correctly according to the player's pointed objects. In this case, I allocated different layers and set the ray casting functions to detect certain layers, but they didn't works as I expected
    a. **Solutions:** I need to set a maximum ray cast distance first, then can I let the raycasting function to detect certain layer
4. **Problem:** The code structure design of the people's script and the building script, and how to make them cooperation to achieve interaction logic and building placement logic
    a. **Solutions:** make good use of List<> structure in C# scripts to implement right data identify, write and read logic
5. **Problem:** Raycast function costs high calculation resources, which means we need to limit the usage of this function to improve game performance
    a. **Solutions:** prevent using raycasting function in main loop or other loop logics that operate in default case

Matija

In my last Practical the fast prototyping approach of writing code led to a totally messy project which I had to refactor to be able to add new features. Learning from my previous mistakes I choose to start this time by using proper Software Patterns to design an extensible code. This approach has a slower start which postponed some of the other progress but will be beneficial in the middle term and long run.

Custom Extension for Mlagents framework

Unity's ML-AgentsFramework enables games and simulations to serve as environments for training intelligent agents using machine learning. It provides a toolkit for integrating state-of-the-art deep learning technologies like PPO or POCA into game and simulation

environments for training autonomous agents to perform complex tasks, enabling advancements in AI research and application within the Unity platform.

It functions as a mediator between Python and Unity.

Design

In my last semester the amount of work to set up a new Agent was very high - about 20 hours. This was caused by several reasons:

- The code for Training the Agent and to use it in the game is difficult to separate
- every Agent needs a new script and set up with a lot of redundant code
- additionally the usual problems with states, resets etc.

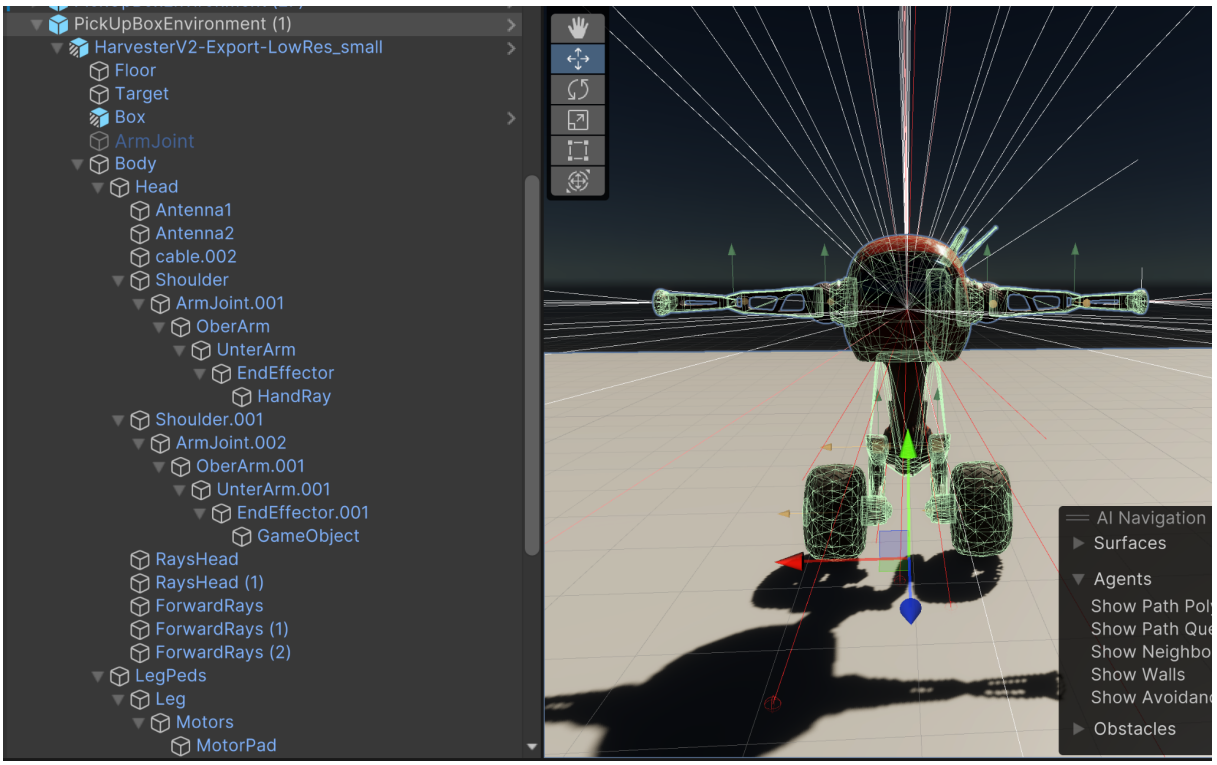
Bc this project will require at least 5 different physical animations done by ML I decided to go the difficult way and to design a more abstract state based Agent System which aims to be easy to set up and reusable.

My Extension Code is several thousand lines long so here are only the basic components:

- **AgentState:** implements a State Pattern
- **Actuator**: Abstract class that takes an input and interacts the the game world somehow (JointActuator, TireActuator etc.)
- **ISense**: Interface to observe the environment (e.g. TransformSensor, TireActuator who return the current speed etc.)
- **AgentTriggers**: react to contact with objects (reset Agent, Reward etc.)
- **ObserveManager**: just catches all ISense of the current Agent and calls their function; saves it a in custom DictionaryList
- **ActuatorManger**: Same as observe Manager
- **TransformFunction**: Kind of Factory Pattern that lets the user define a list of variables that it wants to observe from a TransformSensor. Very handy bc it's required to get the Transform Data from different reference frames etc. Factory Patterns helps to abstract it from the specific implementation
- **Randomspawner**: Manipulates the rotation and position at every Episode

Harvester Setup

The Harvest Robot is a difficult Robot to program: it is an inverse pendulum or balanced robot, having only 2 wheels for keeping balance and its arms.
In Robotics a PID controller can solve this, but needs a lot of tweaking and some tricks (like non linear scaling according to lean rotation).
The simple Catch the Box Scene (next section) showed how well PPO works in such cases, where it has only the torque as a controllable input for the wheels and its arms (arm has 3 DoF in total).

