A Stowaway Contribution…

By Andrew McAllister…

For Major Game Project (601165_A24_T12).

Source: www.pexels.com/photo/3d-render-of-pink-and-purple-pixels-12843338/

Project Overview

What is our game about?

Stowaway is a Survival Horror game made with Unity 3D where the player has to run around their space vessel and complete tasks all whilst an alien is hunting them. The game will take place within several days of which the ship will continue its journey to it's destination planet.

What are my responsibilities?

My roles within the group are as follows:

- Game Mechanics
- Compiling everything in the Engine
- Pixel Art (Minor additions)
- Programming
- Audio

What software am I using?

- Unity 2022.3.5f1: Unity is what I will be using to build the game along with Visual Studio for coding. This was chosen since it is the game engine I'm most comfortable with.

- Aseprite: A pixel art software that I'll be using to create any computer screens in the game.

- Audacity: A free software that allows to customise sounds that I've recorded into proper sound effects to use in the game.

Audacity Logo: https://simple.wikipedia.org/wiki/Audacity
Aesperite Logo: commons.wikimedia.org/wiki/File:Logo_Aseprite.svg
Unity Logo: commons.wikimedia.org/wiki/File:Unity_Technologies_logo.svg

Game Mechanics

Stealth and Audio Detection: A mechanic where each action the player makes a sound that can be detected by the alien, drawing it to the player's location. Louder the sound the further the alien can hear from.

Ship Repair and Degradation: The game starts out with a couple of damaged ship parts throughout the ship which the player must fix. The longer the player takes the lower the condition the other parts can lower to. Ship repair inspired by Among Us.

Multi-day Ship Voyage: Inspired by Lethal Company, the player needs to survive a minimum of three days so the ship's navigation can send the player to their final destination, which is when the game ends. The condition of the ship will determine how long the journey will take.

Safe Room: The safe room is the player's spawn location and is where they can check the condition of the ship's equipment from their computer as well as sleep to the next day.

Stealth and Audio Detection

There are two foreseeable methods I could implement this: the first is by assigning each object that could produce noise, including the player, with noise values for each of its associated actions.

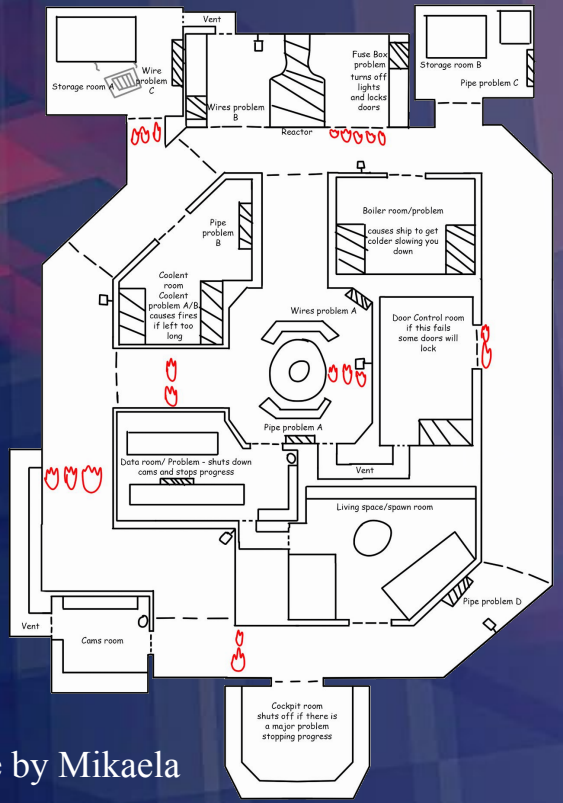
The second is more foreign to me, where I may be able to use the decibels of the sounds used in the game for the noise level instead of fixed values.

The noise level of items (if there is any) will change the scale of a sphere collider around the object any if the collider reaches the alien, it will cause the alien to detect it and head to the location the noise was made.

As part of the UI, we will have a sound bar so the player will be able to see how much noise they are making then committing actions such as running or fixing a piece of the ship.

Ship Repair and Degradation

There are many tasks that the player will need to complete throughout the ship in order to reach their destination. The condition of each ship part will be controlled by a master script, which will gradually reduce each part's individual condition. Every so often, each task will have a chance of losing condition until it reaches zero, at which point it must be repaired. This system introduces a random element to which parts the player will need to fix.



Map made by Mikaela

Starting work on Player Movement

Since at this point I didn't have any puzzle assets or the map, best thing to start out working on is the player movement.

This script updates the player body's y and camera x rotation as the player moves their mouse based on the mouse sensitivity and if or not the camera lock bool is enabled.

I added the cameraLock bool in when I've been working on the puzzle scripts which require the camera to be still when interacting with them.

```
public class scr_mouseLook : MonoBehaviour
{
    public float mouseSensitivity = 4f;

    public Transform playerBody;

    float xRotation = 0f;

    public bool cameraLock = false;

    // Start is called before the first frame update
    void Start()
    {
        Cursor.lockState = CursorLockMode.Locked;
        Cursor.visible = false;
    }

    // Update is called once per frame
    void Update()
    {
        if (!cameraLock)
        {
            float mouseX, mouseY;

            mouseX = Input.GetAxis("Mouse X") * mouseSensitivity;
            mouseY = Input.GetAxis("Mouse Y") * mouseSensitivity;

            xRotation -= mouseY;
            xRotation = Mathf.Clamp(xRotation, -90f, 90f);

            transform.localRotation = Quaternion.Euler(xRotation, 0f, 0f);
            playerBody.Rotate(Vector3.up * mouseX);
        }
    }
}
```

Starting work on Player Movement

The player movement mostly uses the character controller to apply the movement to the player, though through this script it determines the players speed, handles the player's gravity and allows the player to jump based on the jumpHeight variable. When making this script I had help with ChatGPT to create jump logic and alter the animator's speed based on whether the player was running or not.

```
// Update is called once per frame
void Update()
{
    isGrounded = Physics.CheckSphere(groundCheck.position, groundDistance, groundMask);

    if (isGrounded && yVelocity < 0)
    {
        yVelocity = -1f;
    }

    float x = Input.GetAxis("Horizontal");
    float z = Input.GetAxis("Vertical");

    Vector3 movement = transform.right * x + transform.forward * z;

    bool isRunning = Input.GetKey(KeyCode.LeftShift) && movement.magnitude > 0f;
    float currentSpeed = isRunning ? speed * runningMultiplier : speed;

    if (Input.GetButtonDown("Jump") && isGrounded)
    {
        yVelocity = Mathf.Sqrt(jumpHeight * -2f * gravity);
    }

    // Apply gravity
    yVelocity += gravity * Time.deltaTime;

    // Apply movement
    Vector3 velocity = movement * currentSpeed;
    velocity.y = yVelocity;
    controller.Move(velocity * Time.deltaTime);

    // Adjust animation speed based on grounded state
    animator.speed = isGrounded ? 1f : 0f;
```

Starting work on Player Movement

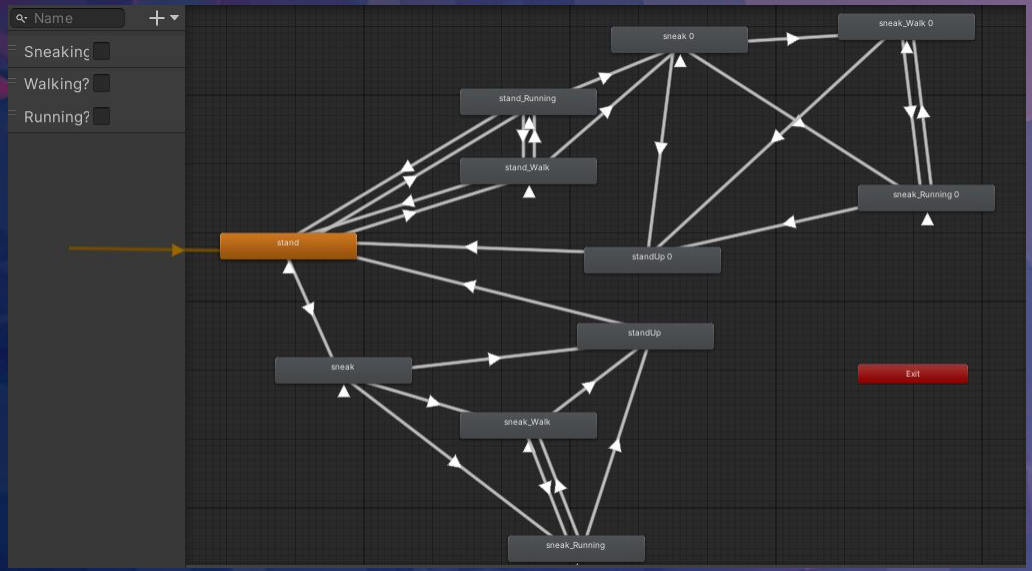
I then added some if statements to determine the player's state, meaning whether or not they're walking, walking, sneaking or still. This is mostly to set bools for the camera animations but will also come in handy when I start work on the audio detection for the player and when adding player footstep audio so I know how often a step sound effect has to play.

```
// Sneaking logic
if (Input.GetKey(KeyCode.LeftControl))
{
    isSneaking = true;
    animator.SetBool("Sneaking?", true);
}
else
{
    isSneaking = false;
    animator.SetBool("Sneaking?", false);
}

// Update playerMovementState
if (movement.magnitude == 0f)
{
    playerMovementState = "Still";
    animator.SetBool("Walking?", false);
    animator.SetBool("Running?", false);
}
else if (isRunning)
{
    playerMovementState = "Running";
    animator.SetBool("Walking?", false);
    animator.SetBool("Running?", true);
```

Player Camera Animations

I have created animations for the main camera so that it will bob up and down when the player is walking or running. The setup is quite messy but it does track what animation should be playing based on the setbools from the player movement script well.

I did notice however an issue with the still and run animation transition where if the player is running and stops moving near the beginning of the running animation the animation will need to finish until it goes onto the still animation, making the camera bob even when the player is still. I don't want to have to make the transition instant because that will make the camera snap in place. For now I'll disable the animations during testing and find a work around in the future.



Player Interaction

Before working on the puzzle scripts I need a way of interacting with objects by looking at them, and that's where raycast comes in. Like the name suggests creating a raycast casts out a ray in a direction that can collide with gameobjects, in this case those gameobjects are interactables. So using this script I'm able to interact with objects with the scr_interactable script attached to them.

It's important to note I got this script and scr_Interactable from a YouTube tutorial by Kabungus, which can be found here: youtube.com/watch?v=b7Yf6BFx6js

```
public class scr_playerInteraction : MonoBehaviour
{
    public float playerReach;
    scr_Interactable currentInteractable;

    // Update is called once per frame
    void Update()
    {
        CheckInteraction();

        if (Input.GetKeyDown(KeyCode.E) && currentInteractable != null)
        {
            currentInteractable.Interact();
        }
    }
```

```
void CheckInteraction()
{
    RaycastHit hit;
    Vector3 rayOrigin = Camera.main.transform.position + Camera.main.transform.up * 0.1f;
    Ray ray = new Ray(rayOrigin, Camera.main.transform.forward);

    if (Physics.Raycast(ray, out hit, playerReach))
    {
        if (hit.collider.CompareTag("Interactable"))
        {
            scr_Interactable newInteractable = hit.collider.GetComponent<scr_Interactable>();

            if (newInteractable != null && newInteractable.enabled)
            {
                if (newInteractable != currentInteractable)
                {
                    Debug.Log("New interactable detected: " + newInteractable.name);
                }
                SetNewCurrentInteractable(newInteractable);
            }
            else
            {
                DisableCurrentInteractable();
            }
        }
        else
        {
            DisableCurrentInteractable();
        }
    }
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Events;

// https://www.youtube.com/watch?v=b7Yf6BFx6js

public class scr_Interactable : MonoBehaviour
{

    public string message;

    public UnityEvent onInteraction;


    public void Interact()
    {
        onInteraction.Invoke();
    }

    void Update()
    {

    }
}
```

Camera To Object Script

First thing I have to make for the puzzles is a camera to object script, this script moves the camera to a new position based on a Vector3 variable put into the inspector. This script works with the interactable script to call the MoveCamera() function and when doing so will also trigger a bool to true to tell the puzzle script the player is interacting with that they are indeed interacting with the puzzle.

When interacting, I have also made it so the script will grab the player movement script and disable it so the player cannot move while interacting, it's also similar with the mouse look script but with that I just enable the camera lock bool to prevent the script from detecting the player's mouse inputs.

```
using System.Collections;
using UnityEngine;
using TMPro;

public class scr_cameraToObject : MonoBehaviour
{
    [Header("Player Data")]
    [SerializeField] public GameObject player;
    [SerializeField] public bool playerCurrentlyInteracting;

    [Header("Camera Data")]
    [SerializeField] Transform cameraTransform;
    [SerializeField] public Vector3 targetRotations;
    [SerializeField] public float cameraSpeed;
    [SerializeField] Vector3 targetCameraPosition;
    [SerializeField] public Vector3 originalCameraPosition;

    [Header("Mouse and UI Data")]
    [SerializeField] public scr_mouseLook mouseLookScript;
    [SerializeField] TMP_Text interactionIndicatorText;

    public void StartMoveCamera()
    {
        if (mouseLookScript != null)
        {
            mouseLookScript.cameraLock = true; // Lock the camera movement
        }

        interactionIndicatorText.gameObject.SetActive(false);

        StartCoroutine(MoveCamera());
    }
```

```
public IEnumerator MoveCamera()
{
    originalCameraPosition = cameraTransform.position;

    player.GetComponent<scr_playerMovement>().enabled = false;

    Quaternion startRotation = cameraTransform.rotation;
    Quaternion targetRotation = Quaternion.Euler(targetRotations);

    float timeElapsed = 0f;
    while (timeElapsed < cameraSpeed)
    {
        cameraTransform.position = Vector3.Lerp(originalCameraPosition, targetCameraPosition, timeElapsed / cameraSpeed);
        cameraTransform.rotation = Quaternion.Lerp(startRotation, targetRotation, timeElapsed / cameraSpeed);
        timeElapsed += Time.deltaTime;
        yield return null;
    }

    cameraTransform.position = targetCameraPosition;
    cameraTransform.rotation = targetRotation;

    playerCurrentlyInteracting = true;
}
```

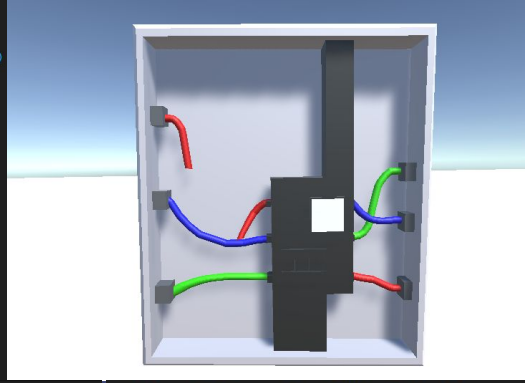
Wire Puzzle

Now that I have the puzzle models from Mikaela I can start work on coding the game's puzzle logic, the first I decided to work on is the wire puzzle.

For this puzzle the player has to repair the red wire by holding down the E or R buttons depending if they want to go for a complete or temporary fix. As the player holds down one of the buttons a meter will go up that will go back to zero if the player lets go.

