CJ Karinen
CS-485
Prof. Turini
Summer 2024

CS-485 Project Proposal: Liminality

Figure 1: Liminality Title Banner

Description:

Liminality is a linear first-person horror game inspired by liminal space aesthetics and so-called "Non-Places." Explore labyrinth-like environments that will pit the player against their own sense of paranoia. Using a Hi-8 tape camera with a built-in flashlight reveals a path in darkness, but be sure to manage its battery. Utilize stealth to avoid being caught by pursuing creatures in a similar fashion to Outlast. Furthermore, environments will be populated with a variety of non-hostile scripted events to keep the player on their toes, much like how the environment changes and deceives the player in Layers of Fear and PT.

Developer Description:

Liminality is a 3D First-Person horror game with linear levels where players navigate through maze-like levels while solving puzzles and avoiding entities. The game will be structured as a short story campaign consisting of 4 levels, each about 10-15 minutes in length. Levels will be heavily inspired by the phenomena of empty spaces bearing an unsettling feeling, also known as liminal space. A major benefit of this art style is that the simple and often barren architecture of these spaces is well-suited to Unity's Pro Builder packages. Coupled with the use of free textures and minimal 3D props, I can create relatively realistic-looking locations without the need for an extreme amount of clutter or detail. Levels will each have their own distinct entity that the player will encounter throughout. The player won't have a means of defense against hostiles and will have to rely on stealth and evasion. Getting caught and attacked will result in immediate game over. In terms of controls, the player is currently able to walk forwards, backward, side strafe, jump, crouch, interact with scripted objects, toggle the flashlight, and toggle a camera zoom. Additional controls will be added in the form of the ability to lean left and right, as well as the ability to raise and lower the camera. Raising and lowering the camera will be implemented similarly to the flashlight, causing a brief fade to black as the UI and camera post-processing effects are disabled via a script. Lastly, the levels will be absent from any traditional music, instead placing heavy emphasis on environmental ambience and additional sound effects to create tension and atmosphere.

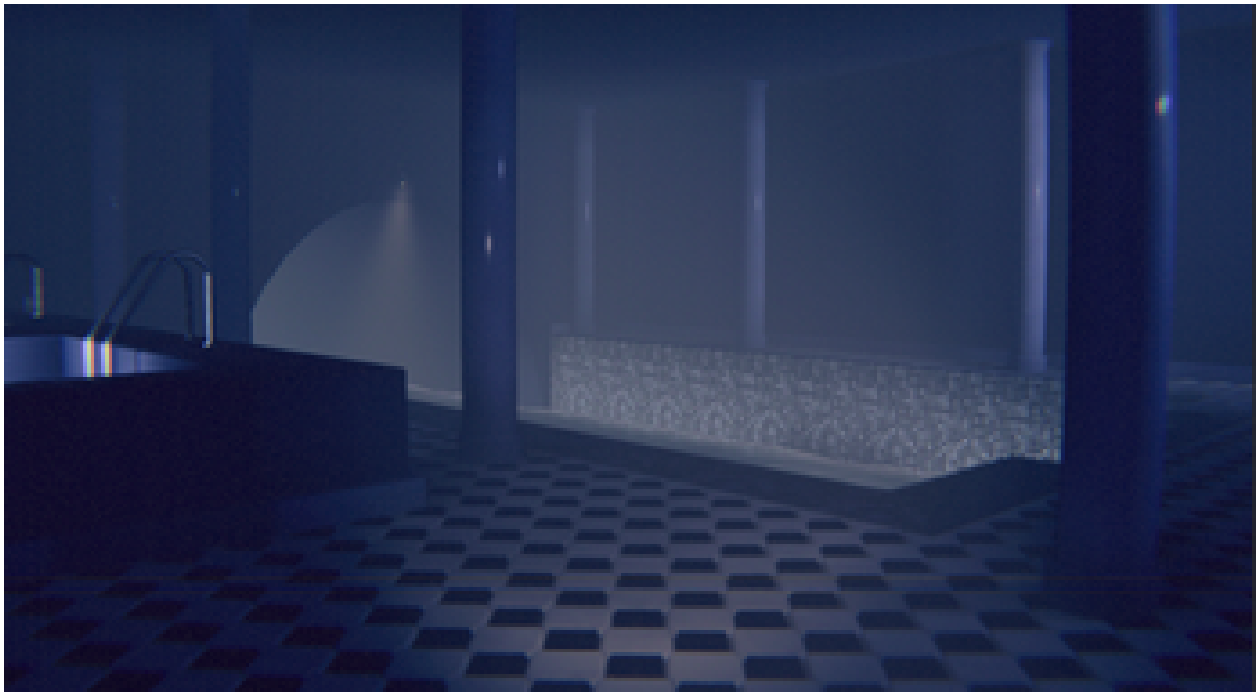

Figure 2: Liminality Tunnel Room, March 26, 2024

Visual Components:

-Player View

The player's view will be from the first-person perspective, and the camera will emulate the Found Footage style, which is popular in horror films and online animations. The camera is a cinemachine camera component hooked up to a custom first-person character controller. In order to replicate the look of a Hi-8 Tape camera, a paid asset called VHS Pro was used. This, in conjunction with handheld camera bob and additional audio effects, creates a convincing analog camera aesthetic.

-UI

The UI mimics the user interface found on the viewfinder of a video camcorder, similar to that of Outlast. Tracked HUD elements will include the camera's battery level, Camera Zoom Level, and the in-game time.

-Levels

The game is set to have 4 levels (this number may change depending on the scope, and each will be designed to be completed within 10-15 minutes each. The player's main goal will be to find the end of each level, navigating around loops and dead ends along the way. The end of each level will smoothly transition to the next, giving the impression of a seamless journey through the game world throughout the campaign. The player will start in a padded cell, where they will be given a short tutorial before being released into the rest of the first level. Chapter 1 has the player navigating through sterile asylum hallways and corridors. In Chapter 2, maintenance rooms and accessways slowly cross over into the realm of the surreal and become

increasingly liminal as the player progresses. Chapter 3 continues into the realm of liminal space as the player navigates through the first layer of the "poolrooms." Lastly, Chapter 4 rounds the campaign off by forcing the player through darker and more claustrophobic "poolrooms." Chapter 4 will end on a cliffhanger to leave room for future content.

Level List:

1. Chapter 1 - The Cell
    a. The player is guided through a short tutorial and then navigates through the asylum environment.
2. Chapter 2 - Breaking Through
    a. The player obtains the video camera and faces a monster that only appears on camera and must rely on sound to track it.
3. Chapter 3 - Shallow Waters
    a. The player navigates through pool environments while evading a monster in the water.
4. Chapter 4 - The Deep
    a. The player navigates through the dark while avoiding contact with areas of deep, dark water.