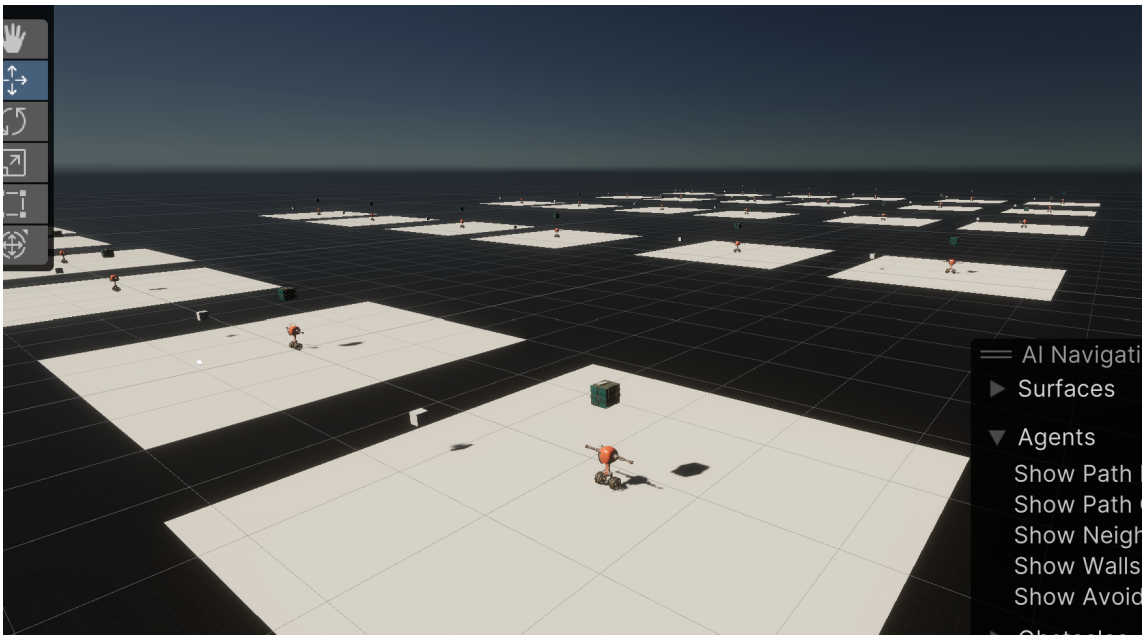


The Setup of the Rigidbodies and Joints was a nightmare with exploding Simulations, Colliding parts, and setting up individual joints' axes.
The solution was that compound Rigidbodies' Colliders had to be set to ignore each other (inner collisions caused exploding), that for some reason the z-axis cannot be free when setting x,y for the shoulder (weird unstoppable movement) and to make in blender an Empty Object to show where the joints rotating axis will be.
I gave the robot additional springs in the legs to make it behave more interestingly/realistically.

Catch the Box Scene Setup

I had no time training the Harvester Movement bc first the detailed surface map should be defined to make the training stable (bumps).

I implemented a simple Catch The Box scene instead to test the framework.



It indeed delivers a faster setup, even though the Trigger and respawn mechanism are to tied together; sometimes you want to restart the Episode (one round intraining) without respawning the agent's position.

The catch the box scene isn't trained yet, but after 3 hours it showed that the agent understands somewhat the assignment (tries to balance the box and keep it in the air, sometimes it throws it in the direction of the target).
The rewards are: stayStraight + Dot(boxVelocity, normalFromBox2Target).

What it showed is that the difficult task of keeping balance can be trained.

Camera Controller Logic

The Camera Movement is one of the most important tools for the player to interact with the game world; it is the main navigation tool hence it had to adhere to modern RTS standards to not feel "cheap". It has to support a wide range of inputs (e.g. arrows keys and WASD) and multiple functions (rotate, zoom, follow unit) while handling contexts well (cannot click on a Unit while in main menu).

How it works

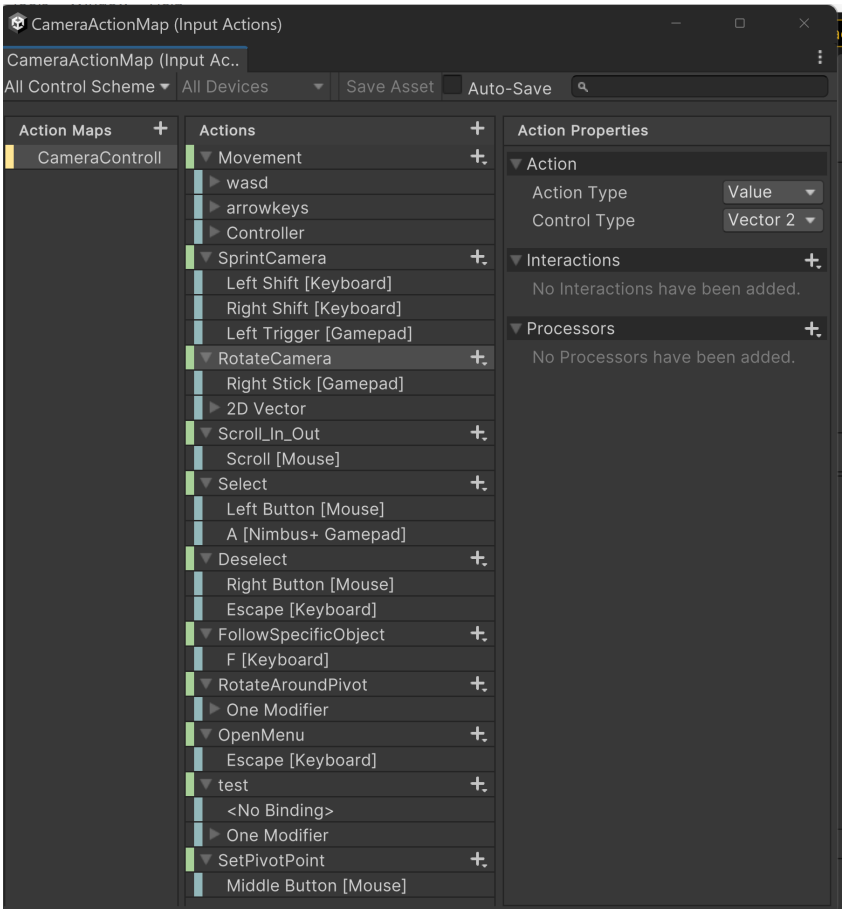
The Camera uses several PID controllers for rotation (Quaternion PID from asset store) and position (custom, for each axis). The player controls the target which the controller tries to reach.
This approach has several benefits:

- a realistic feel to the camera without having to implement custom animation curves, slow start, etc.
- Events like Explosions that give a shock to the camera by simple force.

- no camera clipping bc of the rigidbody

Set of Controlls

- Movement: wasd for 2D plane movement, control for down and space for up
- Sprint: faster movement like in FPS
- Rotate: about global up axis
- Rotate Around pivot: click middle mouse to look at an object or click and drag to rotate around an pivot (raycast to place where you clicked)
- etc.



Game State Logic

To be able to do so I invested more time into designing an abstract Game State Handler.

The Game State Handler uses 4 parts:

- Unity’s advanced New Input System [1] which is an event based flexible framework.
- UniRx, short for "Reactive Extensions for Unity," is a library for asynchronous programming in Unity offering reactive and functional techniques.
- A custom approach that combines a:
  - State Pattern
  - and a not correctly used Command Pattern

The system works as follows: the Unity Input system generates a new event triggering a subscribed function, which then changes the value of UniRx’s ReactiveProperty<Vector3D>.

E.g. the player presses a move forward key, which is mapped to both w and forward arrow; the event then triggers the OnMovePerformed function, which changes the value of the Reactive Property.

The GameLogicHandler has subscribed to the ReactiveProperty movement which triggers an event when its value is changed.

This event again generates a new MovementCommand, which is handled by the current GameState of the GameStateHandler.

This seems daunting at first, but it produces very clean code which is totally context sensitive.

The current state can decide what to do with the command object:

- stack it (e.g. Pause the game, do commands and click play)

