ASSIGNMENT 1 REPORT

VR & 3D GAMES

Introduction

The concept of the game is a 3D puzzle game, with a variety of obstacles which require different abilities to overcome. The game features a character switching system, and each character has a unique ability that will allow them to overcome certain obstacles.

The theme of this puzzle game is based around drones that the player is controlling from a workstation. The workstation has multiple screens, each of which corresponds to one of the drones, and by pressing a button the player can change the drone being controlled.

How to play

Use Right Click to toggle cursor for some menus.

Use ESC to open “pause” menu which has all other controls listed.

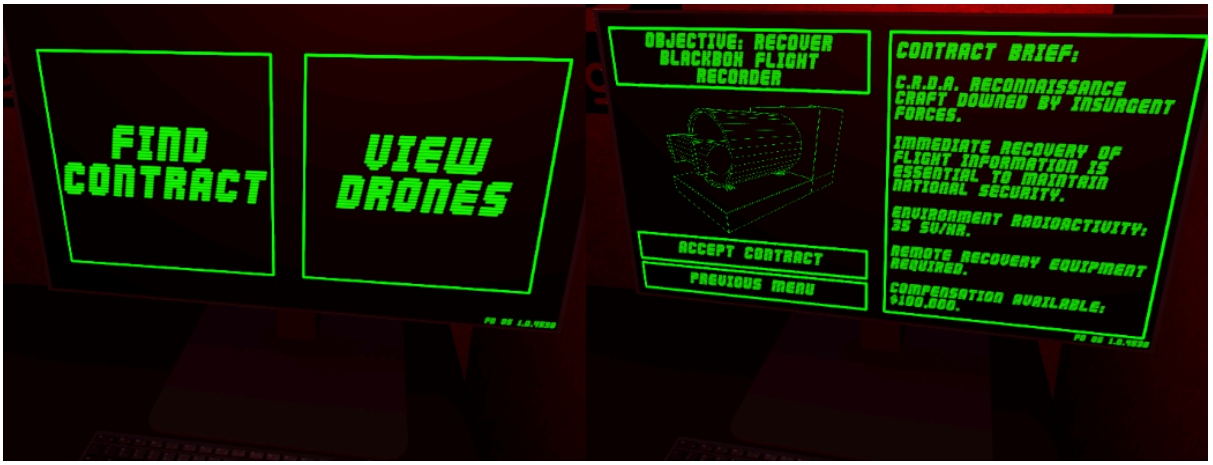
Game Menus and Instructions

Start menu

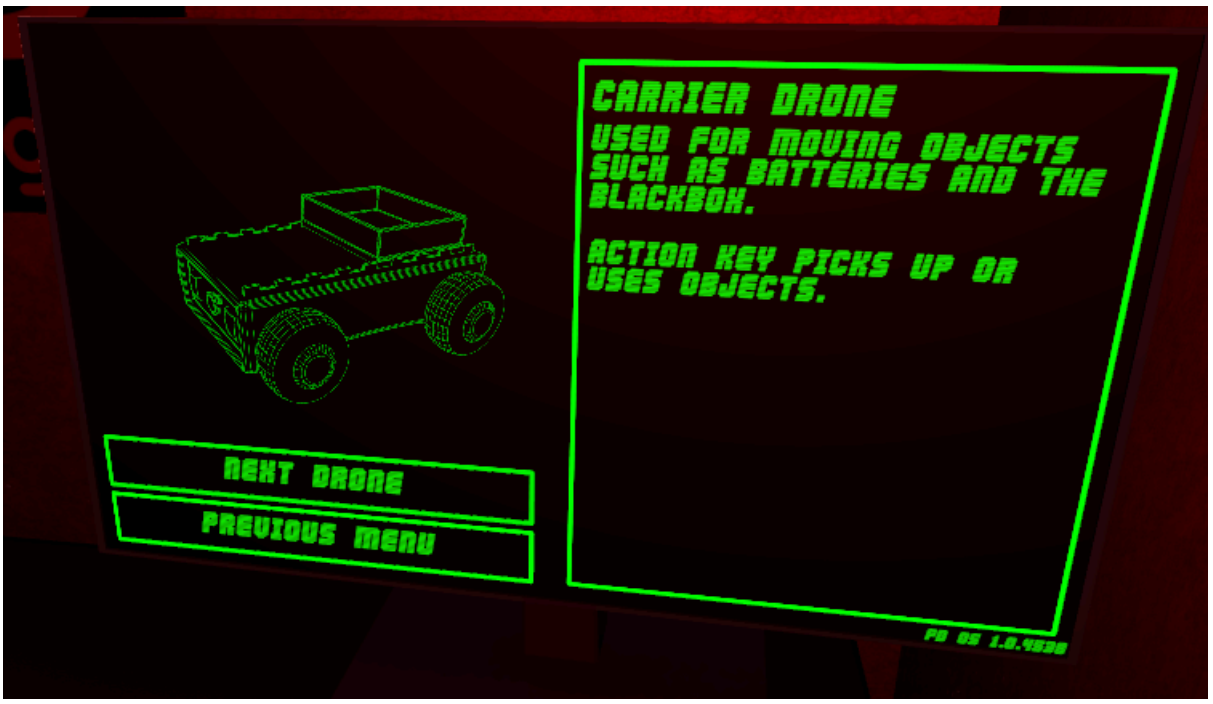
The first thing the player sees once the game starts is the start menu, where they can choose between starting the game or quitting.

3D main menu

For the 3D menu, I created a small level which the player can move around in. This area gives the player some time before starting to familiarise themselves with the theme of the game. In the level there is a screen which they can interact with and select menu options from the screen.



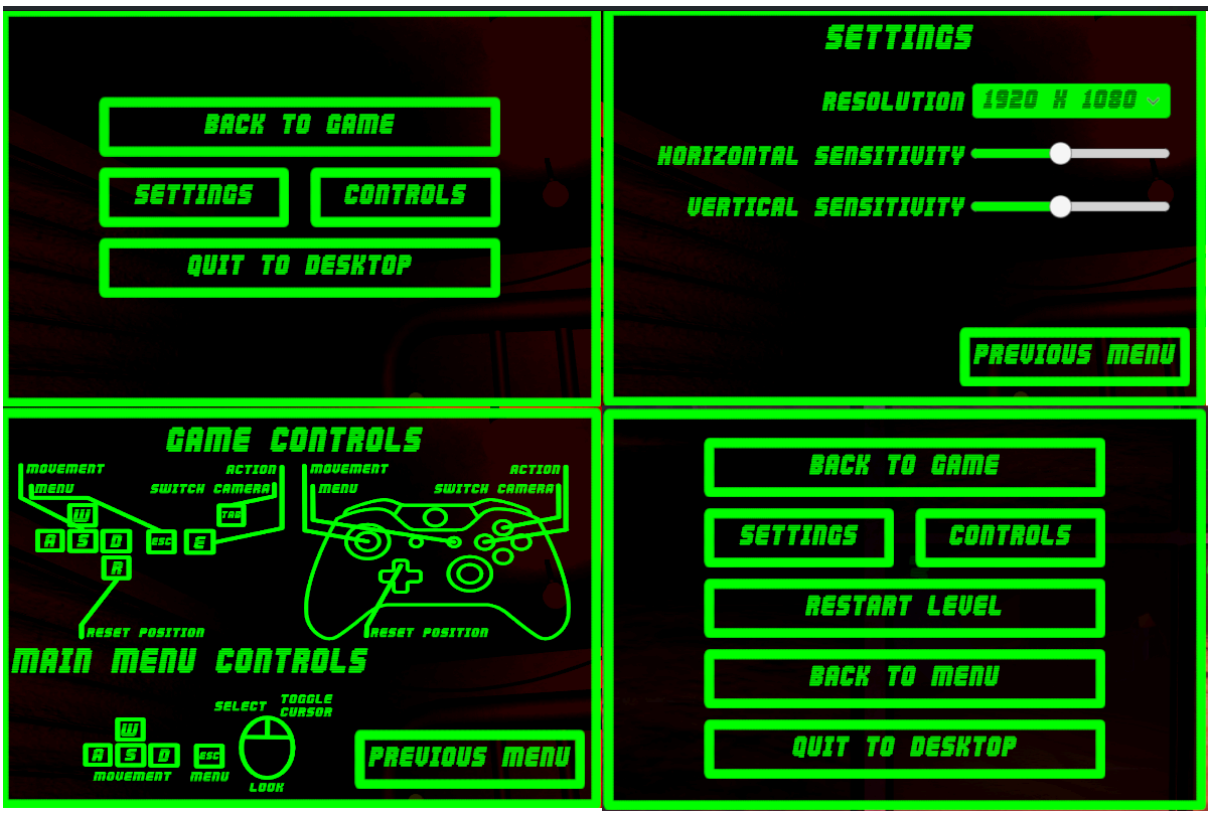
On this in-game menu, the player can select and start a mission, as well as viewing tutorial information on the various drone characters in the game.