One the puzzle is completed or the player presses the return key they can stop interacting with the puzzle and regain control of their character.

```
// Update is called once per frame
void Update()
{
    if (cameraToObjectScript.playerCurrentlyInteracting && Input.GetKeyDown(KeyCode.Return))
    {
        cameraToObjectScript.playerCurrentlyInteracting = false;
        StartCoroutine(ReturnCameraToOriginalPosition());
    }

    if (cameraToObjectScript.playerCurrentlyInteracting)
    {
        if (!puzzleUI.activeSelf) puzzleUI.SetActive(true);

        // Fully Fix
        if (puzzleStatus == PuzzleStatus.Broken && Input.GetKey(KeyCode.E))
        {
            StartCoroutine(BrokenToPerfect());
        }
        // Temporary Fix
        else if (puzzleStatus == PuzzleStatus.Broken && Input.GetKey(KeyCode.R))
        {
            StartCoroutine(BrokenToFixed());
        }
    }
```



```
private IEnumerator ReturnCameraToOriginalPosition()
{
    Vector3 originalCameraLocalPosition = cameraTransform.localPosition;
    Vector3 targetLocalPosition = new Vector3(0f, 1.78f, 0f);

    Quaternion originalCameraLocalRotation = cameraTransform.localRotation;
    Quaternion targetLocalRotation = Quaternion.identity;

    float timeElapsed = 0f;
    float cameraSpeed = cameraToObjectScript.cameraSpeed;

    while (timeElapsed < cameraSpeed)
    {
        cameraTransform.localPosition = Vector3.Lerp(
        originalCameraLocalPosition,
        targetLocalPosition,
        timeElapsed / cameraSpeed);

        cameraTransform.localRotation = Quaternion.Lerp(
        originalCameraLocalRotation,
        targetLocalRotation,
        timeElapsed / cameraSpeed);

        timeElapsed += Time.deltaTime;
        yield return null;
    }

    cameraTransform.localPosition = targetLocalPosition;
    cameraTransform.localRotation = targetLocalRotation;

    cameraToObjectScript.mouseLookScript.cameraLock = false;
    cameraToObjectScript.player.GetComponent<scr_playerMovement>().enabled = true;

    cameraToObjectScript.playerCurrentlyInteracting = false;
```

```
if (puzzleStatus == PuzzleStatus.Perfect)
{
    if (!perfectWire.activeSelf)
    {
        perfectWire.SetActive(true);
        tapedWire.SetActive(false);
        brokenWire.SetActive(false);
    }
}
else if (puzzleStatus == PuzzleStatus.Fixed)
{
    if (!tapedWire.activeSelf)
    {
        perfectWire.SetActive(false);
        tapedWire.SetActive(true);
        brokenWire.SetActive(false);
    }
}
else if (puzzleStatus == PuzzleStatus.Broken)
{
    if (!brokenWire.activeSelf)
    {
        perfectWire.SetActive(false);
        tapedWire.SetActive(false);
        brokenWire.SetActive(true);
    }
}
```

```
private IEnumerator BrokenToPerfect()
{
    puzzleStatus = PuzzleStatus.FixingInProgress;

    fixMeter = 0;

    while (Input.GetKey(KeyCode.E) && puzzleStatus == PuzzleStatus.FixingInProgress && cameraToObjectScript.playerCurrentlyInteracting)
    {
        fixMeter += 10f;
        Debug.Log("Current Fix Meter: " + fixMeter);
        yield return new WaitForSeconds(1);

        if (fixMeter >= 100f) // Check if fix is complete
        {
            puzzleStatus = PuzzleStatus.Perfect;
            Debug.Log("Wires are Perfect");
            lightToEffect.puzzleBroken = false;
            StartCoroutine(ReturnCameraToOriginalPosition());
        }
    }
```

```
private IEnumerator BrokenToFixed()
{
    puzzleStatus = PuzzleStatus.FixingInProgress;

    fixMeter = 0;

    while (Input.GetKey(KeyCode.R) && puzzleStatus == PuzzleStatus.FixingInProgress && cameraToObjectScript.playerCurrentlyInteracting)
    {
        fixMeter += 17.5f;
        Debug.Log("Current Fix Meter: " + fixMeter);
        yield return new WaitForSeconds(1);

        if (fixMeter >= 100f) // Check if fix is complete
        {
            puzzleStatus = PuzzleStatus.Perfect;
            Debug.Log("Wires are Fixed");
            lightToEffect.puzzleBroken = false;
            StartCoroutine(ReturnCameraToOriginalPosition());
        }
    }
```

Fuse Puzzle

With the fuse puzzle, I decided to make it quite a short puzzle where the player just has to remove and replace the broken fuses.

This puzzle uses the enum, “Puzzle Status” quite heavily as it’s checked every time Replace Fuse() function is called and will perform the action based on the puzzle status. Each of these actions do use a lerp to move fuses inside or outside their slots and enable or disable the affected fuse object based if they’re inserting or removing.

This script also uses the same “Return to Original Position” function as the wire script as well as most of the other puzzles will in the game.

I’ve also added a new section that disables the interactable script so it makes it possible to move the camera to the object if the puzzle is fixed and the player will only be able to interact with it if the puzzle is broke.

```
// Update is called once per frame
void Update()
{
    if (cameraToObjectScript.playerCurrentlyInteracting && Input.GetKeyDown(KeyCode.Return))
    {
        cameraToObjectScript.playerCurrentlyInteracting = false;
        StartCoroutine(ReturnCameraToOriginalPosition());
    }

    if (cameraToObjectScript.playerCurrentlyInteracting)
    {
        if (puzzleStatus != PuzzleStatus.TwoNew && puzzleStatus != PuzzleStatus.WIP && Input.GetKey(KeyCode.E))
        {
            StartCoroutine(ReplaceFuses());
        }
    }

    if (puzzleStatus == PuzzleStatus.TwoNew)
    {
        interactableScript.enabled = false;
    }
    else
    {
        interactableScript.enabled = true;
    }
}
```

```
public IEnumerator ReplaceFuses()
{
    float timer = 0f;

    if(puzzleStatus == PuzzleStatus.TwoBroken)
    {
        puzzleStatus = PuzzleStatus.WIP;

        Vector3 startPosition = emptyLeftFuse.transform.localPosition;
        Vector3 targetPosition = startPosition + new Vector3(0, 0, 0.2f);

        while (timer < 0.3f)
        {
            emptyLeftFuse.transform.localPosition = Vector3.Lerp(startPosition, targetPosition, timer / 0.3f);
            timer += Time.deltaTime;
            yield return null;
        }

        emptyLeftFuse.transform.localPosition = targetPosition;

        yield return new WaitForSeconds(0.3f);

        emptyLeftFuse.SetActive(false);
        emptyLeftFuse.transform.localPosition = startPosition;

        puzzleStatus = PuzzleStatus.OneBroken;
    }
    else if(puzzleStatus == PuzzleStatus.OneBroken)
    {
```

```
public enum PuzzleStatus { TwoBroken, OneBroken, Empty, OneNew, TwoNew, WIP };
```

Valve Puzzle

For this puzzle the player has to press Q and E back and forth to turn a valve until completion. For each correct input the player presses it will rotate the valve's z rotation.

I've also added a cooldown to the script so that the player must wait 0.15 seconds until the next input can be registered, this is so that there is some kind of limit when spamming Q and E.

There is also the valve meter variable where when that variable hits 660 the puzzle is complete. Each correct input will result in roughly 15 added to the meter so it should take around 60 inputs until the puzzle is complete.

```
void Update()
{
    if (cameraToObjectScript.playerCurrentlyInteracting && Input.GetKeyDown(KeyCode.Return))
    {
        cameraToObjectScript.playerCurrentlyInteracting = false;
        StartCoroutine(ReturnCameraToOriginalPosition());
    }

    if (cameraToObjectScript.playerCurrentlyInteracting)
    {
        if (puzzleStatus == PuzzleStatus.Broken)
        {
            StartCoroutine(FixValve());
        }
    }
}
```

```
IEnumerator FixValve()
{
    puzzleStatus = PuzzleStatus.WIP;
    string buttonToPress = "Q";
    rotateMeter = 0;
    puzzleUI.SetActive(true);

    valveRotating = false;

    while (puzzleStatus == PuzzleStatus.WIP)
    {
        if (buttonToPress == "Q" && Input.GetKeyDown(KeyCode.Q))
        {
            buttonToPress = "E";

            if(!valveRotating)
            {
                StartCoroutine(ValveRotateTimerReset());
                StartCoroutine(RotateValve());
            }

            Debug.Log("Q Pressed");
        }
        else if (buttonToPress == "E" && Input.GetKeyDown(KeyCode.E))
        {
            buttonToPress = "Q";

            if(!valveRotating)
            {
                StartCoroutine(ValveRotateTimerReset());
                StartCoroutine(RotateValve());
```

```
IEnumerator ValveRotateTimerReset()
{
    valveRotating = true;

    yield return new WaitForSeconds(0.15f);
    valveRotating = false;
}
```

```
IEnumerator RotateValve()
{
    while (valveRotating)
    {
        yield return new WaitForSeconds(0.01f);
        valve.rotation *= Quaternion.Euler(0, 0, 1.5f);
        rotateMeter += 1;

        if(rotateMeter >= 660)
        {
            StopCoroutine(FixValve());
            StopCoroutine(ValveRotateTimerReset());

            puzzleStatus = PuzzleStatus.Fixed;
            rotateMeter = 0;
            Debug.Log("Valve Fixed!");

            yield return new WaitForSeconds(0.2f);

            StartCoroutine(ReturnCameraToOriginalPosition());
            StopCoroutine(RotateValve());
        }
    }
}
```

> Users > amcal > OneDrive > Documents > Unity Projects > Space Ship Collapse > Assets > Sc

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class scr_valvePuzzle : MonoBehaviour
6   {
7       public enum PuzzleStatus { Broken, Fixed, WIP };
8
9       [Header("Puzzle Data")]
0       public PuzzleStatus puzzleStatus;
1       [SerializeField] Transform valve;
2
3       [Header("Camera Data")]
4       [SerializeField] scr_cameraToObject cameraToObjectScript;
5       [SerializeField] Transform cameraTransform;
6
7       bool valveRotating = false;
8       int rotateMeter = 0;
```

Pick Up Items Script

At this point, I won't be able to start work on the generator puzzle until I have some way of picking up the power cores so I can place them into the generator to fix it.

So first I needed to make a script that deals with what items can be picked up and where and how to drop them. The first of two script deals with this using variables from the HeldItemVariants list that I have which store the prefabs, target drop y position and not to forget the name of the object.

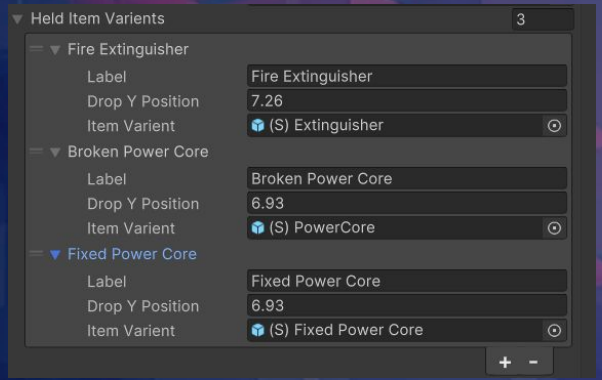
```
public class scr_holdAndDropItem : MonoBehaviour
{

    public enum HeldItem { None, Extinguisher, PowerCore_F, PowerCore_E, PowerCore_Broken };

    [Header("Holding Data")]

    public HeldItem heldItem;
    public bool isHoldingItem = false;
    private GameObject itemToDrop;
    private float itemToDropYTarget;

    [System.Serializable]
    public class HeldItemVarients
    {
        public string label;
        public float dropYPosition;
        public GameObject itemVarient;

        public HeldItemVarients(string _label, float _dropYPosition, GameObject _itemVarient)
        {
            label = _label;
            dropYPosition = _dropYPosition;
            itemVarient = _itemVarient;
        }
    }

    [SerializeField]
    public List<HeldItemVarients> heldItemVarients = new List<HeldItemVarients>();

    [Header("Player Data")]

    [SerializeField] Transform playerTransform;
```

```
if(Input.GetKeyDown(KeyCode.G) && isHoldingItem)
{
    FindDropObject();

    Vector3 dropLocation = new Vector3(playerTransform.position.x, itemToDropYTarget, playerTransform.position.z);
    Quaternion dropRotation = itemToDrop.transform.rotation;

    GameObject staticGO = Instantiate(itemToDrop, dropLocation, dropRotation);

    isHoldingItem = false;
    heldItem = HeldItem.None;
}
```

Pick Up Items Script

```
void FindDropObject()
{
    // Add code when new objects are added to list
    if(heldItem == HeldItem.Extinguisher)
    {
        itemToDrop = heldItemVarients[0].itemVarient;
        itemToDropYTarget = heldItemVarients[0].dropYPosition;
    }
    else if (heldItem == HeldItem.PowerCore_Broken)
    {
        itemToDrop = heldItemVarients[1].itemVarient;
        itemToDropYTarget = heldItemVarients[1].dropYPosition;
    }
    else if (heldItem == HeldItem.PowerCore_F)
    {
        itemToDrop = heldItemVarients[2].itemVarient;
        itemToDropYTarget = heldItemVarients[2].dropYPosition;
    }
}
```

When the player presses G to drop the item, FindDropObject() will be called to find which object prefab to instantiate and what Y position to drop it at.



Pick Up Items Script

The PickUpItem script which is attached to the object in the world before the player picks it up unlike the Hold and Drop Item script which is attached to the player, holds the data what the item will use to instantiate the object that is in the player's hand as a child of the player.

This includes holding the target position, rotation and scale of the item while it's being held aswell as what kind of item that is to be picked up as it's selected from the reference of Held Item enum from the previous script.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class scr_pickUpItem : MonoBehaviour
{
    public enum HeldItem { Extinguisher, PowerCore_Broken };

    [Header("Pick Up Data")]

    [SerializeField] GameObject itemPrefab;

    [SerializeField] Vector3 objectPositionTarget;
    [SerializeField] Vector3 objectRotationTarget;
    [SerializeField] Vector3 objectScaleTarget;

    private Transform parentObject;

    public bool destroyItem = true;

    [Header("Drop Script")]

    public scr_holdAndDropItem dropScript;
    public scr_holdAndDropItem.HeldItem itemToPickUp;

    public void Start()
    {
        parentObject = Camera.main.transform;
        dropScript = GameObject.Find("First Person Player").GetComponent<scr_holdAndDropItem>();
    }
```

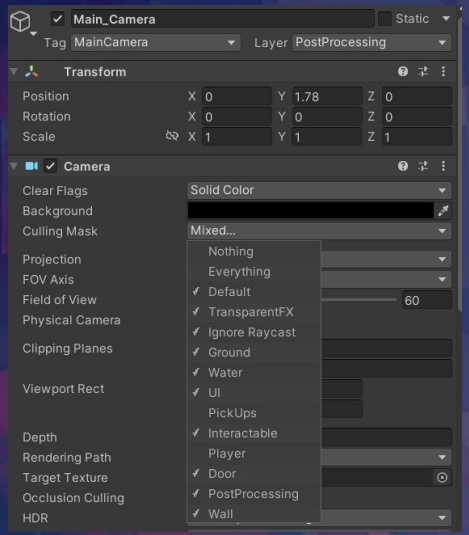
```
public void PickUpItem()
{
    if(!dropScript.isHoldingItem)
    {
        GameObject item = Instantiate(itemPrefab, parentObject.position + parentObject.TransformVector(objectPositionTarget),
        parentObject.rotation * Quaternion.Euler(objectRotationTarget), parentObject);

        item.transform.localScale = objectScaleTarget;

        dropScript.isHoldingItem = true;
        dropScript.heldItem = (scr_holdAndDropItem.HeldItem)itemToPickUp;


        if(destroyItem)
        {
            Destroy(gameObject);
        }

        destroyItem = true;
    }
}
```

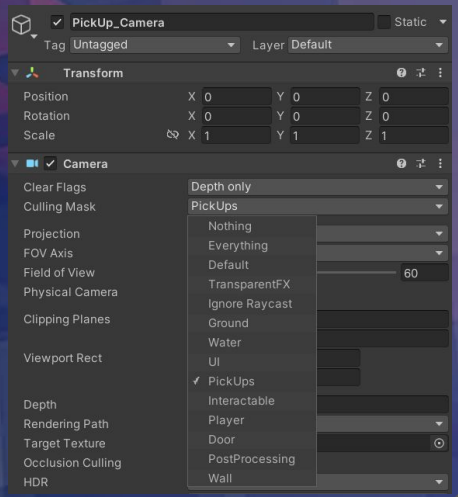
Pick Up Camera

The pick up camera is a child of the Main Camera and its purpose is to layer the picked up item in the player's hand on top of the main camera view.

This is done by altering the culling masks of the main camera and pickup camera to make sure the main camera doesn't show any gameobjects with the layer "PickUps" and the pickup camera culls everything but objects with the PickUp layer assigned.

The reason why I have done this is so the items when they're picked up won't clip through any other objects when the player is holding them.





Pick Up Camera

Fire Extinguisher held up against wall while using the pick up camera

Fire Extinguisher held up against wall without the pick up camera

Generator Puzzle

The first thing I have done with the generator puzzle is to make each of the shield doors interactable and able to open so the player can remove and replace the power cores.

I do this by adding the scr_Interactable script to each shield doors as well as add a new script called scr_shieldDoor.