- reverse actions/undo

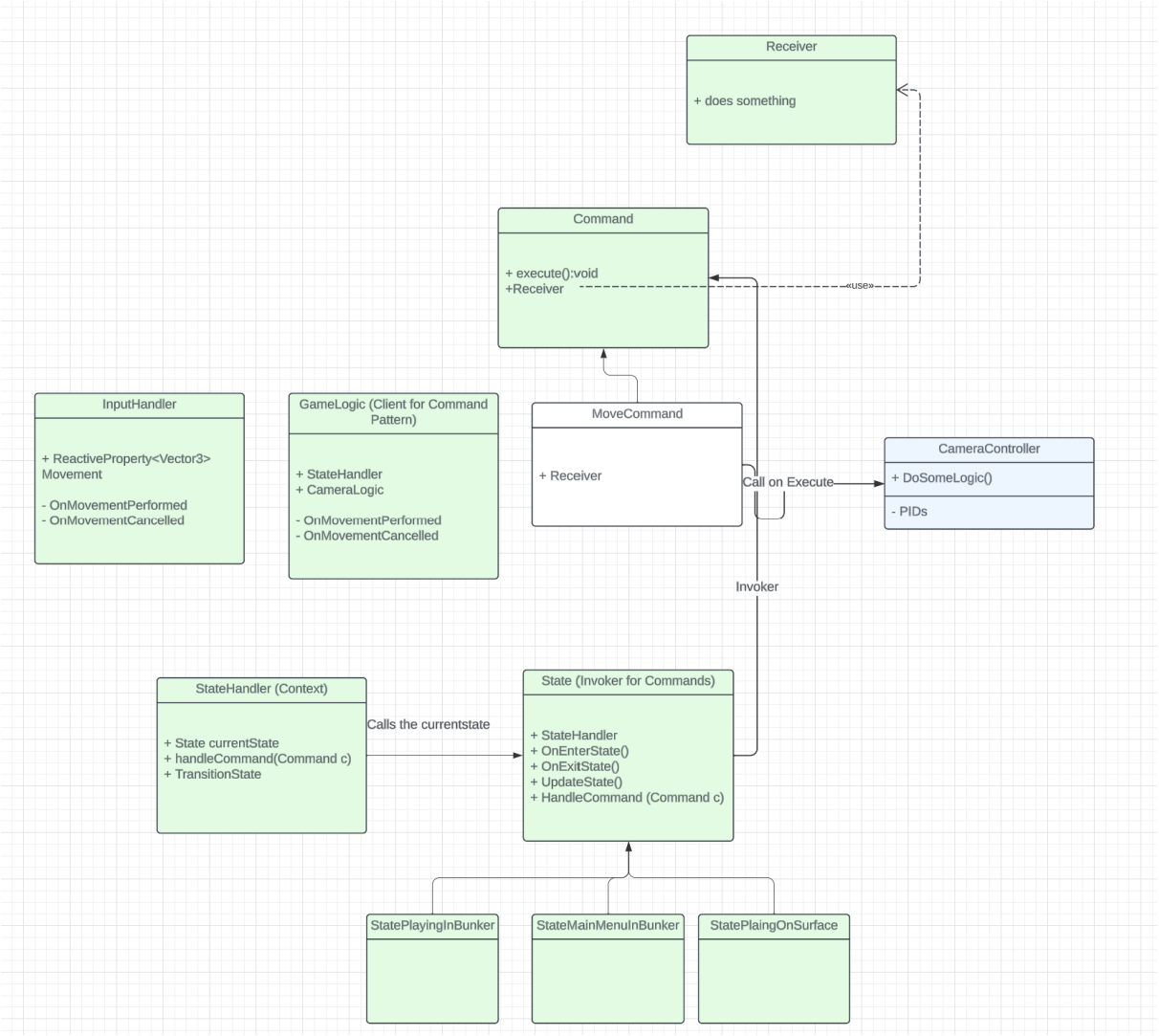
- react to/invoke only certain command types (e.g. don’ react to MoveCommand when in MainMenuState)

The downside of my system is that the system requires a lot of writing/biolercode which is more than compensated by the clean and extensible code.

3D Assets

I designed, modeled and textured all models from scratch to adhere to our realistic style with some stylized features. The postprocess Graphic Novel effect (like outlines) will be implemented at a later stage and should fit with the realistic models and rendering.

[1] Unity's new input system offers a flexible and extensible framework for managing inputs from various devices, allowing developers to easily define and respond to different control schemes. It supports a wide range of input devices and enables more complex input handling, such as mapping multiple actions to a single key or handling simultaneous inputs from different devices.