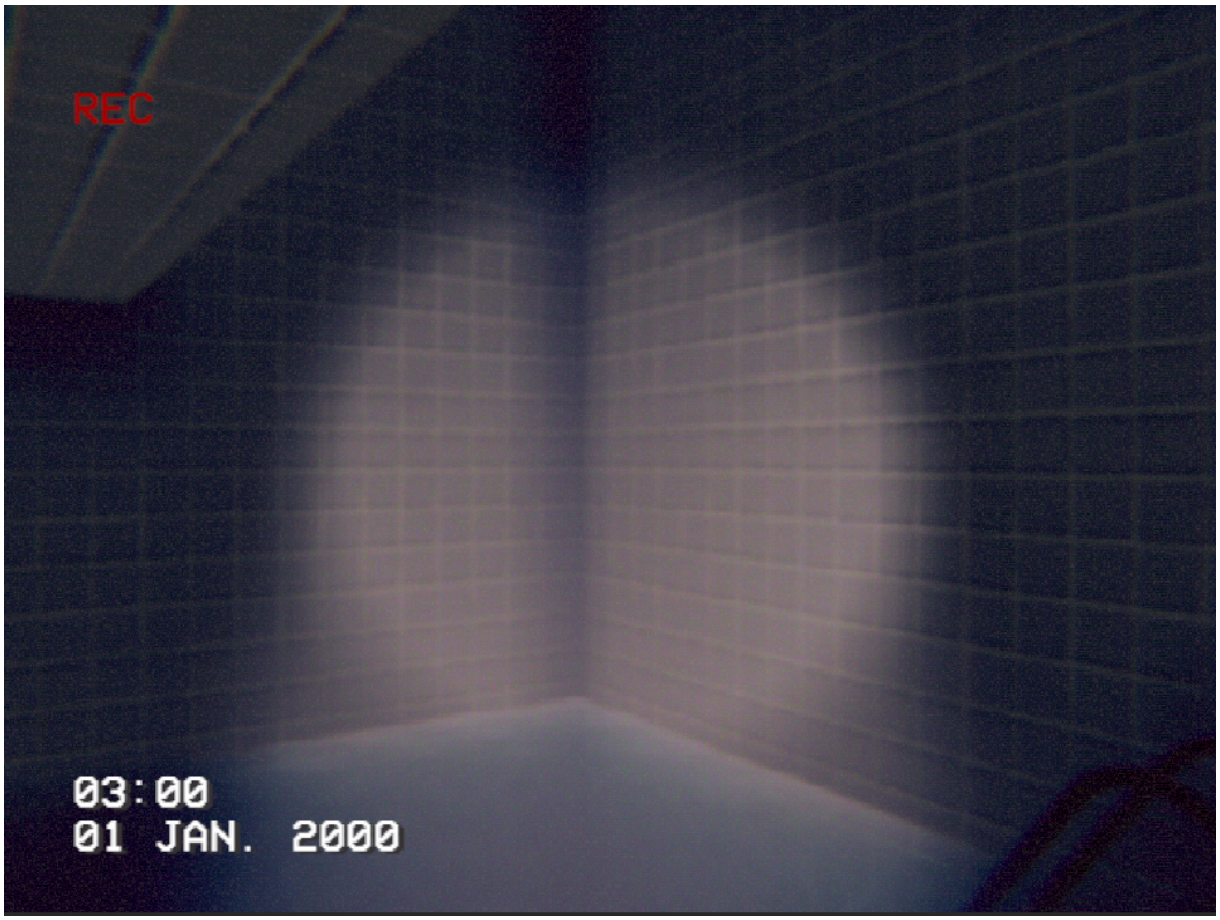

Figure 3: Camera Effects and WIP HUD, July 17, 2024

Figure 4: Outlast 2 HUD, Released April 24, 2017

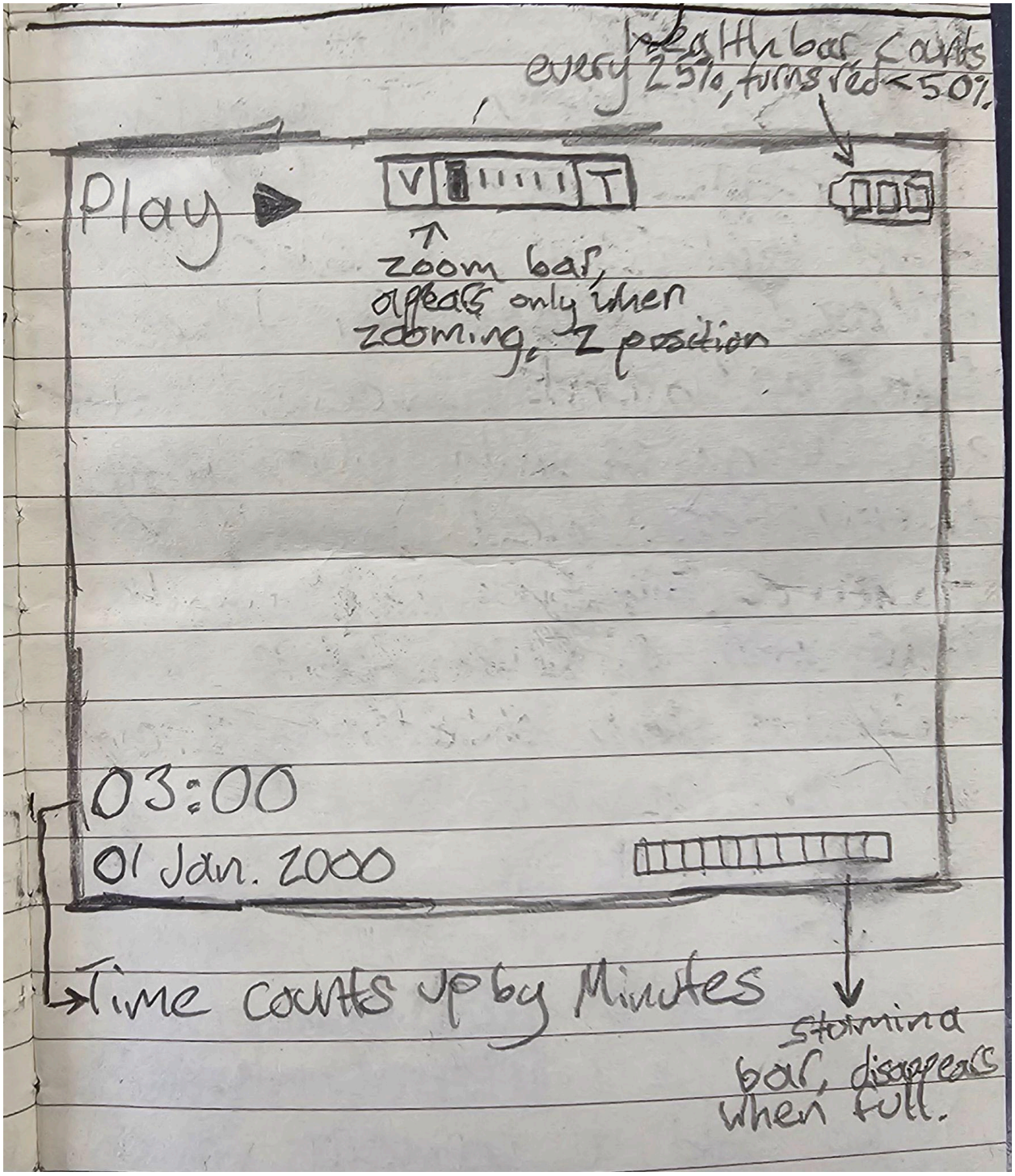

Figure 5: Liminality HUD Concept Sketch, December 17, 2023

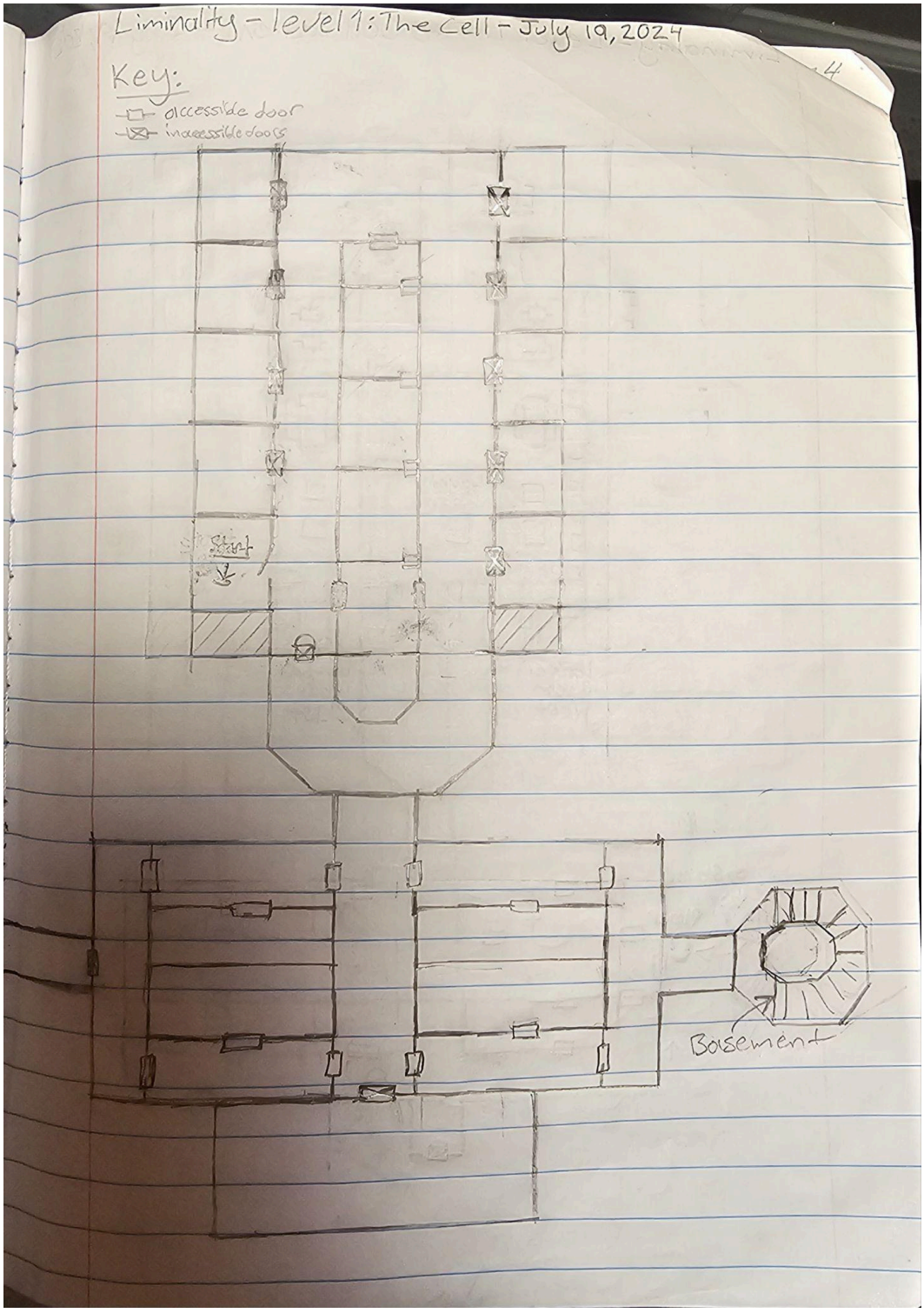

Figure 6: Level 1 Layout L1 Sketch, July 19, 2024

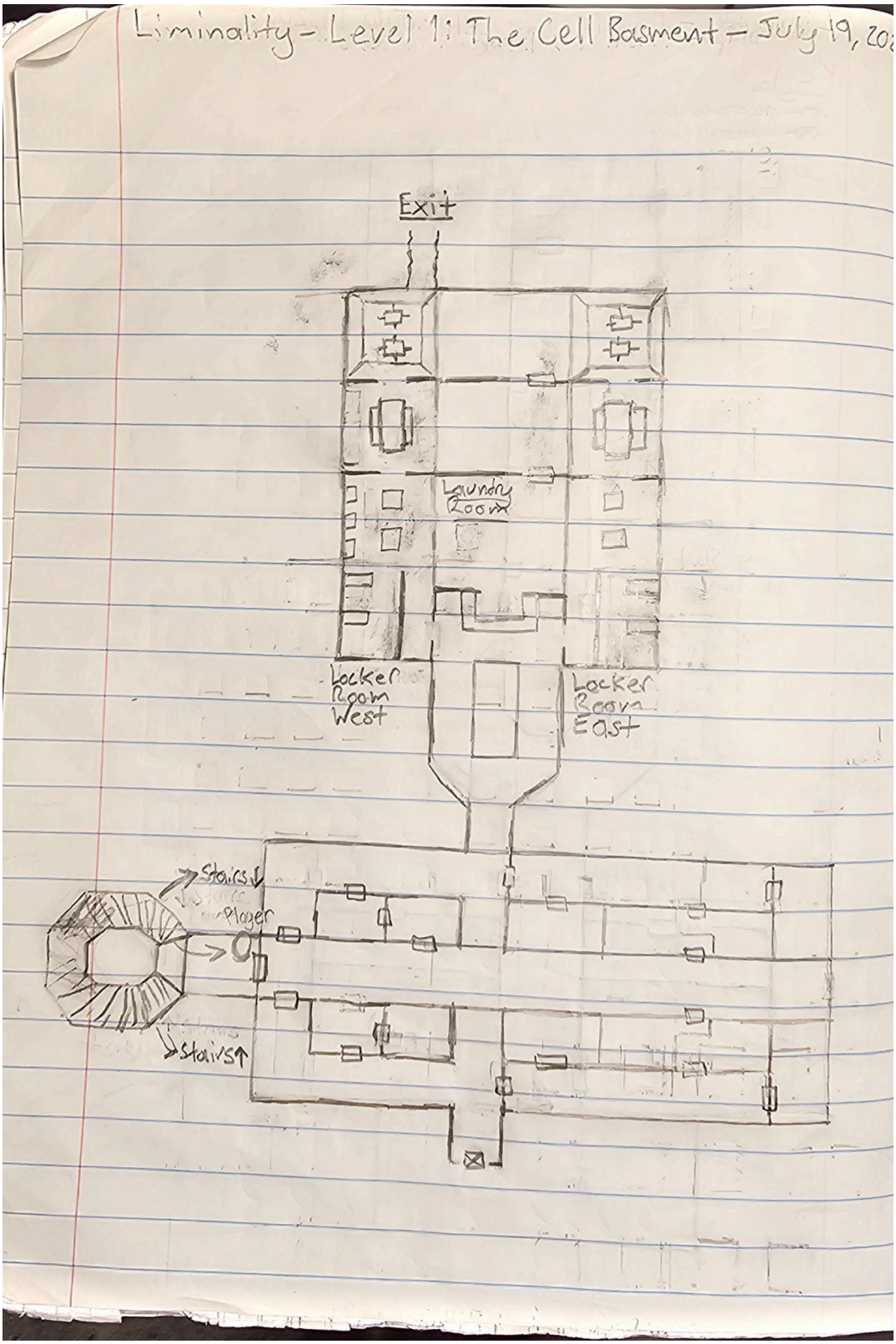

Figure 7: Level 1 Layout L0 Sketch, July 19, 2024

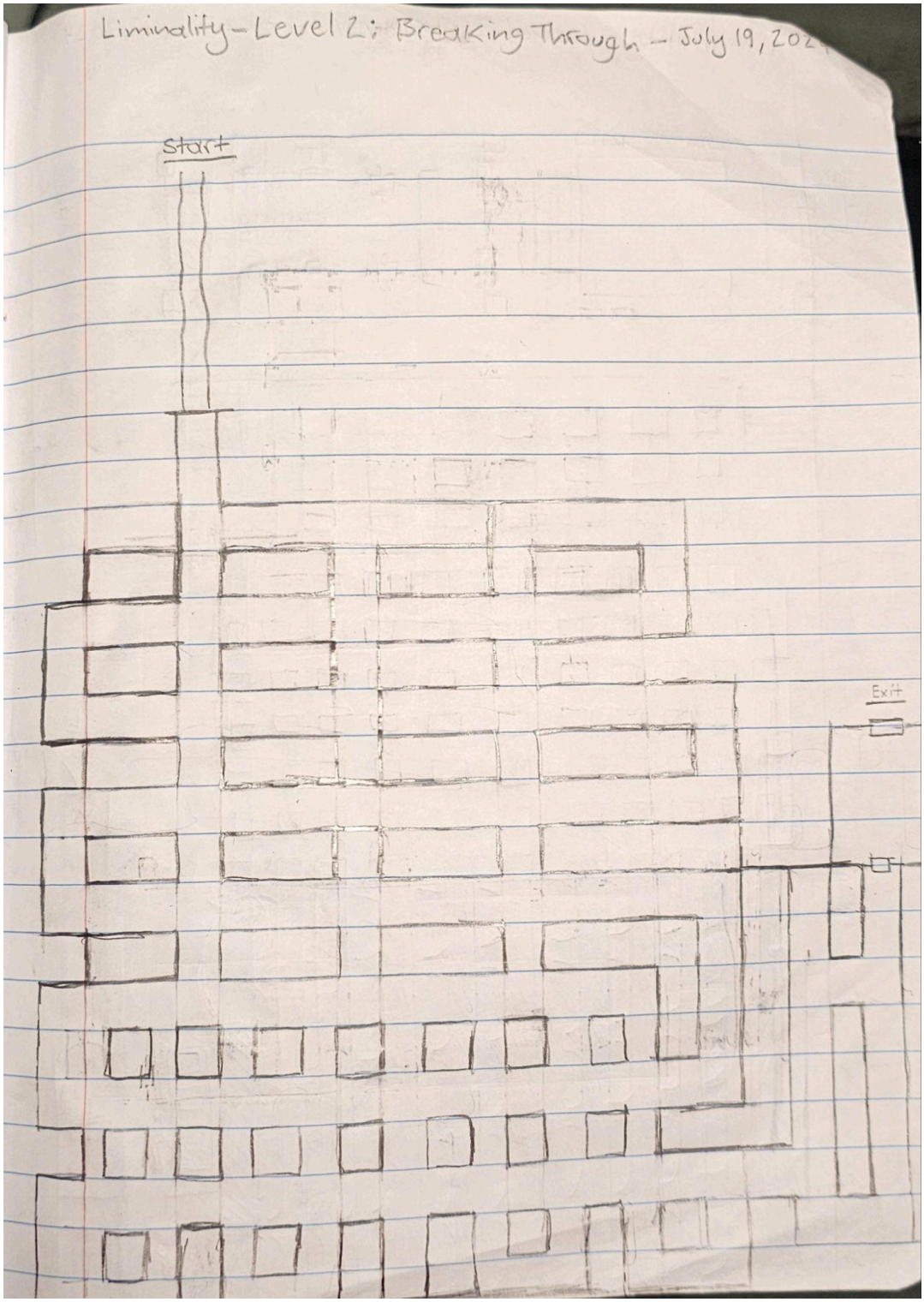

Figure 8: Level 2 Layout Sketch, July 19, 2024

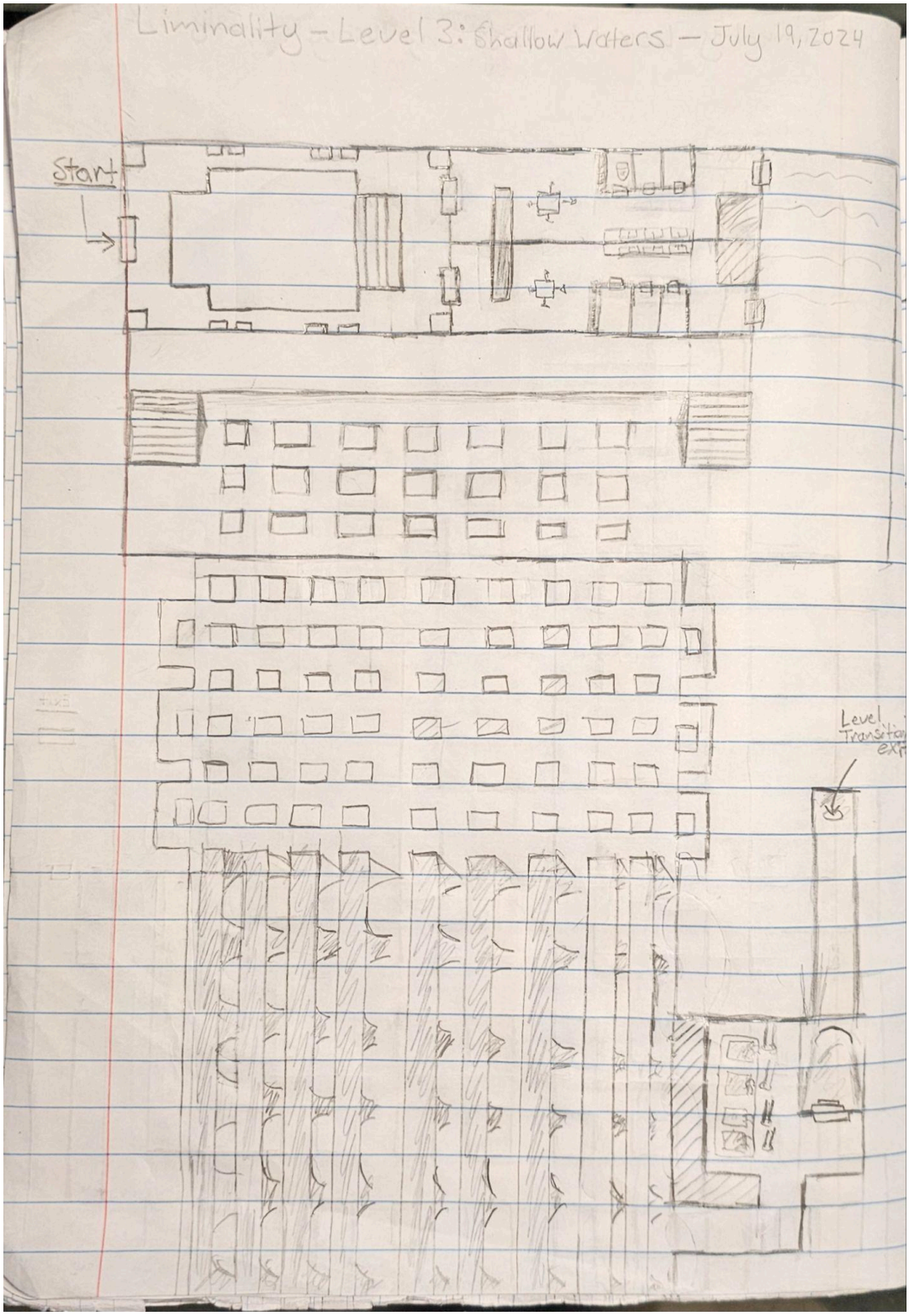

Figure 9: Level 3 Layout Sketch, July 19, 2024

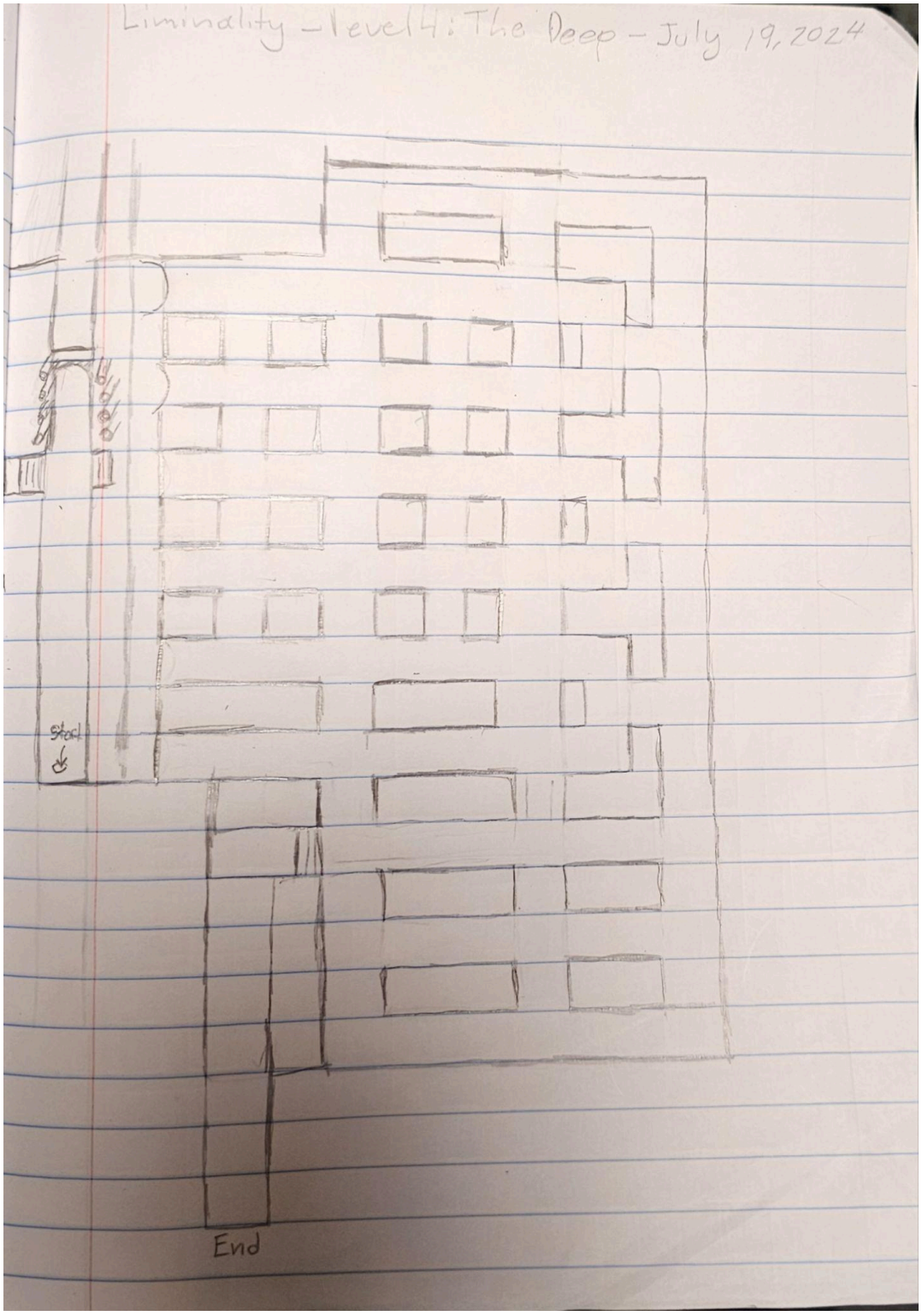

Figure 10: Level 4 Layout Sketch, July 19, 2024

Required Modules:

- **Assets:**
    - **Needed:**
        - **Environmental Textures and Materials (Partially complete)**
        - **Wall Signage and misc decals**
        - **Better Locker Room 3D Models**
        - **UI Icons and Indicators**
    - **Completed:**
        - **Pool Railing 3D Models**
        - **Placeholder Locker Room 3D Models**
        - **UI Text and Fonts**
        - **Camera Filter and Effects**
- **Environments:**
    - **Needed:**
        - **Main Menu Screen Environment**
        - **Chapter 1 Asylum Environment**
        - **Chapter 2 Basement Environment**
        - **Chapter 3 Poolrooms Environment**
        - **Chapter 4 Dark Poolrooms Environment**
    - **Completed:**
        - **Poolrooms Alpha Environment (Not sure where to fit this in with new project direction)**
        - **AI Testing Ground Scene (Used to test an early version of enemy chase and patrol behavior)**
        - **FPS Character Controller Test Level (Used to test custom character controller features)**
- **Characters:**
    - **Needed:**
        - **Final Character Models and shaders**
    - **Completed:**
        - **Acquired Animations for Characters**
