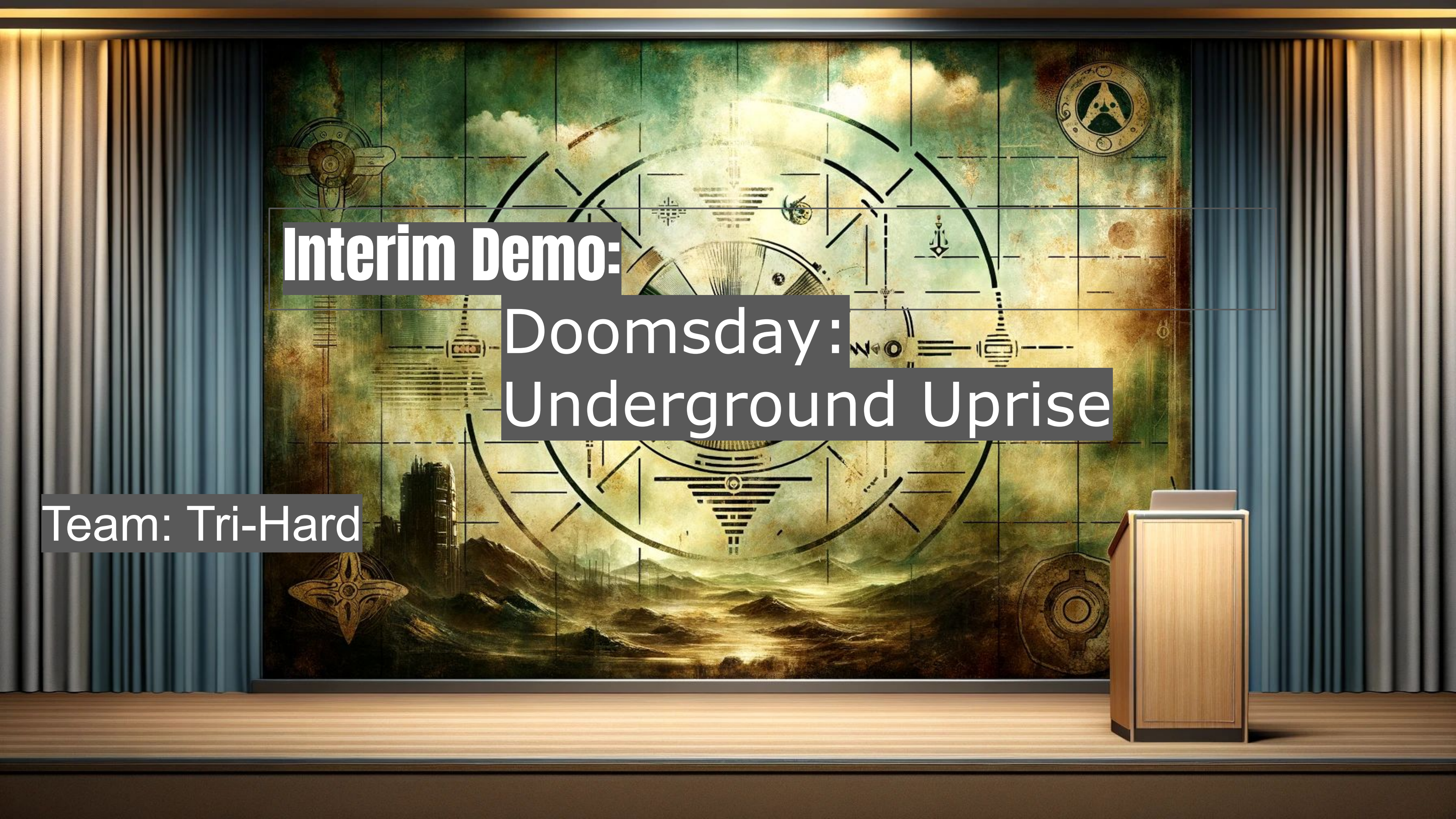
Interim Demo:

Doomsday: Underground Uprise

Team: Tri-Hard

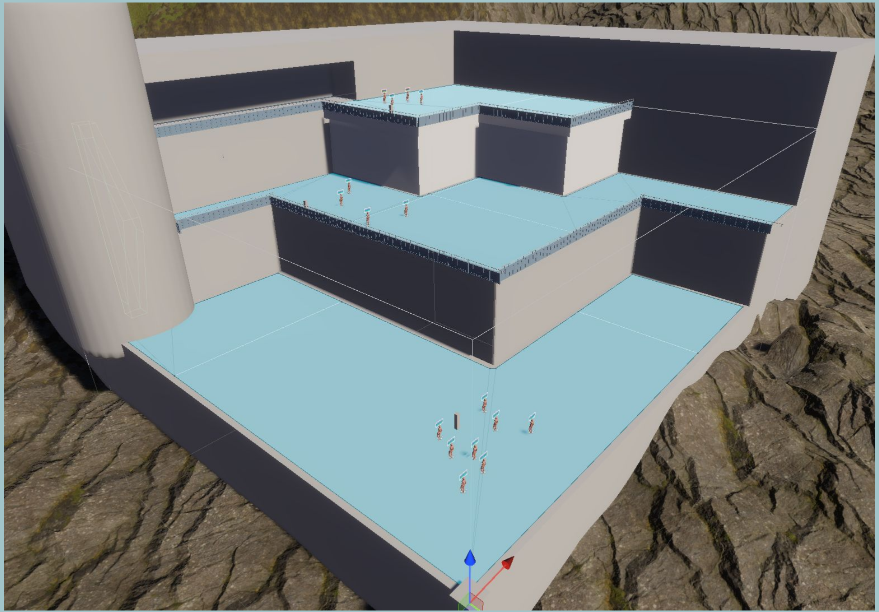


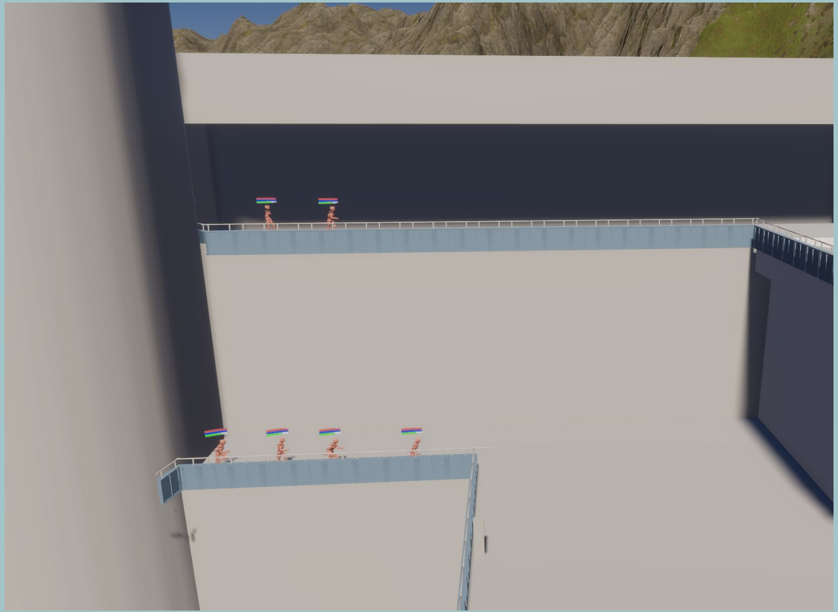
Remember? Gameplay

- build your bunker and surface buildings
- assign people to it, or build robots to operate on surface
- surface absolute essential to survive but attack regularly by enemies