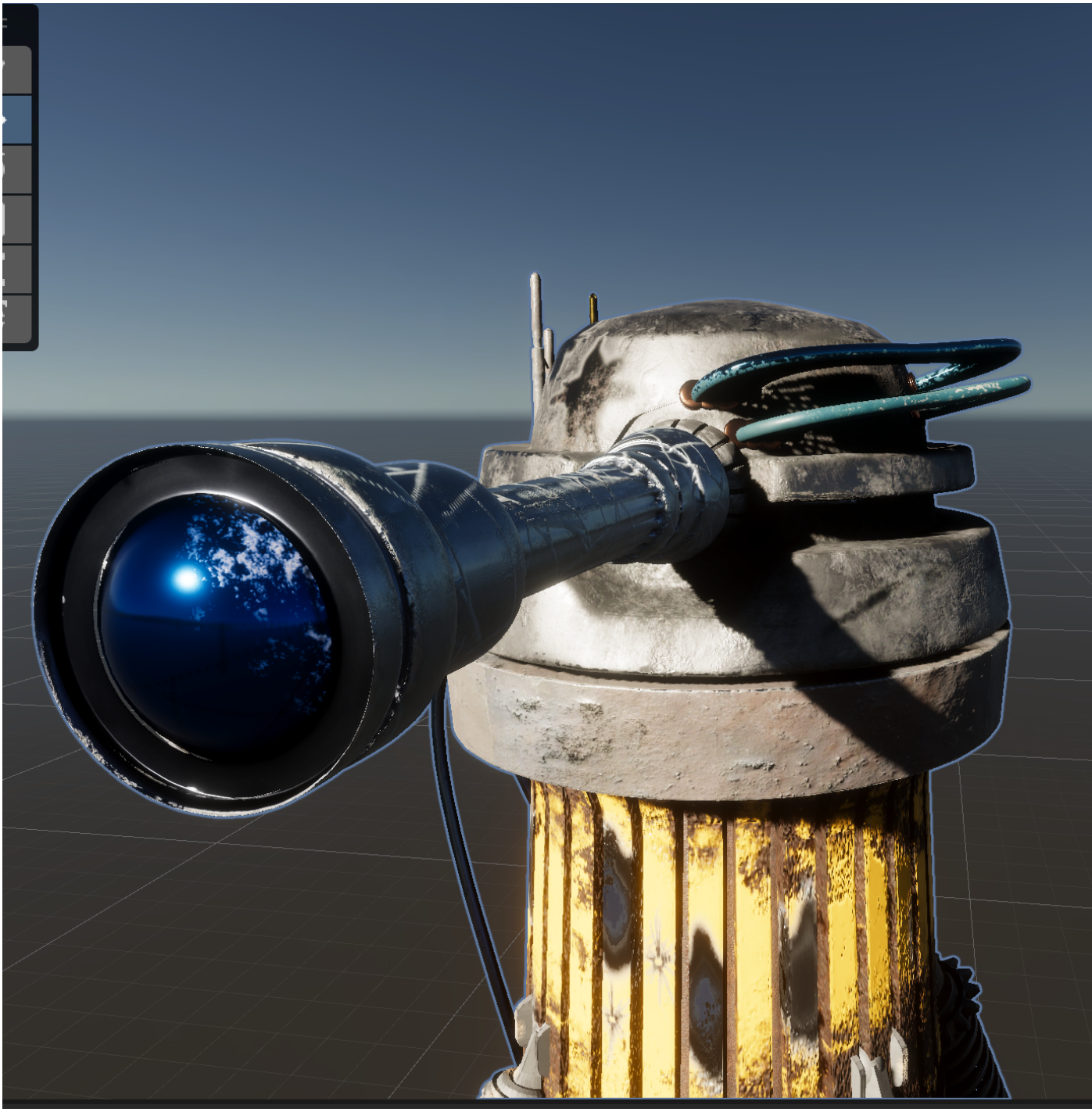


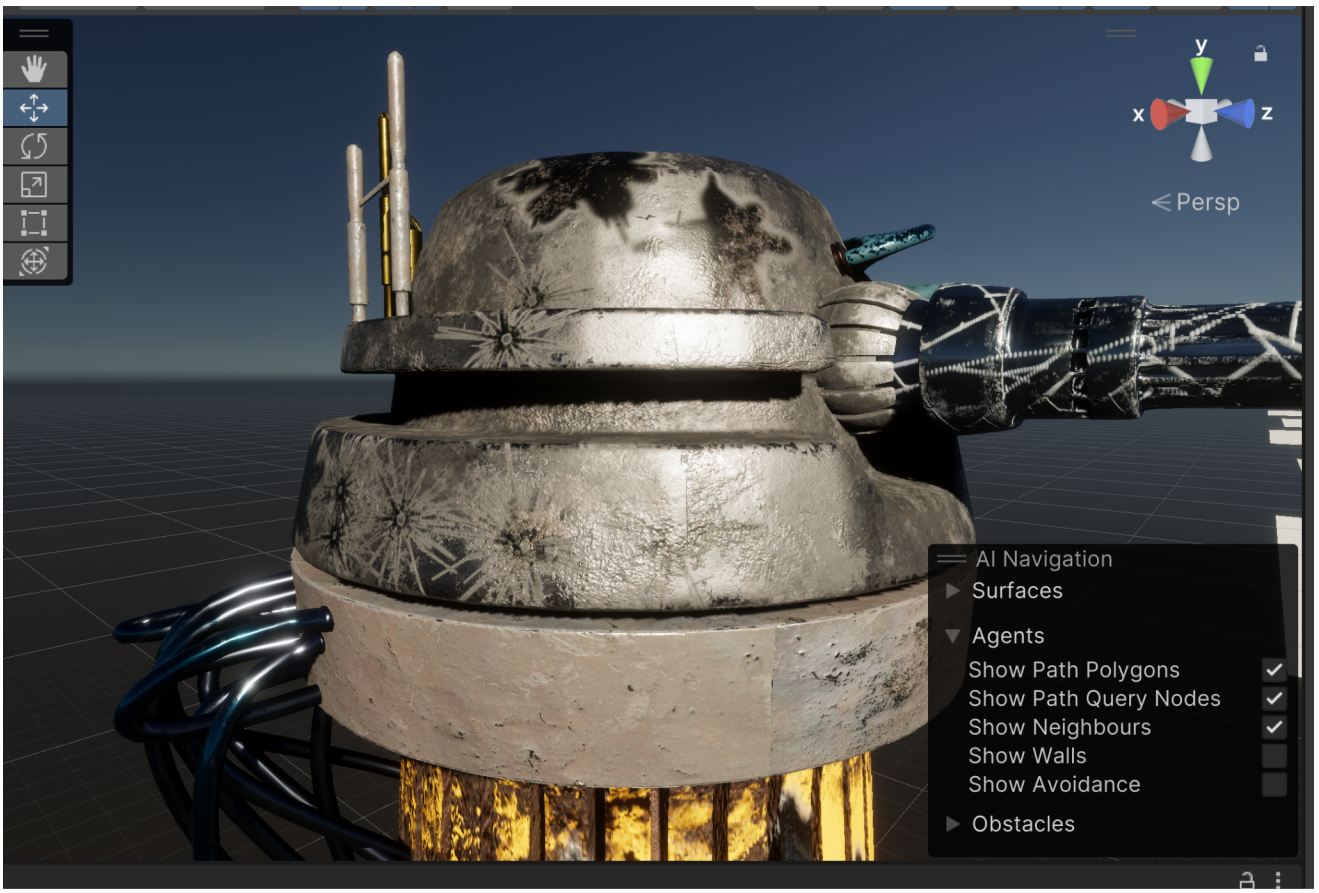
Models

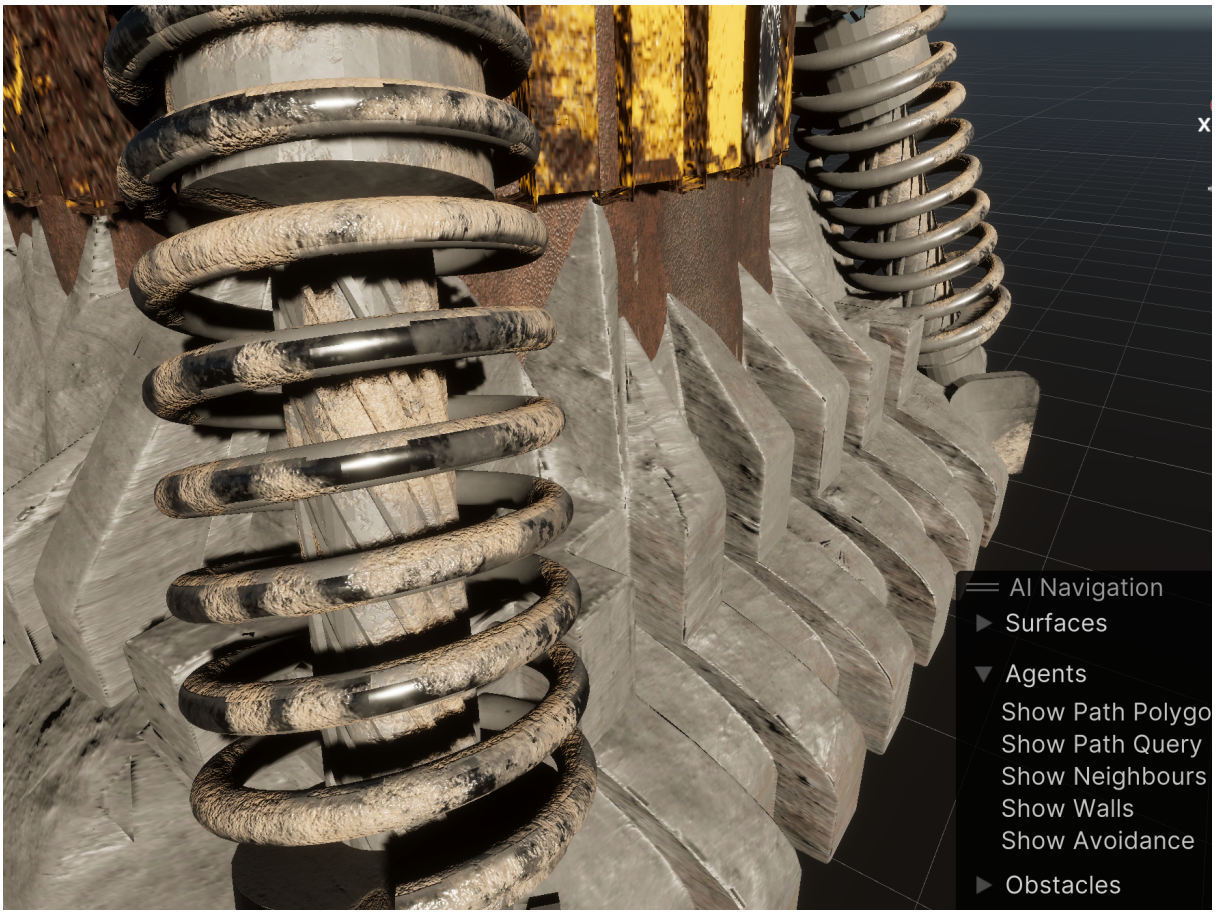
I had some experience with Blender before (Blender 4.0 used in this project), but every Version changes controls etc., so I needed some getting used to. I managed to create somewhat interesting models that fit the game world. They are high poly and textured with Substance Painter 2023 which was a new tool for me to learn. Compared to Blender it is much simpler and accelerates both the content creation and the quality.

LaserTower

The Laser Tower has a rotatable top (up-axis) and a rotatable muzzle (right-axis).