In the script I included float variables that store the open and closed rotations of each shield door. Then when the shield door is interacted with it will call either OpenShieldLatch() or CloseShieldLatch() depending on the current state of the door, though both coroutines are the same in every way but their target rotation which for OpenShieldLatch() the target rotation would be openZRotation and CloseShieldLatch would be closedZRotation.

```
public class scr_shieldDoor : MonoBehaviour
{
    public bool doorOpen;
    [SerializeField] float openZRotation;
    [SerializeField] float closedZRotation;
    bool latchMoving;

    Transform shieldDoorTransform;

    public scr_Interactable interactionScript;

    void Start()
    {
        shieldDoorTransform = GetComponent<Transform>();
    }

    public void StartLatch()
    {
        if(!latchMoving)
        {
            interactionScript.message = "";

            if(doorOpen)
            {
                StartCoroutine(CloseShieldLatch());
            }
            else
            {
                StartCoroutine(OpenShieldLatch());
            }
        }
    }
```

```
public IEnumerator OpenShieldLatch()
{
    latchMoving = true;

    Quaternion startRotation = shieldDoorTransform.rotation;
    Quaternion targetRotation = Quaternion.Euler(shieldDoorTransform.eulerAngles.x, shieldDoorTransform.eulerAngles.y, openZRotation);

    float timeElapsed = 0f;
    float duration = 0.75f;

    while (timeElapsed < duration)
    {
        shieldDoorTransform.rotation = Quaternion.Lerp(startRotation, targetRotation, timeElapsed / duration);
        timeElapsed += Time.deltaTime;
        yield return null;
    }

    shieldDoorTransform.rotation = targetRotation;
    latchMoving = false;

    doorOpen = true;

    interactionScript.message = "Close (E)";
}
```

Generator Puzzle

The next thing I made is the generator puzzle script which stores each of the power cores that are part of the generator and keeps track of the generator's status by checking in a FixedUpdate if all of the power module script's core status is full and if any of them aren't then it will mark the PuzzleStatus enum to Broken.

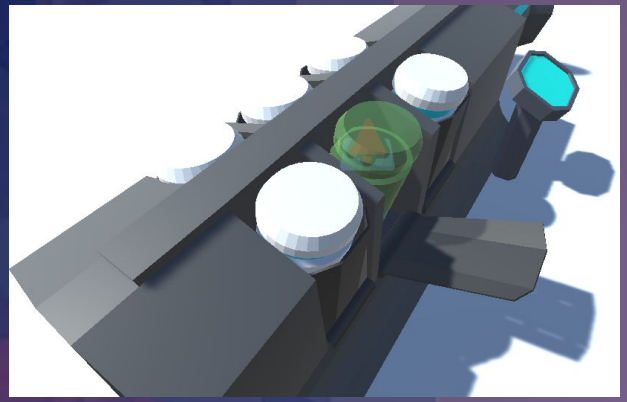
This means that my other script which I'll talk about next is the most important within the generator puzzle for the game since it handles the player's interaction with the generator by letting them insert the power cores.

```
public class scr_generatorPuzzle : MonoBehaviour
{

    public enum PuzzleStatus {Fixed, Broken};

    public enum CoreStatus {Full, Empty};

    [Header("Puzzle Data")]
    public PuzzleStatus puzzleStatus;

    [SerializeField] private GameObject powerCorePrefab;

    [System.Serializable]
    public class BatterySections
    {
        public string label;
        public GameObject batteryGameObject;
        public CoreStatus coreStatus;

        public BatterySections(string _label, GameObject _battery, CoreStatus _coreStatus)
        {
            label = _label;
            batteryGameObject = _battery;
            coreStatus = _coreStatus;
        }
    }

    [SerializeField]
    public List<BatterySections> batterySections = new List<BatterySections>();
```

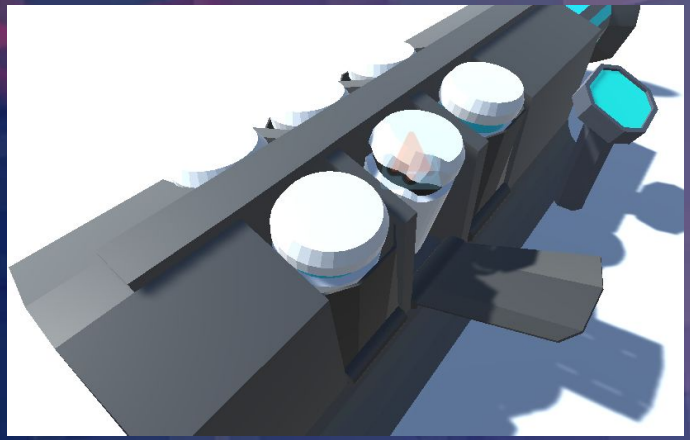
```
void FixedUpdate()
{
    bool allSectionsFull = true;

    foreach (BatterySections section in batterySections)
    {
        if (section.coreStatus != CoreStatus.Full)
        {
            allSectionsFull = false;
            break;
        }
    }

    if (allSectionsFull) puzzleStatus = PuzzleStatus.Fixed;
}
```

Generator Puzzle

Now with the powercore script, the power core doesn't get destroyed when it's picked up from the generator, instead it changes mesh. When the generator is broken the broken power cores will change it's mesh to the empty variant. If the player removes the core from the slot the powercore in the generator will change it's mesh to null, but the gameobject is still there and the script is still functioning. The script will also change the mesh to fixed and the materials to a green translucent material that I made so signify the player can place the power core as long as the interaction raycast is colliding with the gameobject and the player has a fixed power core as their held item.



Power Core when player is interacting, power core status is none and player is holding a fixed power core



Broken Power Core

Generator Puzzle

```
public class scr_powerCore : MonoBehaviour
{
    [Header("Player Interaction Scripts")]

    [SerializeField] private scr_playerInteraction playerInteractionScript;
    [SerializeField] private scr_holdAndDropItem playerPickUpScript;
    [SerializeField] private scr_pickUpItem pickUpItemScript;
    [SerializeField] private scr_Interactable interactableScript;

    [Header("Generator Script")]

    [SerializeField] private scr_generatorPuzzle generatorScript;
    private int powerCoreID;

    [Header("Power Core Materials")]
    [SerializeField] private Material selectMaterial;
    [SerializeField] private Material element1;
    [SerializeField] private Material element2;
    [SerializeField] private Material lightOffMaterial;

    [Header("Power Core Meshes")]
    [SerializeField] private Mesh fixedMesh;
    [SerializeField] private Mesh brokenMesh;

    [Header("Shield and Power Core Settings")]
    [SerializeField] private scr_shieldDoor shieldLatch;
    [SerializeField] public PowerCoreStatus powerCoreStatus;

    public enum PowerCoreStatus { None, Empty, Running };
```

```
    void Start()
    {
        int count = 0;

        while (count < generatorScript.batterySections.Count)
        {
            if (generatorScript.batterySections[count].batteryGameObject == gameObject)
            {
                powerCoreID = count;
                break;
            }
            count++;
        }

        Debug.Log(gameObject.name + " Power Core ID = " + powerCoreID);
    }
```

```
void Update()
{
    if(playerInteractionScript.currentInteractable != null)
    {
        if(playerInteractionScript.currentInteractable.gameObject == gameObject)
        {
            if(powerCoreStatus == PowerCoreStatus.None && playerPickUpScript.heldItem == scr_holdAndDropItem.HeldItem.PowerCore_F && shieldLatch.doorOpen)
            {
                Material[] mats = GetComponent<Renderer>().materials;
                mats[0] = selectMaterial;
                mats[1] = selectMaterial;

                GetComponent<Renderer>().materials = mats;
                GetComponent<MeshFilter>().mesh = fixedMesh;

                interactableScript.message = "Place (E)";

                if(Input.GetKeyDown(KeyCode.E))
                {
                    generatorScript.batterySections[powerCoreID].coreStatus = scr_generatorPuzzle.CoreStatus.Full;
                    powerCoreStatus = PowerCoreStatus.Running;
                    playerPickUpScript.heldItem = scr_holdAndDropItem.HeldItem.None;
                    playerPickUpScript.isHoldingItem = false;

                    mats[0] = element1;
                    mats[1] = element2;
                    GetComponent<Renderer>().materials = mats;

                    Destroy(FindObjectOfType<scr_destroyItem>().gameObject);
                }
            }
            else if(powerCoreStatus == PowerCoreStatus.Empty && playerPickUpScript.heldItem == scr_holdAndDropItem.HeldItem.None && shieldLatch.doorOpen)
            {
                interactableScript.message = "Pick Up (E)";

                if(Input.GetKeyDown(KeyCode.E))
                {
                    powerCoreStatus = PowerCoreStatus.None;

                    playerPickUpScript.heldItem = scr_holdAndDropItem.HeldItem.PowerCore_Broken;
                    pickUpItemScript.itemToPickUp = scr_holdAndDropItem.HeldItem.PowerCore_Broken;
                    pickUpItemScript.PickUpItem();
                }
            }
            else if(powerCoreStatus == PowerCoreStatus.None || powerCoreStatus == PowerCoreStatus.Running)
            {
                interactableScript.message = "";
            }
        }
    }
}

void FixedUpdate()
{
    if(generatorScript.batterySections[powerCoreID].coreStatus == scr_generatorPuzzle.CoreStatus.Empty && powerCoreStatus != PowerCoreStatus.None)
    {
        powerCoreStatus = PowerCoreStatus.Empty;

        Material[] mats = GetComponent<Renderer>().materials;
        mats[1] = lightOffMaterial;

        GetComponent<Renderer>().materials = mats;

        if(generatorScript.puzzleStatus != scr_generatorPuzzle.PuzzleStatus.Broken)
        {
            generatorScript.puzzleStatus = scr_generatorPuzzle.PuzzleStatus.Broken;
        }
    }
    else if(powerCoreStatus != PowerCoreStatus.None)
    {
        powerCoreStatus = PowerCoreStatus.Running;
    }

    if(powerCoreStatus == PowerCoreStatus.Running)
    {
        GetComponent<MeshFilter>().mesh = fixedMesh;
    }
    else if(powerCoreStatus == PowerCoreStatus.Empty)
    {
        GetComponent<MeshFilter>().mesh = brokenMesh;
    }
    else
    {
        GetComponent<MeshFilter>().mesh = null;
    }
}
```

Interaction Script Alteration

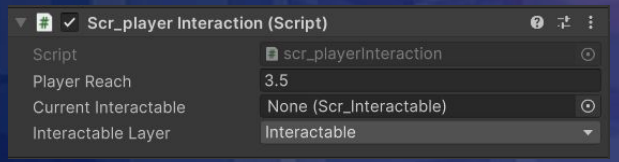
Recently when testing the generator puzzle I had noticed that when I try and pick up the fixed power core off the ground and even the fire extinguisher item, that the interaction text wouldn't show when facing the item where the raycast hits the top of the collider. This meant I couldn't interact with any of the items if I was looking at them from a certain angle even if it was in range.

To fix this issue I had to revamp the player interaction script to detect objects that were part of a layermask that I made instead of trying to look for objects with the tag "Interactable".

All I had to do for this was create a LayerMask variable, set it to only pick up objects on the "Interactable" Layer and change the if statement in the update function to:

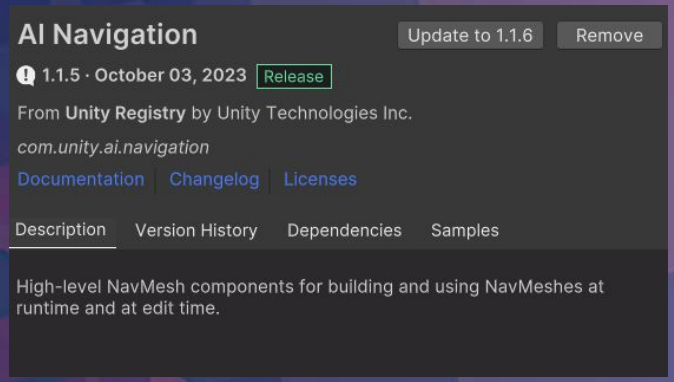
```
if (Physics.Raycast(ray, out hit, playerReach, interactableLayer))
```

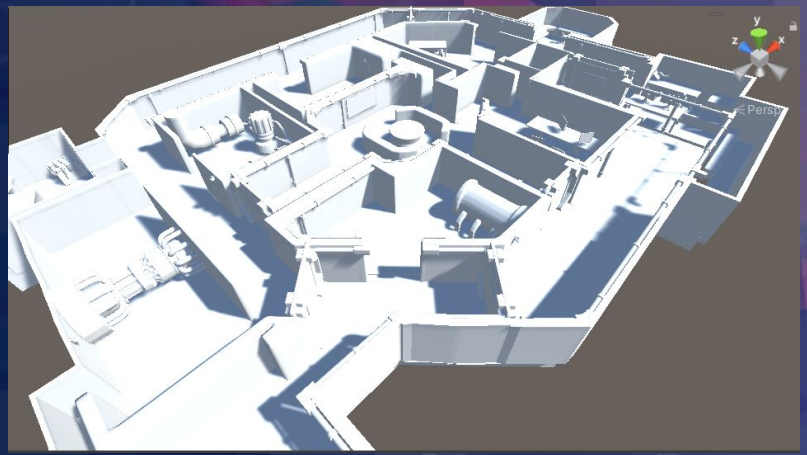
```
void CheckInteraction()
{
    Vector3 rayOrigin = Camera.main.transform.position + Camera.main.transform.forward * 0.1f;
    Vector3 rayDirection = Camera.main.transform.forward.normalized;
    Ray ray = new Ray(rayOrigin, rayDirection);

    RaycastHit hit;
    if (Physics.Raycast(ray, out hit, playerReach, interactableLayer))
    {
        if (hit.collider.CompareTag("Interactable"))
        {
            scr_Interactable newInteractable = hit.collider.GetComponent<scr_Interactable>();

            if (newInteractable != null && newInteractable.enabled)
            {
                if (newInteractable != currentInteractable)
                {
                    Debug.Log("New interactable detected: " + newInteractable.name);
                }
                SetNewCurrentInteractable(newInteractable);
            }
            else
            {
                DisableCurrentInteractable();
            }
        }
```



Alien AI

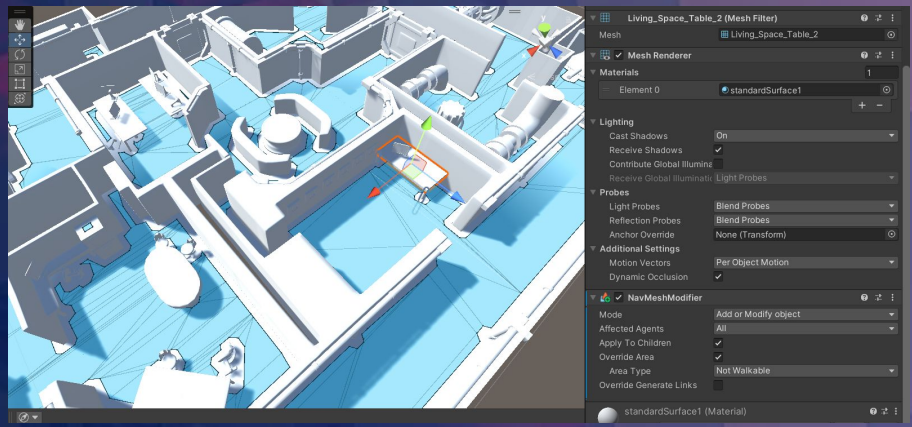
Now that Hayden has sent over the map for the game, and I have imported that map into the scene I can start work on the Alien movement behaviour. For this I will be using the AI Navigation Unity package to move the alien around the map without walking straight into walls.





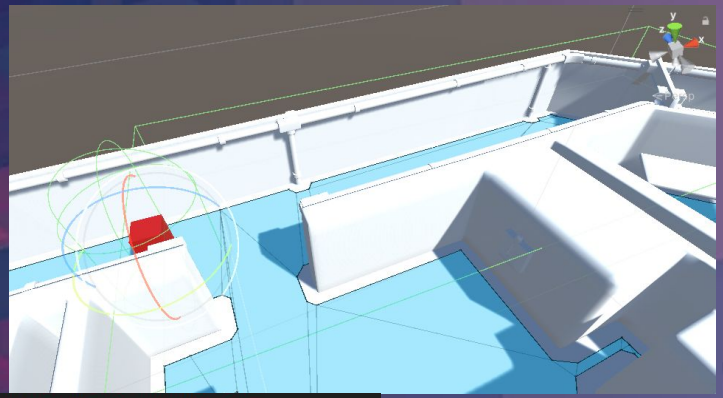
Alien AI

I decided to start off with making a NavMesh surface for the map, I did this by creating a new empty GameObject and adding the NavMesh surface component onto it. From there I clicked on bake which then lined out an area within the map that will act as a plane for where the alien will be able to walk. I did notice that some of the tables and walls had some blue areas signifying the alien could walk on it so I selected them and added the NavMeshModifier component, enabled override area and set it to not walkable, this made it so these table, box and wall objects weren't accessible to the alien to walk on in the future.