=> employ Tower Defense and Defend Bots

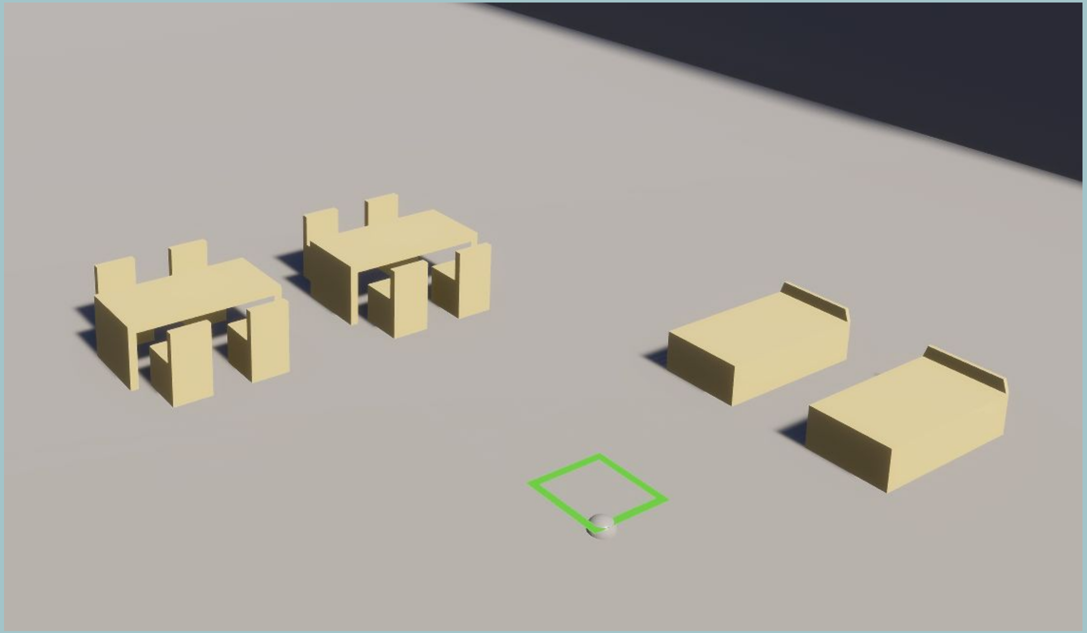
Underground citizens’ navigation system with elevator logic



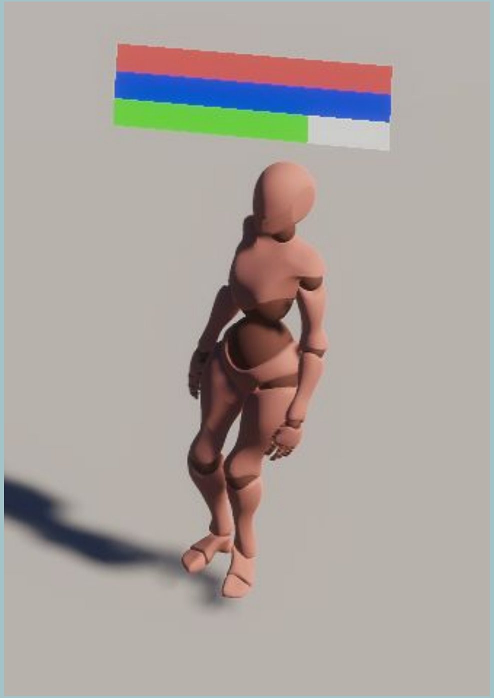


underground basement building system

grids based construction

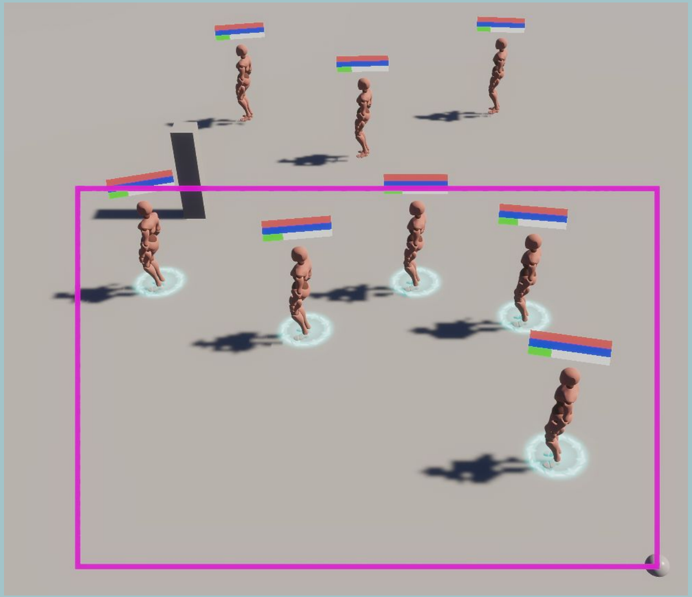


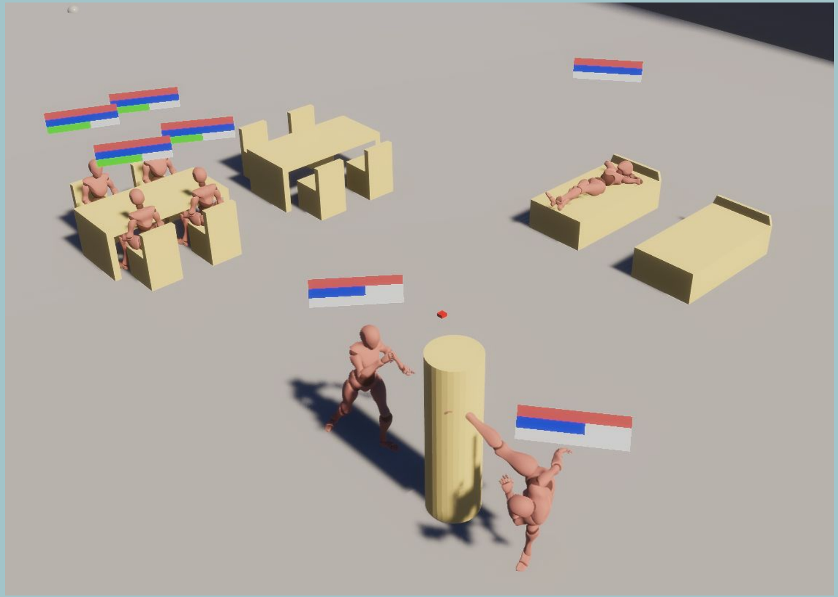
People’s status bar



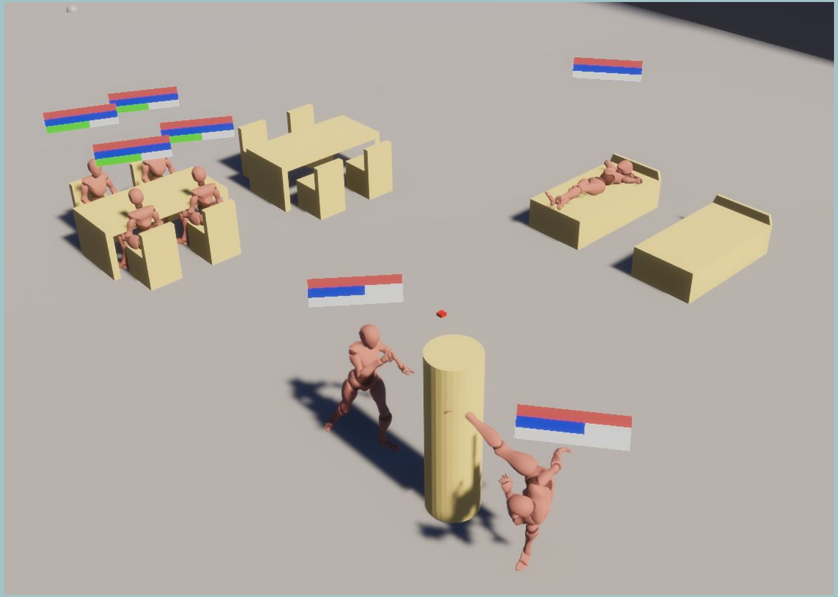
Red:	Health Bar
Blue:	Energy Bar
Green:	Hunger Bar