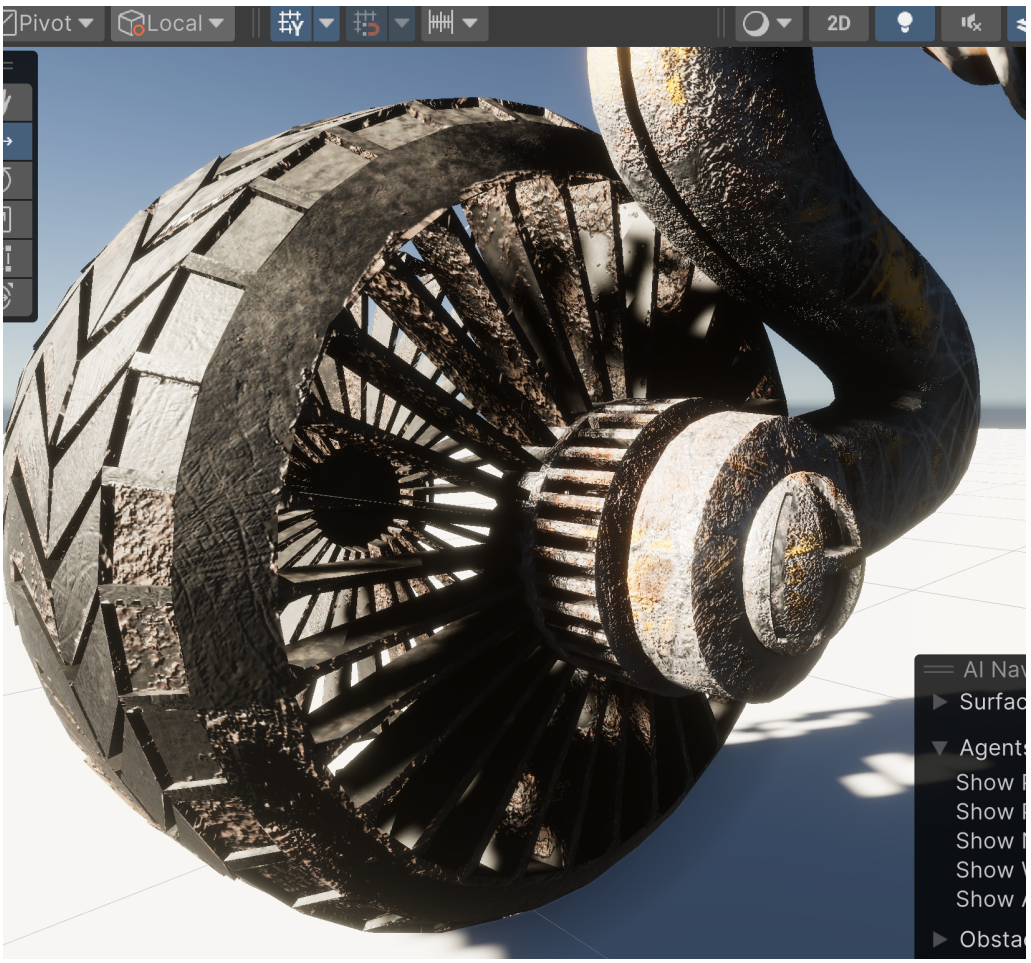

y
x
z
Persp
AI Navigation
Surfaces
Agents
Show Path Polygons
Show Path Query Nodes
Show Neighbours
Show Walls
Show Avoidance
Obstacles

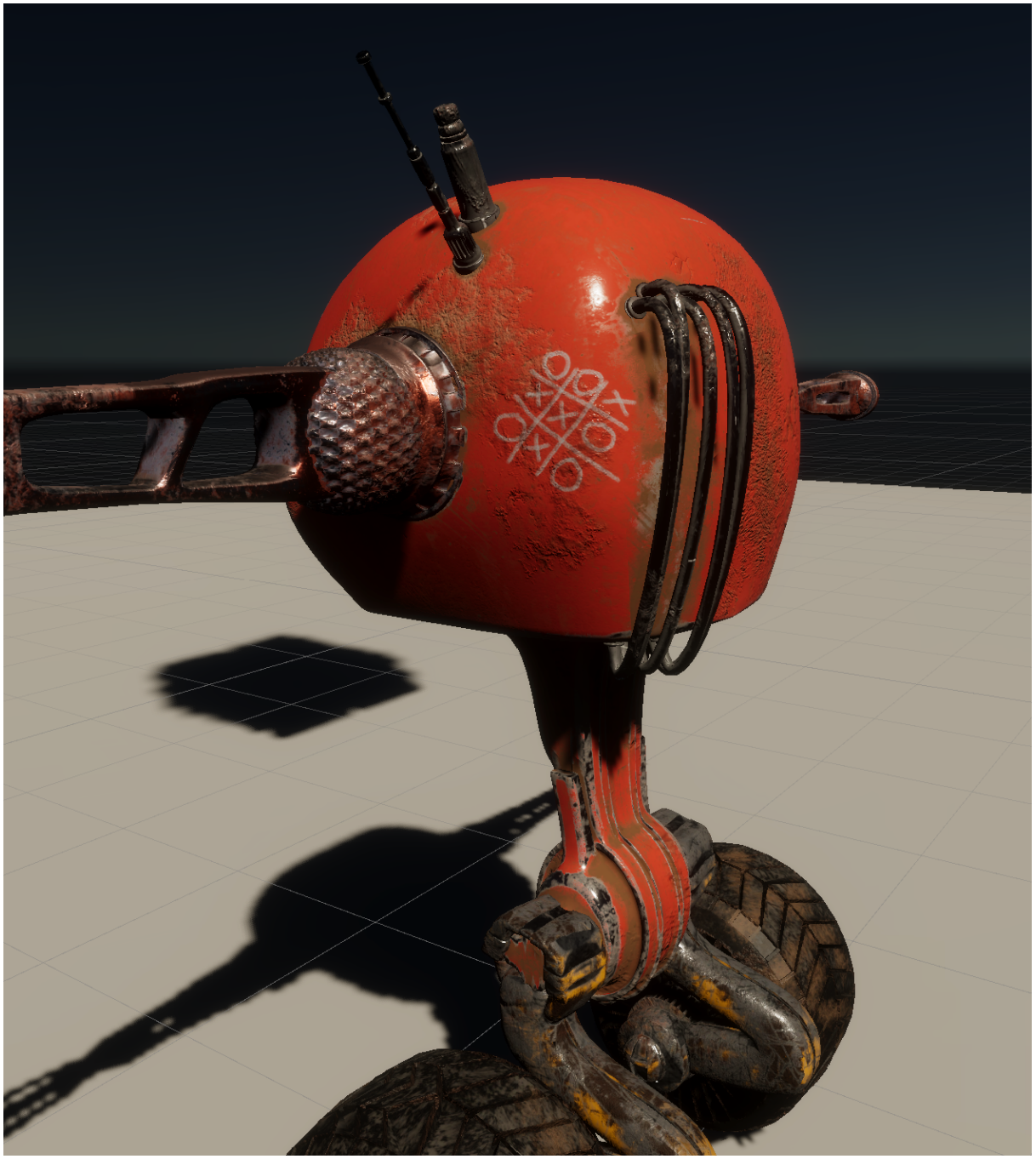

AI Navigation
Surfaces
Agents
Show Path Polygo
Show Path Query
Show Neighbours
Show Walls
Show Avoidance
Obstacles