Alien AI

The way on how the Alien Behaviour script is going to work is I have added a child game object to the alien (which is currently just a cube) called “Sight which has a box and capsule collider attached to it. There is also a script attached to it that tracks if the player is within one of the colliders. If the player is in a collider, it will enable a bool within the alien behaviour script that will then draw a raycast from the alien to the player.



```
public class DetectionZoneCheck : MonoBehaviour
{

    public AlienBehaviour alienBehaviour;

    void OnTriggerEnter(Collider collider)
    {
        if(collider.gameObject.CompareTag("Player"))
        {
            alienBehaviour.playerInDetectionZone = true;
            Debug.Log("In Detection Zone");
        }
    }

    void OnTriggerExit(Collider collider)
    {
        if(collider.gameObject.CompareTag("Player"))
        {
            alienBehaviour.playerInDetectionZone = false;
            Debug.Log("Exit Detection Zone");
        }
    }
}
```

Alien AI

The Alien Behaviour script will then cause the alien to engage the player if the raycast hits the player gameobject. Engage() is a coroutine in the script that will continue running as long as the isEngaging bool is true and during which will continuously set the NavMesh agent's Set Destination to the player's position. This coroutine will only be turned off when the Disengage() coroutine is called, in which after the bool is disabled, the SetDestination will be set to the alien's current location after 2 seconds which stops the alien from moving since they're reached their destination.

During the 2 second wait the alien will continue moving to the player's location meaning that if the player stays out of sight for those 2 seconds the alien will stop chasing them.

I also have added a conditional that will adjust the alien's rotation to the direction they're moving so that when they're roaming around the ship or chasing the player, they'll be looking where they're going.

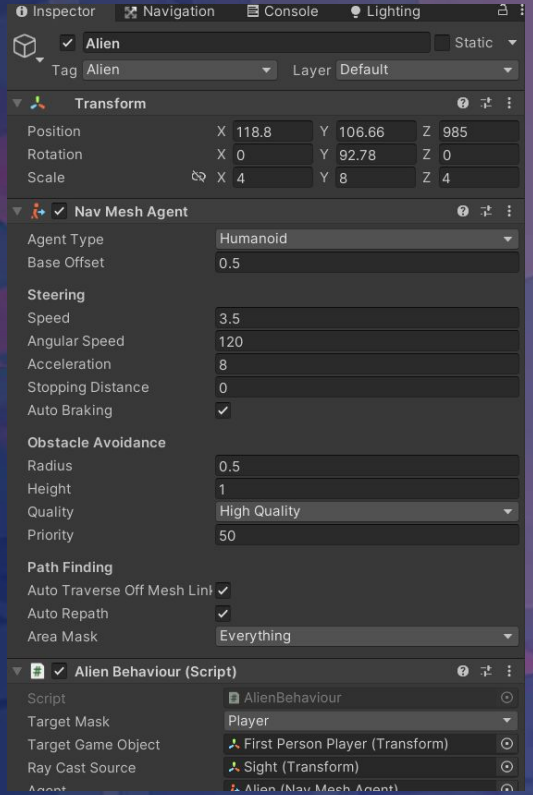
```
public class AlienBehaviour : MonoBehaviour
{
    public LayerMask targetMask;
    [SerializeField] private Transform targetGameObject;

    [SerializeField] private Transform rayCastSource;
    [SerializeField] private BoxCollider detectionZone;
    [SerializeField] private NavMeshAgent agent;

    private bool isEngaging = false;
    public bool playerInDetectionZone;
```

```
void Update()
{
    if (playerInDetectionZone)
    {
        Vector3 direction = (targetGameObject.position - rayCastSource.position).normalized;

        RaycastHit hit;
        if (Physics.Raycast(rayCastSource.position, direction, out hit, Mathf.Infinity))
        {
            Debug.DrawRay(rayCastSource.position, direction * 100f, Color.red);
            Debug.Log("Drawing Raycast");
        }

        if (hit.collider != null && hit.collider.gameObject == targetGameObject.gameObject)
        {
            Debug.Log("Player in direct line of sight");
            if (!isEngaging)
            {
                StartCoroutine(Engage());
            }
        }
        else
        {
            Debug.Log("Player no longer in sight");
            if (isEngaging)
            {
                StartCoroutine(Disengage());
            }
        }
    }
```

```
if (agent.velocity.sqrMagnitude > 0.1f)
{
    Quaternion targetRotation = Quaternion.LookRotation(agent.velocity.normalized, Vector3.up);
    transform.rotation = Quaternion.Slerp(transform.rotation, targetRotation, Time.deltaTime * 5f);
}
```

Alien AI

```
public IEnumerator Engage()
{
    isEngaging = true;
    agent.speed = 3.5f; // Running speed

    while (isEngaging)
    {
        Debug.Log("Is Engaging");
        agent.SetDestination(targetGameObject.position);
        yield return null;
    }
}

public IEnumerator Disengage()
{
    isEngaging = false;
    agent.speed = 1.5f; // Walking speed

    yield return new WaitForSeconds(2f);
    agent.SetDestination(transform.position);
}
```

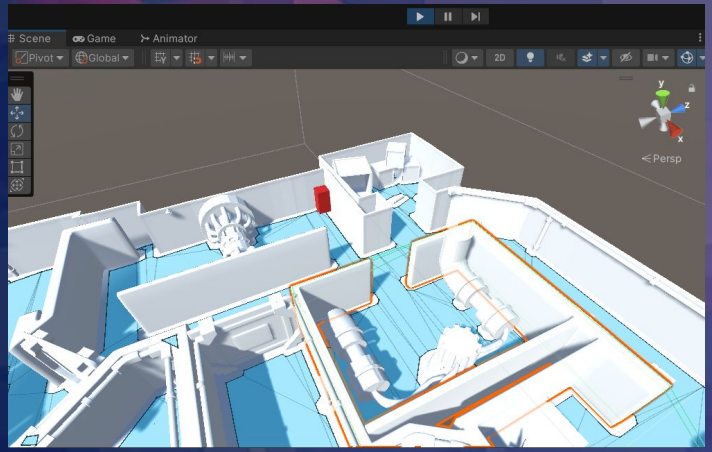


Alien AI

After writing the alien behaviour script and tested it in the scene I have noticed something wrong with the alien, for some reason the alien kept teleporting to the exterior wall of the ship right when the game started, I figured that the alien agent just wasn't snapping to the NavMeshSurface, not knowing what to do I asked ChatGPT how to make sure the alien snaps to the NavMesh and I added what it gave me to the script.

This however didn't work, though I did end up finding what was wrong because when I walked around the ship using the first person player I had trouble entering rooms and noticed I was colliding with something. The issue I discovered was I set all of the walls mesh colliders' corvex to true and disabling it seems to have fixed the issue.

```
void Start()
{
    NavMeshHit hit;
    if (NavMesh.SamplePosition(transform.position, out hit, 10f, NavMesh.AllAreas))
    {
        agent.Warp(hit.position);
        Debug.Log("Warped agent to closest NavMesh point.");
    }
    else
    {
        Debug.LogWarning("No NavMesh nearby to warp to.");
    }
```



Alien AI

Since finishing the issue with the ship colliders I have added a bit more to the Alien Behaviour script. The feature that I've added is the roaming mechanic, this mechanic checks every second a 1 in 4 chance to send the alien to a random room within the ship using SetDestination and a list of Vector3s.

Once the alien reaches the destination they will stay there until the MovementCheck() coroutine sends the alien somewhere else.

This feature is required for the final product since it wouldn't be very entertaining if the alien stayed in one place until they see the player from which they start chasing them.

```
void Update()
{
    if (alienStatus != AlienStatus.Idle)
    {
        StopCoroutine(MovementCheck());
        isCheckingMovement = false;
    }

    if (!agent.pathPending && agent.remainingDistance <= agent.stoppingDistance + 0.1f)
    {
        alienStatus = AlienStatus.Idle;
        agent.SetDestination(transform.position);
    }
```

```
IEnumerator MovementCheck()
{
    if (isCheckingMovement) yield break;

    isCheckingMovement = true;

    while (true)
    {
        int randomNumber = Random.Range(1, 5);
        Debug.Log("Movement Check Ran. Number: " + randomNumber);

        if (randomNumber == 1 && alienStatus == AlienStatus.Idle)
        {
            int destinationNumber = Random.Range(0, possibleRoamPositions.Length);
            Vector3 newDestination = new Vector3(
                possibleRoamPositions[destinationNumber].x,
                transform.position.y,
                possibleRoamPositions[destinationNumber].z
            );

            agent.SetDestination(newDestination);
            alienStatus = AlienStatus.Roaming;
            agent.speed = 8f;
        }

        yield return new WaitForSeconds(1f);
    }
}
```

Alien AI

The added variable AlienStatus was also added to make the roam feature possible which I'm sure would be useful to have whenever I start working on implementing the actual alien model and animations once Leon gets them finished and textured.

I also feel it's important to reference the video I found out about the AI Navigation package from and how to use it so here it is below:

youtube.com/watch?v=atCOd4o7tG4

```
public enum AlienStatus { Roaming, Idle, Engaging, Disengaging };

public bool isEngaging = false;
public bool playerInDetectionZone;
public AlienStatus alienStatus = AlienStatus.Idle;
```

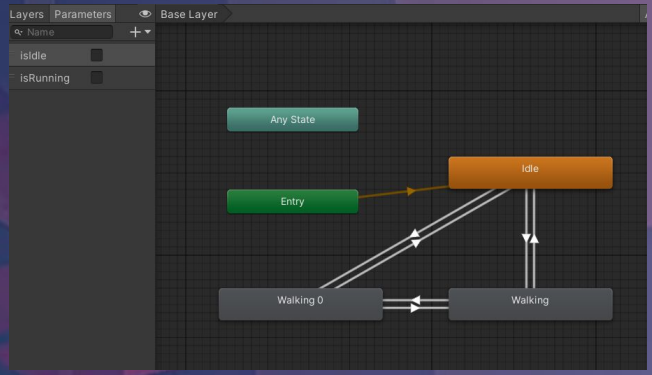
Alien Model and Animations

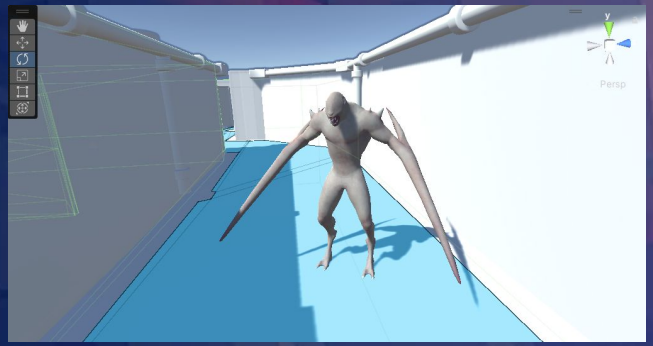
Since I wrote the alien behaviour script I had taken a break to do work on my Professional Portfolio Project module. This gave Leon to get the alien model done and Mikaela who took up doing the animations to have them finished.

Leon sent me over a fbx of the model first which I imported into the scene and applied the textures to the mesh that came with the alien that Leon sent over.

A week later I got sent two more fbx files of the alien but with a walk/run animation and idle animation which I was able to move onto the animator window of the gameobject I already had in the scene.

Since Mikaela didn't make a separate animation for running since they needed to also do work for their portfolio, I just used the same animation as walking but upped the animation speed for it to match the speed of the alien when engaging.





Alien Model and Animations

After adding the animations I now have added some SetBools into the Alien Behaviour script to control which animation is playing.

Animation Bool Conditions:

- Idle Animation: isIdle = true & isRunning = false.
- Walk Animation: isIdle = false & isRunning = false.
- Running Animation = isIdle = false & isRunning = true.

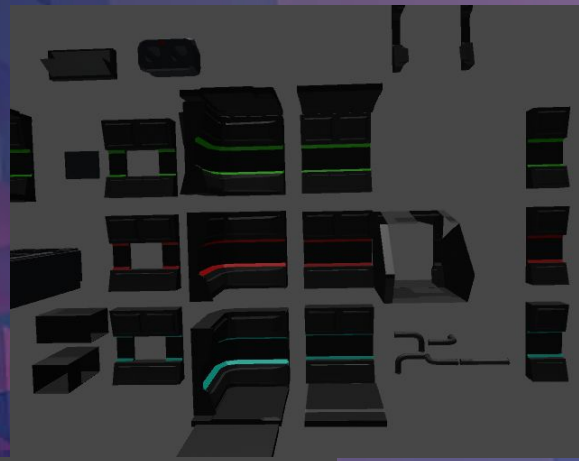
```
public IEnumerator Engage()
{
    alienStatus = AlienStatus.Engaging;
    animator.SetBool("isIdle", false);
    animator.SetBool("isRunning", true);
    isEngaging = true;
    agent.speed = 10f; // Running speed

    int random = Random.Range(0, 2);

    while (isEngaging)
    {
        Debug.Log("Is Engaging");
        agent.SetDestination(targetGameObject.position);
        yield return null;
    }
}
```

```
public IEnumerator Disengage()
{
    isEngaging = false;

    yield return new WaitForSeconds(4f);
    agent.SetDestination(transform.position);

    alienStatus = AlienStatus.Idle;
    animator.SetBool("isIdle", true);
    animator.SetBool("isRunning", false);
}
```

```
void Start()
{
    NavMeshHit hit;
    if (NavMesh.SamplePosition(transform.position, out hit, 10f, NavMesh.AllAreas))
    {
        agent.Warp(hit.position);
        Debug.Log("Warped agent to closest NavMesh point.");
    }
    else
    {
        Debug.LogWarning("No NavMesh nearby to warp to.");
    }

    animator = GetComponent<Animator>();
    animator.SetBool("isIdle", true);
    animator.SetBool("isRunning", false);

    StartCoroutine(MovementCheck());
```

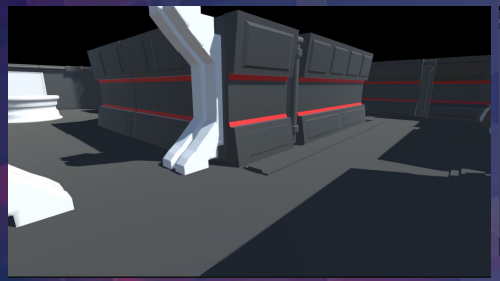
Adding Modular Walls

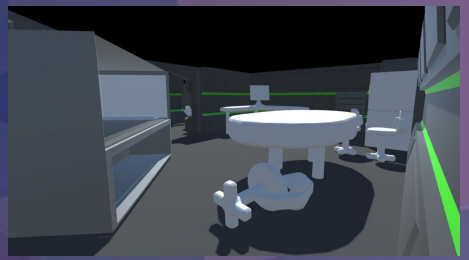
It was decided early on in the project that we were going to use modular walls in addition to the map that Hayden made for the scene, this meant that someone had to make the walls which Mikaela was decided to be in charge of. Mikaela sent over some additional assets for the map such as vents, the alien cage, some assets for the cockpit and some smaller items such as energy drink cans and folders. I used some of these assets as well as the assets Mikaela sent alongside the puzzles to the environment for the safe room and cockpit.

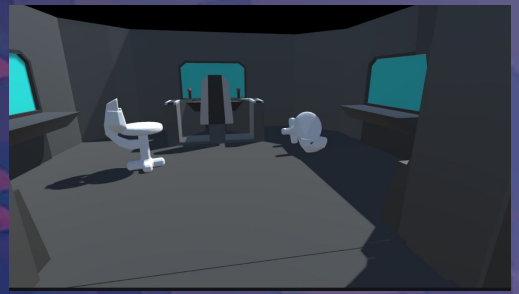


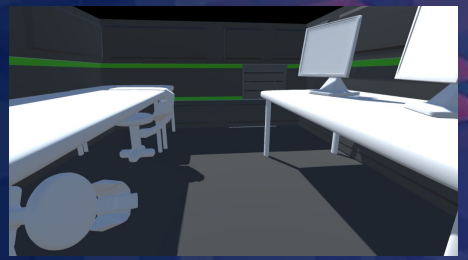
Adding Modular Walls

After revamping the cockpit and the safe room I added the modular walls in the scene and placed them throughout. The walls for the vents were also added during this process which were outlined on the map design Mikaela made in the beginning for the proposal. One of the challenges that I faced was that some of the base walls weren't all straight so I had to rotate the walls in non-90 degree so that the base walls of Haydens map wouldn't clip through the modular walls. This came with the side effect that some of the modular walls don't connect seamlessly together, but aesthetically didn't turn out that bad.