Pause menu

An additional menu is available to the player when they press the escape button. This menu allows the player to quit the game, as well as changing some basic settings and viewing the controls.

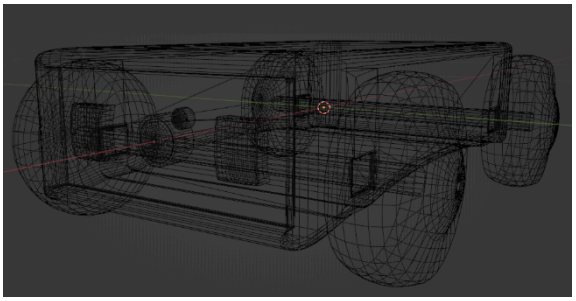
When a level is selected, a separate version of the pause menu is used, which also allows the user to reset the level in case there is an issue encountered, as well as returning back to the main menu.

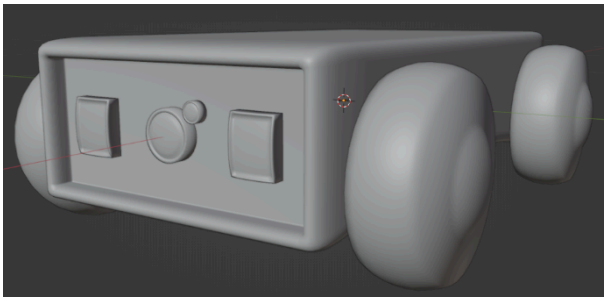


Complexity and Sophistication of Scenes and Sound

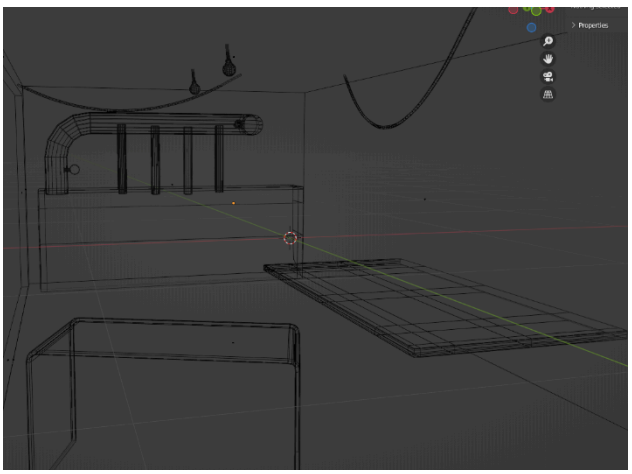
3D Modelling

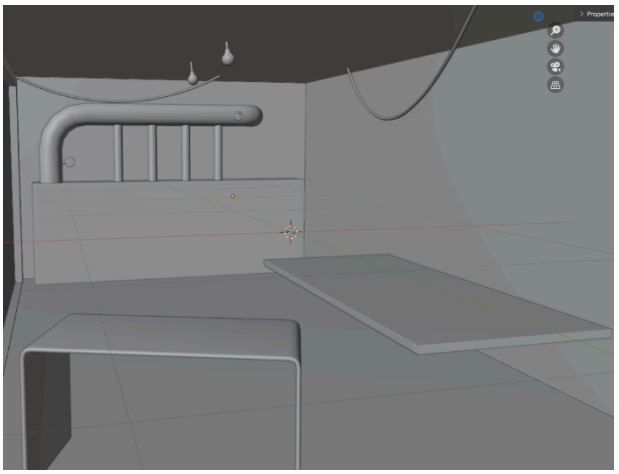
I created the models in this game using Blender. The first 3d model I worked on was for the base drone model, which will be used for the characters.





Interior space for the menu, and for the main camera and monitor station to be situated in.