        - **Placeholder Models for Test AI**
- **User Interface:**
    - **Needed:**
        - **Camera Battery Tracker**
        - **Camera Zoom Indicator**
        - **Flashlight Indicator Icon**
        - **Scripted Functionality for HUD Text**
        - **Main Menu Interface**

        - **Pause Menu Interface**
    - **Completed:**
        - **HUD Text (No Functionality)**
- **Sounds:**
    - **Needed:**
        - **Player Breathing/Running Sounds**
        - **Monster Sounds**
        - **Additional Background Ambience Tracks**
    - **Completed:**
        - **Footstep Sounds**
        - **Water Sounds**
        - **Primary Ambience Tracks**
        - **Flashlight Toggle Sound Effect**

Development:

Player Asset & Controls:

- **Overview**

For the player asset, I have been utilizing a custom written first-person controller that I had created over 9 months ago by following a series of tutorials from the Youtube channel Comp-3 Interactive. This movement controller has been modularly written, allowing each feature to be toggled on or off. This makes it easy to reuse this system for other projects or to add new features in the future. Pretty much every aspect of the movement system (control bindings, movement variables, etc) can be adjusted using serialized fields in the inspector. Modular features of the movement system include: walking, running, crouching, jumping, headbob, footstep sound system, zoom/aim-down-sights, interaction framework, health system, and a stamina system. Two other features worth noting are slope sliding and ceiling detection while crouched (prevents the player from standing up and clipping into ceilings). In addition to these base features, my movement controller has since been modified to incorporate a blood overlay to indicate damage, a fall damage system, and to fix the previous issue with footstep sounds not playing properly when sprinting.