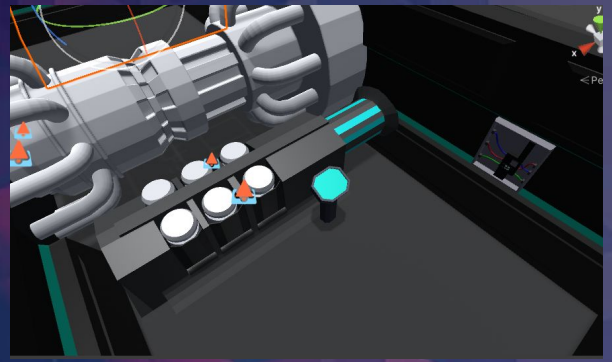




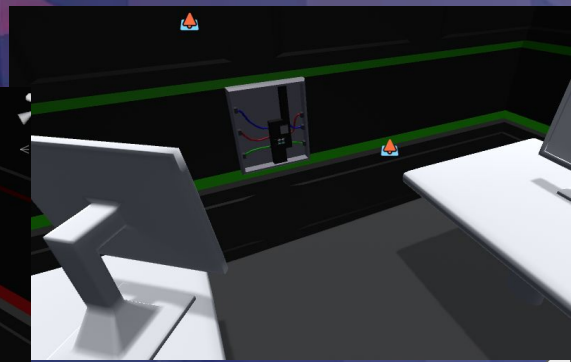
To match the original map walls Hayden sent I changed the materials of them to the same materials used in Mikaela assets so that base walls don't stand out too much whenever they're visible.

Adding the Puzzles to the Scene

During adding the modular walls, I also duplicated the puzzles that I made already onto the ship as well the section for a pipe puzzle that Mikaela sent with the original puzzles that I haven't got around to making a script for yet but I will after working on the map some more.



At this point none of the puzzles have any impact on the ship's environment like stopping doors from opening or starting fires, features that were mentioned in the original map design Mikaela made, and at this point I'm not sure if I'm going to be able to add all those features in the game by deadline.



Vent Behaviour

After finishing adding the map I have moved on to making a script that allows the player to travel through the vents, the way I decided to do this is by creating an empty game object called Vent Hinge which will act as the hinge of the vent door.

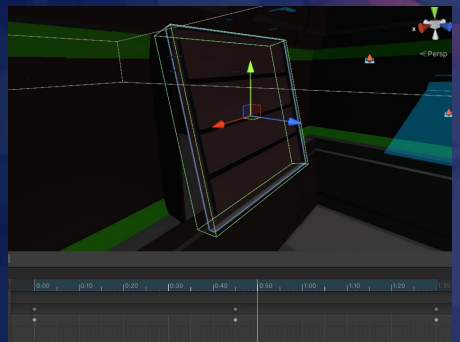
I've also added the interactable script to the hinge and extended a box collider to the dimensions of the vent model, this is so that even though the gameobject doesn't have a mesh it can still be interacted with by the player.

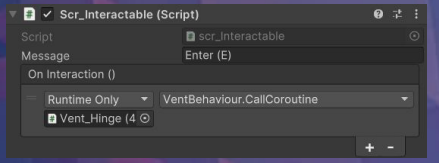
The way the inside of the vents work is that they don't actually have any collisions, instead I use the collisions of the base map model Hayden made to act as the walls of the vent, this means I don't have to shrink the player or anything to make them fit inside of the vent or even make a separate camera each vent.

When the player interacts with the vent they will teleport them inside or outside the vent depending of a new bool I added to player movement called isInVent. In addition, I have also made a short animation that will play when the vent is opened.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class VentBehaviour : MonoBehaviour
{

    [SerializeField] private Vector3 insideVentPos;
    [SerializeField] private Vector3 outsideVentPos;
    [SerializeField] private Transform player;
    [SerializeField] private scr_playerMovement movementScript;
    private CharacterController controller;

    private Animator animator;
    private bool ventOpening;

    // Start is called before the first frame update
    void Start()
    {
        animator = GetComponent<Animator>();
        controller = player.gameObject.GetComponent<CharacterController>();
    }

    public void CallCoroutine()
    {
        if (ventOpening) return;

        if (player.gameObject.GetComponent<scr_holdAndDropItem>().isHoldingItem) return;

        StartCoroutine(OpenVent());
        ventOpening = true;
        animator.SetBool("isOpen", true);
    }
```

```
IEnumerator OpenVent()
{
    movementScript.enabled = false;
    controller.enabled = false;

    yield return new WaitForSeconds(0.05f);
    animator.SetBool("isOpen", false);

    yield return new WaitForSeconds(1.55f);

    movementScript.isInVent = !movementScript.isInVent;

    Vector3 placeToMove = movementScript.isInVent ? insideVentPos : outsideVentPos;
    player.position = placeToMove;

    ventOpening = false;

    movementScript.enabled = true;
    controller.enabled = true;
}
```





Door Behaviour

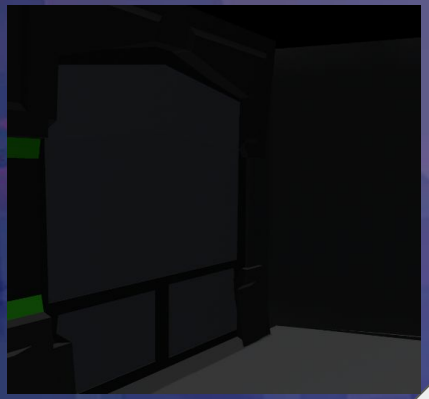
Like the vent behaviour, the door script triggers an animation based when interacted with but instead of using the interaction script it's triggered by a trigger box collider to see whether or not the player or the alien is inside. If either characters are inside of the trigger the door will play the rise animation and when done will transfer to the open animation which is just a short animation for the door position to be while it's open until there's no more characters in the collider.

In the script I made it so it tracks how many characters are in the collider, this was so if the player and alien are in the collider at the same time and the player leaves the door won't shut until the alien leaves as well. Right now this script is only attached to the doors for the different rooms and not the large doors placed in the map that splits the corridors that hayden made with the map.

The door models that are used for these doors were made by Mikaela after I asked for them to be made to match the door frame model they'd already given me.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class AutomatedDoor : MonoBehaviour
{
    private int collidingCharacters;
    private Animator animator;

    void Start()
    {
        animator = GetComponent<Animator>();
        StartCoroutine(StatusCheck());
    }

    void OnTriggerEnter(Collider collider)
    {
        if (collider.CompareTag("Player") || collider.CompareTag("Alien"))
        {
            collidingCharacters++;
        }
    }

    void OnTriggerExit(Collider collider)
    {
        if (collider.CompareTag("Player") || collider.CompareTag("Alien"))
        {
            collidingCharacters--;

            if(collidingCharacters < 0)
            {
                collidingCharacters = 0;
            }
        }
    }
```

```
IEnumerator StatusCheck()
{
    while(true)
    {
        if(collidingCharacters == 0)
        {
            animator.SetBool("doorOpen", false);
        }
        else
        {
            animator.SetBool("doorOpen", true);
        }

        yield return new WaitForSeconds(0.1f);
    }
}
```

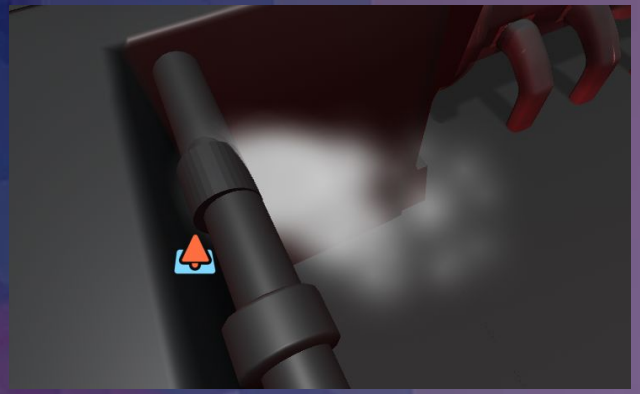


Valve Puzzle Particles

Next up, I figured that I should make use of the particle system in the game, so I created some particle systems in the rooms where the valve puzzles are placed at the pipes and machines within either room.

The particle effect that I made is a steam effect that is going to only trigger when the valve puzzle is broken and it should be easy to get an audio for since I could do something with a hairspray or deodorant can.

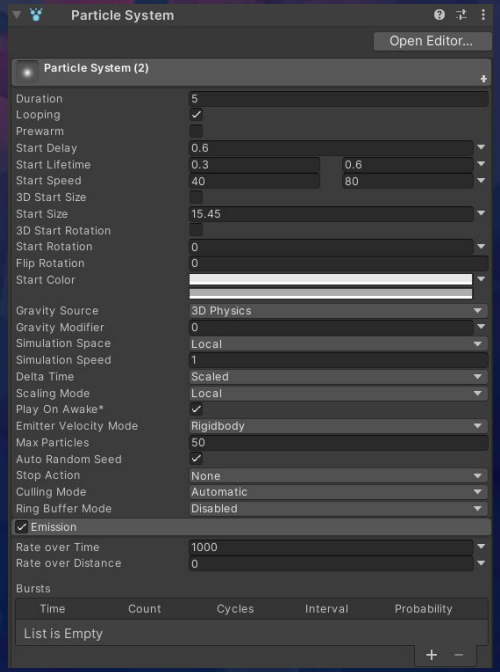
To make this work, all I needed to do in the script was add a if and if else statement to activate and deactivate the gameobjects the particle systems are attached to based on whether or not the puzzle is broken or fixed.

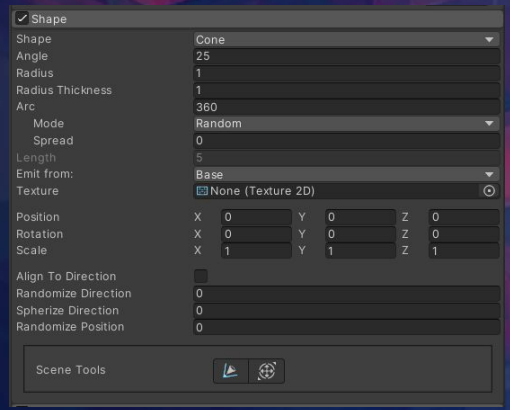


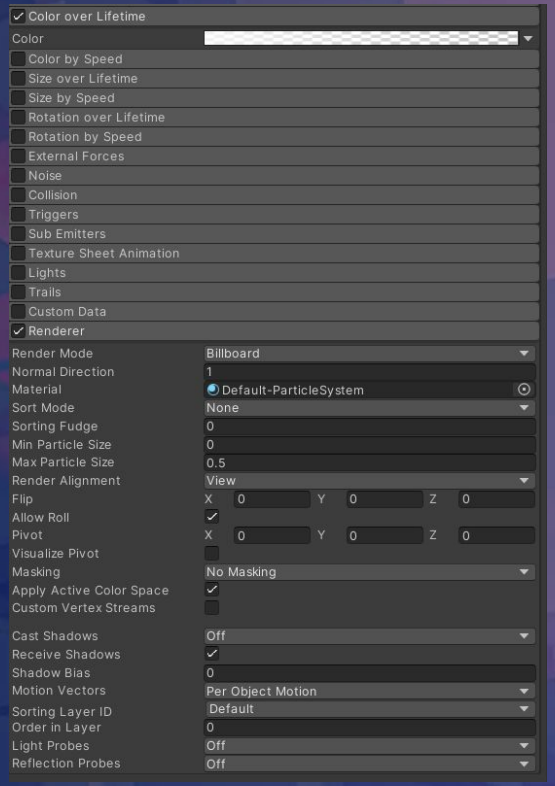
```
[SerializeField] List<GameObject> particleSystems = new List<GameObject>();
```

```
void FixedUpdate()
{
    if (puzzleStatus == PuzzleStatus.Fixed)
    {
        foreach (GameObject smoke in particleSystems)
        {
            smoke.SetActive(false);
        }
    }
    else if (puzzleStatus == PuzzleStatus.Broken)
    {
        foreach (GameObject smoke in particleSystems)
        {
            smoke.SetActive(true);
        }
    }
```

Valve Puzzle Particles







Pipe Script

With the last puzzle model that Mikaela sent, I made a script that lets the player replace a broken pipe when the it's broken. For the puzzle however, when removing the broken pipe it only has a 1 in 6 chance of succeeding, meaning the player has to keep trying to remove it until it succeeds.

If the removal doesn't succeed, the script will run the Shake() coroutine which will as you guessed, shake the broken pipe. For the removal and replacement of the pipe the script uses the same kind of coroutines as the fuse puzzle script does by using lerp.

Since I also have a steam particle effect in the scene, I decided that it would be a good move to incorporate it into the pipe puzzle as well by adding one particle system that is enabled when the pipe gets broken coming out of the broken pipe, and two more that are enabled when the broken pipe is removed that come out of the two ends of the adjacent pipe exits.

```
public enum PuzzleStatus { Perfect, Broken, Fixed, Empty };

[Header("Puzzle Data")]
public PuzzleStatus puzzleStatus;

[System.Serializable]
public class Pipe
{
    public string label;
    public GameObject pipeGameObject;
}

[SerializeField] private List<Pipe> pipes = new List<Pipe>();
[SerializeField] private GameObject[] particleSystems;
[SerializeField] private GameObject[] fires;

private GameObject perfectPipe;
private GameObject brokenPipe;
private GameObject fixedPipe;

[SerializeField] GameObject brokenParticles;

[Header("Camera Data")]
[SerializeField] scr_cameraToObject cameraToObjectScript;
[SerializeField] Transform cameraTransform;

private bool isShaking;
private bool pipeInPlace;
private bool restrictExit;
private bool settingFires;
```

```
IEnumerator Shake (GameObject shakeObject)
{
    isShaking = true;
    restrictExit = true;

    Vector3 originalPos = shakeObject.transform.localPosition;
    Quaternion originalRot = shakeObject.transform.localRotation;
    int shakes = 0;

    while (shakes < 5)
    {
        float x = Random.Range(-1f, 1f) * 0.0005f;
        float y = Random.Range(-1f, 1f) * 0.0005f;
        float z = Random.Range(-1f, 1f) * 0.00005f;

        shakeObject.transform.localPosition = originalPos + new Vector3(x, y, z);

        float rotZ = Random.Range(-1f, 1f) * 0.005f;

        yield return new WaitForSeconds(0.08f);
        shakes++;
        yield return null;
    }

    shakeObject.transform.localPosition = originalPos;
    shakeObject.transform.localRotation = originalRot;
    isShaking = false;
    restrictExit = false;
}
```

```
if(cameraToObjectScript.playerCurrentlyInteracting && Input.GetKeyDown(KeyCode.E) && !isShaking)
{
    int random = Random.Range (0, 6);

    GameObject pipeToAlter = null;

    switch (puzzleStatus)
    {
        case PuzzleStatus.Perfect:
            pipeToAlter = perfectPipe;
            break;
        case PuzzleStatus.Broken:
            pipeToAlter = brokenPipe;
            break;
        case PuzzleStatus.Fixed:
            pipeToAlter = fixedPipe;
            break;
    }

    if(random == 0)
    {
        isShaking = true;
        StartCoroutine(RemovePipe(pipeToAlter));
    }
    else StartCoroutine(Shake(pipeToAlter));
}
else if (cameraToObjectScript.playerCurrentlyInteracting && Input.GetKeyDown(KeyCode.E) && puzzleStatus == PuzzleStatus.Empty)
{
    StartCoroutine(AddPipe(fixedPipe));
}
```

Part of the Update() function

Pipe Script

I have also added a system that sets fires around the map if the puzzle is broken for too long. I created the fire particle system before adding this addition though I don't have a way of actually extinguishing the fires yet.