people selection box and interaction between buildings and people





people selection box and interaction between buildings and people

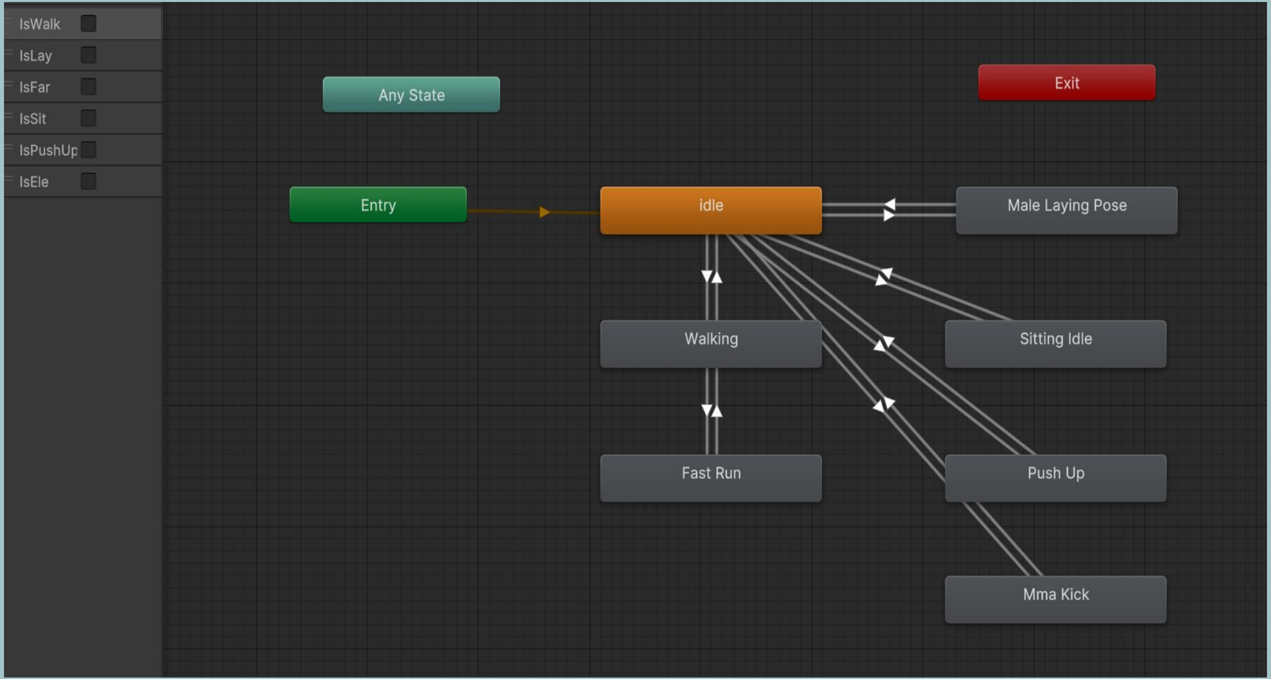


Place the facilities may be funny

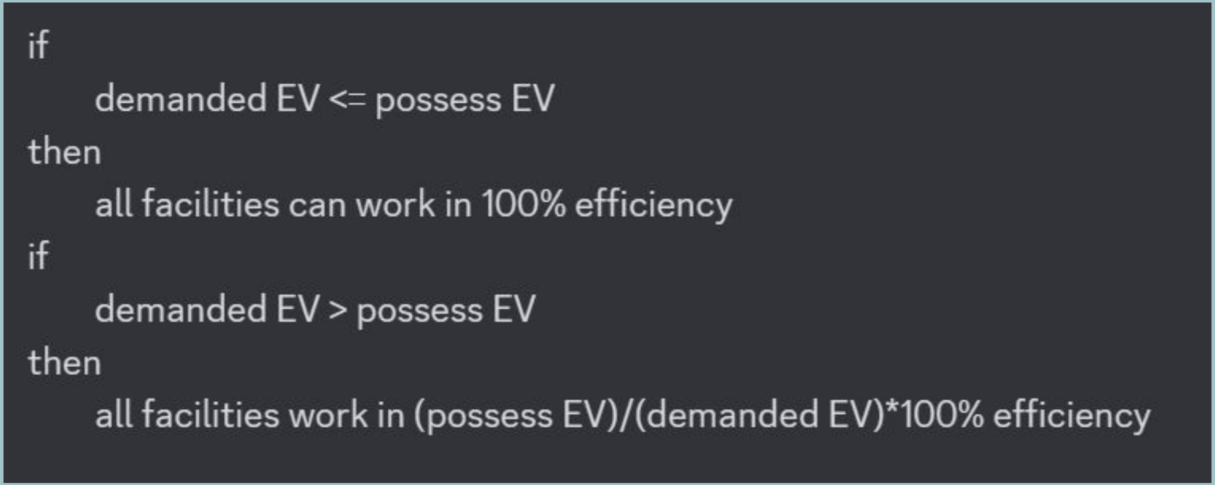
But doing all the deployments manually consumes a lot of attention and isn’t funny

The people should resupply themselves automatically

animation control logic



Electricity Voltage



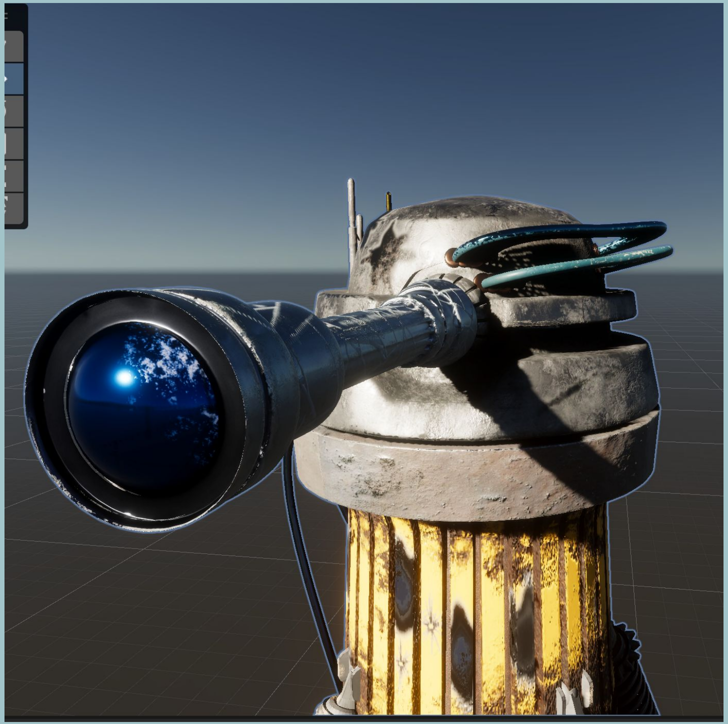
```
if
    demanded EV <= possess EV
then
    all facilities can work in 100% efficiency
if
    demanded EV > possess EV
then
    all facilities work in (possess EV)/(demanded EV)*100% efficiency
```