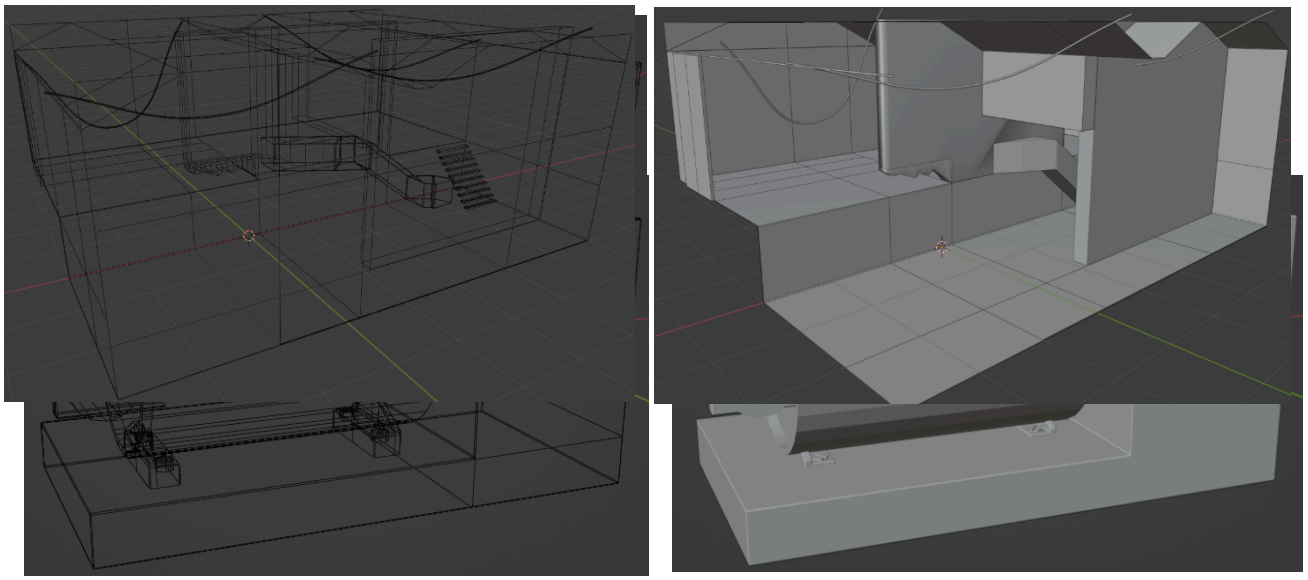




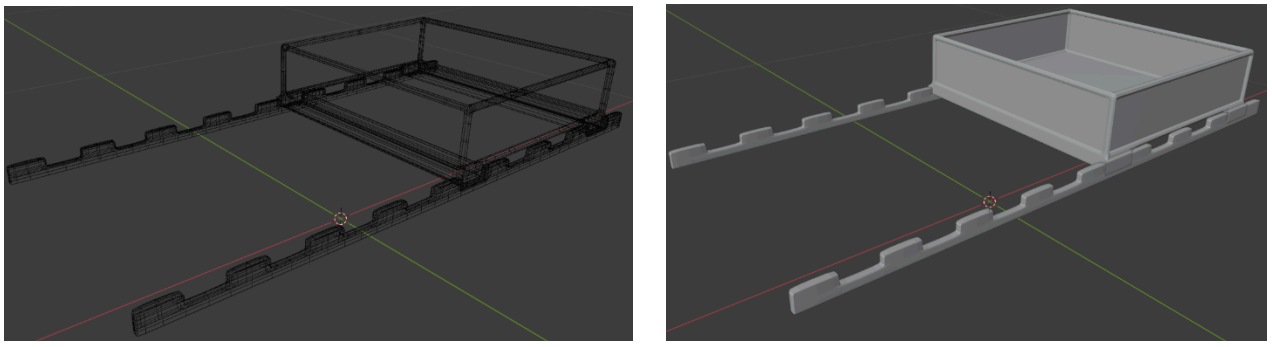
Monitor Station which the player will be viewing the drone’s cameras from.

Flight recorder black box which will be an item that is collected by the player.

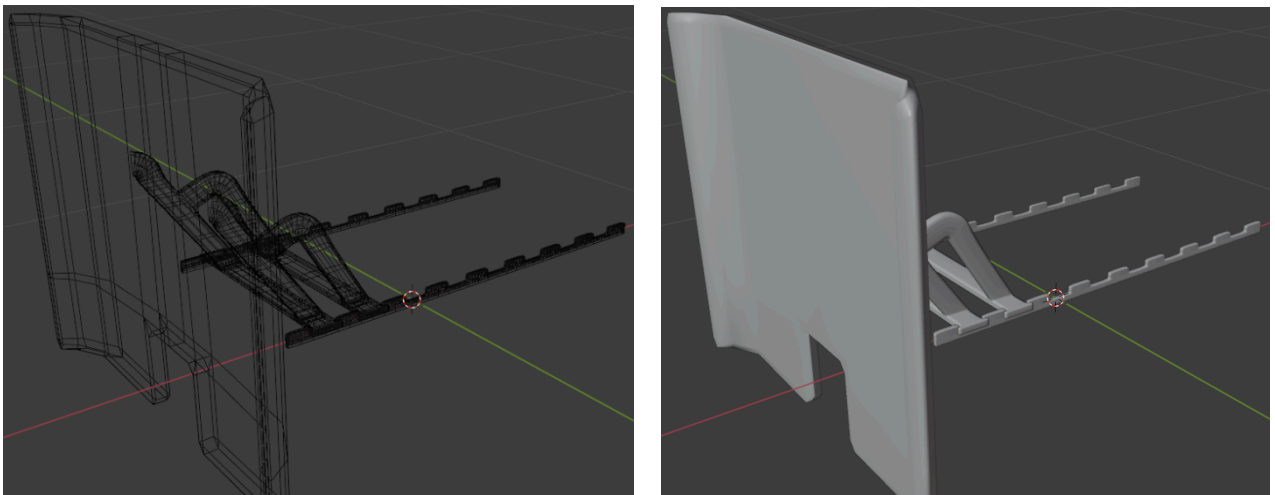
The interior of the game level.



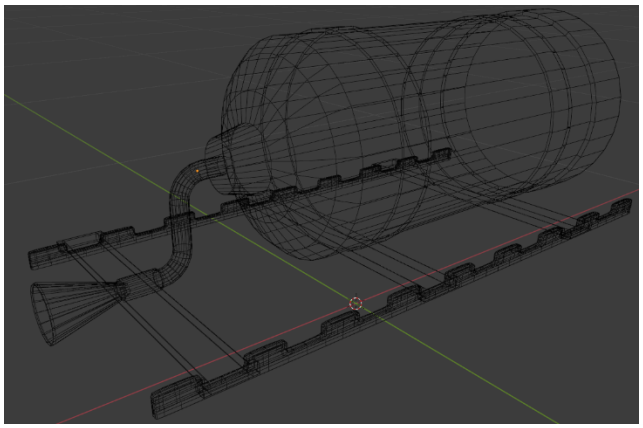
Carrier attachment for a drone.

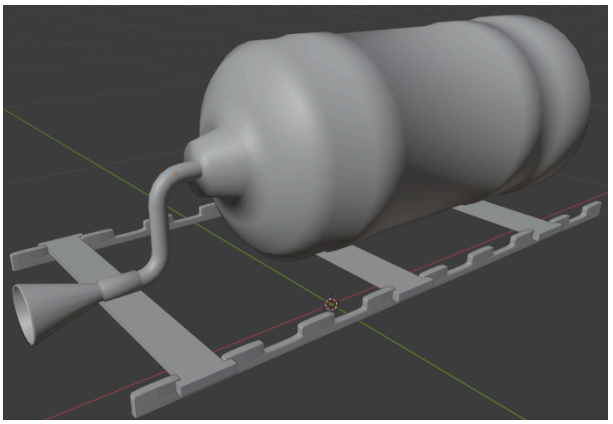


Dozer attachment for a drone.

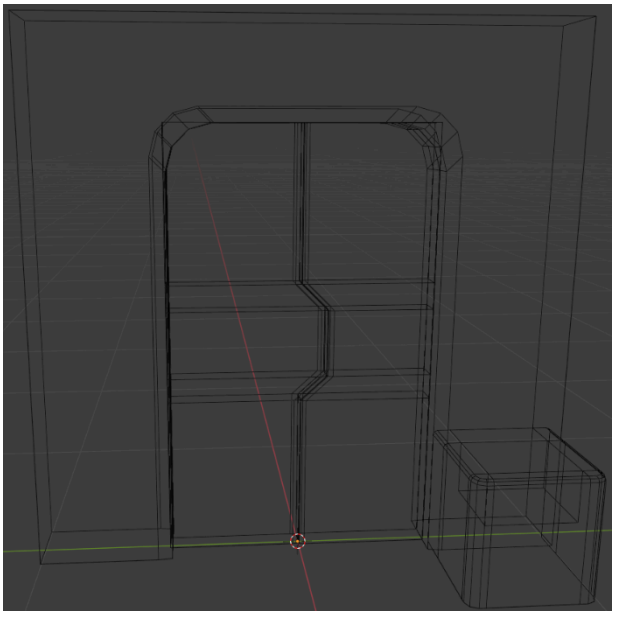


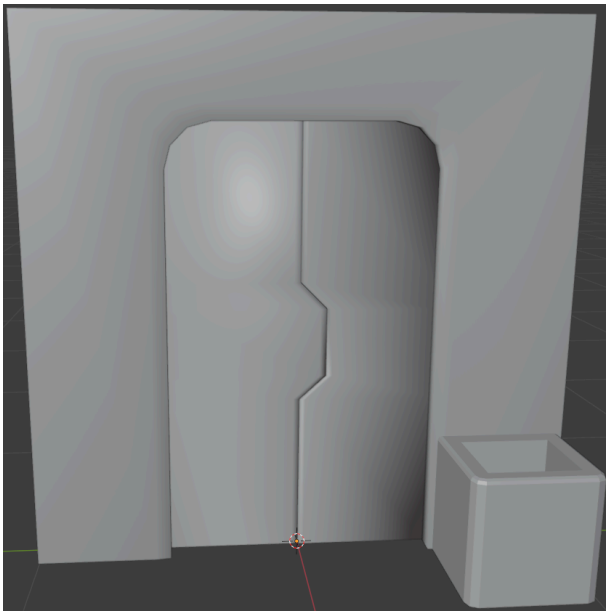
Extinguisher attachment for a drone.