The SetFires() IEnumerator is called when the puzzle is first broken and contains a while statement that will keep running until the puzzle is fixed. In the while statement (`while (settingFires == true)`) the first line is a WaitForSeconds and then the fires are enabled after, this means that the fires won't be enabled if the puzzle is fixed before the waitforseconds is complete. This is because the settingFires bool will be disabled when the puzzle status is fixed or perfect within the Fixed Update.

```
void FixedUpdate()
{
    if (puzzleStatus == PuzzleStatus.Perfect || puzzleStatus == PuzzleStatus.Fixed)
    {
        gameObject.GetComponent<scr_Interactable>().enabled = false;
    }
    else
    {
        gameObject.GetComponent<scr_Interactable>().enabled = true;
    }


    brokenParticles.SetActive(brokenPipe.activeSelf);

    if(!settingFires && puzzleStatus == PuzzleStatus.Broken ||
        !settingFires && puzzleStatus == PuzzleStatus.Empty) StartCoroutine(SetFires());
    else if (settingFires && puzzleStatus == PuzzleStatus.Perfect
        || settingFires && puzzleStatus == PuzzleStatus.Fixed) settingFires = false;
```

```
public IEnumerator SetFires()
{
    settingFires = true;
    int random = Random.Range(240, 361);

    while (settingFires)
    {
        yield return new WaitForSeconds(random);

        foreach(GameObject fire in fires)
        {
            fire.SetActive(true);
        }
    }
}
```

Fire Behaviour

Now that I have the fire gameobjects I need some way of extinguishing them. Firstly, I attached a particle system to the player character that will only be active when the player's left mouse button is down and they're holding the fire extinguisher.

Second part of the added code on the player hold and drop item script is a check that enables and disables a box collider attached to the same object as the particle effect that will act as a trigger to decrease the fire's health.

After adding the bit of code to the item script I wrote the fire behaviour script to check if the box collider from the extinguisher particle effect gameobject is colliding with the fire gameobject and if it is it will decrease its health until 0 at which then it will disable itself.

```
if (heldItem == HeldItem.Extinguisher)
{
    emission = extinguisherEffect.GetComponent<ParticleSystem>().emission;

    if (Input.GetMouseButton(0))
    {
        emission.rateOverTime = 900f;
        extinguisherEffect.gameObject.GetComponent<BoxCollider>().enabled = true;
    }
    else
    {
        emission.rateOverTime = 0f;
        extinguisherEffect.gameObject.GetComponent<BoxCollider>().enabled = false;
    }

}
else
{
    extinguisherEffect.gameObject.GetComponent<BoxCollider>().enabled = false;
}
```

```
void OnEnable()
{
    health = 100;
    gettingExtinguished = false;
    StartCoroutine(HealthManager());
}

void OnTriggerStay(Collider collider)
{
    if(collider.CompareTag("Extinguisher Foam"))
    {
        gettingExtinguished = true;
        Debug.Log("Fire is being extinguished");
    }
}

void OnTriggerExit(Collider collider)
{
    if(collider.CompareTag("Extinguisher Foam"))
    {
        gettingExtinguished = false;
    }
}
```

```
IEnumerator HealthManager()
{
    while (true)
    {
        fireMapUI.gameObject.SetActive(true);

        if (gettingExtinguished)
        {
            health -= 3;
            Debug.Log("Current fire health: " + health);

            if (health <= 0)
            {
                Debug.Log("Fire extinguished!");

                gameObject.SetActive(false);
                yield break;
            }
        }

        yield return new WaitForSeconds(0.1f);
    }
}
```

Player Death

With the alien working and the fire in the game, it is about time I have a way to call the player's death. To do this, I decided to make a post process volume to layer over the player camera when the player dies. The volume's intensity will gradually increase when the player kill coroutine is called during which the character controller will be disabled and the mouseLock will be enabled.

The PlayerDeath() Coroutine is within a new script called playerBehaviour which will handle the player's interactions outside of the environment (like puzzles or vents) and movement such as menus and the player's status.

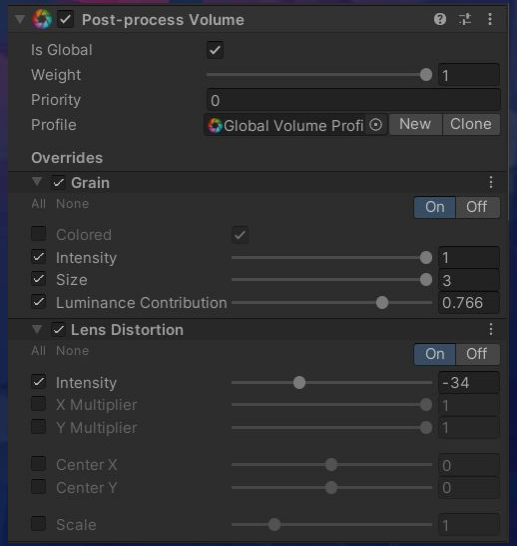
After making the coroutine I created another new script called the alien kill trigger which is attahed to the alien mesh. The point of this script is to call playerBehaviour's PlayerDeath() when it's collider collides with the player.

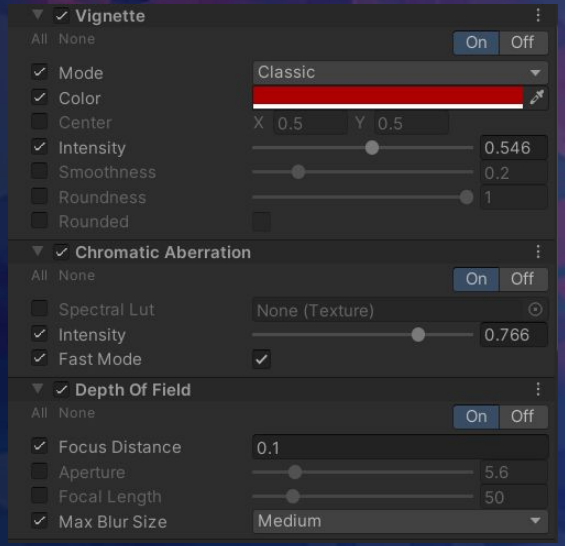
Instead of adding a kill trigger script to the fire I called PlayerDeath() from the FireBehaviour script.

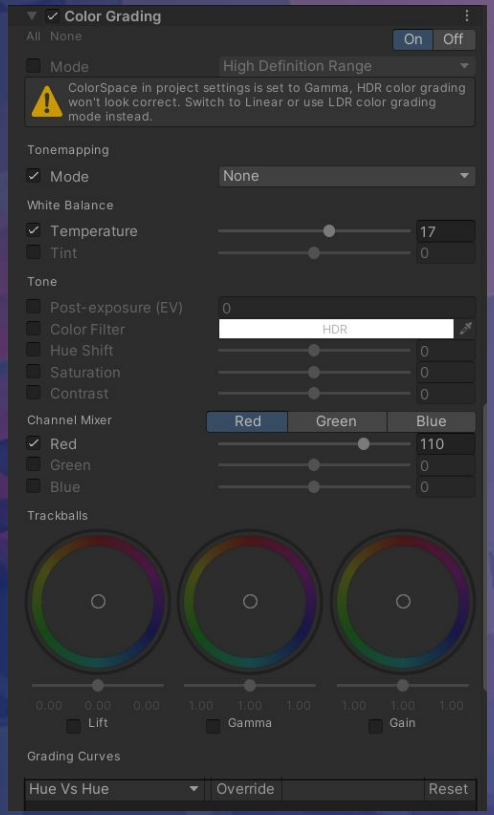
```
public IEnumerator PlayerDeath()
{
    mouseScript.cameraLock = true;
    charController.enabled = false;

    while (postProcessing.weight < 1f)
    {
        postProcessing.weight += 0.05f;
        yield return new WaitForSeconds(0.01f);
    }

    postProcessing.weight = 1f;

    yield return new WaitForSeconds(2.5f);

    Scene currentScene = SceneManager.GetActiveScene();
    SceneManager.LoadScene(currentScene.name);
}
```

```
void OnTriggerStay(Collider collider)
{
    if(collider.CompareTag("Extinguisher Foam"))
    {
        gettingExtinguished = true;
        Debug.Log("Fire is being extinguished");
    }

    if(collider.CompareTag("Player"))
    {
        Debug.Log(collider.gameObject);
        playerBehaviour.StartCoroutine(playerBehaviour.PlayerDeath());
    }
}
```

Player Death







Starting work on Audio

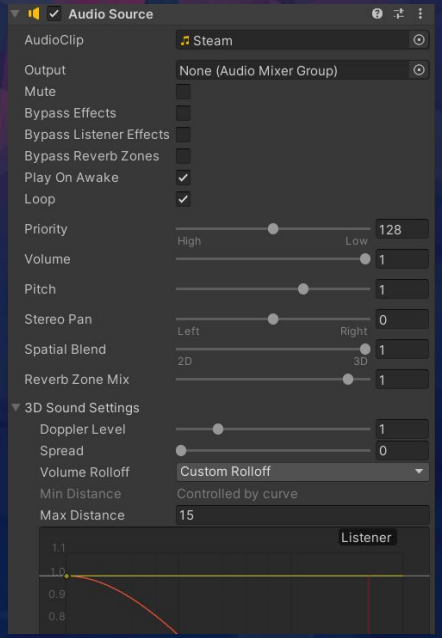
During the project, I've been recording some audio files to use as sound effects in the game, so since the game has been feeling a bit quite I decided to add some. All of the audio files I recorded are edited using Audacity to add reverb, change pitch and such, here are some of the files that I have added so far:

Steam sound effect: This file was originally recorded from me spraying a hairspray can, I also had a recording of a deodorant can but decided that the hairspray made a closer sound to steam in its raw form. In audacity, I slowed the audio down a bit added some reverb and lowered the pitch a little bit. After editing I added the it to a new audio source to each of the steam particle effect objects, and since the valve script disables the objects when it's not broken, the audio is only audible when it *is* broken.

Player Steps: For the player's steps I recorded an audio of me stepping on the side of one of my radiators since it gave a good echo and metallic sound. I edited the file by changing the pitch and adding some more reverb. From this I made two step files that will be triggered in the player movement script.

Alien Steps: The alien footsteps are also the radiator steps the player's use except I used different samples and added more reverb and lowered the pitch more since the alien would be heavier and would make a lower pitched and heavier footsteps.

Starting work on Audio




```
IEnumerator PlayFootsteps (string movementType)
{
    audioPlaying = true;
    AudioSource source = gameObject.AddComponent<AudioSource>();

    if(movementType == "Running" && !outOfStamina)
    {
        source.clip = clipToPlay;
        source.volume = 0.3f;
        source.Play();

        Destroy(source, clipToPlay.length);

        yield return new WaitForSeconds(0.3f);

        audioPlaying = false;
    }
    else if (movementType == "Walking" || movementType == "Running" && outOfStamina)
    {
        source.clip = clipToPlay;
        source.volume = 0.23f;
        source.Play();

        Destroy(source, clipToPlay.length);

        yield return new WaitForSeconds(0.55f);

        audioPlaying = false;
    }

    if (clipToPlay == step1) clipToPlay = step2;
        else clipToPlay = step1;
```

```
else if (isRunning)
{
    playerMovementState = "Running";
    animator.SetBool("Walking?", false);
    animator.SetBool("Running?", true);

    if(!audioPlaying  && !isInVent)
    {
        StartCoroutine(PlayFootsteps("Running"));
    }
}
else
{
    if(!audioPlaying && !isInVent)
    {
        StartCoroutine(PlayFootsteps("Walking"));
    }

    playerMovementState = "Walking";
    animator.SetBool("Walking?", true);
    animator.SetBool("Running?", false);
}
```

Player footsteps

Starting work on Audio

```
IEnumerator StepSounds()
{
    while (true)
    {
        float timeBetweenSteps = 0f;

        if(alienStatus == AlienStatus.Roaming)
        {
            timeBetweenSteps = 1f;
        }
        else if (alienStatus == AlienStatus.Engaging)
        {
            timeBetweenSteps = 0.8f;
        }

        if (alienStatus == AlienStatus.Roaming
            || alienStatus == AlienStatus.Engaging)
        {
            AudioSource source = gameObject.AddComponent<AudioSource>();
            source.clip = clipToPlay;
            source.spatialBlend = 1f;
            source.minDistance = 1f;
            source.maxDistance = 23f;
            source.Play();

            Destroy(source, clipToPlay.length);

            if (clipToPlay == step1) clipToPlay = step2;
                else clipToPlay = step1;
        }

        yield return new WaitForSeconds(timeBetweenSteps);
    }
```

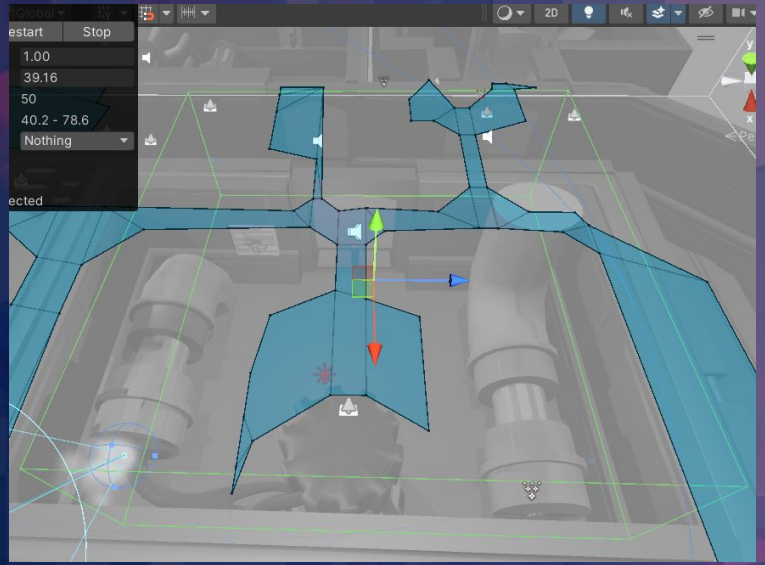
Alien Footsteps

I've also tried to make some kind of audio occlusion where sounds will sound muffled if they're on the other side of a wall. This is because I'm not using FMOD unlike we originally planned because I don't have the time to set up the FMOD project and get it integrated into the scene since I only really have little less than a month left to get the game done.

Instead of audio occlusion, I instead made a temporary solution by creating box colliders in the areas I want the player to be able to hear the steam sound effect from for each particle effect. The way it works is that the volume of the audio source will reduce to 0 if the the player is not in the collider and back to 1 if they are, this stops some issues where the player could hear the steam loudly through walls.

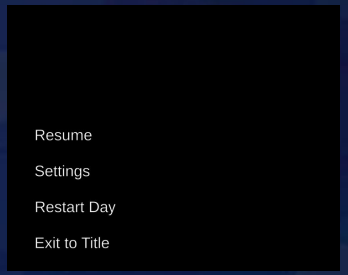
Audio Fall Off

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class AudioFallOff : MonoBehaviour
{
    private AudioSource audioSource;

    void Start()
    {
        audioSource = transform.parent.GetComponent<AudioSource>();
        if (audioSource == null)
        {
            Debug.LogWarning("No AudioSource found on the parent object!");
        }
        audioSource.volume = 0f;
    }

    void OnTriggerStay(Collider collider)
    {
        if (collider.CompareTag("Player"))
        {
            audioSource.volume = 1f;
        }
    }

    void OnTriggerExit(Collider collider)
    {
        if (collider.CompareTag("Player"))
        {
            audioSource.volume = 0f;
        }
    }
}
```



Pause Menu and Settings

```
private void Update()
{
    if (Input.GetKeyDown(KeyCode.Escape) && !restrictMenuOpen)
    {
        if (gamePaused)
        {
            Time.timeScale = 1f;

            ResumeFromPause();
        }
        else
        {
            Time.timeScale = 0f;
        }
    }
}
```



Next, I need to work on a way for the player to change settings such as inverting mouse axes, sensitivity and FOV. To start work on this I created a gameobject with an image and on that I have 4 buttons that will each perform an action when clicked.

To handle the pause menu, I created a MenuController to hold each of the button's actions and the effect of the pause menu on the game. One of the first things I need to do it when the menu is opened Time.timescale needs to be set to 0. This will stop scripts that work on IEnumators, Animators, etc from running until it's set back to 1 or any number higher than 0.

From there I also made the settings menu which will open when the settings button from the pause menu is clicked.

The settings menu has 4 settings, invert axes, mouse sensitivity, camera fov and volume (for audio), and I have set up the menu controller to update these settings when they leave the settings menu by pressing escape, or clicking the back button.

Pause Menu and Settings

When the settings update, I also have them update the values of a scriptable object I made called Settings Storage, this is so that the settings can save between scenes or if the scene gets reloaded which is expected to happen.

After the settings logic was done I made the functions to handle resume game and restart day, and I'll add the Back To Title logic when I've actually added a title screen.