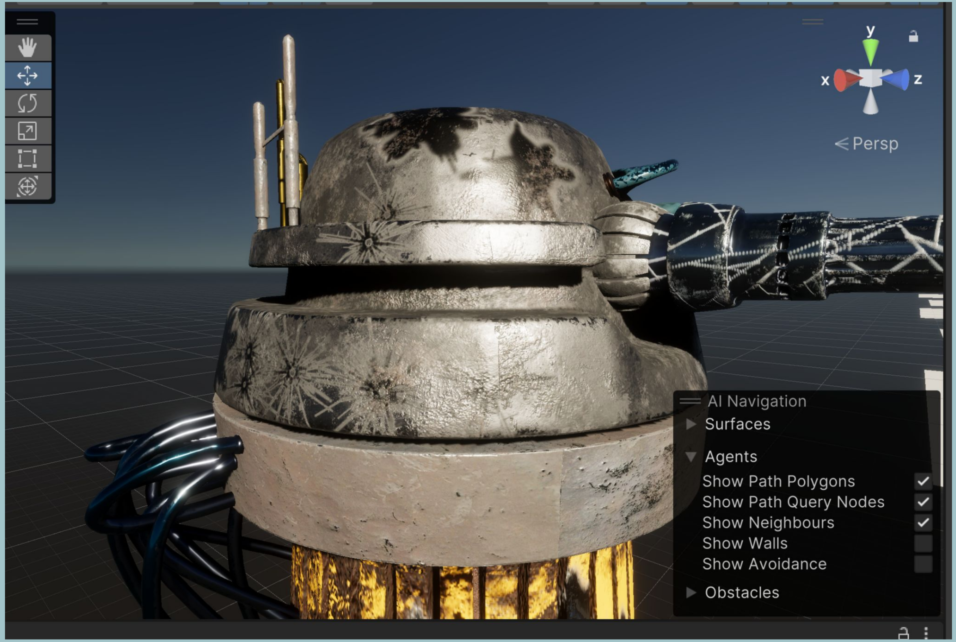
First steps: 3D Asset Creation

Highly detailed Assets bc of low object count

- full control over parts, style, polycount
- worth the time to make unique game
- took me a lot of time though …
- used Substance Painter and Blender 4.0

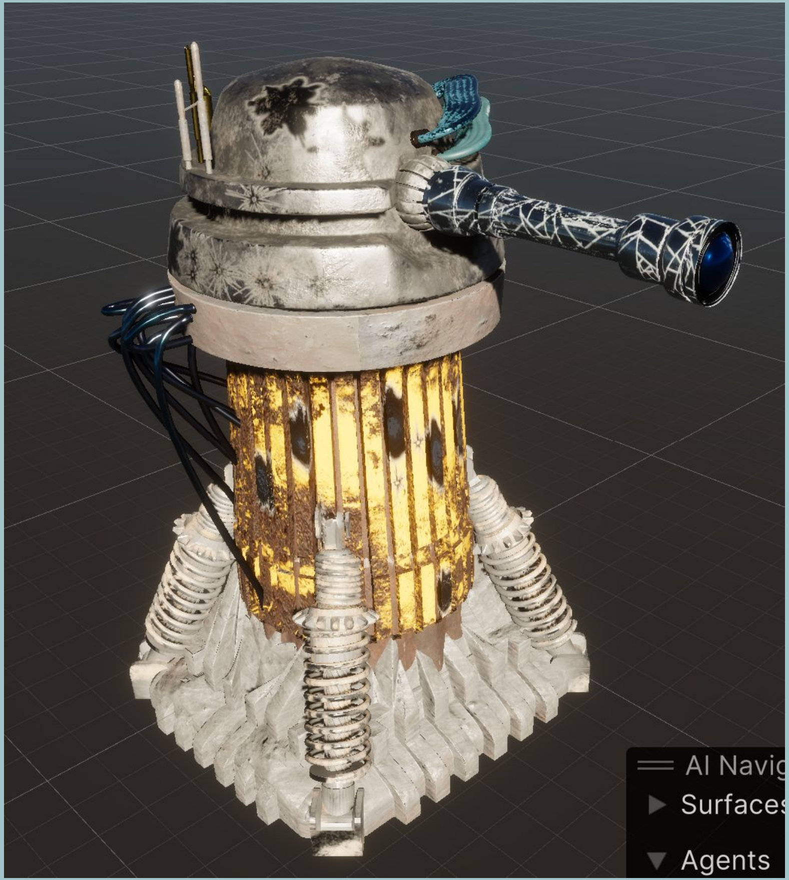
Tower



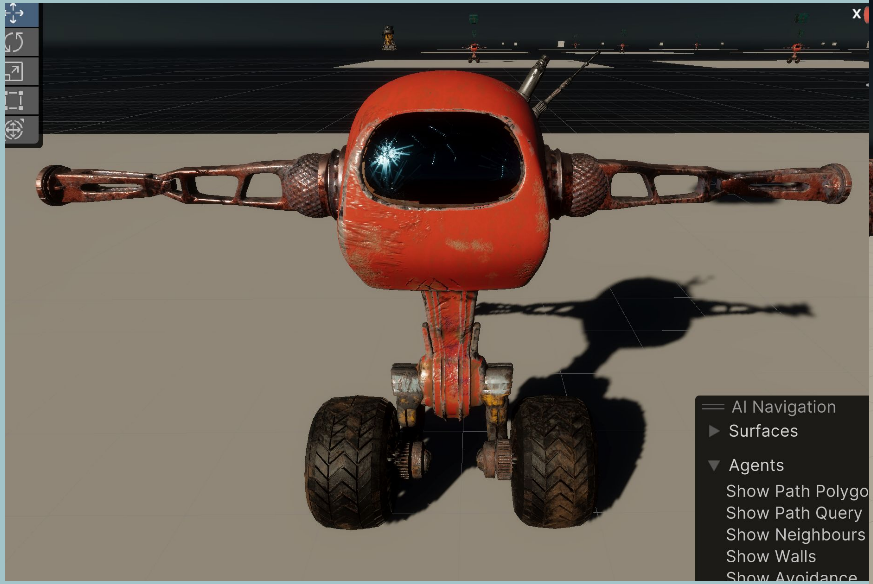


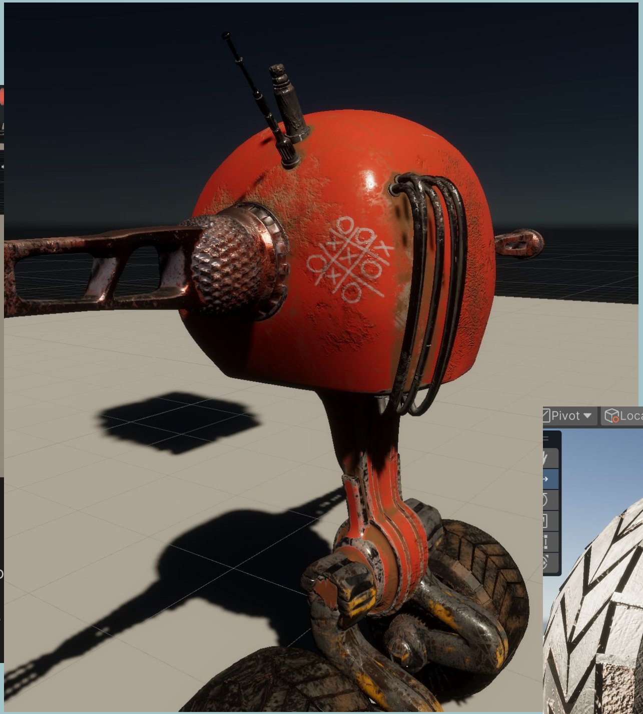
AI Navigation
- Surfaces
- Agents
  - Show Path Polygons ✓
  - Show Path Query Nodes ✓
  - Show Neighbours ✓
  - Show Walls
  - Show Avoidance
- Obstacles