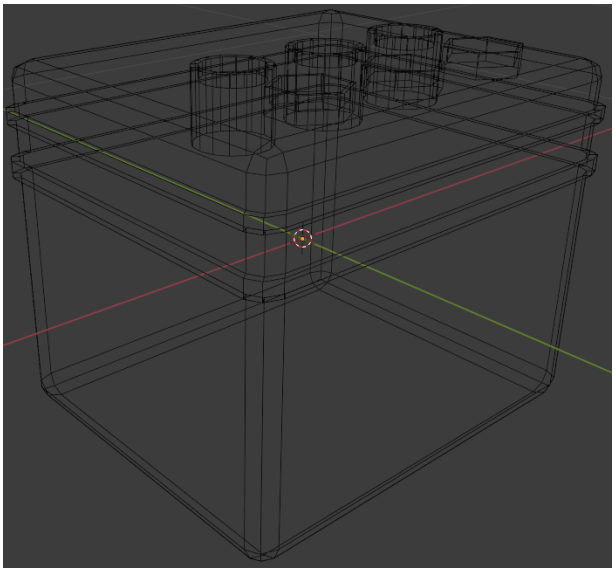


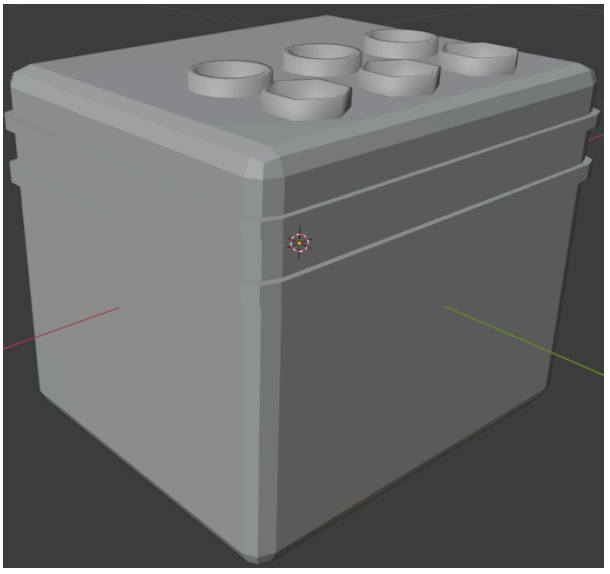
Door and doorway, with animations, which will be openable by the player in some way.



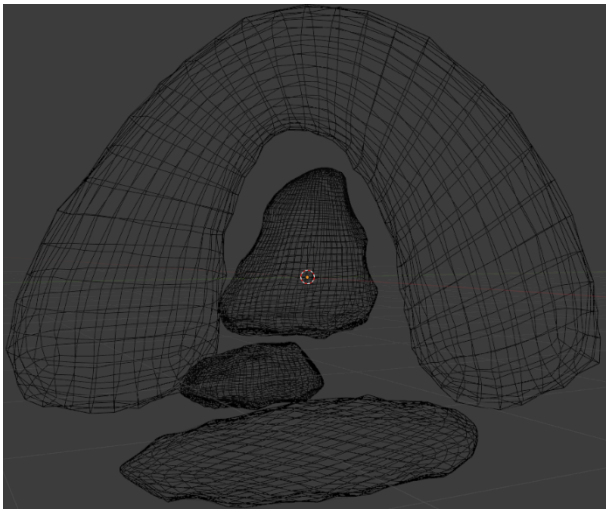


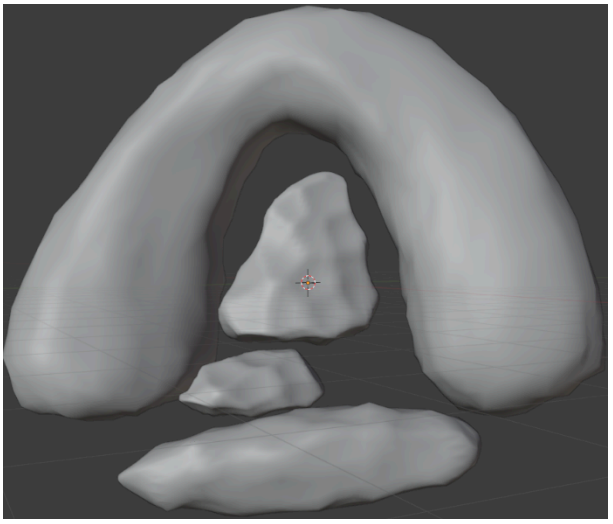
Battery item which can be picked up by the player and used on a door.