- **Controls**
    - Walk: WASD
    - Run: Left Shift
    - Jump: Space
    - Crouch: Left Control (short press for toggle, hold for momentary)
    - Zoom: Right Mouse Button (hold to zoom)
    - Interact: Left Mouse Button
    - Flashlight: F

- **Character Controller Code**

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

public class NewFPSController : MonoBehaviour
{
    public bool canMove = true;
    private bool isSprinting => sprintEnabled &&
Input.GetKey(sprintKey) && Input.GetKey(KeyCode.W) && !isCrouching;
    private bool shouldJump => Input.GetKeyDown(jumptKey) &&
characterController.isGrounded;
    private bool shouldCrouch => Input.GetKeyDown(crouchKey) ||
Input.GetKeyUp(crouchKey) && !ongoingCrouchAnimation &&
characterController.isGrounded;

    [Header("Functional Options")]
    [SerializeField] public bool mouseLookEnabled = true;
    [SerializeField] private bool sprintEnabled = true;
    [SerializeField] private bool jumpEnabled = true;
    [SerializeField] private bool crouchEnabled = true;
    [SerializeField] private bool headBobEnabled = true;
    [SerializeField] private bool slopeSlidingEnabled = true;
    [SerializeField] private bool zoomEnabled = true;
    [SerializeField] private bool interactionEnabled = true;
    [SerializeField] private bool footstepsEnabled = true;
    [SerializeField] private bool staminaEnabled = true;
    [SerializeField] private bool fallDamageEnabled = true;

    [Header("Controls")]
    [SerializeField] private KeyCode sprintKey = KeyCode.LeftShift;
    [SerializeField] private KeyCode jumptKey = KeyCode.Space;
    [SerializeField] private KeyCode crouchKey = KeyCode.LeftControl;
    [SerializeField] private KeyCode zoomKey = KeyCode.Mouse1;
    [SerializeField] private KeyCode interactKey = KeyCode.Mouse0;
```

```
    [Header("Movement Parameters")]
    [SerializeField] public float walkSpeed = 4.5f;
    [SerializeField] private float sprintSpeed = 6.2f;
    [SerializeField] private float crouchSpeed = 1.5f;
    [SerializeField] private float slopeSpeed = 8f;


    [Header("Look Parameters")]
    [SerializeField, Range(1, 10)] private float lookSpeedX = 2.0f;
    [SerializeField, Range(1, 10)] private float lookSpeedY = 2.0f;
    [SerializeField, Range(1, 100)] private float upperLookLimit =
80.0f;
    [SerializeField, Range(1, 100)] private float lowerLookLimit =
80.0f;

    [Header("Health Parameters")]
    [SerializeField] private float maxHealth = 100;
    [SerializeField] private float regenCooldown = 10;
    [SerializeField] private float healthIncreaseIncrement = 1;
    [SerializeField] private float healthTimeIncrement = 0.1f;
    private float currentHealth;
    private Coroutine regeneratingHealth;
    public static Action<float> OnTakeDamage;
    public static Action<float> OnDamage;
    public static Action<float> OnHeal;
    public Image bloodOverlay;

    [Header("Fall Damage Parameters")]
    [SerializeField] private float fallDamage = 50;
    [SerializeField] private float minFallHeight = 5f;
    Rigidbody rigidBody;
    private bool _grounded;
    private bool wasGrounded;
    private bool wasFalling;
    private float beginFallHeight;

    [Header("Stamina Parameters")]
    [SerializeField] private float maxStamina = 100;
```

```
    [SerializeField] private float staminaUseMultiplier = 5;
    [SerializeField] private float timeBeforeStaminaRegenStarts = 5;
    [SerializeField] private float staminaIncreaseIncrement = 2;
    [SerializeField] private float staminaTimeIncrement = 0.1f;
    private float currentStamina;
    private Coroutine regeneratingStamina;
    public static Action<float> OnStaminaChange;

    [Header("Jump Parameters")]
    [SerializeField] private float jumpForce = 8.0f;
    [SerializeField] private float gravity = 30.0f;

    [Header("Crouch Parameters")]
    [SerializeField] private float crouchHeight = 0.5f;
    [SerializeField] private float standingHeight = 2f;
    [SerializeField] private float timeToCrouch = 0.25f;
    [SerializeField] private Vector3 crouchingCenter = new Vector3
(0, 0.5f, 0);
    [SerializeField] private Vector3 standingCenter = new Vector3(0,
0, 0);
    private bool isCrouching;
    private bool ongoingCrouchAnimation;

    [Header("Headbob Parameters")]
    [SerializeField] private float walkbobSpeed = 14f;
    [SerializeField] private float walkbobAmount = 0.5f;
    [SerializeField] private float sprintbobSpeed = 18f;
    [SerializeField] private float sprintbobAmount = 0.11f;
    [SerializeField] private float crouchbobSpeed = 8f;
    [SerializeField] private float crouchbobAmount = 0.025f;
    private float defaultYPos = 0;
    private float Timer;

    [Header("Zoom Parameters")]
    [SerializeField] private float timeToZoom = 0.3f;
    [SerializeField] private float zoomFOV = 40f;
    private float defaultFOV;
    private Coroutine zoomRoutine;
```

```
    [Header("Footstep Parameters")]
    [SerializeField] private float baseStepSpeed = 0.7f;
    [SerializeField] private float sprintStepSpeed = 0.3f;
    [SerializeField] private float crouchStepMultiplier = 1.5f;
    [SerializeField] private float sprintStepMultiplier = 0.3f;
    [SerializeField] private AudioSource footstepAudioSource =
default;
    [SerializeField] private AudioClip[] grassSounds = default;
    [SerializeField] private AudioClip[] grassRunSounds = default;
    [SerializeField] private AudioClip[] dirtSounds = default;
    [SerializeField] private AudioClip[] dirtRunSounds = default;
    [SerializeField] private AudioClip[] tileSounds = default;
    [SerializeField] private AudioClip[] tileRunSounds = default;
    [SerializeField] private AudioClip[] waterSounds = default;
    [SerializeField] private AudioClip[] waterRunSounds = default;



    private float footstepTimer = 0;
    private float GetCurrentOffset => isCrouching ? baseStepSpeed *
crouchStepMultiplier : isSprinting ? baseStepSpeed =
sprintStepMultiplier : baseStepSpeed;


    // SLIDING PARAMETERS
    private Vector3 hitPointNormal;

    private bool isSliding
    {
        get
        {
            if (characterController.isGrounded &&
Physics.Raycast(transform.position, Vector3.down, out RaycastHit
slopeHit, 2f))
            {
                hitPointNormal = slopeHit.normal;
                return Vector3.Angle(hitPointNormal, Vector3.up) >
characterController.slopeLimit;
            }
```

```
            else
            {
                return false;
            }
        }
    }
    [Header("Interaction Parameters")]
    [SerializeField] private Vector3 interactionRayPoint = default;
    [SerializeField] private float interactionDistance = default;
    [SerializeField] private LayerMask interactionLayer = default;
    private Interactable currentInteractable;

    private Camera playerCamera;
    private CharacterController characterController;

    private Vector3 moveDirection;
    private Vector2 currentInput;

    private float rotationX = 0;

    public static NewFPSController instance;

    private void OnEnable()
    {
        OnTakeDamage += ApplyDamage;
    }
    private void OnDisable()
    {
        OnTakeDamage -= ApplyDamage;
    }

    void Awake()
    {
        instance = this;
        rigidBody = GetComponent<Rigidbody>();
        playerCamera = GetComponentInChildren<Camera>();
        characterController =
GetComponentInChildren<CharacterController>();
        defaultYPos = playerCamera.transform.localPosition.y;
```

```
        defaultFOV = playerCamera.fieldOfView;
        currentHealth = maxHealth;
        currentStamina = maxStamina;
        Cursor.lockState = CursorLockMode.Locked;
        Cursor.visible = false;
    }

    void Update()
    {
        //Specifies when to update movement
        if (canMove)
        {
            HandleMovementInput();

            if(mouseLookEnabled)
                HandleMouseLook();
            if (jumpEnabled)
                HandleJump();
            if (crouchEnabled)
                HandleCrouch();
            if (headBobEnabled)
                HandleHeadBob();
            if (zoomEnabled)
                HandleZoom();
            if (footstepsEnabled)
                HandleFootsteps();
            if (interactionEnabled)
            {
                HandleInteractionCheck();
                HandleInteractionInput();
            }
            if (staminaEnabled)
                HandleStamina();
            if (fallDamageEnabled)
                FallDamage(fallDamage);

            DamageOverlay();

            Debug.Log("Health: " + currentHealth);
```

```
            Debug.Log("Stamina: " + currentStamina);

            ApplyFinalMovement();
        }
    }
    private void HandleMovementInput()
    {
        currentInput = new Vector2((isSprinting ? sprintSpeed :
walkSpeed) * Input.GetAxis("Vertical"), (isSprinting ? sprintSpeed :
walkSpeed) * Input.GetAxis("Horizontal"));

        float moveDirectionY = moveDirection.y;
        moveDirection =
(transform.TransformDirection(Vector3.forward) * currentInput.x) +
(transform.TransformDirection(Vector3.right) * currentInput.y);
        moveDirection.y = moveDirectionY;
    }
    private void HandleMouseLook()
    {
        rotationX -= Input.GetAxis("Mouse Y") * lookSpeedY;
        rotationX = Mathf.Clamp(rotationX, -upperLookLimit,
lowerLookLimit);
        playerCamera.transform.localRotation =
Quaternion.Euler(rotationX, 0, 0);
        transform.rotation *= Quaternion.Euler(0,
Input.GetAxis("Mouse X") * lookSpeedX, 0);
    }
    private void HandleJump()
    {
        if (shouldJump)
            moveDirection.y = jumpForce;
    }
    private void CheckGround()
    {
        _grounded =
Physics.Raycast(characterController.transform.position + Vector3.up,
-Vector3.up, 1.01f);
    }
    private bool isFalling { get { return (!_grounded &&
```

```
rigidBody.velocity.y < 0); } }

    private void HandleCrouch()
    {
        if (shouldCrouch)
            StartCoroutine(CrouchStand());
    }
    private void HandleHeadBob()
    {
        if (!characterController.isGrounded) return;

        if (Mathf.Abs(moveDirection.x) > 0.1f ||
Mathf.Abs(moveDirection.z) > 0.1f)
        {
            Timer += Time.deltaTime + (isCrouching ? crouchbobSpeed :
isSprinting ? sprintbobSpeed : walkbobSpeed);
            playerCamera.transform.localPosition = new Vector3(
                playerCamera.transform.localPosition.x,
                defaultYPos + Mathf.Sin(Timer) * (isCrouching ?
crouchbobAmount : isSprinting ? sprintbobAmount : walkbobAmount),
                playerCamera.transform.localPosition.z);

            playerCamera.transform.localPosition = new Vector3(
                playerCamera.transform.localPosition.x,
                defaultYPos + Mathf.Sin(Timer) * (isCrouching ?
crouchbobAmount : isSprinting ? sprintbobAmount : walkbobAmount),
                playerCamera.transform.localPosition.z);
        }

    }
    private void HandleStamina()
    {
        if (isSprinting && currentInput != Vector2.zero)
        {
            if (regeneratingStamina != null)
            {
                StopCoroutine(regeneratingStamina);
                regeneratingStamina = null;
            }
```

```
            currentStamina -= staminaUseMultiplier * Time.deltaTime;

            if (currentStamina < 0)
                currentStamina = 0;

            OnStaminaChange?.Invoke(currentStamina);

            if (currentStamina <= 0)
                sprintEnabled = false;
        }
        if (!isSprinting && currentStamina < maxStamina &&
regeneratingStamina == null)
        {
            regeneratingStamina =
StartCoroutine(RegenerateStamina());
        }
    }

    private void HandleZoom()
    {
        if (Input.GetKeyDown(zoomKey))
        {
            if (zoomRoutine != null)
            {
                StopCoroutine(zoomRoutine);
                zoomRoutine = null;
            }
            zoomRoutine = StartCoroutine(ToggleZoom(true));

        }
        if (Input.GetKeyUp(zoomKey))
        {
            if (zoomRoutine != null)
            {
                StopCoroutine(zoomRoutine);
                zoomRoutine = null;
            }
            zoomRoutine = StartCoroutine(ToggleZoom(false));
```

```
        }
    }
    private void ApplyFinalMovement()
    {
        if (!characterController.isGrounded)
            moveDirection.y -= gravity * Time.deltaTime;

        if (slopeSlidingEnabled && isSliding)
            moveDirection += new Vector3(hitPointNormal.x,
-hitPointNormal.z) * slopeSpeed;


            characterController.Move(moveDirection * Time.deltaTime);
    }
    private void HandleFootsteps()
    {
        if (!characterController.isGrounded) return;
        if (currentInput == Vector2.zero) return;

        footstepTimer -= Time.deltaTime;

            if (footstepTimer <= 0)
            {
                footstepAudioSource.pitch =
UnityEngine.Random.Range(0.9f, 1.1f);
                if
(Physics.Raycast(characterController.transform.position,
Vector3.down, out RaycastHit hit, 3))
                {
                    switch (hit.collider.tag)
                    {
                        case "Grass":
                            if (isSprinting)

footstepAudioSource.PlayOneShot(grassRunSounds[UnityEngine.Random.Ran
ge(0, grassSounds.Length - 1)]);
                            else
                            {
```

```
footstepAudioSource.PlayOneShot(grassSounds[UnityEngine.Random.Range(
0, grassSounds.Length - 1)]);
                            }
                            break;
                        case "Dirt":
                            if (isSprinting)

footstepAudioSource.PlayOneShot(dirtRunSounds[UnityEngine.Random.Rang
e(0, grassSounds.Length - 1)]);
                            else
                            {

footstepAudioSource.PlayOneShot(dirtSounds[UnityEngine.Random.Range(0
, dirtSounds.Length - 1)]);
                            }
                            break;
                        case "Tile":
                            if (isSprinting)

footstepAudioSource.PlayOneShot(tileRunSounds[UnityEngine.Random.Rang
e(0, grassSounds.Length - 1)]);
                            else
                            {

footstepAudioSource.PlayOneShot(tileSounds[UnityEngine.Random.Range(0
, tileSounds.Length - 1)]);
                            }
                            break;
                        case "Water":
                            if (isSprinting)

footstepAudioSource.PlayOneShot(waterRunSounds[UnityEngine.Random.Ran
ge(0, grassSounds.Length - 1)]);
                            else
                            {

footstepAudioSource.PlayOneShot(waterSounds[UnityEngine.Random.Range(
0, waterSounds.Length - 1)]);
                            }
```