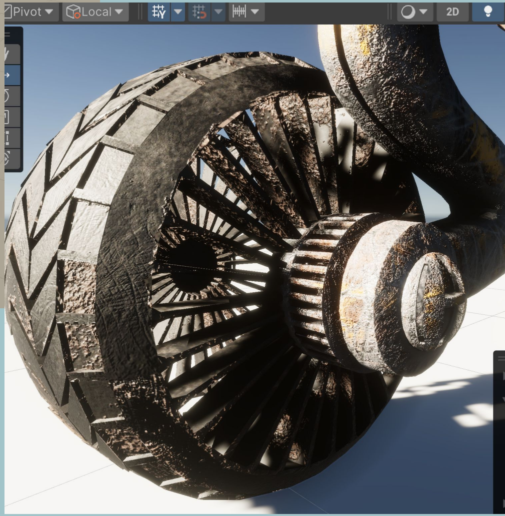
Tower



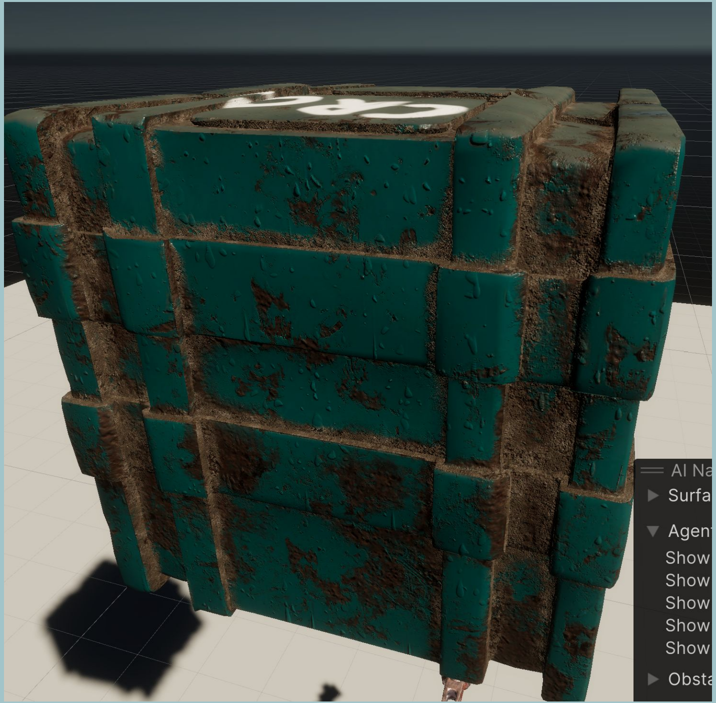
Harvester


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

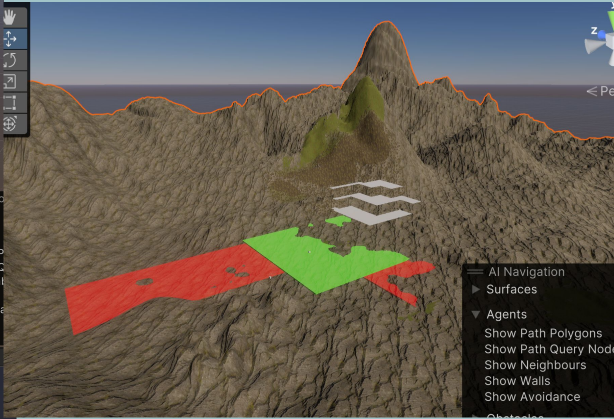


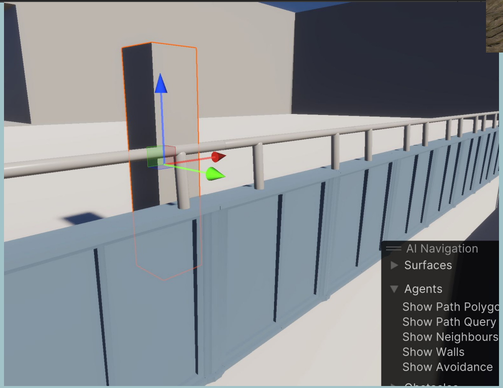

Pivot
Local
2D

Other


AI Na
Surfa
Agen
Show
Show
Show
Show
Show
Obsta

AI Navigatio
Surfaces
Agents
Show Path P
Show Path Q
Show Neigh
Show Walls
Show Avoid
Obstacles


AI Navigation
Surfaces
Agents
Show Path Polygons
Show Path Query Node
Show Neighbours
Show Walls
Show Avoidance


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

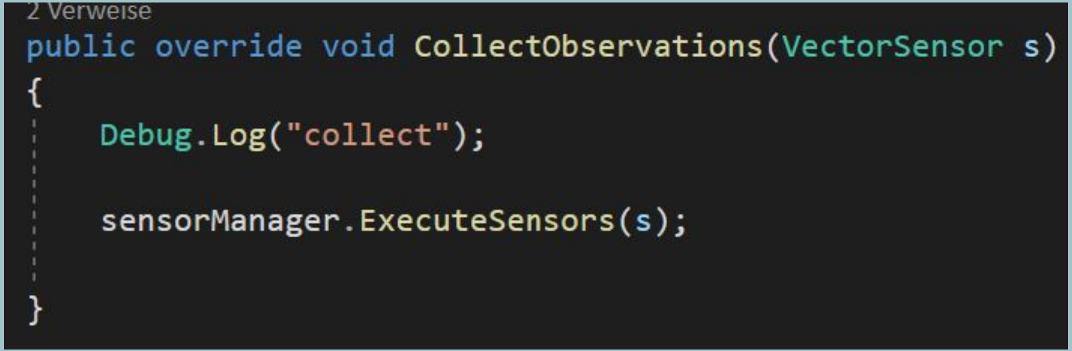
Mlagents

- Last Semester experience was very bad using plane MLagents -> too explicit code, hard to setup new things
- tried to abstract away as much as possibe

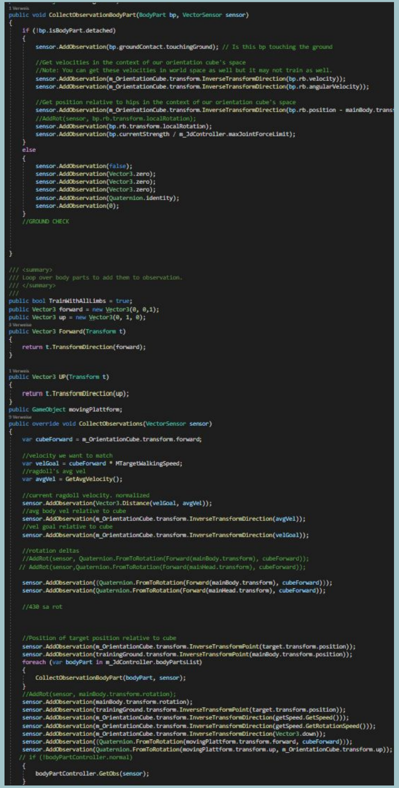
My Mlagents Abstractions

- **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

E.g. Collectobservations, got better, right?