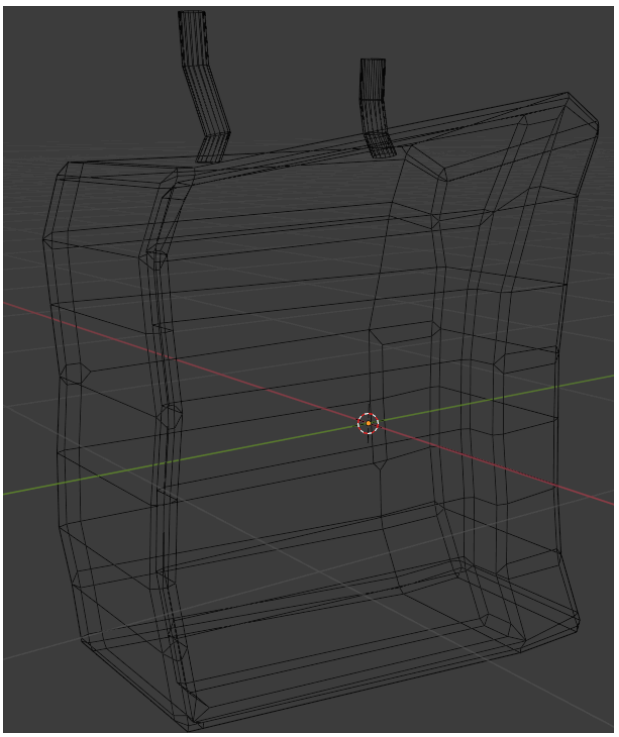


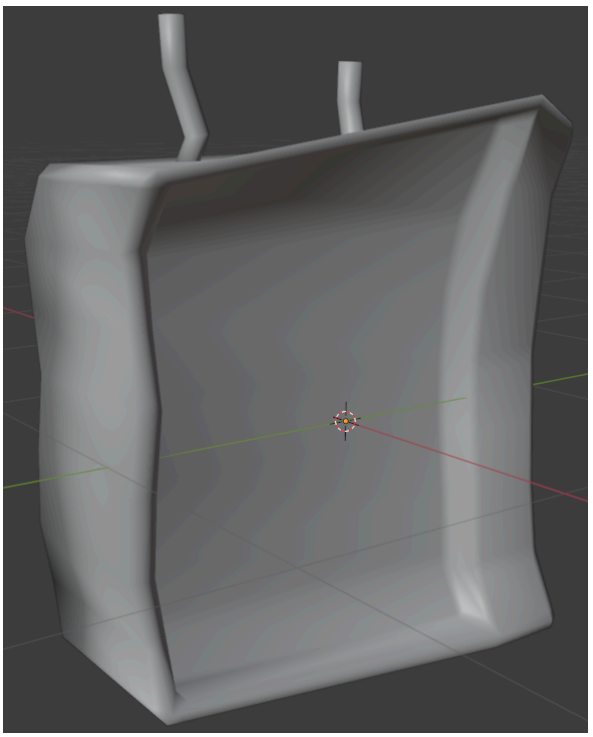
Various rocks, to be used to guide the player's direction by creating clear pathways.



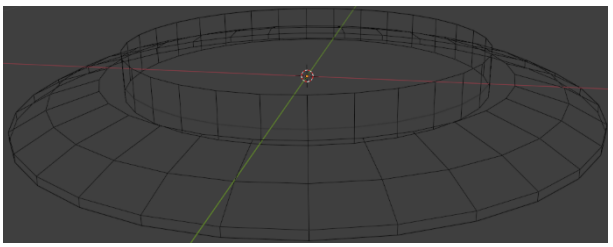


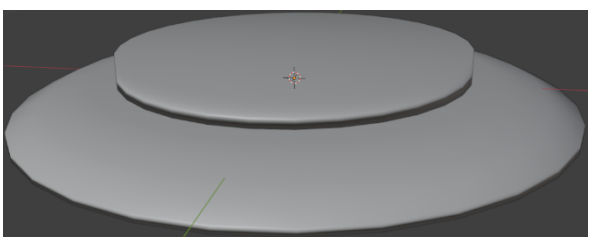
Debris which can be pushed out of the way by the Dozer type drone, but not the others.



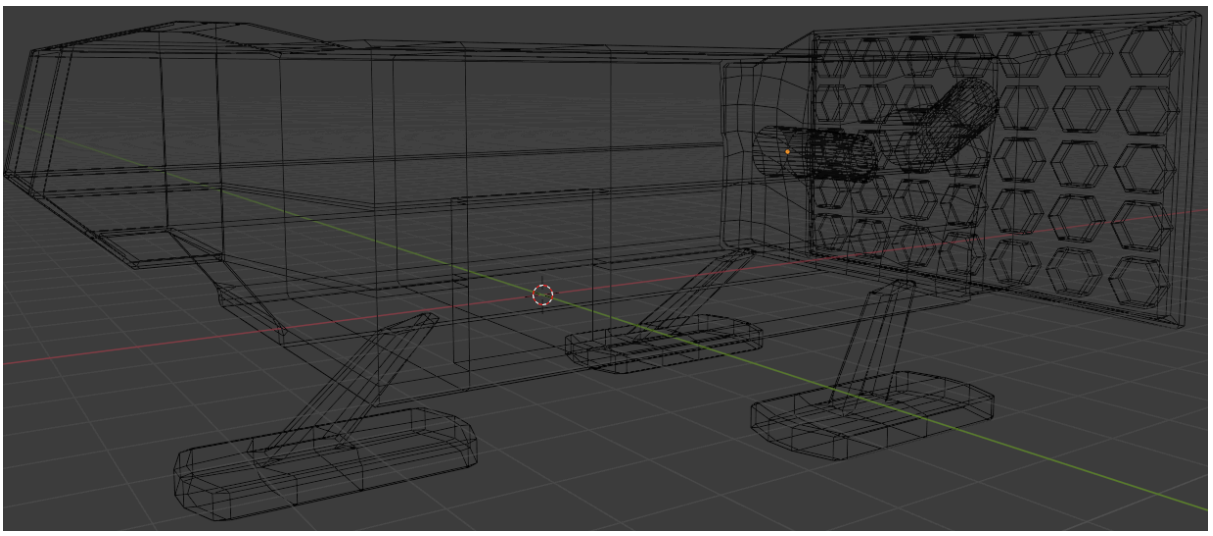


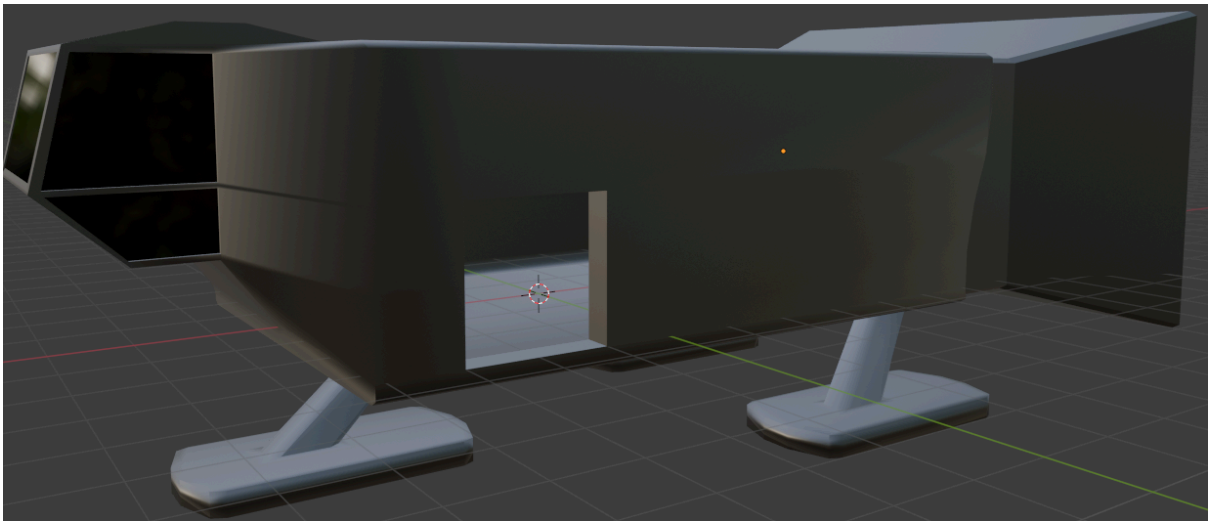
Button, with animation, which can be pressed by drones to have some action in the world.



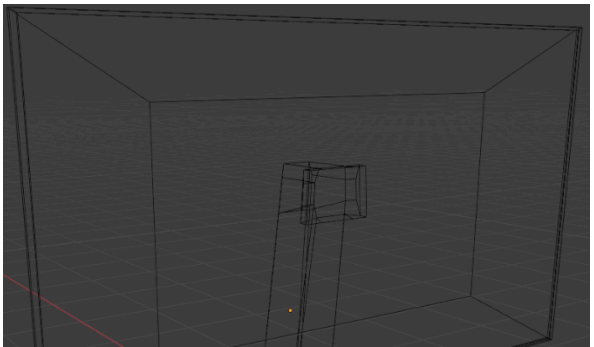


Ship which can be broken apart to act as set decoration, the main body section is designed to contain the interior of the game level.