```
private void Update()
{
    if (Input.GetKeyDown(KeyCode.Escape) && !restrictMenuOpen)
    {
        if (gamePaused)
        {
            Time.timeScale = 1f;

            settingsStorage.cameraFOV = FOVSlider.value;
            settingsStorage.cameraSensitivity = SensitivitySlider.value;
            settingsStorage.volume = VolumeSlider.value;
            settingsStorage.invertedMouse = cameraScript.invertAxes;

            ResumeFromPause();
        }
        else
        {
            Time.timeScale = 0f;
            PauseGame();
        }
    }
}
```

```
private void ResumeFromPause()
{
    Time.timeScale = 1f;

    cameraScript.mouseSensitivity = SensitivitySlider.value;
    cameraScript.cameraLock = false;
    mainCamera.fieldOfView = FOVSlider.value;
    AudioListener.volume = VolumeSlider.value / 100;

    pauseMenu.SetActive(false);
    settingsMenu.SetActive(false);

    gamePaused = false;
    playerController.enabled = true;

    Cursor.visible = false;
    Cursor.lockState = CursorLockMode.Locked;
}
```

Pause Menu and Settings

```
private void PauseGame()
{
    pauseMenu.SetActive(true);
    settingsMenu.SetActive(false);

    gamePaused = true;
    cameraScript.cameraLock = true;
    playerController.enabled = false;

    Cursor.visible = true;
    Cursor.lockState = CursorLockMode.None;

    cameraScript.mouseSensitivity = SensitivitySlider.value;
    mainCamera.fieldOfView = FOVSlider.value;
    AudioListener.volume = VolumeSlider.value / 100;
}
```

```
public void OpenSettings()
{
    pauseMenu.SetActive(false);
    settingsMenu.SetActive(true);

    Debug.Log("Open Settings");

    FOVSlider.value = settingsStorage.cameraFOV;
    SensitivitySlider.value = settingsStorage.cameraSensitivity;
    VolumeSlider.value = settingsStorage.volume;

    invertedMouseCheck.SetActive(settingsStorage.invertedMouse);
}

public void BackToPauseMenu()
{
    settingsStorage.cameraFOV = FOVSlider.value;
    settingsStorage.cameraSensitivity = SensitivitySlider.value;
    settingsStorage.volume = VolumeSlider.value;
    settingsStorage.invertedMouse = cameraScript.invertAxes;

    ApplySettingsFromStorage();

    pauseMenu.SetActive(true);
    settingsMenu.SetActive(false);
}
```

```
private void ApplySettingsFromStorage()
{
    invertedMouseCheck.SetActive(settingsStorage.invertedMouse);
    mainCamera.fieldOfView = settingsStorage.cameraFOV;
    cameraScript.mouseSensitivity = settingsStorage.cameraSensitivity;
    AudioListener.volume = settingsStorage.volume / 100;
    cameraScript.invertAxes = settingsStorage.invertedMouse;
}
```

Stamina UI

The next thing I wanted to do is add a system that depletes and regenerates the player's stamina, as well as show the player's stamina as a meter on the Canvas.

So I made two images in the canvas hierarchy, one to be the background of the bar and the other to be the bar which will scale based on the player's stamina.

This mechanic is controlled by a new script called PlayerStamina and in it, it has two coroutines that are constantly running in IEnumerators.

The first coroutine handles with updating the stamina bar and scaling the gameobject based on the player's stamina.

The other deals with fading the stamina bar and it's parent's image alpha value when the stamina bar is at 100 or 0 so that it is only visible when the bar is decreasing or increasing.

```
using System.Collections;
using UnityEngine;
using UnityEngine.UI;

public class PlayerStamina : MonoBehaviour
{
    [SerializeField] private GameObject staminaBar;
    [SerializeField] private scr_playerMovement movementScript;

    public float staminaValue = 100f;

    private float staminaBarBaseOpacity;
    private Image staminaBarImage;
    private Image staminaBarChildImage;

    void Start()
    {
        staminaBarImage = staminaBar.GetComponent<Image>();
        staminaBarBaseOpacity = staminaBarImage.color.a;

        staminaBarChildImage = staminaBar.transform.GetChild(0).GetComponent<Image>();

        Color parentColor = staminaBarImage.color;
        parentColor.a = 0f;
        staminaBarImage.color = parentColor;

        Color childColor = staminaBarChildImage.color;
        childColor.a = 0f;
        staminaBarChildImage.color = childColor;

        StartCoroutine(UpdateStamina());
        StartCoroutine(UpdateStaminaBarFade());
    }
private IEnumerator UpdateStamina()
{
    while (true)
    {
        if (movementScript.playerMovementState == "Running" && !movementScript.isInVent)
        {
            if (staminaValue > 0) staminaValue -= 2f;

            if (staminaValue <= 0) movementScript.outOfStamina = true;
        }
        else
        {
            if (staminaValue < 100) staminaValue += 0.8f;
            else staminaValue = 100;

            if (staminaValue > 0) movementScript.outOfStamina = false;
        }

        float scaleFactor = Mathf.Clamp(staminaValue / 100f, 0f, 1f);
        staminaBarChildImage.transform.localScale = new Vector3(
            scaleFactor * 0.99f,
            staminaBarChildImage.transform.localScale.y,
            staminaBarChildImage.transform.localScale.z);

        yield return new WaitForSeconds(0.1f);
    }
}
```

Stamina UI

```
private IEnumerator UpdateStaminaBarFade()
{
    while (true)
    {
        Color parentColor = staminaBarImage.color;

        if (staminaValue >= 100f || staminaValue <= 0)
        {
            parentColor.a -= 0.05f * Time.deltaTime * 20f;
            if (parentColor.a < 0f) parentColor.a = 0f;
        }
        else
        {
            parentColor.a += 0.05f * Time.deltaTime * 20f;
            if (parentColor.a > staminaBarBaseOpacity) parentColor.a = staminaBarBaseOpacity;
        }

        staminaBarImage.color = parentColor;

        if (staminaBarChildImage != null)
        {
            Color childColor = staminaBarChildImage.color;

            if (staminaValue >= 100f)
            {
                childColor.a -= 0.05f * Time.deltaTime * 20f;
                if (childColor.a < 0f) childColor.a = 0f;
            }
            else
            {
                childColor.a += 0.05f * Time.deltaTime * 20f;
                if (childColor.a > staminaBarBaseOpacity) childColor.a = staminaBarBaseOpacity;
            }

            staminaBarChildImage.color = childColor;
        }

        yield return null;
    }
}
```

```
bool isRunning = Input.GetKey(KeyCode.LeftShift) && movement.magnitude > 0f;
float currentSpeed = isRunning ? speed * runningMultiplier : speed;

if (outOfStamina) currentSpeed = speed;
if (isInVent) currentSpeed = 2.5f;
```

Above I have added some lines to the player movement that restrict sprinting when the stamina script's stamina value is 0, also I forgot to mention earlier in the log that I reduce the player's movement when they are in the vent.

Player Sleeping and Puzzle Breaking

Early on when making this system of breaking the puzzles via sleeping to work on getting a gameplay section into the game, I just added an interaction script to the bed which broke a random amount of puzzles based on an int added to the function's brackets.

This however evolved when I set the function to break a certain amount of puzzles throughout the game day depending on what day it was. To do this I added the day variable to the settings storage and then in the MasterPuzzleScript it will use that to break some puzzles on Start() and then it might break a few more if it's not the first day after so many seconds.

The bed instead now was just a way to check if all the current puzzles are fixed and if so, it will increase the day count and restart the scene.

For this I also needed to add a break function to each of the puzzles so that I can call them from the master puzzle script when they have been chosen to break. Well, I say each of the puzzles, but some didn't need that function since their condition updates automatically based on the puzzle status, so I just had to change the puzzle status to PuzzleStatus.Broken.

```
foreach (MasterPuzzleScript.Puzzle puzzle in masterPuzzleScript.puzzles)
{
    if (puzzle.puzzleScript is scr_pipePuzzle pipe)
    {
        if (pipe.puzzleStatus != scr_pipePuzzle.PuzzleStatus.Fixed && pipe.puzzleStatus != scr_pipePuzzle.PuzzleStatus.Perfect)
        {
            allEquipmentFixed = false;
            break;
        }
    }
    else if (puzzle.puzzleScript is scr_fusePuzzle fuse)
    {
        if (fuse.puzzleStatus != scr_fusePuzzle.PuzzleStatus.TwoNew)
        {
            allEquipmentFixed = false;
            break;
        }
    }
    else if (puzzle.puzzleScript is scr_valvePuzzle valve)
    {
        if (valve.puzzleStatus != scr_valvePuzzle.PuzzleStatus.Fixed)
        {
            allEquipmentFixed = false;
            break;
        }
    }
    else if (puzzle.puzzleScript is scr_wirePuzzle wire)
    {
        if (wire.puzzleStatus != scr_wirePuzzle.PuzzleStatus.Perfect && wire.puzzleStatus != scr_wirePuzzle.PuzzleStatus.Fixed)
        {
            allEquipmentFixed = false;
```

Sleep Interaction Script

```
if (allEquipmentFixed)
{
    GetComponent<scr_Interactable>().enabled = false;
    imageFader.StartFadeIn();
    StartCoroutine(GoToSleep());
}
```

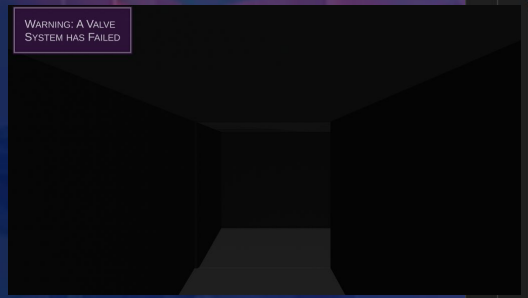
Master Puzzle Script

```
IEnumerator StartDamagePuzzles()
{
    if (playerSettings.day == 0)
    {
        yield return new WaitForSeconds(1);
        DamagePuzzle(2);
    }

    if (playerSettings.day != 0) yield return new WaitForSeconds(4.5f);


    if (playerSettings.day == 1)
    {
        DamagePuzzle(1);
        yield return new WaitForSeconds(Random.Range(13, 21));
        DamagePuzzle(1);
    }
    else if (playerSettings.day == 2)
    {
        DamagePuzzle(2);
        yield return new WaitForSeconds(Random.Range(23, 40));
        DamagePuzzle(2);
    }
    else if (playerSettings.day != 0)
    {
        DamagePuzzle(3);
        yield return new WaitForSeconds(Random.Range(30, 60));
        DamagePuzzle(2);
        yield return new WaitForSeconds(Random.Range(30, 60));
        DamagePuzzle(2);
    }
}
```

```
public void DamagePuzzle(int puzzlesToBreak)
{
    int attempts = 0;
    int maxAttempts = 100; // prevent infinite loop if none are breakable

    while (puzzlesToBreak > 0 && attempts < maxAttempts)
    {
        MonoBehaviour script = puzzles[Random.Range(0, puzzles.Count)].puzzleScript;

        if (script is scr_pipePuzzle pipe &&
            (pipe.puzzleStatus == scr_pipePuzzle.PuzzleStatus.Perfect || pipe.puzzleStatus == scr_pipePuzzle.PuzzleStatus.Fixed))
        {
            pipe.DamagePipe();
            notificationSystem.messages.Add("Pipe Damage Detected - Maintenance Required");
            puzzlesToBreak--;
        }
        else if (script is scr_fusePuzzle fuse &&
                fuse.puzzleStatus == scr_fusePuzzle.PuzzleStatus.TwoNew)
        {
            notificationSystem.messages.Add("Fuses Damaged - Replacements Required");
            fuse.DamageFuses();
            puzzlesToBreak--;
        }
        else if (script is scr_valvePuzzle valve &&
                valve.puzzleStatus == scr_valvePuzzle.PuzzleStatus.Fixed)
        {
            notificationSystem.messages.Add("Warning: A Valve System has Failed");
            valve.puzzleStatus = scr_valvePuzzle.PuzzleStatus.Broken;
            puzzlesToBreak--;
        }
        else if (script is scr_wirePuzzle wire &&
                (wire.puzzleStatus == scr_wirePuzzle.PuzzleStatus.Perfect || wire.puzzleStatus == scr_wirePuzzle.PuzzleStatus.Fixed))
        {
            notificationSystem.messages.Add("Electrical Wiring Damage Detected");
            wire.puzzleStatus = scr_wirePuzzle.PuzzleStatus.Broken;
            puzzlesToBreak--;
        }
        else if (script is scr_generatorPuzzle generator &&
                generator.puzzleStatus == scr_generatorPuzzle.PuzzleStatus.Fixed)
        {
            notificationSystem.messages.Add("System Warning: Generator Cores have Degraded");
            generator.DegradePowerCores();
            puzzlesToBreak--;
        }

        if (puzzlesToBreak <= 0)
        {
            Debug.Log("Damaged Puzzle: " + script);
        }

        attempts++;
```

Notification System

Now, you may have noticed in the masterPuzzle script the reference called notificationSystem. This is a script that I made that sends notifications on the player's canvas. The way it works is that the script is constantly checking if there are any strings in the messages list, if there is the notification text will change to message[0] and will play the notification box's animation till completion and it will keep doing that until there are no messages. The good thing about having this is that I can add messages to the list from any script which is helpful since I won't have to store every possible message all in the NotificationSystem script.




```
using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;

public class NotificationSystem : MonoBehaviour
{
    private Animator messageAnimator;
    [SerializeField] private TMP_Text messageText;

    public List<string> messages = new List<string>();

    void Start()
    {
        messageAnimator = GetComponent<Animator>();
        messageAnimator.enabled = false;

        if (messageText == null)
            messageText = GetComponentInChildren<TMP_Text>();

        StartCoroutine(UpdateMessages());
    }

    IEnumerator UpdateMessages()
    {
        while (true)
        {
            if (messages.Count != 0)
            {
                messageText.text = messages[0];
                messageAnimator.enabled = true;

                yield return new WaitForSeconds(3f);

                messageAnimator.enabled = false;

                messages.RemoveAt(0);

                yield return new WaitForSeconds(1f);
            }
            else
            {
                yield return null;
            }
        }
    }
```

Map UI

When a puzzle breaks or a fire is started, icons will appear on the map which appears in the pause menu. This will assist the player in finding where the puzzles that are broken are or with the generator puzzle where the power cores are since I'm sure it wouldn't be that fun if the player had to look around the whole map for one power core and end up getting a game over after running out of time.

The map image I made in Aesprite by tracing over Mikaela's map design.

Also I mentioned earlier about the power cores on the map, I forgot to mention that I added a part of the script where if the power core breaks it will spawn a new one at one of the predetermined locations so the player can repair the generator.

The icons for the map were also made by me in aesprite, though specifically for the power core icon, to make that I just scaled down an image of the power core model.