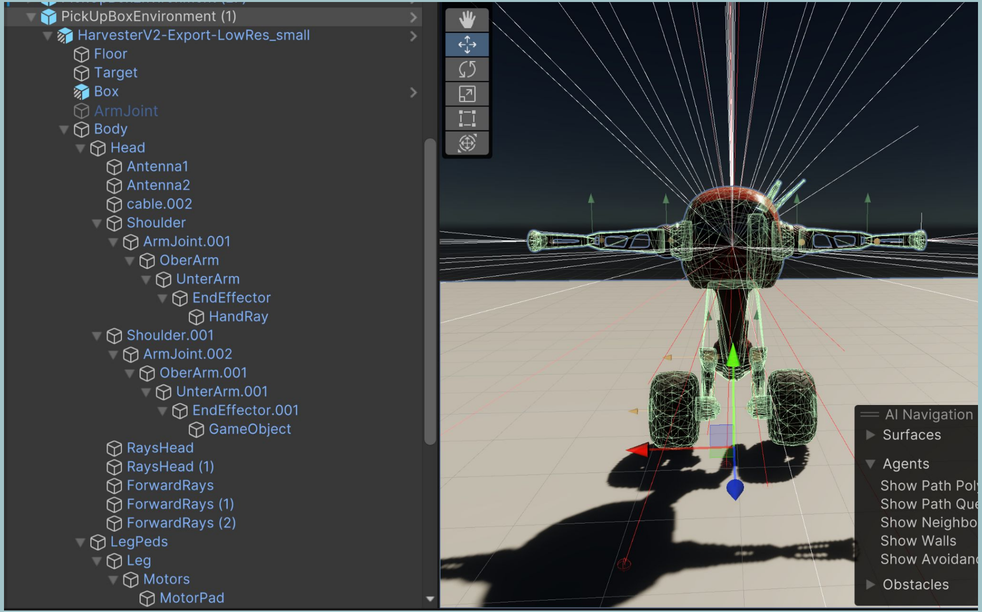


```
2 Verweise
public override void CollectObservations(VectorSensor s)
{
    Debug.Log("collect");

    sensorManager.ExecuteSensors(s);

}
```

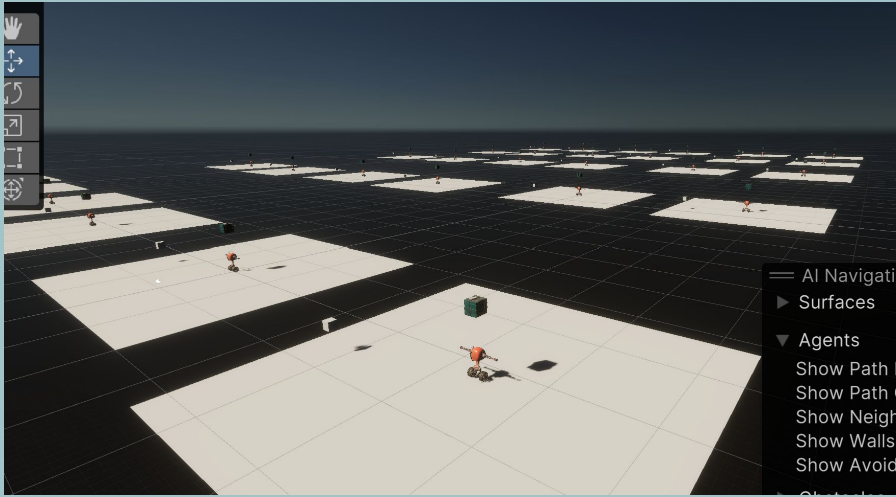


```
1 Verweis
public void CollectObservationBodyPart(BodyPart bp, VectorSensor sensor)
{
    if (!bp.isBodyPart.detached)
    {
        sensor.AddObservation(bp.groundContact.touchingGround); // Is this bp touching the ground

        //Get velocities in the context of our orientation cube's space
        //Note: You can get these velocities in world space as well but it may not train as well.
        sensor.AddObservation(m_OrientationCube.transform.InverseTransformDirection(bp.rb.velocity));
        sensor.AddObservation(m_OrientationCube.transform.InverseTransformDirection(bp.rb.angularVelocity));

        //Get position relative to hips in the context of our orientation cube's space
        sensor.AddObservation(m_OrientationCube.transform.InverseTransformDirection(bp.rb.position - mainBody.trans
        //AddRot(sensor, bp.rb.transform.localRotation);
        sensor.AddObservation(bp.rb.transform.localRotation);
        sensor.AddObservation(bp.currentStrength / m_JdController.maxJointForceLimit);
    }
    else
    {
        sensor.AddObservation(false);
        sensor.AddObservation(Vector3.zero);
        sensor.AddObservation(Vector3.zero);
        sensor.AddObservation(Vector3.zero);
        sensor.AddObservation(Quaternion.identity);
        sensor.AddObservation(0);
    }
    //GROUND CHECK



}

/// <summary>
/// Loop over body parts to add them to observation.
/// </summary>
///
public bool TrainWithAllLimbs = true;
public Vector3 forward = new Vector3(0, 0,1);
public Vector3 up = new Vector3(0, 1, 0);
3 Verweise
public Vector3 Forward(Transform t)
{
    return t.TransformDirection(forward);
}

1 Verweis
public Vector3 UP(Transform t)
{
    return t.TransformDirection(up);
}
public GameObject movingPlattform;
9 Verweise
public override void CollectObservations(VectorSensor sensor)
{
    var cubeForward = m_OrientationCube.transform.forward;

    //velocity we want to match
    var velGoal = cubeForward * MTargetWalkingSpeed;
    //ragdoll's avg vel
    var avgVel = GetAvgVelocity();

    //current ragdoll velocity. normalized
    sensor.AddObservation(Vector3.Distance(velGoal, avgVel));
    //avg body vel relative to cube
    sensor.AddObservation(m_OrientationCube.transform.InverseTransformDirection(avgVel));
    //vel goal relative to cube
    sensor.AddObservation(m_OrientationCube.transform.InverseTransformDirection(velGoal));

    //rotation deltas
    //AddRot(sensor, Quaternion.FromToRotation(Forward(mainBody.transform), cubeForward));
   // AddRot(sensor,Quaternion.FromToRotation(Forward(mainHead.transform), cubeForward));

    sensor.AddObservation((Quaternion.FromToRotation(Forward(mainBody.transform), cubeForward)));
    sensor.AddObservation(Quaternion.FromToRotation(Forward(mainHead.transform), cubeForward));

    //430 sa rot



    //Position of target position relative to cube
    sensor.AddObservation(m_OrientationCube.transform.InverseTransformPoint(target.transform.position));
    sensor.AddObservation(trainingGround.transform.InverseTransformPoint(mainBody.transform.position));
    foreach (var bodyPart in m_JdController.bodyPartsList)
    {
        CollectObservationBodyPart(bodyPart, sensor);
    }
    //AddRot(sensor, mainBody.transform.rotation);
    sensor.AddObservation(mainBody.transform.rotation);
    sensor.AddObservation(trainingGround.transform.InverseTransformPoint(target.transform.position));
    sensor.AddObservation(m_OrientationCube.transform.InverseTransformDirection(getSpeed.GetSpeed()));
    sensor.AddObservation(m_OrientationCube.transform.InverseTransformDirection(getSpeed.GetRotationSpeed()));
    sensor.AddObservation(m_OrientationCube.transform.InverseTransformDirection(Vector3.down));
    sensor.AddObservation((Quaternion.FromToRotation(movingPlattform.transform.forward, cubeForward)));
    sensor.AddObservation(Quaternion.FromToRotation(movingPlattform.transform.up, m_OrientationCube.transform.up));
   // if (!bodyPartController.normal)
    {
        bodyPartController.GetObs(sensor);
    }
```

Setup

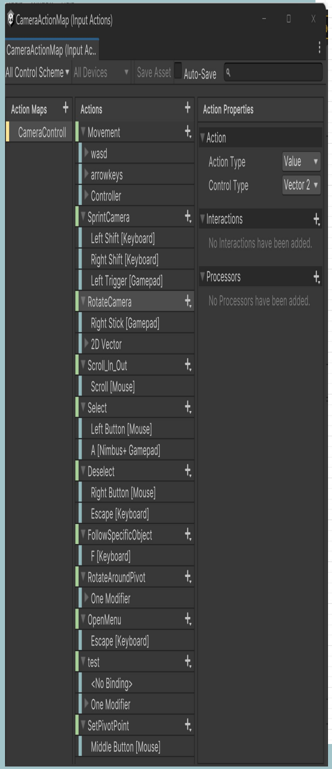


PickUpBoxEnvironment (1)
HarvesterV2-Export-LowRes_small
Floor
Target
Box
ArmJoint
Body
Head
Antenna1
Antenna2
cable.002
Shoulder
ArmJoint.001
OberArm
UnterArm
EndEffector
HandRay
Shoulder.001
ArmJoint.002
OberArm.001
UnterArm.001
EndEffector.001
GameObject
RaysHead
RaysHead (1)
ForwardRays
ForwardRays (1)
ForwardRays (2)
LegPeds
Leg
Motors
MotorPad