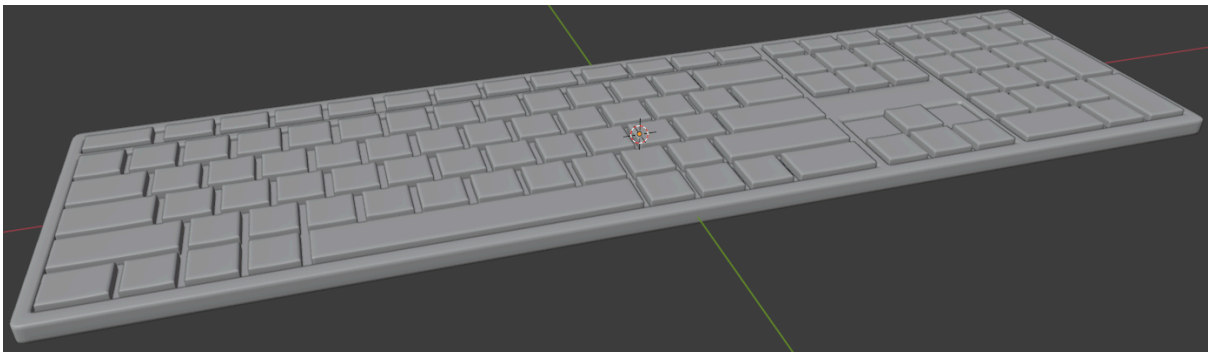




Desktop monitor for the menu decoration.



Keyboard for the menu decoration.



Sound Effects

To create sound effects, I used Audacity to prepare audio recordings I have made.

Effect	Source	Processing
Wind Loop	Wind	Added a fade in and fade out at the edges of the track, then spliced the beginning and the end together to make a seamless loop.
Drone Motor	Xbox controller rumble motors	Pitched up and amplified.
Door Open	Hiss from opening bottle, rubbing a tin along microphone.	Increased the gain on the second audio clip, then put the two sounds to play together in the same audio track, timed with the door animation.
Item pickup	Dropping a small tin onto a larger tin.	No processing.
Jump Piston	Bike pump	Increased tempo and amplified.
Screen Switch	Synthetic	Generated a short clip of white noise in Audacity and applied a sharp fade in and out.
Fire	Wind	Edited the wind loop: normalised loudness, reduced tempo, reduced pitch, added distortion.
Fire Extinguish	Synthetic	Generated pink noise, added a fade in and out, increased tempo.
Extinguisher Foam	Squirty cream dispenser	Reduced pitch and amplified.
Menu Click	Clicking ends of a charger together	Amplified.

Audio Variation

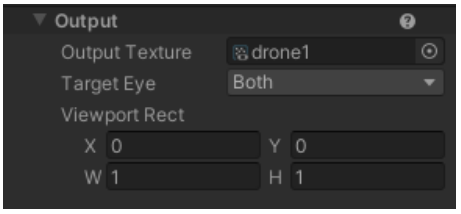
For the drone's motor, the audio varies based on how far the joystick is pushed from the centre.

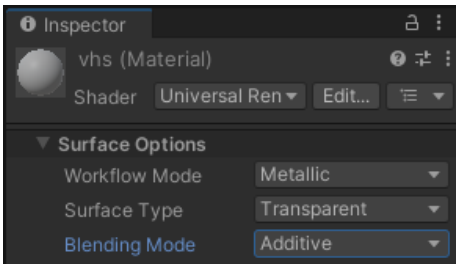
```
motorMagnitude = Mathf.Clamp(Mathf.Abs(input.x) + Mathf.Abs(input.y), 0.0f, 1.0f);
GetComponent<AudioSource>().volume = motorMagnitude;
GetComponent<AudioSource>().pitch = motorMagnitude + 1f;
```

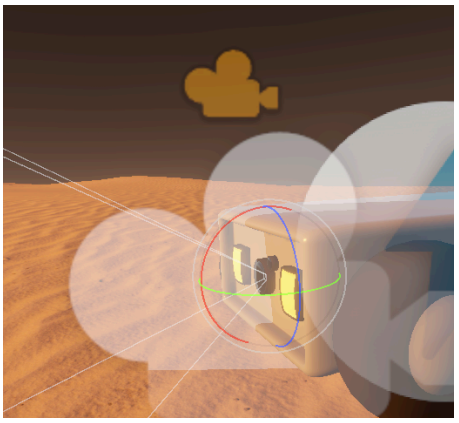
Similarly, the jump sound effect pitch is based on how much charge the jump has gathered.

```
jumpAudioSource.GetComponent<AudioSource>().pitch = jumpCharge;
jumpAudioSource.GetComponent<AudioSource>().PlayOneShot(jumpSound);
```

Screen effect





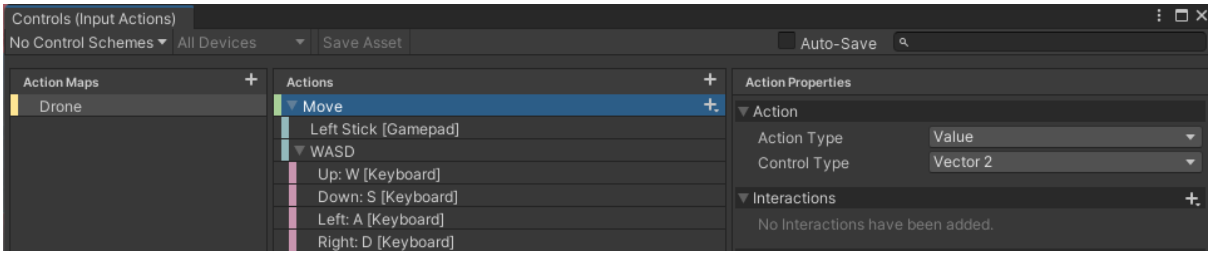


The screen effect is achieved with two plane game objects. The first plane is given a material created by using a Render Texture which receives the output from a camera attached to one of the drones.

This first plane completes the minimum requirements for a screen, but to improve the sophistication, I added a second plane slightly above the image, which is configured with a material with transparent surface type and blending mode set to additive. The plane has a video player component which has a loop of a VHS effect I found online playing. On the main screen there is an additional overlay which uses a Coroutine to display an additional effect temporarily when switching cameras.

Movement and Collision Detection

For the player to control the character in the game, I used unity's new User Input system to create an InputActions map for movement. I configured it to have as default input via the left stick on a game controller, or alternatively using W, A, S and D keys on a keyboard to emulate that for those without a controller. The input from this control scheme is delivered as a parameter to the method OnMove().