```
                        break;
                    default:
                    if (isSprinting)
                        footstepTimer = sprintStepSpeed;
                    else
                        footstepTimer = baseStepSpeed;
                    break;
                }
            }
        if (isSprinting)
            footstepTimer = sprintStepSpeed;
        else
            footstepTimer = baseStepSpeed;
        }
    }
    private void DamageOverlay()
    {
        float transparency = 1f - (currentHealth / 100f);
        Color imageColor = Color.white;
        imageColor.a = transparency;
        bloodOverlay.color = imageColor;
    }
    public void ApplyDamage(float damage)
    {
        currentHealth -= damage;
        OnDamage?.Invoke(currentHealth);

        if (currentHealth <= 0)
            KillPlayer();
        else if (regeneratingHealth != null)
            StopCoroutine(regeneratingHealth);

        regeneratingHealth = StartCoroutine(RegenerateHealth());
    }
    public void FallDamage(float fallDamage)
    {
        CheckGround();

        if(!wasFalling && isFalling)
```

```
        {
            beginFallHeight =
characterController.transform.position.y;
        }
        if (!wasGrounded && _grounded)
        {
            float fallDistance = beginFallHeight -
characterController.transform.position.y;
            if (fallDistance > minFallHeight)
            {
                ApplyDamage(fallDamage);
            }
        }
        wasGrounded = _grounded;
        wasFalling = isFalling;

    }
    public void KillPlayer()
    {
        currentHealth = 0;

        if (regeneratingHealth != null)
            StopCoroutine(regeneratingHealth);

        Debug.Log("DEAD");


SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex);

        //Add further implementation later
        //Make a "fade to black" animation play here and restart the
scene
    }
    private void HandleInteractionCheck()
    {
        if
(Physics.Raycast(playerCamera.ViewportPointToRay(interactionRayPoint)
, out RaycastHit hit, interactionDistance))
        {
```

```
            if (hit.collider.gameObject.layer == 8 &&
(currentInteractable == null ||
hit.collider.gameObject.GetInstanceID() !=
currentInteractable.GetInstanceID()))
            {
                hit.collider.TryGetComponent(out
currentInteractable);

                if (currentInteractable)
                    currentInteractable.OnFocus();
            }
        }
        else if (currentInteractable)
        {
            currentInteractable.OnLoseFocus();
            currentInteractable = null;
        }
    }
    private void HandleInteractionInput()
    {
        if (Input.GetKeyDown(interactKey) && currentInteractable !=
null &&
Physics.Raycast(playerCamera.ViewportPointToRay(interactionRayPoint),
out RaycastHit hit, interactionDistance, interactionLayer))
        {
            currentInteractable.OnInteract();
        }

    }
    private IEnumerator CrouchStand()
    {
        if (isCrouching &&
Physics.Raycast(playerCamera.transform.position, Vector3.up, 1f))
            yield break;


        ongoingCrouchAnimation = true;

        float timeElapsed = 0;
```

```
        float targetHeight = isCrouching ? standingHeight :
crouchHeight;
        float currentHeight = characterController.height;
        Vector3 targetCenter = isCrouching ? standingCenter :
crouchingCenter;
        Vector3 currentCenter = characterController.center;

        isCrouching = !isCrouching;

        while (timeElapsed < timeToCrouch)
        {
            characterController.height = Mathf.Lerp(currentHeight,
targetHeight, timeElapsed / timeToCrouch);
            characterController.center = Vector3.Lerp(currentCenter,
targetCenter, timeElapsed / timeToCrouch);
            timeElapsed += Time.deltaTime;
            yield return null;
        }

        characterController.height = targetHeight;
        characterController.center = targetCenter;

        ongoingCrouchAnimation = false;
    }
    private IEnumerator ToggleZoom(bool enterZoom)
    {
        float targetFOV = enterZoom ? zoomFOV : defaultFOV;
        float startingFOV = playerCamera.fieldOfView;
        float timeElapsed = 0;

        while (timeElapsed < timeToZoom)
        {
            playerCamera.fieldOfView = Mathf.Lerp(startingFOV,
targetFOV, timeElapsed / timeToZoom);
            timeElapsed += Time.deltaTime;
            yield return null;
        }

        playerCamera.fieldOfView = targetFOV;
```

```
        zoomRoutine = null;
    }
    private IEnumerator RegenerateHealth()
    {
        yield return new WaitForSeconds(regenCooldown);
        WaitForSeconds timeToWait = new
WaitForSeconds(healthTimeIncrement);

        while(currentHealth < maxHealth)
        {
            currentHealth += healthIncreaseIncrement;

            if (currentHealth > maxHealth)
                currentHealth = maxHealth;

            OnHeal?.Invoke(currentHealth);
            yield return timeToWait;
        }

        regeneratingHealth = null;
    }
    private IEnumerator RegenerateStamina()
    {
        yield return new
WaitForSeconds(timeBeforeStaminaRegenStarts);
        WaitForSeconds timeToWait = new
WaitForSeconds(staminaTimeIncrement);

        while(currentStamina < maxStamina)
        {
            if (currentStamina > 0)
                sprintEnabled = true;
            currentStamina += staminaIncreaseIncrement;

            if (currentStamina > maxStamina)
                currentStamina = maxStamina;

            yield return timeToWait;
        }
```

```
        regeneratingStamina = null;
    }
}
```

Player Abilities:

- **Overview**

This section is going to be relatively brief as most of the player's abilities were covered in the previous section. Firstly, the player is able to crouch underwater in order to crawl through flooded passages in some areas. The original plan as of submission 2, was to add an oxygen system but seeing as how navigating underwater was never a major aspect of the level design, this feature has been cut for the sake of time. The only other non-movement related player ability is the flashlight. The player's flashlight can be toggled using F and will play an essential role in navigating sparsely lit environments. Originally, the flashlight was infinite, with no limit to how long the player could use it. As of this submission, the flashlight now features a simple battery system in which the flashlight will "drain" at an adjustable rate and will turn off when its battery reaches 0. This is currently accompanied by a "flickering" sound effect but with no visual flickering effect, meaning that the light simply turns off (A proper flickering effect is something I plan to implement later). Once the battery is depleted, it will slowly recharge as long as it remains off. I coded this to allow for easy changes in case I want to add physical battery item pick-ups later.

- **Code**

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

public class flashlight : MonoBehaviour
{
    [Header("Flashlight Parameters")]
    public AudioSource AudioSource;
    public AudioClip flashlightClick;
    public AudioClip flashlightFlicker;
    public GameObject followTarget;
    public Light flashLight;
    public KeyCode lightToggle = KeyCode.F;
    private Vector3 vectorOffset;
    [SerializeField] private float speed = 3.0f;
```

```
    [Header("Battery Life Parameters")]
    [SerializeField] private bool batteryLifeEnabled = true;
    [SerializeField] private float maxBattery = 100;
    [SerializeField] private float batteryUseMultiplier = 1;
    [SerializeField] private float timeBeforeBatteryRegenStarts = 5;
    [SerializeField] private float batteryIncreaseIncrement = 2;
    [SerializeField] private float batteryTimeIncrement = 0.1f;
    private float currentBatteryLevel;
    private Coroutine rechargingBattery;
    public static Action<float> OnBatteryChange;




    void Start()
    {
        currentBatteryLevel = maxBattery;
        flashLight = GetComponent<Light>();
        GetComponent<Light>().enabled = false;
        followTarget = Camera.main.gameObject;
        vectorOffset = transform.position -
followTarget.transform.position;
    }


    void Update()
    {
       HandleFlashlight();
       HandleBattery();

       Debug.Log("Battery: " + currentBatteryLevel);
    }
    private void HandleBattery()
    {
        if (batteryLifeEnabled && GetComponent<Light>().enabled)
```

```
        {
            if (rechargingBattery != null)
            {
                StopCoroutine(rechargingBattery);
                rechargingBattery = null;
            }
            currentBatteryLevel -= batteryUseMultiplier *
Time.deltaTime;

            if (currentBatteryLevel < 0)
                currentBatteryLevel = 0;

            OnBatteryChange?.Invoke(currentBatteryLevel);

            if (currentBatteryLevel <= 0)
            {
                StartCoroutine(BatteryDeath());
            }
        }
        if (!GetComponent<Light>().enabled && currentBatteryLevel <
maxBattery && rechargingBattery == null)
        {
            rechargingBattery = StartCoroutine(RechargeBattery());
        }
    }
    private void HandleFlashlight()
    {
        transform.position = followTarget.transform.position +
vectorOffset;
        transform.rotation = Quaternion.Slerp(transform.rotation,
followTarget.transform.rotation, speed * Time.deltaTime);
        if (Input.GetKeyDown(lightToggle))
        {
            if (GetComponent<Light>().enabled)
            {
                AudioSource.PlayOneShot(flashlightClick);
                GetComponent<Light>().enabled = false;
            }
            else if (!GetComponent<Light>().enabled &&
```

```
currentBatteryLevel > 0)
            {
                AudioSource.PlayOneShot(flashlightClick);
                GetComponent<Light>().enabled = true;
            }

        }
    }
    private IEnumerator RechargeBattery()
    {
        yield return new
WaitForSeconds(timeBeforeBatteryRegenStarts);
        WaitForSeconds timeToWait = new
WaitForSeconds(batteryTimeIncrement);

        while (currentBatteryLevel < maxBattery)
        {
            currentBatteryLevel += batteryIncreaseIncrement;

            if (currentBatteryLevel > maxBattery)
                currentBatteryLevel = maxBattery;

            yield return timeToWait;
        }
        rechargingBattery = null;
    }
    private IEnumerator BatteryDeath()
    {
        AudioSource.PlayOneShot(flashlightFlicker);
        yield return new WaitForSeconds(2);
        GetComponent<Light>().enabled = false;
    }
}
```

Environments:

- **Overview**

Originally, the plan was to create 4 separate levels for this project, however I have since reduced that scope to primarily focus on only 3 levels. The original level plan was the following: The Asylum (1), The Tunnels (2), Shallow Poolrooms (3), and the Dark Poolrooms (4). Due to a lack of compelling assets, the entire concept for scene 2 was reworked into dark and empty city streets. The idea here was to reuse the modular city building assets that I had purchased for my CS-420 VR project. Due to the time constraints of the term, it was decided prior to midterms to postpone this level entirely. Currently, the Asylum, the Shallow Poolrooms, and the Dark Poolrooms are all about 90% completed. Firstly, the Asylum has been expanded and polished considerably, with the design being in line with the original plan that I had laid out in the second submission. The level layout starts out as entirely maze-like but then ends in a more open ended looping corridor that is dotted with many explorable side rooms. At the current moment in time, most of these side rooms are furnished but there are still quite a few that need further detail and polish. As this doesn't affect the gameplay of the level, it was not considered a priority for this submission as there were functional aspects of the level that needed attention. Several of the more open ended areas have had numerous deadends inserted in order to make navigating to the exit far less straightforward and to increase the tension once enemies are added to the mix. Furthermore, the third scene aka the Shallow Poolrooms has remained largely unchanged except for the addition of more side rooms and deadends being added to prevent players from being able to walk straight to the exit. Lastly, the third level aka the Dark Poolrooms has had additional floors added to several of its rooms along with two stairways that lead players to the exit in two distinct paths. These extra floors help to give the level a more open ended and unique feel compared to the previous two scenes. Lastly, both of the two distinct routes through the level end in another maze-like area which stands apart from other areas by having massive corridors that increase discomfort by making the player feel extremely small and insignificant. This sense of scale is something that I wanted to increase as the level progressed in order to contrast the darker and more claustrophobic areas at the beginning.