Camera Controller: State and Command Pattern

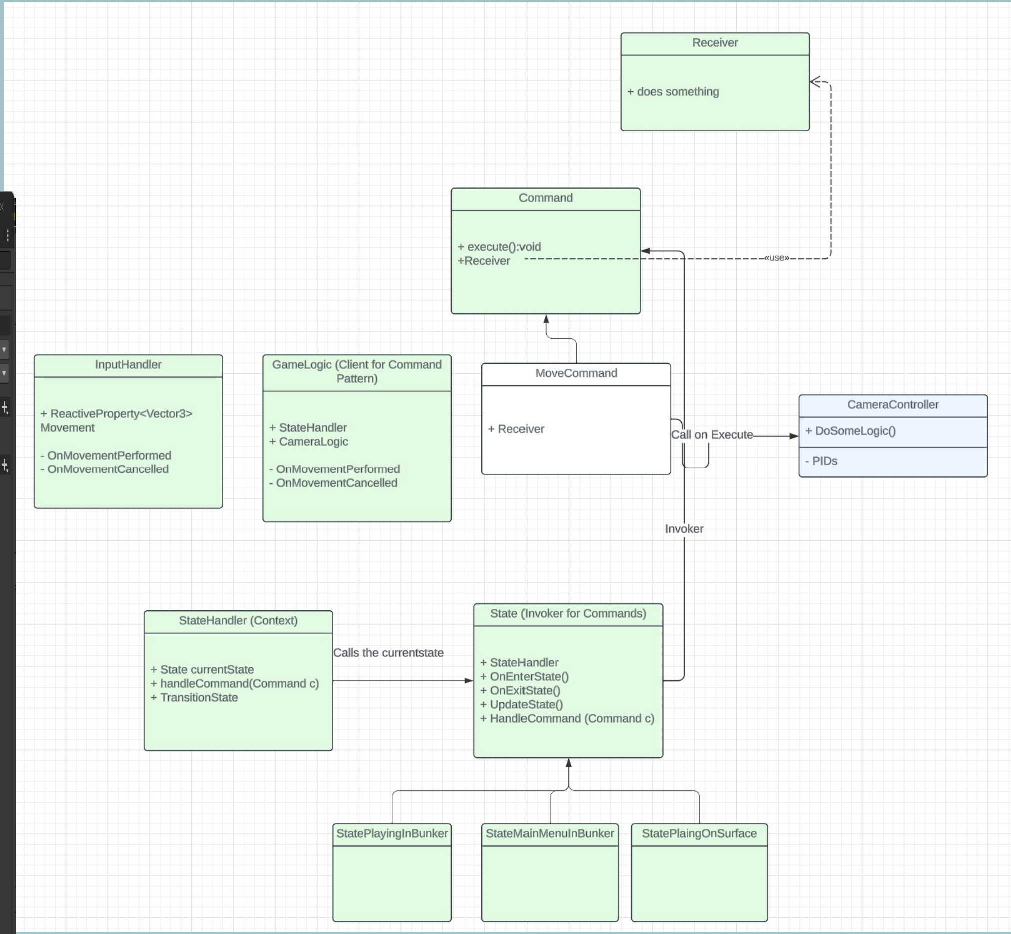
Again bad experience last year -> more patterns


CameraActionMap (Input Actions)

CameraActionMap (Input Ac..

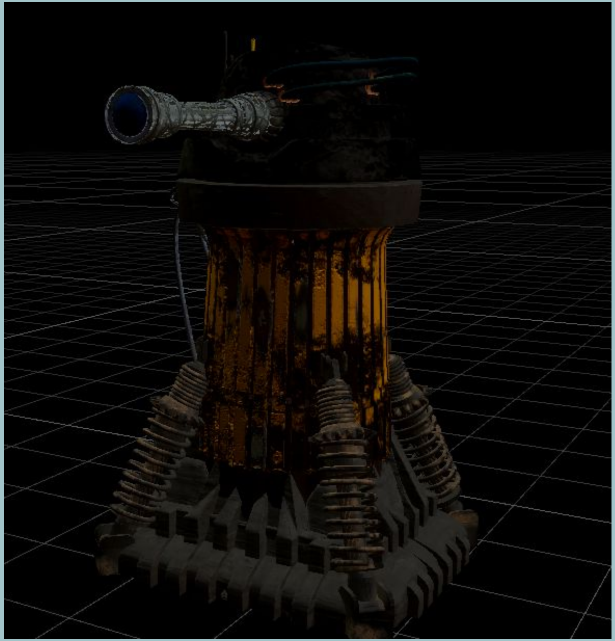
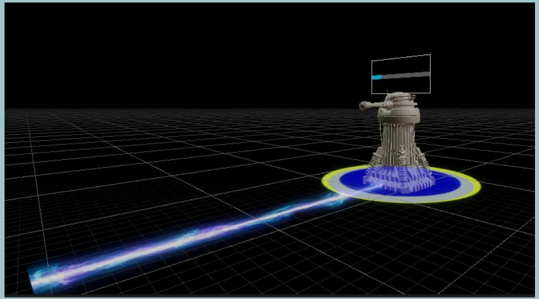
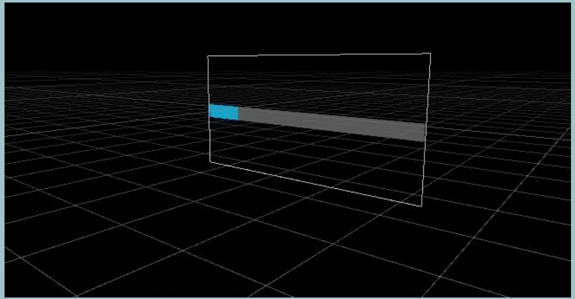
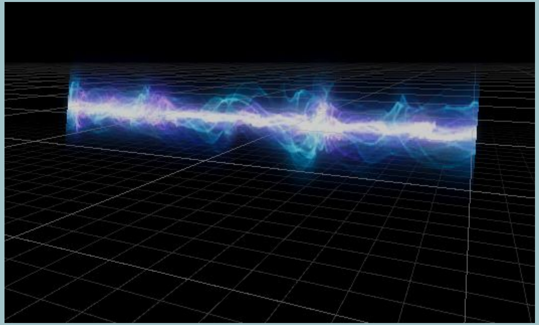
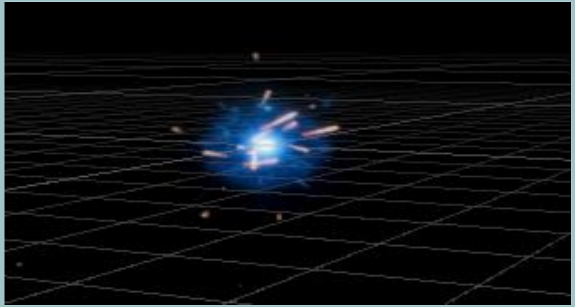
All Control Scheme ▾ All Devices ▾ Save Asset Auto-Save