Whilst the usual usage of this input is to have the left and right directions for strafing, this wouldn't make sense for my type of character which has wheels and shouldn't be capable of moving in that way. Instead, I opted to use the left and right direction to rotate the character on the spot.

```
private void OnMove(InputValue value)
{
    input = value.Get<Vector2>();
}
1 reference
private void Move()
{
    rb.MovePosition(transform.position + transform.forward * input.y * movementSpeed * Time.deltaTime);
    transform.Rotate(transform.up, input.x * rotationSpeed * Time.deltaTime);
}
```

I created a script DroneController, which would be a component of each of the drone characters in the game, to manage the operation of the drones. I decided to use the RigidBody MovePosition for controlling forwards and backwards movement, which allows colliders to push into each other smoothly unlike the Translate method of Transform, but this takes as the parameter a location in world space to move towards, rather than just a vector. To get this value, the current position of the transform can be used additively to the vector. This is less of a concern for rotation, so I used the simpler Rotate method for that.

```
void FixedUpdate()
{
    if (Vector3.Dot(transform.up, Vector3.up) > 0)
    {
        Move();
    }
}
```

A problem that I discovered was that if the character became flipped upside-down, they would still be able to move around as normal, which doesn't make any sense. This issue was fixed by checking the dot product of the up vector of the character itself and the up vector in world space. If this calculation resulted in a value between 0 and -1, it would mean that the object had been flipped onto its back or side, and shouldn't be able to move so the Move() method is only called if this is not the case.

Now that the user loses control if the character is flipped, there needs to be a way to unflip the vehicle. For this I added a new input control for R key, or D-Pad Up on a controller, which calls a simple method which resets the x and z rotation of the character,

so that they can get themselves unstuck.

```
private void OnUnstuck()
{
    if (controlled)
    {
        transform.rotation = Quaternion.Euler(0, transform.eulerAngles.y, 0);
    }
}
```

Game Play Features

Switching between drones

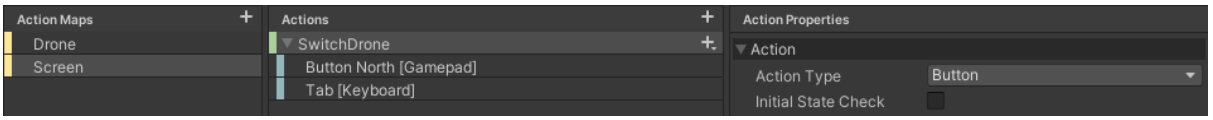
One of the core features of the game is being able to switch between the four available drone characters at will, so that the player can use them to co-operate to solve puzzles.

To achieve this, first I had to create a Boolean for the DroneController script which would track whether the drone it is currently attached to is being controlled, and only allow movement if it is.

Next, I created a new script ScreenSwitcher with arrays for both the drone GameObjects themselves, and for the material created from a RenderTexture of the output of each drone's Camera and linked these up appropriately in editor.

```
[SerializeField] private GameObject drone1, drone2, drone3, drone4;
[SerializeField] private Material matDrone1, matDrone2, matDrone3, matDrone4;
void Start()
{
    drones = new GameObject[] { drone1, drone2, drone3, drone4 };
    cameraFeeds = new Material[] { matDrone1, matDrone2, matDrone3, matDrone4 };
}
```

I also added a new input control for the switching of drones, by default linked to the TAB key, or the “North” button on a controller (Y for Xbox, Triangle for PlayStation).



When the user presses this button, it calls the OnSwitchDrone() method, which has code for looping around from the value 0 (start of array) to the value at the end of the array. This gives a reference value that can be used by the arrays to dictate which item in the array should be in use.

```
if (controlledDrone == drones.Length - 1)
{
    controlledDrone = 0;
}
else
{
    controlledDrone++;
}
```

After cycling this number, the corresponding drone has its controlled variable altered so that only the drone with the number in the reference is set to true, and whichever was being controlled previously will be set to false.

```
drones[controlledDrone].GetComponent<DroneController>().Controlled = true;
if (controlledDrone == 0)
{
    drones[drones.Length - 1].GetComponent<DroneController>().Controlled = false;
}
else
{
    drones[controlledDrone - 1].GetComponent<DroneController>().Controlled = false;
}
```

Now that the drones can be switched between, this needs to be reflected in the screen system, by setting the main screen’s camera feed to be the appropriate one linked to the drone being controlled.

```
if (drones[controlledDrone].GetComponent<DroneController>().Controlled)
{
    GetComponent<Renderer>().material = cameraFeeds[controlledDrone];
}
```

Carrying Items

One of the drone types has the ability to pick up an item, which can be used elsewhere in the level. I wanted to create a flexible system that can be easily expanded to deal with different pick up items, so I built a simple script “Pickup” which just stores a single variable pickupType. Other scripts can check if this script is attached to an object, which will let them know if the

```
[SerializeField] private string pickupType;
1 reference
public string PickupType
{
    get { return pickupType; }
    set { pickupType = value; }
}
```

```
private void OnTriggerEnter(Collider other)
{
    // when entering a trigger, set a reference to the object it belongs to
    if (other.isTrigger)
    {
        nearbyObject = other.gameObject;
        if (nearbyObject.GetComponent<Pickup>())
        {
            inPickupZone = true;
            if (inventory == "none")
            {
                mainScreenPrompt.text = "Use action to pick up";
                mainScreenPrompt.GetComponent<MeshRenderer>().enabled = true;
            }
            else
            {
                mainScreenPrompt.text = "Inventory at capacity";
                mainScreenPrompt.GetComponent<MeshRenderer>().enabled = true;
            }
```

object is a pickup, and also further query exactly what type of object it is.

To give a drone the ability to interact with pickups, I created the CarrierDrone script, which is attached to any drones that should be of that type. This script keeps track of the last trigger volume it entered, and if it is an item which can be picked up, it gives the relevant prompt on screen depending on whether the inventory is full or not, as well as setting a Boolean value which denotes whether it's possible for the object to be picked up.

This value is utilised in another area of the script, which when called by the user pressing the action button (West button on gamepad, or E on keyboard), causes the item to be destroyed, and variables indicating that it is now stored by the drone itself are toggled on. The item carried by the drone is telegraphed to the player by a model of the item being displayed in the carrying box of the drone, in addition to displaying the item on the info screen at the bottom right

of the player's screen.

Once the player has an item, they can bring it to a delivery zone trigger. If they have the right item for that trigger, then they will be able to press the Action button to deliver the item to that zone, which removes it from the inventory and completes an objective.

If the player brings the blackbox item to the delivery zone at the end of the level, they will be able to deliver it there which ends the level and returns the player to the main menu.

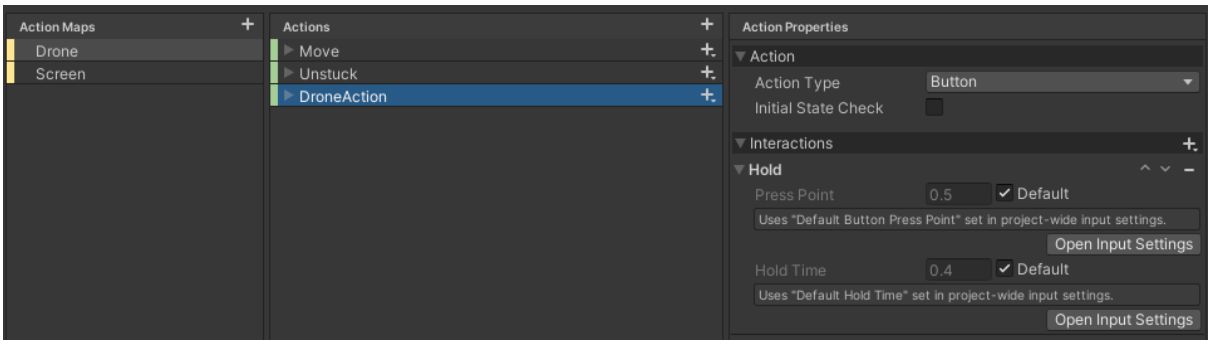
If the player brings a battery item to a door that requires a battery, they will be able to use it to open the door.