- **Level Development**
    - **Scene 1: The Asylum**
        - Work Completed:
            - Level Layout - 95% Done
            - Level Transition Trigger
            - Level Texturing
            - Level Props - 60% Done
            - Level Lighting - 95% Done
        - To-Do:
            - Add Environmental Ambience
            - Add Reverb Zones
    - **Scene 3: Shallow Waters**
        - Work Completed:
            - Level Layout - 95% Done
            - Level Transition Trigger
            - Environmental Ambience
            - Light Sources - 50% Done
            - Reflection Probes
            - Reverb Zones
            - Level Props
    - **Scene 4: Deep Waters**
        - Work Completed:
            - Level Layout - 95% Done
            - Environmental Ambience
            - Light Sources - 10% Done
            - Reflection Probes
            - Reverb Zones
            - Level Props - 90% Done

- **Level Layouts**

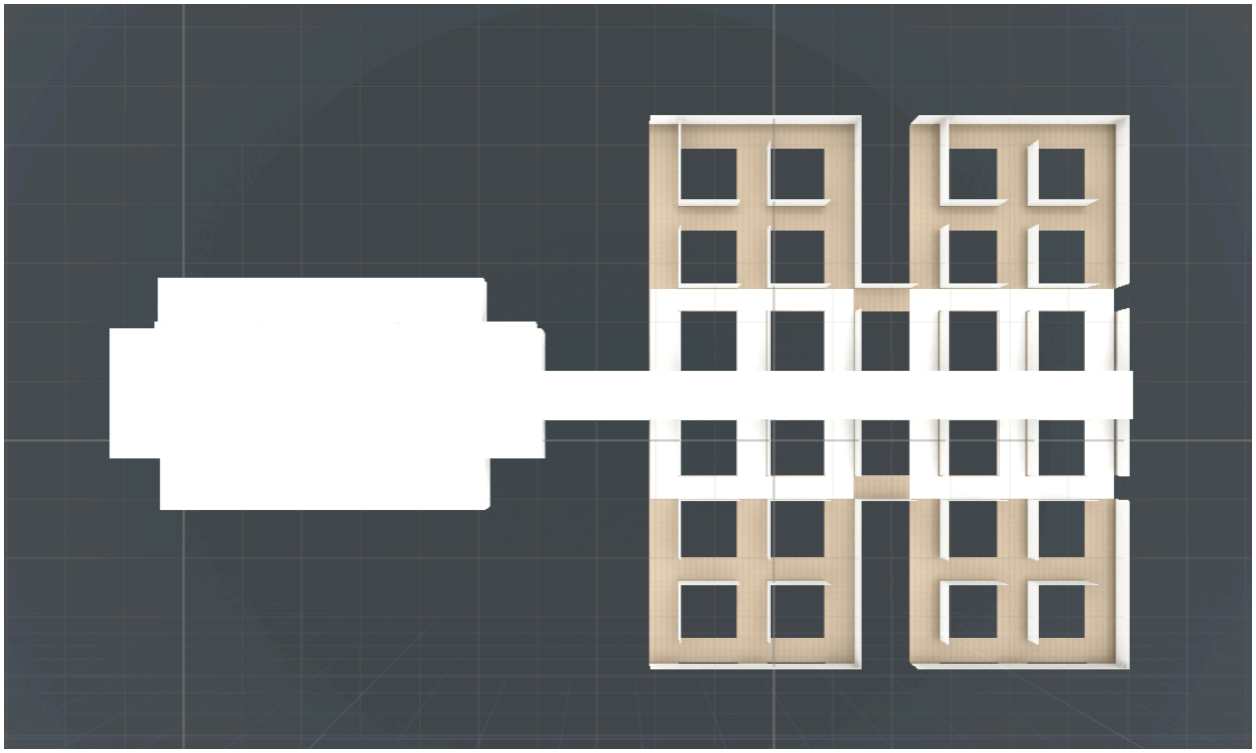

Figure 11: Scene 1 WIP Layout, August 19, 2024, (Submission 2)

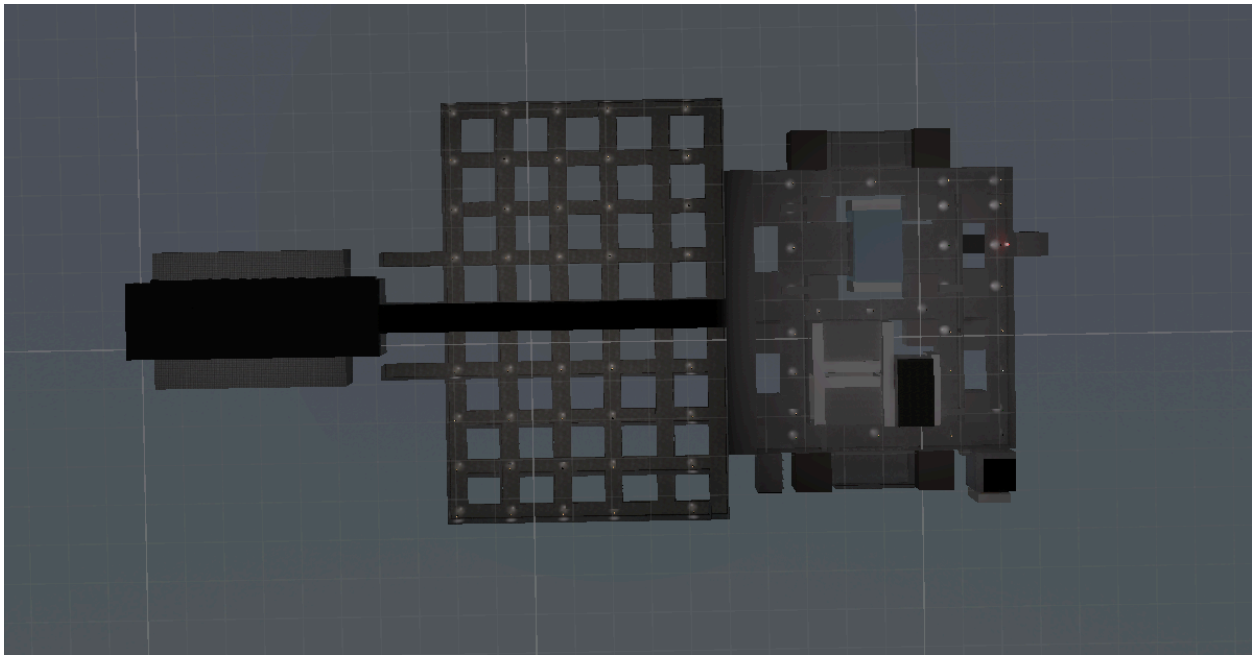

Figure 12: Scene 1 Final Layout, September 29, 2024, (Submission 3)

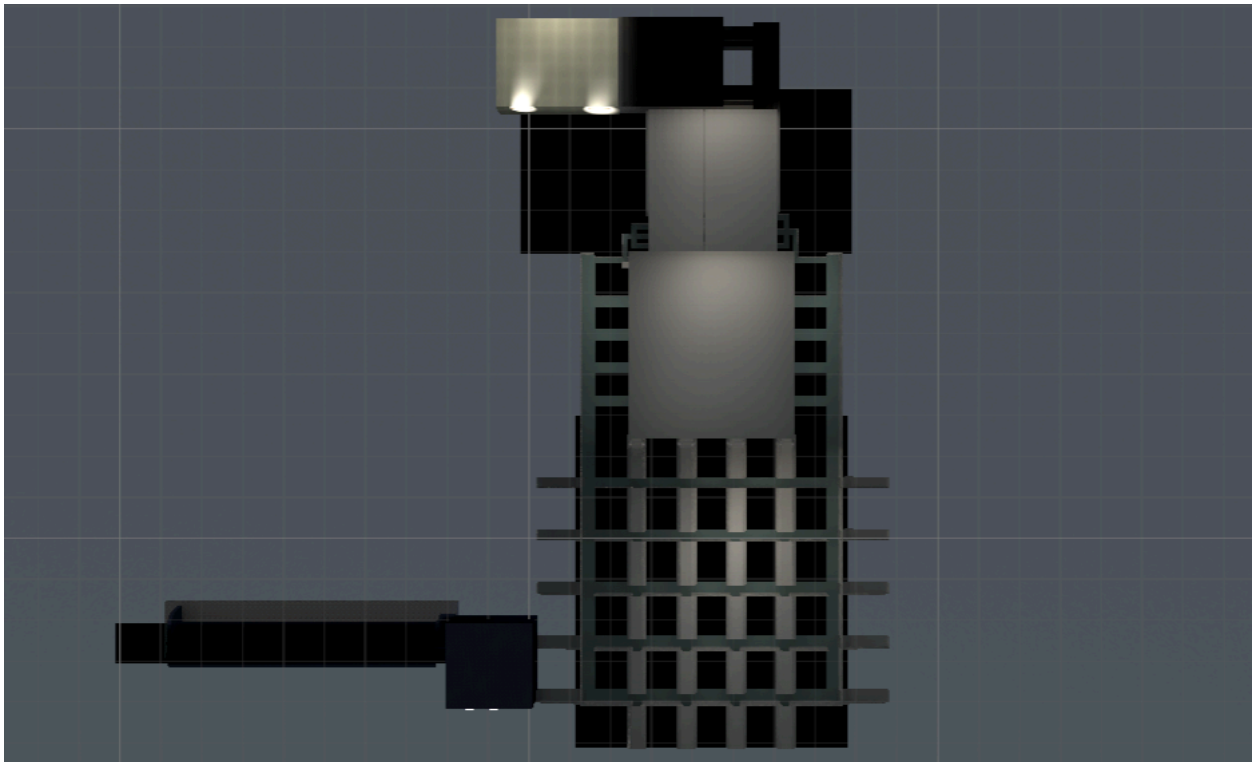

Figure 13: Scene 2 Layout, August 19, 2024, (Submission 2)

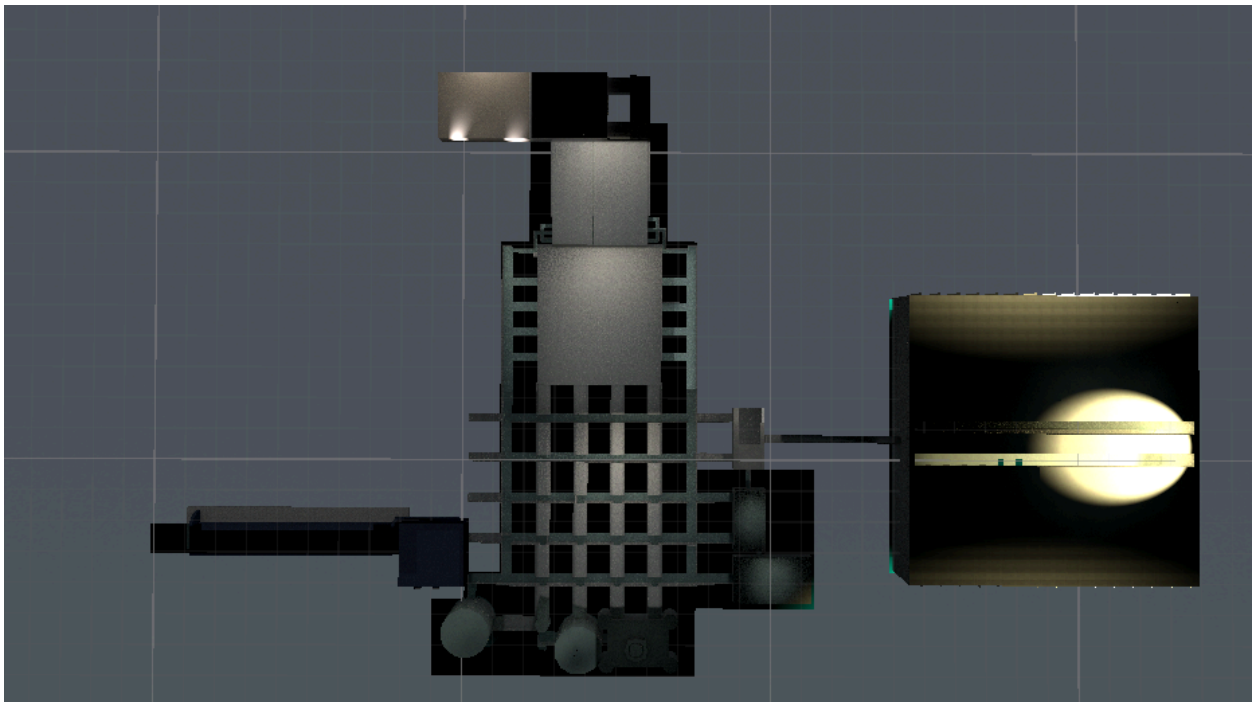

Figure 14: Scene 2 Layout, September 29, 2024, (Submission 3)

Figure 14: Scene 3 Layout, August 19, 2024, (Submission 2)