Action Maps	Actions	Action Properties
CameraControll	Movement	Action
	wasd	Action Type: Value
	arrowkeys	Control Type: Vector 2
	Controller	Interactions
	SprintCamera	No Interactions have been added.
	Left Shift [Keyboard]	Processors
	Right Shift [Keyboard]	No Processors have been added.
	Left Trigger [Gamepad]	
	RotateCamera	
	Right Stick [Gamepad]	
	2D Vector	
	Scroll_In_Out	
	Scroll [Mouse]	
	Select	
	Left Button [Mouse]	
	A [Nimbus+ Gamepad]	
	Deselect	
	Right Button [Mouse]	
	Escape [Keyboard]	
	FollowSpecificObject	
	F [Keyboard]	
	RotateAroundPivot	
	One Modifier	
	OpenMenu	
	Escape [Keyboard]	
	test	
	<No Binding>	
	One Modifier	
	SetPivotPoint	
	Middle Button [Mouse]	

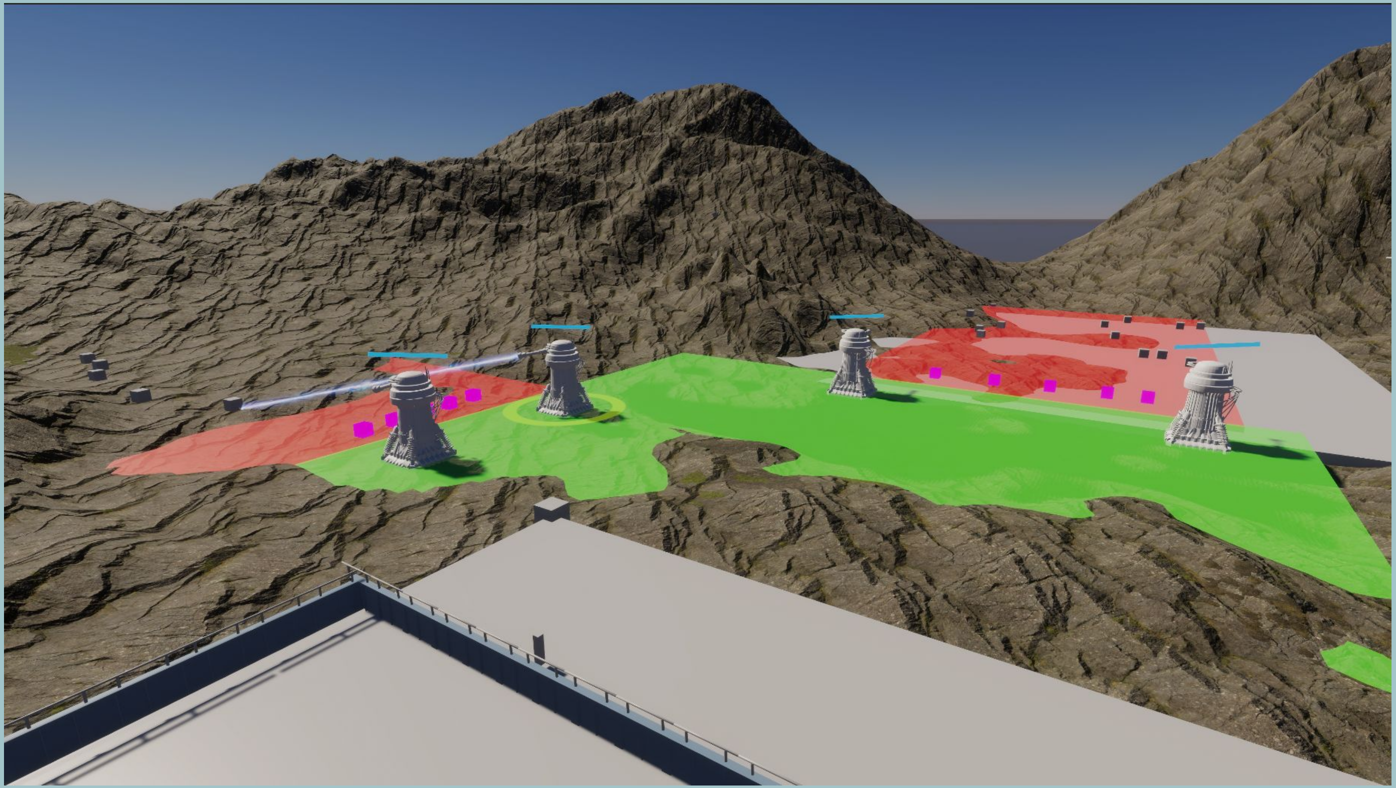


Implementation

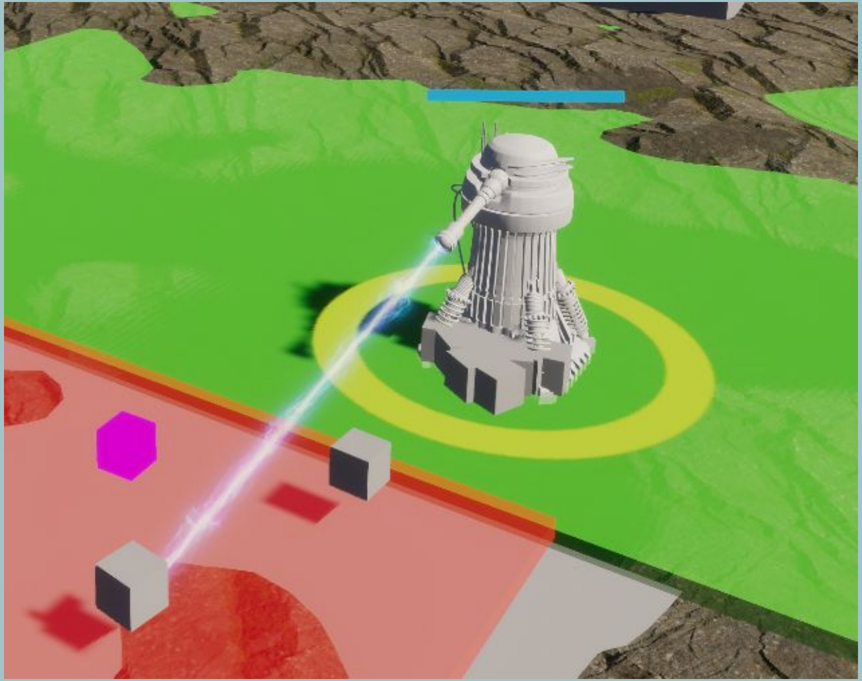
-Laser Tower

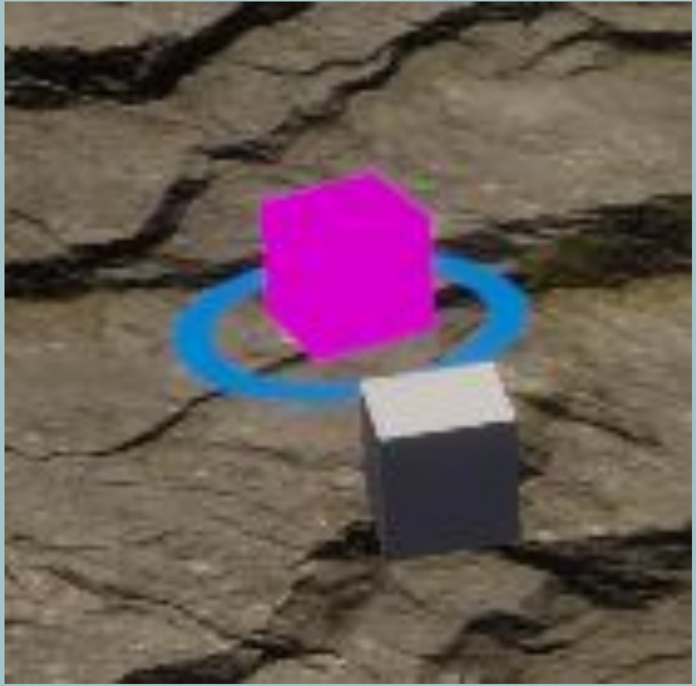






Implementation



Implementation





THANK YOU
FOR YOUR
ATTENTION!