Jumping

Another drone type is the jumper drone, which is smaller and can do either a small jump or charge up a jump up to twice as high. To be able to implement this behaviour, I needed a way to record how long the button was held for. I already knew how to do this using the

legacy input in Unity, but I wanted to utilise the new InputSystem, so did some research and found a useful blog which helped me to reach this goal.

I modified the Input Actions scheme to have the Hold interaction, which enables more complex behaviour than simply checking for a button down event.



Then instead of using the regular OnDroneAction() to trigger on input, I had to create a reference to the moment the action is “canceled”, which allows for the jump to trigger when the button is let go of, as well as being able to gather the duration of the button being held, and therefore be used to modify the force of the jump based on that.

In addition to checking if the drone is currently being controlled, to ensure that the player cannot double jump, I have a Raycast check just underneath the drone, and the jump will only work if there is some floor to collide with within its path.

I also added a UI feature that uses a progress bar to show how much charge is being held for the jump, so that the player knows when they can stop holding the jump button.

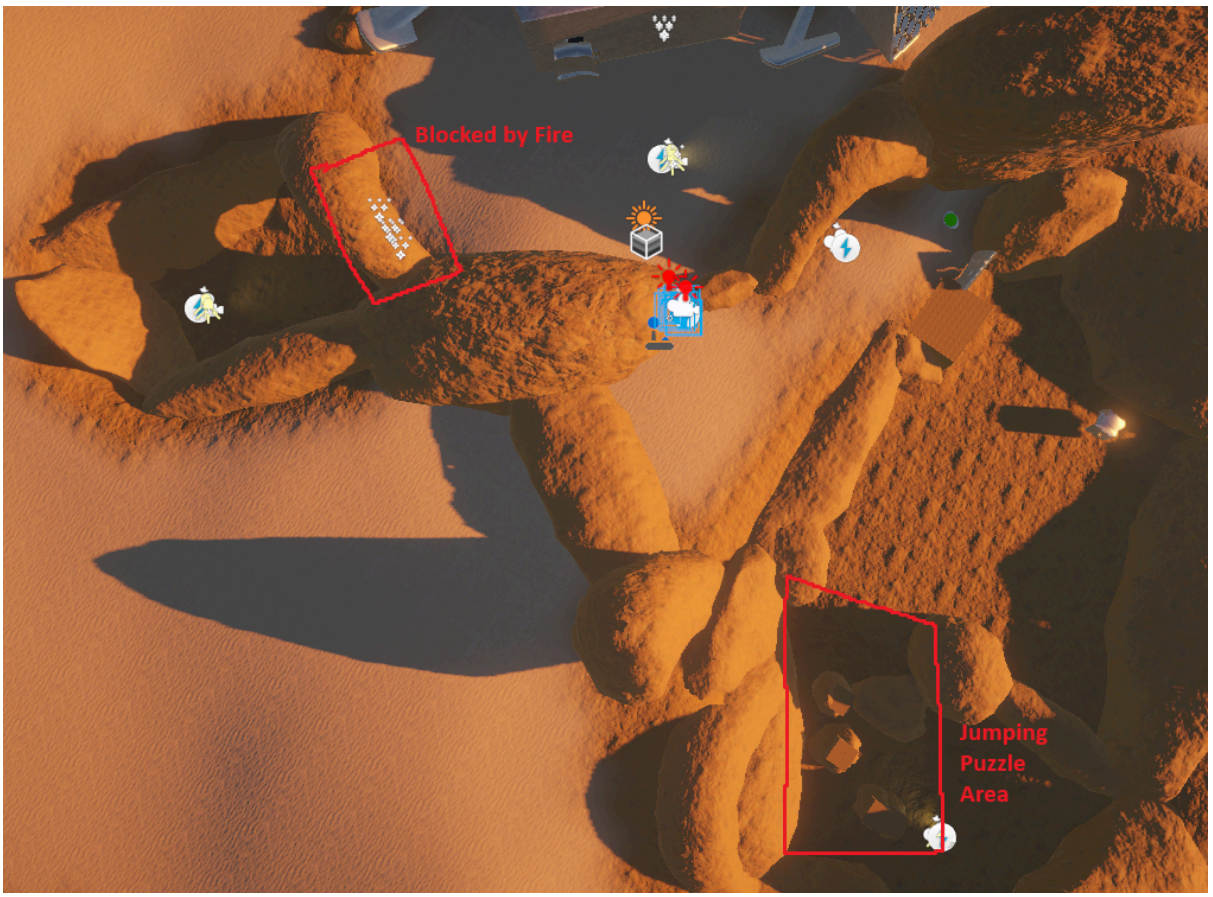
Pushing Objects

One of the drone’s abilities is to move heavy objects that would block other drones from passing through an area. To accomplish this, I added functionality to the action button which lets the user switch this drone to a mode where the mass value of the rigidbody is greatly increased, in addition to reducing the speed that it is capable of moving at.

Extinguishing Fires

Another ability in the game is using a fire extinguisher to remove fire obstacles from the level. When the player holds the Action button the extinguisher will steadily instantiate and move invisible projectiles that have a trigger collider, that when entering a fire obstacle will deal damage to the fire, and eventually cause it to be disabled. In addition to these projectiles, a particle system is used to visually represent them.

Level Design



After completing the initial outdoors area for the level as above, I ran some playtests and found that players were struggling to figure out how to deal with the marked areas. Where the route is blocked by fire, some players could not find the area, due to it being at a lower height than the area just outside it, so unless they went towards it purposefully its unlikely they would see it. To solve this I moved the fire so that it wasn't covered by the rocks, and increased the height of the smoke so that it was more easily visible from a higher elevation.

Another issue was the jumping puzzle area, where players did not recognise that the platforms were intended to be jumped up to and were just trying to jump over the walls instead. To prevent this I added paint decals to the platforms, which makes it very obvious that they are distinct from the regular environment.

The only other issue I found in playtesting was that players would leave the play area after struggling to figure out where they actually needed to go. A simple fix for this is putting up walls on the exits to the play area that aren't blocked already.

The interior area did not have any issues being completed as it is less busy visually.

Sources

https://docs.unity3d.com/Manual/index.html - Unity user manual

https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/using-properties - C# Properties

https://christopherhilton88.medium.com/exploring-tap-hold-interactions-hierarchical-precedence-and-errors-unitys-new-input-system-bf9bc2aa0882 - hold interaction

Third-party Assets

https://giphy.com/gifs/television-music-video-l378vMZ1IbLcmj3H2 - VHS switch effect

https://blog.unity.com/engine-platform/free-vfx-image-sequences-flipbooks - particle effects textures

https://www.pinterest.co.uk/pin/428334614557335847/ - VHS camera effect

https://www.poliigon.com –environment textures

https://www.freepbr.com – environment textures

https://www.bing.com/images/create - logo graphics

https://cpetry.github.io/NormalMap-Online/ - normal maps

https://www.deviantart.com/ottomattick/art/Seamless-Door-Texture-v2-180640178 - menu door texture

https://www.dafont.com/abduction-2002.font - font

https://www.pinterest.co.uk/pin/295196950572177263/ - gamepad icon

https://uppbeat.io/t/jeff-kaale/aeon - Music from #Uppbeat (free for Creators!): License code: CPJTJAGTW38O1C0K – trailer video music