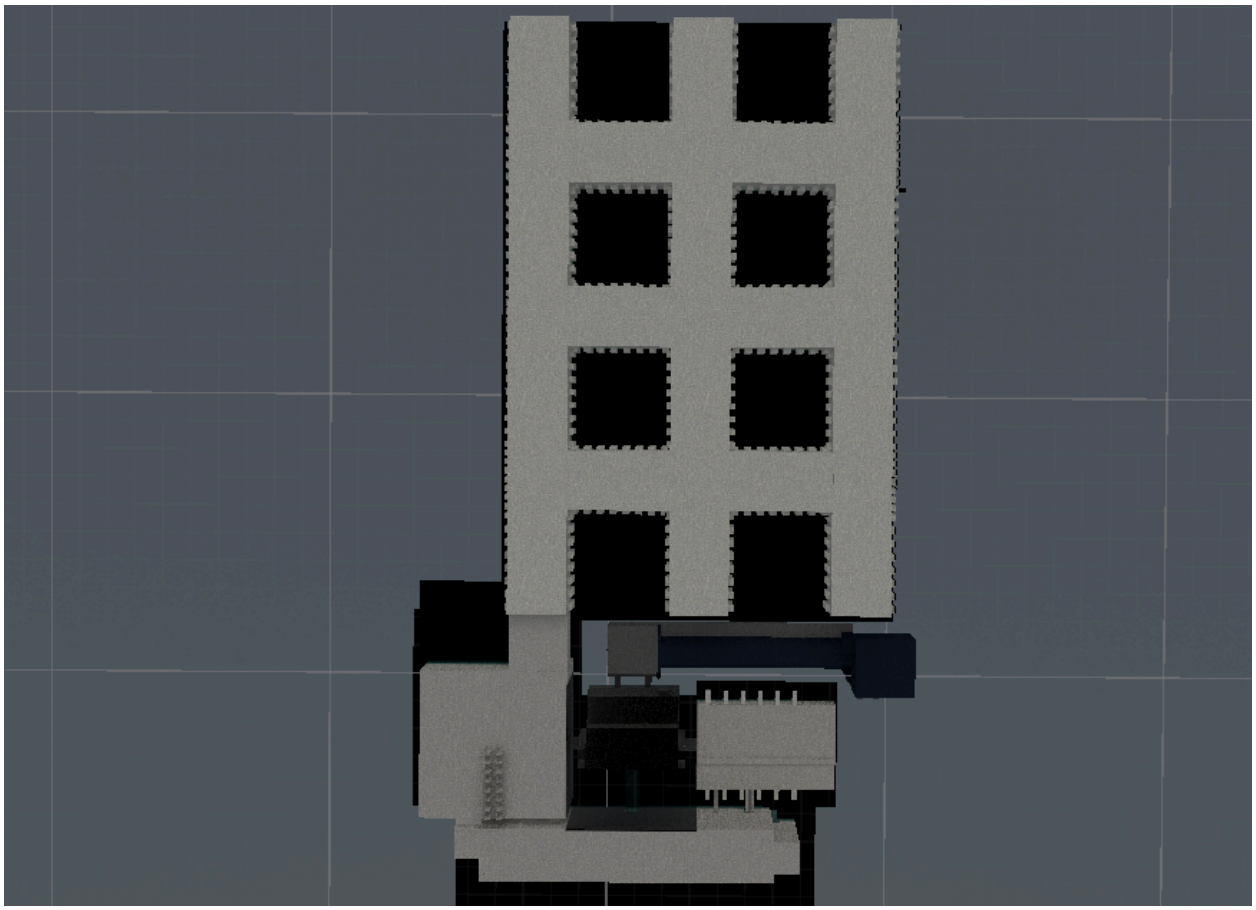

Figure 14: Scene 3 Layout, September 29, 2024, (Submission 3)

User-Interface:

- **Overview**

The user interface has seen some considerable improvement since the last submission. The main menu and pause menu have seen some major artistic changes through the implementation of new fonts and sizing tweaks. The main menu has seen the least amount of change, while the pause menu has been completely overhauled with the addition of an animated background and sound effects. The previously barren settings menu has also been almost entirely finished. The available settings include video resolution, graphics presets, fullscreen toggle and master volume control. The list of video resolutions is automatically generated based on what is available on the target computer. The graphics preset selection has full functionality but at this point in time, I still need to optimize each quality preset individually. Additionally, basic implementation of the required JSON.net package functionality for the save system has been started in the form of the IDataService and JSONDataService scripts, but still needs to be fully implemented for my use-case. Meaning that the saving and loading of level progress is not an available feature quite yet. Lastly, the in-game HUD has seen major improvements through the addition of a working date/time system with adjustable formatting and some tweaked post-processing effects. As a band-aid fix for something I was not satisfied with, the HUD text was changed from a screen overlay to a world space UI element so that it would be affected by the aforementioned post-processing effects. This is not entirely ideal but it has also been extensively tweaked and adjusted so that it won't clip with objects unless the user is trying especially hard to do so. In addition to the tweaked post-processing effects, audio sources were added to the master mixing group and a low-pass filter and compressor were added on the high end in order to create that crunchy VHS camcorder style audio. Lastly, a level manager script was created in order to handle level transitions and loading screen behavior.

- **UI Development**
    - **Main Menu**
        - **Done:**
            - Start Game Button Functionality
            - Settings Button Functionality
            - Quit Button Functionality
            - Settings Menu Functionality
        - **To-Do:**
            - Save/Load System - 50% Done (JSON.net scripts written, still need to be properly implemented)
    - **Pause Menu**
        - **Done:**
            - Resume Game Button Functionality
            - Settings Button Functionality
            - Main Menu Button Functionality
            - Quit Button Functionality
            - Sounds + Background Effects
            - Settings Menu Functionality
    - **Settings Menu**
        - **Done:**
            - Created Canvas Object
            - Audio Settings
            - Graphics Settings
    - **HUD**
        - **Done**
            - Working Clock and Date Text with Adjustable Formatting
            - Faux “Play” indicator
            - Blood on Lens Effect (Advanced Feature)
        - **To-Do**
            - Flashlight Battery Meter
            - Zoom Indicator (Not essential, can use audio queues)
            - Stamina Meter (Not essential, can use audio queues)

Figure 15: WIP HUD, August 19, 2024, (Submission 2)

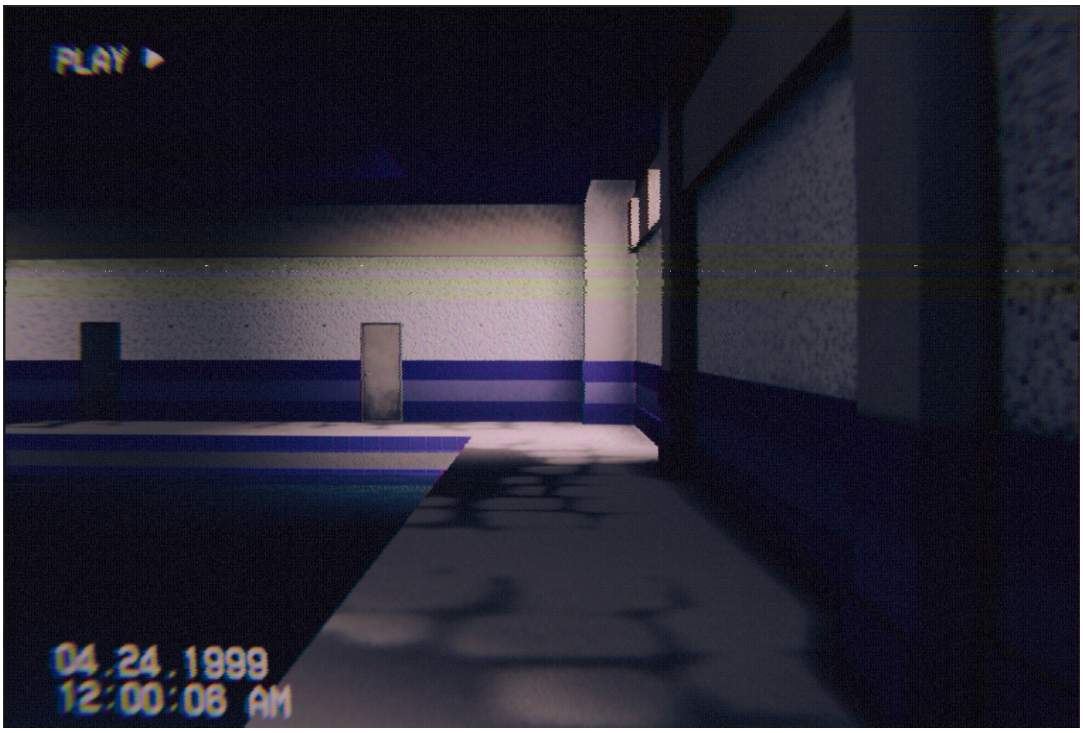

Figure 16: WIP HUD, September 29, 2024, (Submission 3)

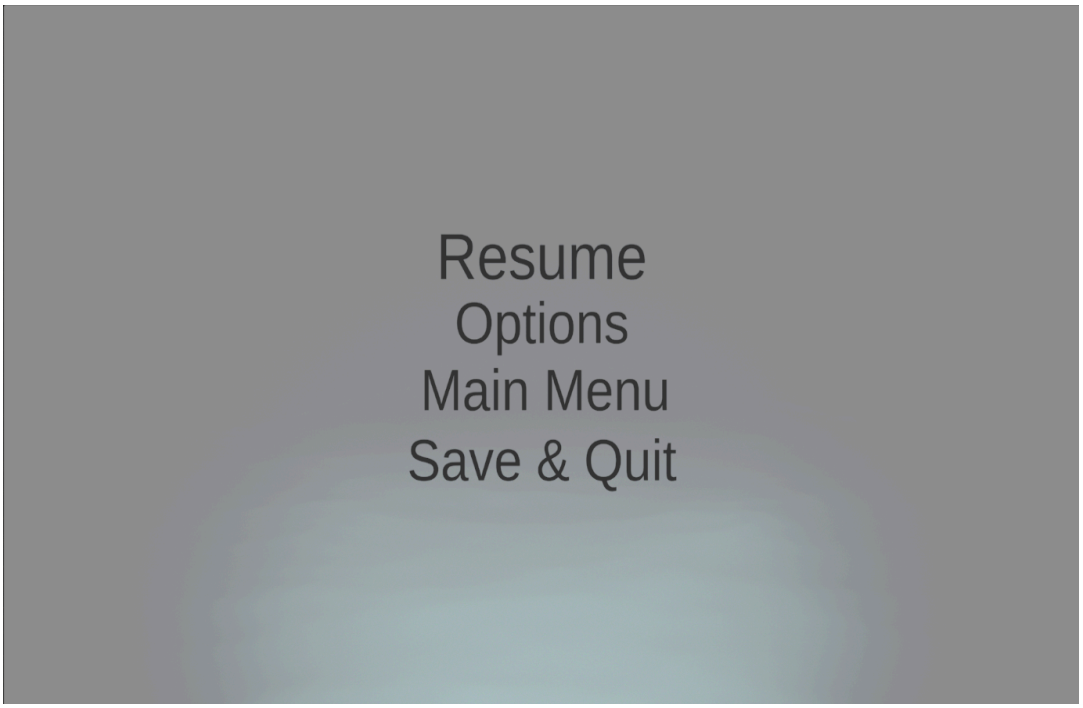

Figure 17: Pause Menu WIP, August 19, 2024, (Submission 2)

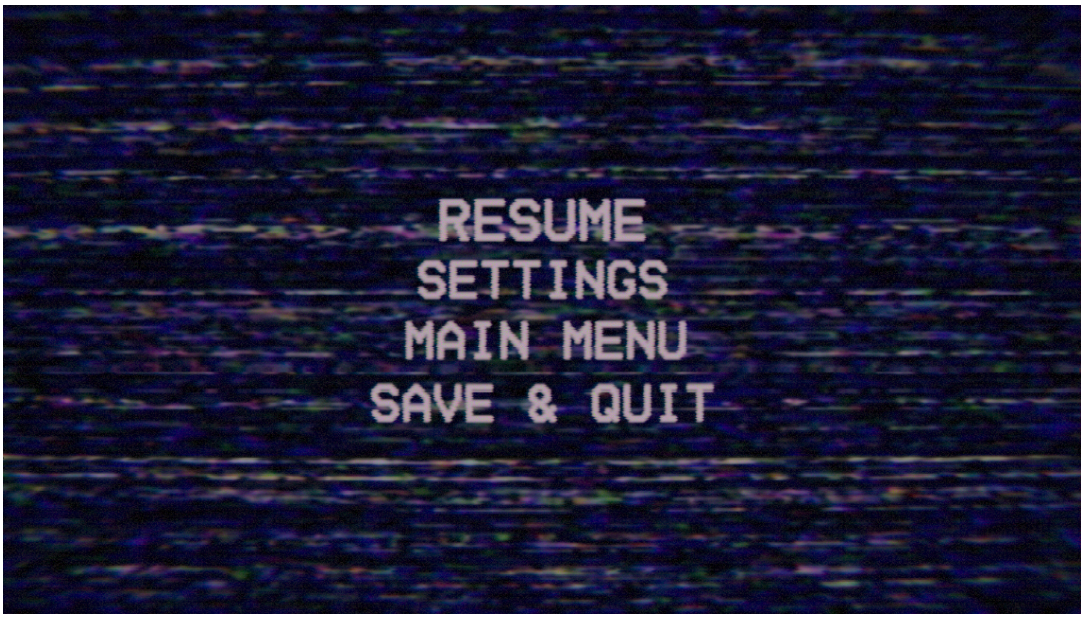

Figure 18: Final Pause Menu, September 29, 2024, (Submission 3)