AI Navig
Surfaces
Agents

Harvester

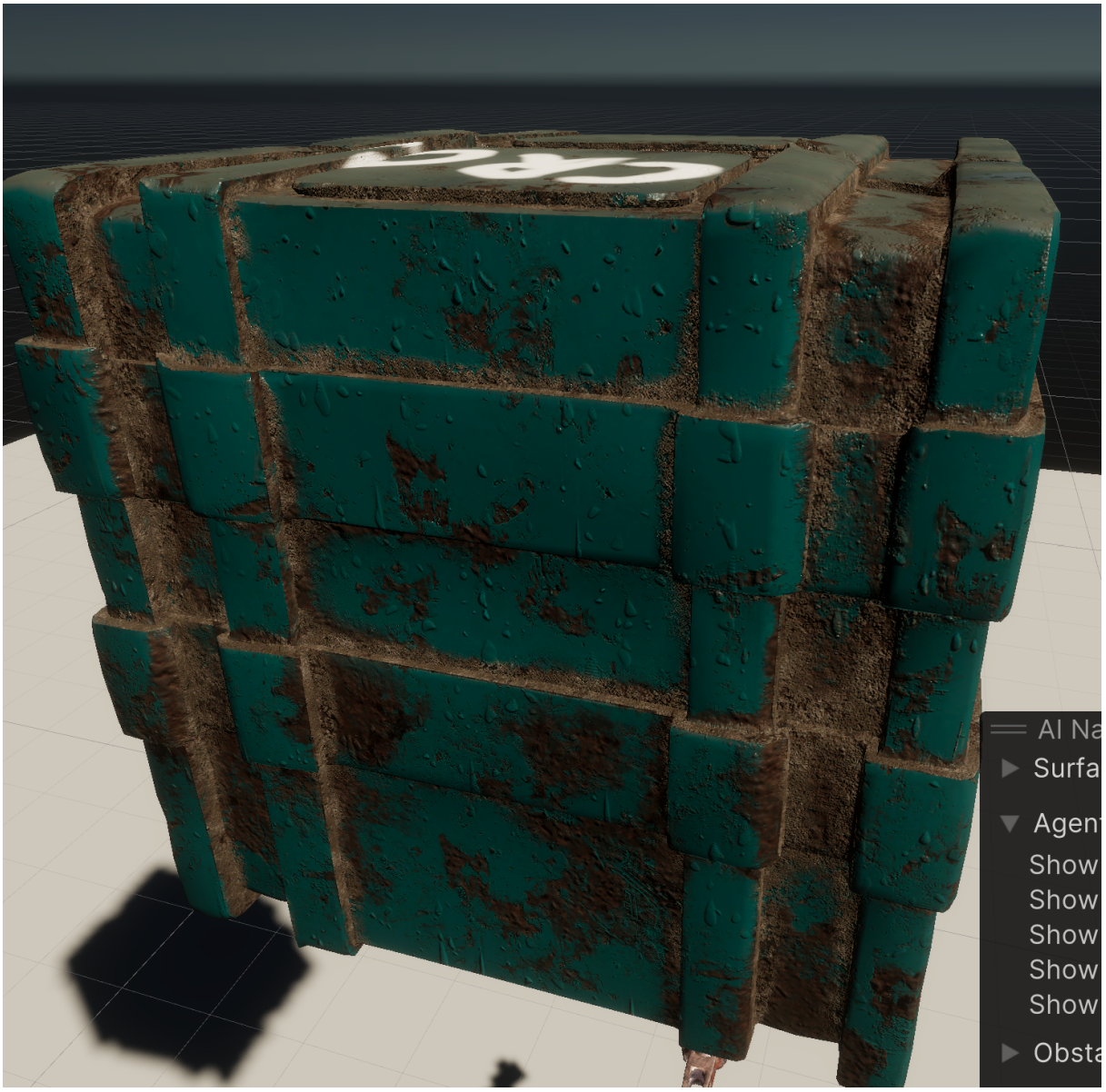
The Harvest model shows the basic surface robot worker.