```
void OnEnable()
{
    GetComponent<Image>().enabled = false;
    blinkAnimator = GetComponent<Animator>();

    blinkAnimator.enabled = false;

    if (generatorPuzzle != null && generatorPuzzle.puzzleStatus == scr_generatorPuzzle.PuzzleStatus.Broken)
    {
        blinkAnimator.enabled = true;
    }

    if (pipePuzzle != null &&
        (pipePuzzle.puzzleStatus == scr_pipePuzzle.PuzzleStatus.Broken || pipePuzzle.puzzleStatus == scr_pip
    {
        blinkAnimator.enabled = true;
    }

    if (fusePuzzle != null && fusePuzzle.puzzleStatus != scr_fusePuzzle.PuzzleStatus.TwoNew)
```

The blink animator plays an animation that enables and disabled the icon

```
public void DegradePowerCores()
{

    int coresToBreak = Random.Range(1, 4);

    while (coresToBreak > 0)
    {
        int coreToBreak = Random.Range(0, batterySections.Count);

        if (batterySections[coreToBreak].coreStatus == CoreStatus.Empty) continue;

        PowerCoreSpawn coreToSpawn = powerCoreSpawns[Random.Range(0, powerCoreSpawns.Count)];
        if (coreToSpawn.hasBeenSpawned) continue;

        batterySections[coreToBreak].coreStatus = CoreStatus.Empty;
        coreToSpawn.hasBeenSpawned = true;

        GameObject newCore = Instantiate(powerCorePrefab, coreToSpawn.powerCoreSpawn, Quaternion.Euler(coreToSpawn.powerCoreRotation));

        ToggleMapIcon toggle = newCore.GetComponent<ToggleMapIcon>();

        if (toggle != null)
        {
            toggle.mapIcon = coreToSpawn.mapIcon;
            toggle.EnableIcon();
        }

        puzzleStatus = PuzzleStatus.Broken;

        coresToBreak--;
    }
```

Title Screen and Credits

After Leon has given me an image for the title screen, I made a new scene and put it on the canvas. The image already had some a couple of buttons' images merged onto the picture so I had to make some unity buttons with no next and place them on top of the already existing images.

The two buttons were and exit and settings button, so I created functions that dealt with those actions but specifically with the settings button, I used the settings menu I already made and implemented the same save to the storage settings code I already had.

I also created a credits button with text on the bottom left to bring the player to the credits scene after I make it.

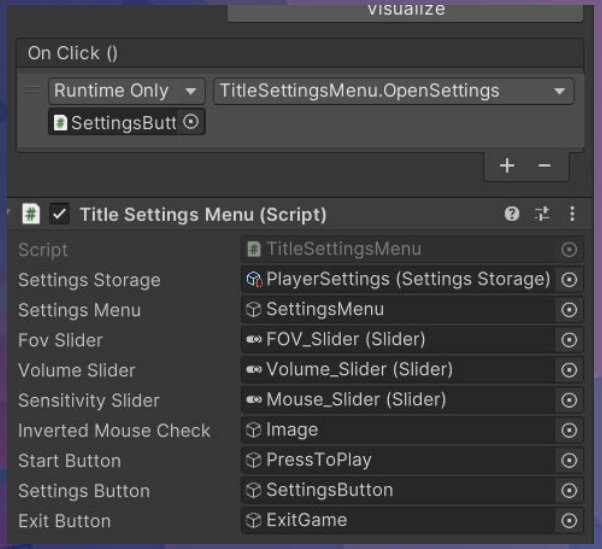
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;

public class TitleMenuButtons : MonoBehaviour
{

    [SerializeField] private Animator fadeAnimator;

    private bool isFading = false;

    void Start()
    {
        fadeAnimator.enabled = false;
        Time.timeScale = 1f;

        Cursor.lockState = CursorLockMode.None;
        Cursor.visible = true;
    }

    public void StartPlayGame()
    {
        if (!isFading)
        {
            StartCoroutine(PlayGame());
        }
    }
```

```
IEnumerator PlayGame()
{
    isFading = true;

    Cursor.lockState = CursorLockMode.Locked;
    //GetComponent<Image>().enabled = false;

    fadeAnimator.enabled = true;

    yield return new WaitForSeconds (2.5f);

    SceneManager.LoadScene("Intro");
}


public void QuitGame()
{
    Application.Quit();
}

public void StartCredits()
{
    if (!isFading)
    {
        StartCoroutine(Credits());
    }
}
```

Title Screen and Credits

```
IEnumerator Credits()
{
    isFading = true;

    Cursor.lockState = CursorLockMode.Locked;

    fadeAnimator.enabled = true;

    yield return new WaitForSeconds (2.5f);

    SceneManager.LoadScene("Credits");
}
```

```
public void OpenSettings()
{
    settingsMenu.SetActive(true);

    startButton.SetActive(false);
    settingsButton.GetComponent<Image>().enabled = false;
    settingsButton.GetComponent<Button>().enabled = false;
    exitButton.SetActive(false);

    invertedMouseCheck.SetActive(settingsStorage.invertedMouse);
}

public void CloseSettings()
{
    settingsStorage.cameraFOV = fovSlider.value;

    settingsStorage.volume = volumeSlider.value;
    AudioListener.volume = volumeSlider.value / 100;

    settingsStorage.cameraSensitivity = sensitivitySlider.value;

    if (invertAxes) settingsStorage.invertedMouse = true;
        else settingsStorage.invertedMouse = false;

    settingsMenu.SetActive(false);

    startButton.SetActive(true);
    settingsButton.GetComponent<Image>().enabled = true;
    settingsButton.GetComponent<Button>().enabled = true;
    exitButton.SetActive(true);
}
```

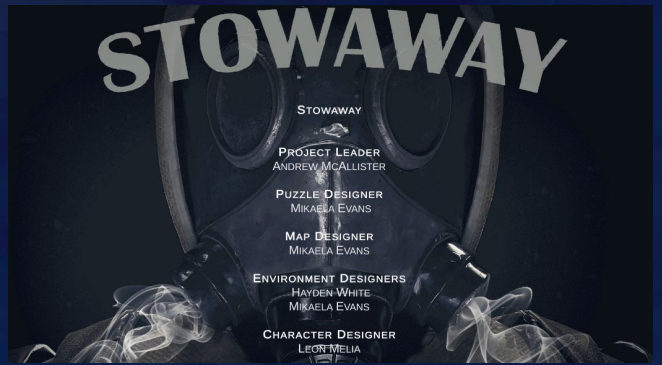


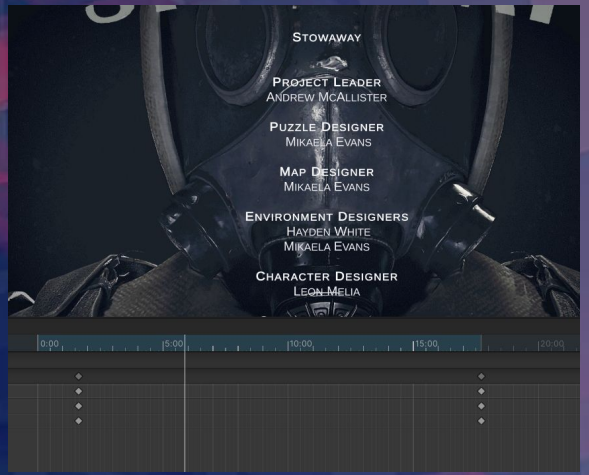
Inspector view for the settings button

Title Screen and Credits

For the Credits scene I used the same image in the title screen but I scaled it up a but so that the merged button images aren't in view, from there I just wrote down who in the project team did what and added a section for any external assets used in the game.

I of course also needed to add an animation to scroll the text from bottom to top.





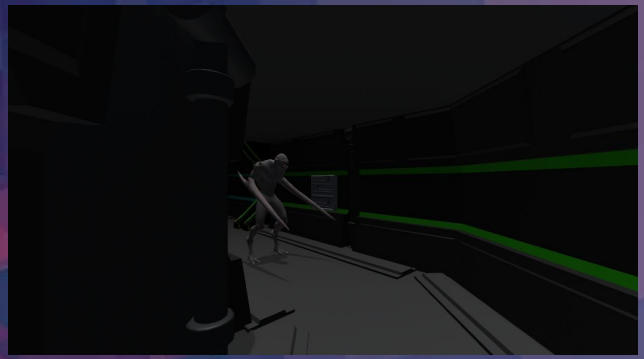
Cutscenes

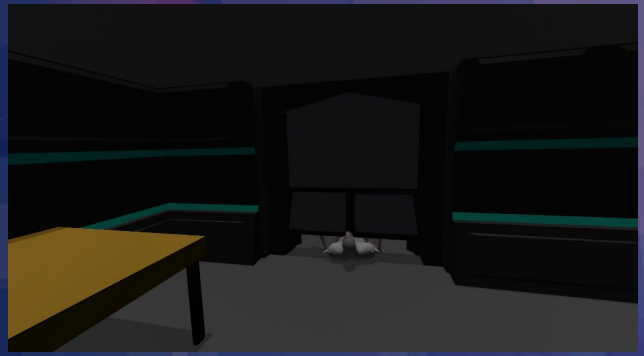
Now that it's near the end of the project I feel I should add some finishing touches to make the game feel complete one of those is narrative. Because of this, I decided to make two short cutscenes for the game.

The first cutscene is the intro cutscene where the player is made aware of the alien and their character enters the vent from the living area. When the settingstorage.day is 0, this will cause the player to spawn in the vent that the player entered in the cutscene, otherwise they'll just wake up in the bed during future days.

For the second cutscene I made the ending where the ship's system is rebooted and the next day the player is chased by the alien into the door control room and crushes them with the door after flicking a lever. I find having an ending cutscene like this contributes greatly to having some resolution after the gameplay is over to what happens after.

To create these scenes I duplicated the game scene and removed some of the scripts essential to the gameplay like the movement script so that the player can't move while the cutscenes are playing.





Game Over

So at this point I have puzzles and a system to break them down, but how about a way to cause the game over via ship implosion? Well, now I've got that too.

By altering the puzzle scripts (except the pipe puzzle because those cause fires), I've made a way to have the main camera start shaking for 5 seconds once the puzzles or a fire is broken (or in fire's case, active) after which will trigger the PlayerDeath() coroutine.

Pair this coroutine with a rumble and explosion sound effect, the game will have a decent game over.

I've also added a method that tracks how many puzzles are about to cause a game over, this means that I can now avoid the issue of the case when two scripts are going to cause player death within 5 seconds and the player fixes one of those puzzles, that script won't cancel the player death since there's still another script in the emergency state. I also have it if the player can fix all the puzzles while the camera is shaking it will stop the implosion.

```
if(puzzleStatus != PuzzleStatus.Fixed && !warningLightTimerCalled)
{
    StartCoroutine(ActivateWarningLight());
    StartCoroutine(TriggerPlayerDeath());
}
```

```
IEnumerator TriggerPlayerDeath()
{
    while (puzzleStatus != PuzzleStatus.Fixed)
    {
        yield return new WaitForSeconds(Random.Range(175, 196));

        if(!playerBehaviour.isEmergencyState) notificationSystem.messages.Add("Valve Heating Failure - IMPLOSION IMMINENT!!");
        playerBehaviour.isEmergencyState = true;
        emergencyStatus = true;
        playerBehaviour.scriptsAffectingRumble++;
        yield return new WaitForSeconds(4);
        cameraShaker.TriggerShake();
        playerBehaviour.rumbleAudioSource.Play();
        yield return new WaitForSeconds(5);
        playerBehaviour.StartCoroutine(playerBehaviour.PlayerDeath());
    }

    if (emergencyStatus)
    {
        notificationSystem.messages.Add("Valve Recalibrated - Implosion has been Avoided");
        playerBehaviour.isEmergencyState = false;
    }
}
```

```
IEnumerator ActivateWarningLight()
{
    warningLightTimerCalled = true;

    while (puzzleStatus != PuzzleStatus.Fixed)
    {
        yield return new WaitForSeconds(90);

        warningLightOn = true;
        lightToEffect.puzzleBroken = true;
    }

    warningLightOn = false;
    warningLightTimerCalled = false;
    if (emergencyStatus) playerBehaviour.scriptsAffectingRumble--;

    if(playerBehaviour.scriptsAffectingRumble == 0) playerBehaviour.rumbleAudioSource.Stop();
}
```

Adding Rest of the Audio

I had been adding audio throughout the final stages of the game but I'm going to list them here instead of scattering them around the DevLog. So here we go:

- To get the externally sourced files out of the way first, I have acquired three files for the game, one is an ambient background audio that plays during the game and both of the cutscenes, I tried to lower the volume of this one so I don't draw too much attention to it. In editing, I did a cross fade for the file in Audacity so that it was loopable and there wasn't any noticeable clicks or sudden changes when it's looping.
- The second file I got from freesound.org was music called "Emotional Mysterious Music" that is used in the title screen and credits scenes, and the only editing for this was a fade out at the end so the file doesn't distort when it looped.
- And the final audio file I sourced was metal door slam sound effect for the doors. I made two files for the game with this through editing for each the open and close sounds that play separately in the game. The way I implemented the door audio into the game is by creating two empty game objects one called CloseAudio and OpenAudio, I then attached each audio file to they're appropriate gameobject. From there I altered the door animations to have these two objects disabled when the door is full open or fully closed and then when it's opening or closing it will enable the appropriate gameobject and because the audio source has Play On Awake the audio file will play. I also used another edited audio file from their original audio for the opening and closing of the vent doors but difference with this from the doors is that I played the audio from the VentBehaviour script.

Adding Rest of the Audio

Now onto my original audio assets that I recorded myself:

- Alien roar: I made this audio by running something across the edge of a sheet of paper, the file was then edited until it sounded like a roar.
- Valve Sound Effect: I got this by editing a file of me running a table knife through the zipper of a suitcase.
- Wire puzzle sound effects: There's two sounds for this puzzle, the first was me just peeling a bit of tape off my desk, this was used for the temporary fix for the puzzle. The second sound was something I used the generate noise tool in audacity and edited it with a modulation effect to get it sound right.
- Generator: This was a tumble dryer sound effect that I modulated and added reverb and echo to to make it sound like a running generator. I also used a cross fade to make it loopable.
- Item Drop and PickUp sound: For this, I recorded me hitting another one of my radiators which makes a different noise than the one I used for the steps. For each the drop and pick up audios I had them at different pitches and reverb so the impact of the action felt different.
- Vent Crawling: These audios will play when the player is navigating through any of the vents and I got them from extracting different samples from an audio of me hitting a baking tray.
- Mouse Click: I just recorded myself clicking my mouse for this one and separated it into two files, one for MouseDown and MouseUp.
- Lever Click: For this one I actually recorded myself clicking my tongue and added some reverb and a pitch change to make it sound right.
- Rumbling sound effect: This one I made by grabbing some items and shaking them inside a plastic box aside with some reverb. I added some volume increases at the end of the file to simulate the sound of the implosion.

Adding Rest of the Audio

Whenever I wanted to add a sound to a puzzle or a footstep instead of using an existing audio source I just added one to the gameObject and deleted it when the clip was over, I found this cleaner than working with and assigning multiple audio sources on the same object onto the script. Specifically when adding audio onto the cutscenes, I made a function called CreateAudio(AudioClip, float) that I could call which would create the audio source and assign it a custom value and after it was done playing it would delete the component.

```
IEnumerator PlayerAudio()
{
    yield return new WaitForSeconds(0.5f);
    int count = 0;

    while (count < 4)
    {
        CreateAudio(step1, 0.25f);
        count++;
        if(count == 4) break;
        yield return new WaitForSeconds(0.65f);

        count++;
        CreateAudio(step2, 0.2f);
        yield return new WaitForSeconds(0.65f);
    }

    yield return new WaitForSeconds(0.9f);

    count = 0;
```

```
void CreateAudio(AudioClip audioClip, float volume)
{
    AudioSource audioSource = gameObject.AddComponent<AudioSource>();

    audioSource.clip = audioClip;
    audioSource.volume = volume;

    audioSource.Play();
    Destroy(audioSource, audioSource.clip.length);
}
```

What would I want to add given I had more time?

- Make the puzzles effect the map more: The original idea was for the puzzles to effect a different aspect of the map like the doors locking or the ship running out of oxygen which I could have done a post process volume where the screen blurs and a vignette appears for.
- Add more audio for UI: I think I missed the opportunity to add a keyboard type effect to when the player presses one of the buttons for the puzzles so it felt more reactive.
- Security Cameras and Map on Monitor: I would have liked it if I had more time to integrate security cameras in the map which can be seen through the monitors in the camera room, I think it would be a good addition to let the player see where the alien is. This also includes the ability to see the ships map and see what puzzles are broken from the monitor in the safe room instead of the pause menu.
- Audio muffling: I mentioned before I wasn't able to get this done and it as an addition would make the environment more immersive for the player.
- Lighting: I wasn't able to work a lot on the lighting since I hadn't worked with light baking or anything like that in Unity before and I didn't have time to learn which is a shame since it would really have enhanced the horror part of the game.
- Tutorial: Some kind of tutorial to let the player grab a hang of things like at the beginning the player can test out fixing the equipment before the alien got loose from it's cage.

Team Collaboration

Compared to last year, I believe that the communication between each other was more frequent when working on the project and because of this our responsibilities for the game were clear as we all knew what we were doing. Though when it came to project management, I found it quite difficult to keep up to date with our original gantt chart since in some cases I would need a piece of work from someone else until I can continue programming the game mechanics or puzzles. I believe why this is the case is that we forgot to account for our other modules.

A lot of times where I have hit a dead end where I can't continue working is because a team member needed to focus on their portfolio project, and this went for me aswell as I've needed to take some time to get up to date on my negotiated project for professional portfolio design.

Because of this, it lead to a bit of cramming within the final stages of the project but I was able to handle it well by scheduling my work day by day on what to get done.

Self-Reflection

Every project that I work on I intend to learn something new, and that I confidently say I have done. Considering that this is my first game that I made in Unity3D I'd say I've done pretty well and have gained more confident in my use of the engine and in C#.

One of the areas that I have picked up on is the Unity Navigation AI package which will prove to have knowledge on for future projects, this as well as using Unity's native audio sources instead of the middleware FMOD, I think is important to know how to use for solo or small scale group projects to avoid it's service fees on games that exceed a certain threshold of income when the project will be no different without it.

Thank you for reading :)