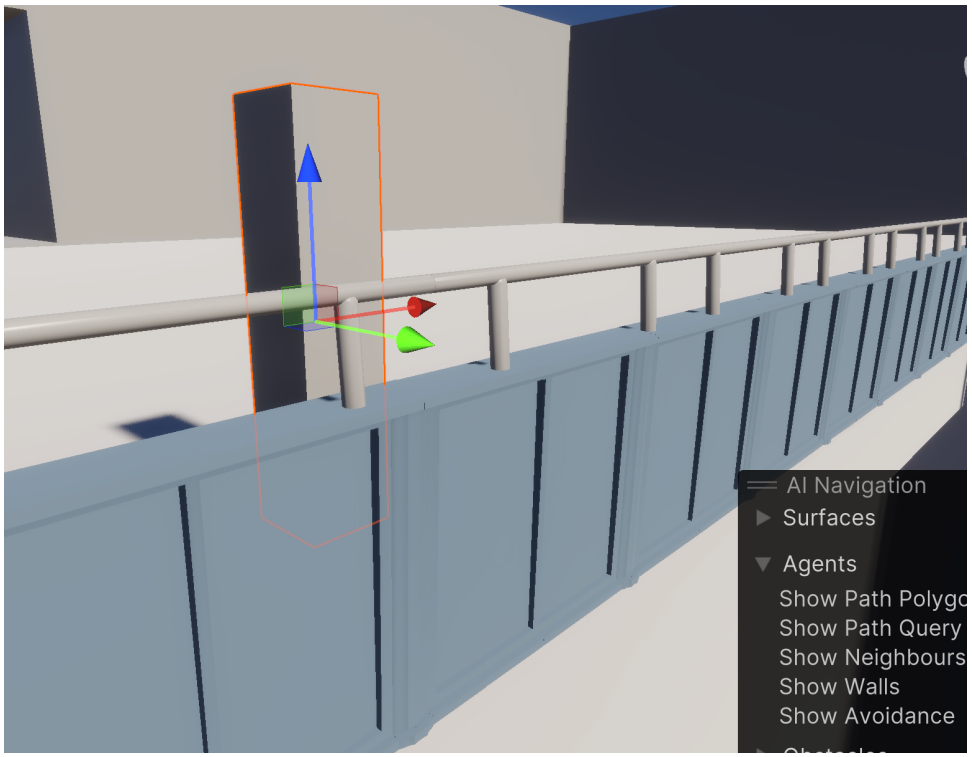


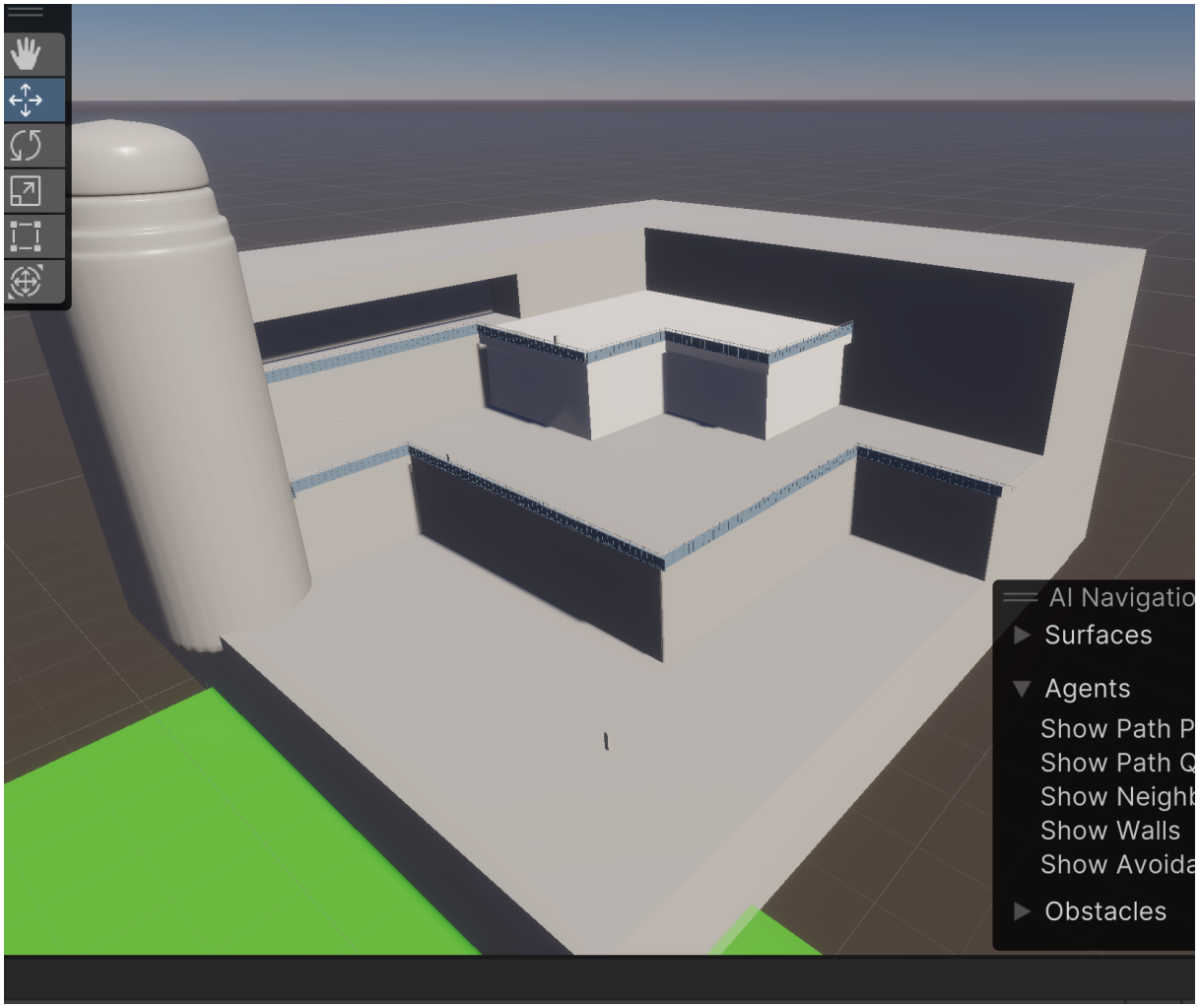
Crate

The box the harvesters will carry.

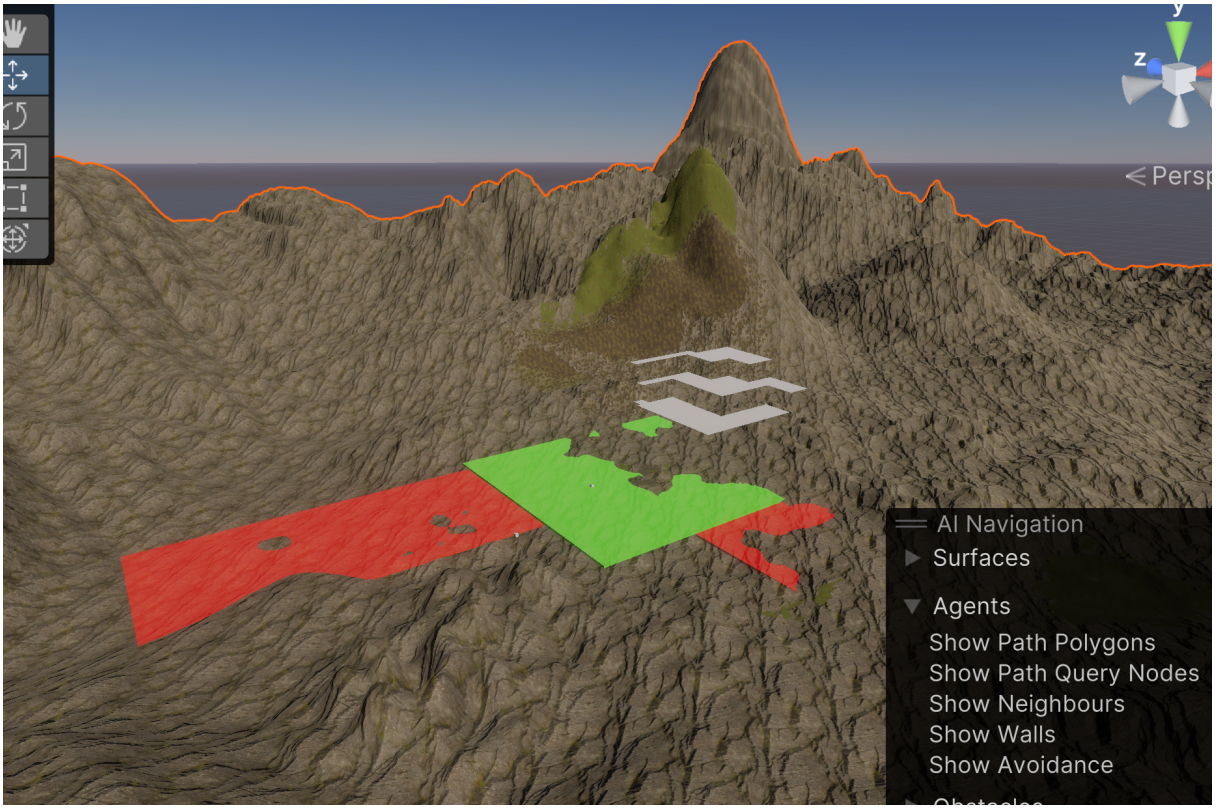


Bunker (First draft)





Terrain (Layout)



Haorui

Because of some troubles, I am lagging behind. I have completed some of the following content:

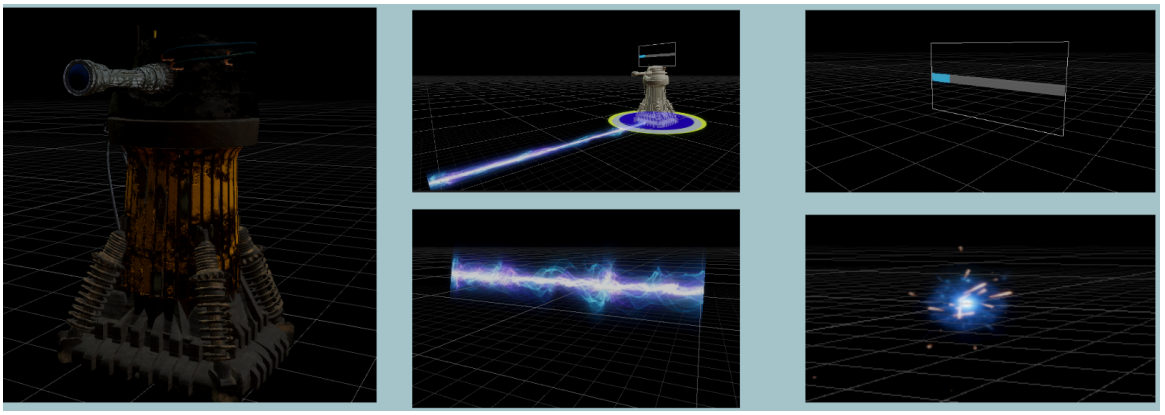
-After getting the model of the laser tower:

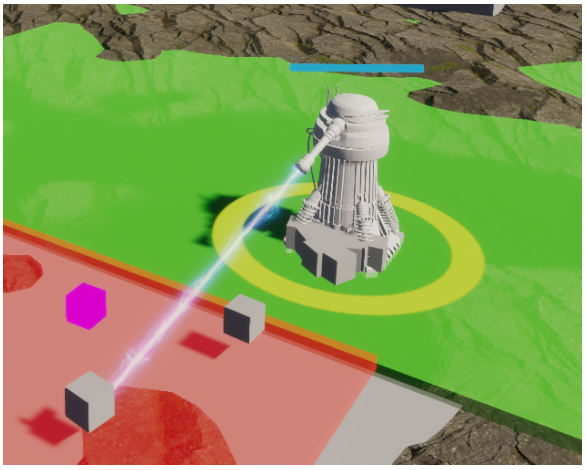
-Laser tower animation

-Top rotation correction

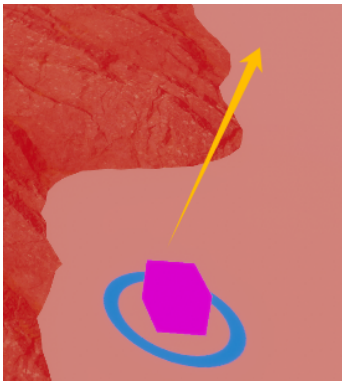
-Cooldown time after shooting

-The effect after the enemy is shot

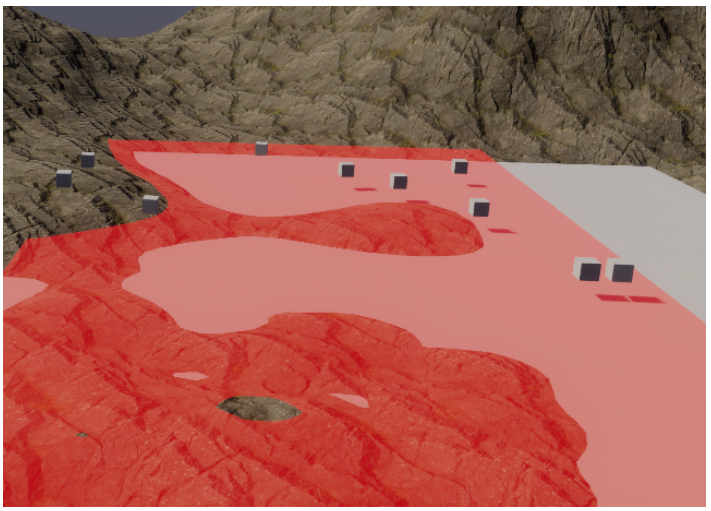




-Initialization of the remaining three defense tower scripts
-Initialization of robot scripts
	-Click to select the robot and select a location to dispatch it



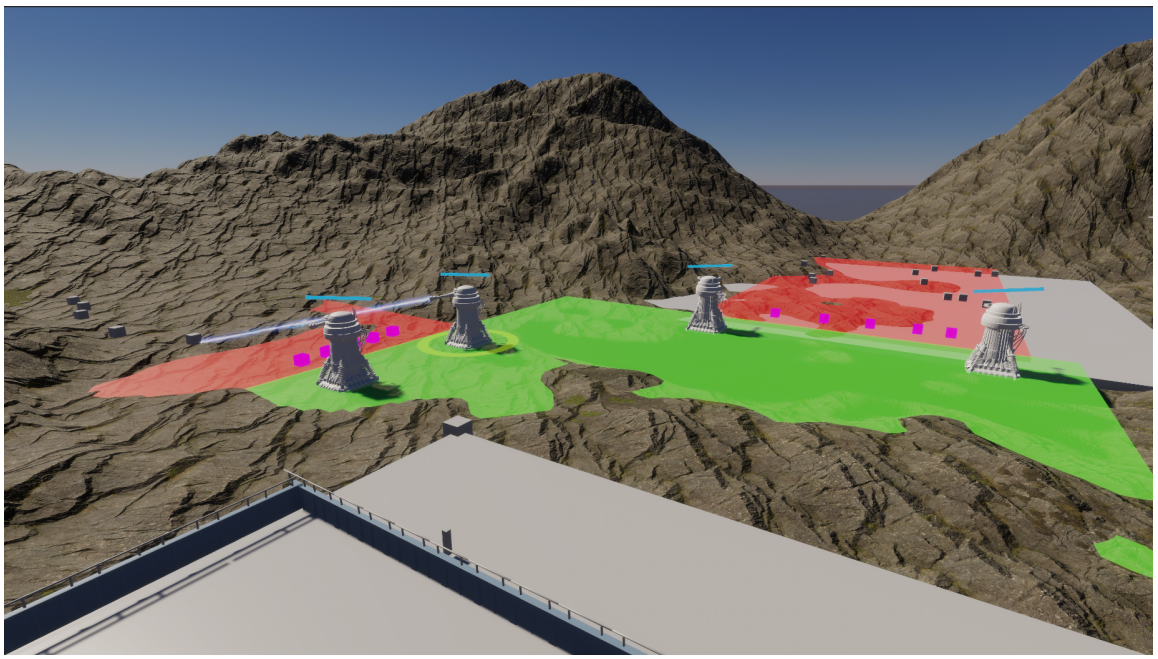
-Initialization of enemy scripts
	-Enemies automatically spawn from both sides of the map and move towards the base



-When encountering a robot, it automatically stops -> enters battle

Progress in Pictures

-Battle scene:



What we still have to implement:

These are the things we will work on next for the game:

- Switching between surface and underground.
- Complete the functions of the remaining defense towers.
- Complete the logic of the remaining robots and enemies
- Complete and polish the construction system
- Add more kinds of underground buildings
- Add more people's animations and interactions behaviors
- Complete the underground basement operation logic
- let the people have their own resupply logic when released from work, or they can automatically operate the basement
- Combine voltage system with tower functionality
- more robots

Updated Timeline

Task	Estimated Time in hours	Time took	importance	Developer	comment
basic Map and buildings model	20	30	1	Matija	
Camera Control for both Surface, Bunker	20	30	1	Matija	Including Switch button between bunker surface, using Unity's Input system
construction system	20		1	Hongbo	let the player place wanted buildings to certain location
Menu for start save	10	10	1	Haorui	
workers selection and deployment	20		1	Hongbo	check workers status and send them to the place to work
Elevator based navigation system	30		1	Hongbo	Elevator only connect layers of underground layers
underground basement operation logic	60		2	Hongbo	What buildings produce what resource / need what resource to operate / how many people can be plugged in the buildings etc
Adding Details to map	30	skipped	2	Matija	
Models for surface buildings	40	skipped	2	Matija	
Resource tab	10		2	Haorui	
advanced Visuals with custom shaders	40	skipped	2	Matija	
Win Lose condition	10	skipped	2	Haorui	
Animations for surface buildings	20	25	3	Haorui	
Animations for underground buildings(prefab)	40	skipped	3	Matija	Making cool animations with Inverse Kinematics
building construction menu	30		3	Hongbo	make it easier for player to interact
MLagents based robo-spider walking	30	60	3	Matija	
Models for details like junk	20		3	Matija	
Models for underground buildings	30		3	Matija	
Music	5	5	3	Haorui	
People's behaviour logic when resting	20		3	Hongbo	People go to kitchen & garden & hospital by themselves when released from jobs
Saving the current game	10	skipped	3	Haorui	saving whole state of every object, saving it to database or file, load it
Sound effects	20		3	Matija	Sounds of machienes when zooming in, Light having buzzing etc.
Tower defense attack math	30	Not all completed	3	Haorui	
worker generation system/birth logic	40		3	Hongbo	child will inherite parents' outlook / maybe some sort of clone device which only needs to put a couple into the container then it can generate a clone kid in a second
Animations for Cursor	5		4	Matija	
Animations for Buildings, Cursor and people	60		4	Matija	work, walk, sit, lie down, get hurt, injured walk, chat | Take from Aasetstore
Complicated enemy behaviour	60		4	Matija	
People animation when resting(chat, play instrument)	20	skipped	4	Haorui	
GPT driven underground Management System	100		4	Hongbo	Only if have time
Dynamic 3D sound			5		
Having Story Modus with Missions			5		
More enemies			5		
Multiplayer in some way			5		
people's animation control logic	10		1	Hongbo	the logic used to switch different animations of the people
Electricity Voltage System	10		2	Hongbo	logic for electricity gain and consume and effects