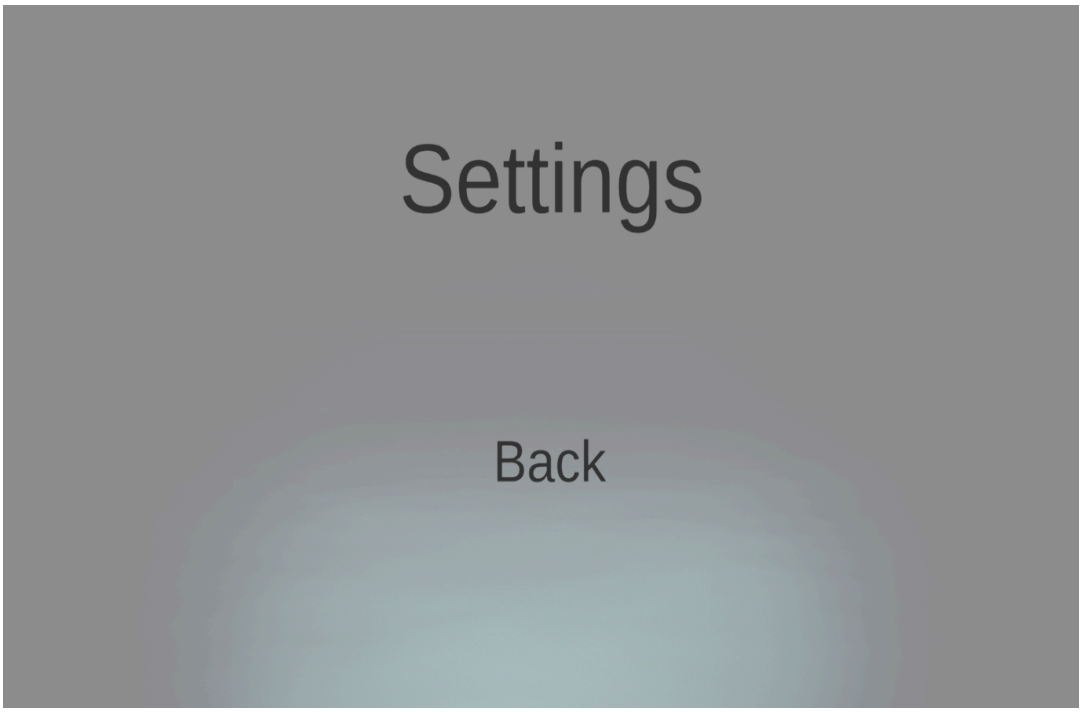

Figure 19: WIP Settings Menu, August 19, 2024, (Submission 2)

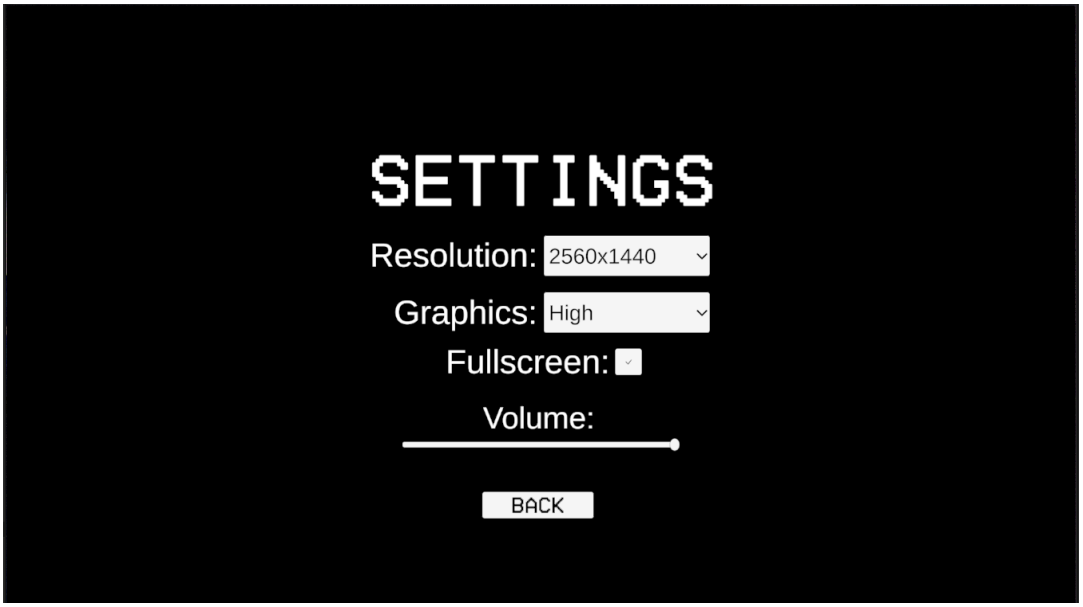

Figure 20: Settings Menu, September 29, 2024, (Submission 3)

Figure 21: WIP Main Menu, August 19, 2024, (Submission 2)

Figure 22: Main Menu, September 29, 2024, (Submission 3)

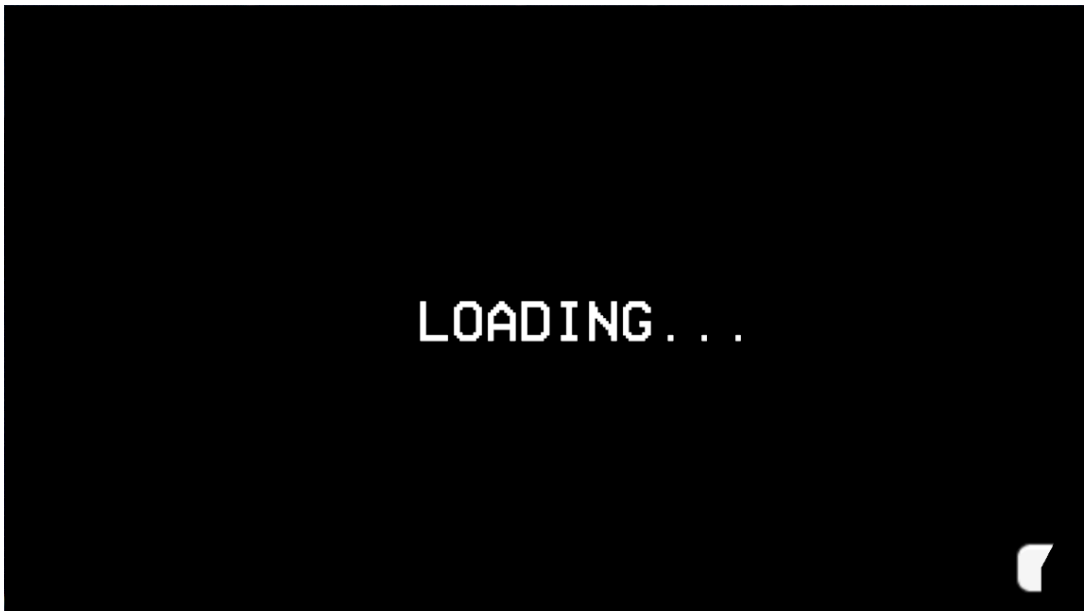

Figure 23: Loading Screen, September 29, 2024

Main Menu Code:

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

public class MainMenu : MonoBehaviour
{
    public LevelManager LevelManager;
    public void NewGame()
    {
        LevelManager.LoadNextLevel();
    }
    public void LoadGame()
    {
        //add load game functionality
    }
    public void OnApplicationQuit()
    {
        Application.Quit();
    }
}
```

Pause Menu Code:

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

public class PauseMenu : MonoBehaviour
{
    public GameObject pauseMenu;
    public GameObject timeText;
    public GameObject playText;
    public GameObject dateText;
    public NewFPSController playerController;

    public bool isPaused;

    // Start is called before the first frame update
    void Start()
    {
        pauseMenu.SetActive(false);
        timeText.SetActive(true);
        playText.SetActive(true);

    }

    // Update is called once per frame
    void Update()
    {
        if(Input.GetKeyDown(KeyCode.Escape))
        {
           if (isPaused)
            {
                ResumeGame();
            }
            else
            {
                PauseGame();
            }
```

```
        }
    }
     public void PauseGame()
    {
        pauseMenu.SetActive(true);
        timeText.SetActive(false);
        playText.SetActive(false);
        Cursor.lockState = CursorLockMode.Confined;
        Cursor.visible = true;
        playerController.mouseLookEnabled = false;
        Time.timeScale = 0f;
        isPaused = true;
    }
    public void ResumeGame()
    {
        pauseMenu.SetActive(false);
        timeText.SetActive(true);
        playText.SetActive(true);
        Cursor.visible = false;
        playerController.mouseLookEnabled = true;
        Time.timeScale = 1f;
        isPaused = false;
    }
    public void QuitGame()
    {
        Application.Quit();
    }
    public void MainMenu()
    {
        Time.timeScale = 1f;
        SceneManager.LoadScene(0);
    }
}
```

Settings Menu Code:

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

public class SettingsMenu : MonoBehaviour
{
    public AudioMixer audioMixer;

    public TMP_Dropdown resolutionDropdown;

    Resolution[] resolutions;

    private void Start()
    {
        resolutions = Screen.resolutions;

        resolutionDropdown.ClearOptions();

        List<string> options = new List<string>();

        int currentResolutionIndex = 0;

        for (int i = 0; i < resolutions.Length; i++)
        {
            string option = resolutions[i].width + "x" +
resolutions[i].height;
            options.Add(option);

            if (resolutions[i].width ==
Screen.currentResolution.width && resolutions[i].height ==
Screen.currentResolution.height)
            {
                currentResolutionIndex = i;
            }
        }
```

```
        resolutionDropdown.AddOptions(options);
        resolutionDropdown.value = currentResolutionIndex;
        resolutionDropdown.RefreshShownValue();
    }
    public void SetVolume (float volume)
    {
        audioMixer.SetFloat("volume", volume);
    }

    public void SetQuality (int qualityIndex)
    {
        QualitySettings.SetQualityLevel(qualityIndex);
    }

    public void SetFullScreen (bool isFullScreen)
    {
        Screen.fullScreen = isFullScreen;
    }

    public void setResolution (int resolutionIndex)
    {
        Resolution resolution = resolutions[resolutionIndex];
        Screen.SetResolution(resolution.width, resolution.height,
Screen.fullScreen);
    }
}
```

HUD Clock Code:

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

public class UIClock : MonoBehaviour
{
    public string clockFormat;

    public TextMeshProUGUI clockText;

    public System.TimeSpan timeSpan = new System.TimeSpan(0, 0, 0, 0,
0);

    public System.DateTime date = new System.DateTime(1999,04,24);

    public float timeRate = 1;

    private void Update()
    {
        float milliseconds = Time.deltaTime * 1000 * timeRate;

        timeSpan += new System.TimeSpan(0, 0, 0, 0,
(int)milliseconds);

        System.DateTime dateTime = date.Add(timeSpan);

        clockText.text = dateTime.ToString(@clockFormat, new
CultureInfo("en-US"));
    }
    public void AddTime(int value)
    {
        timeSpan += new System.TimeSpan(value, 0, 0);
    }
}
```

Advanced Features:

Blood/Damage Overlay - 100% Complete

- The first of my advanced features was to add a blood/damage overlay that appears on screen when damage is taken and slowly fades away as the player heals. This effect has been seen in countless FPS games such as the past 16 years of Call of Duty titles, and was surprisingly straightforward to implement. A UI element to represent the effect was added to the player asset, along with a function in the character controller script. The script method simply detects whether damage has been taken, and scales the opacity of the screen overlay with the player's health. This method is then called within the update method. It should be noted that all of my AI enemies can kill the player in one hit (and utilize separate "jumpscare" cameras), and as such the blood effect isn't used in these cases. Instead, the blood effect is used whenever the player takes fall damage.
- **Damage Overlay Code:**

```
private void DamageOverlay()
{
    float transparency = 1f - (currentHealth / 100f);
    Color imageColor = Color.white;
    imageColor.a = transparency;
    bloodOverlay.color = imageColor;
}
```

Occlusion Culling - 95% Complete

- This was another feature that was fairly straightforward to implement as I am simply using Unity's provided occlusion system for the sake of time and simplicity. I would like to note that I did spend considerable time researching the built-in occlusion system and how it actually functions so that I wasn't being entirely hand held throughout the process. Although, it was as simple as selecting each of the individual level segments and selecting "Occluder Static" for level geometry and "Occludee Static" for level props. In addition to this, occlusion areas were also created for each of the major level segments which acts very similar to a traditional collision based layer occlusion system. Lastly, it is hard to visualize the implementation of this feature as from what I have seen, Unity doesn't show the occlusion in the editor window unless you are in the Occlusion settings tab. I am considering this feature implementation to be 95% complete as there are still some minor tweaks and adjustments that need to be made.

- **Occlusion Culling Example:**

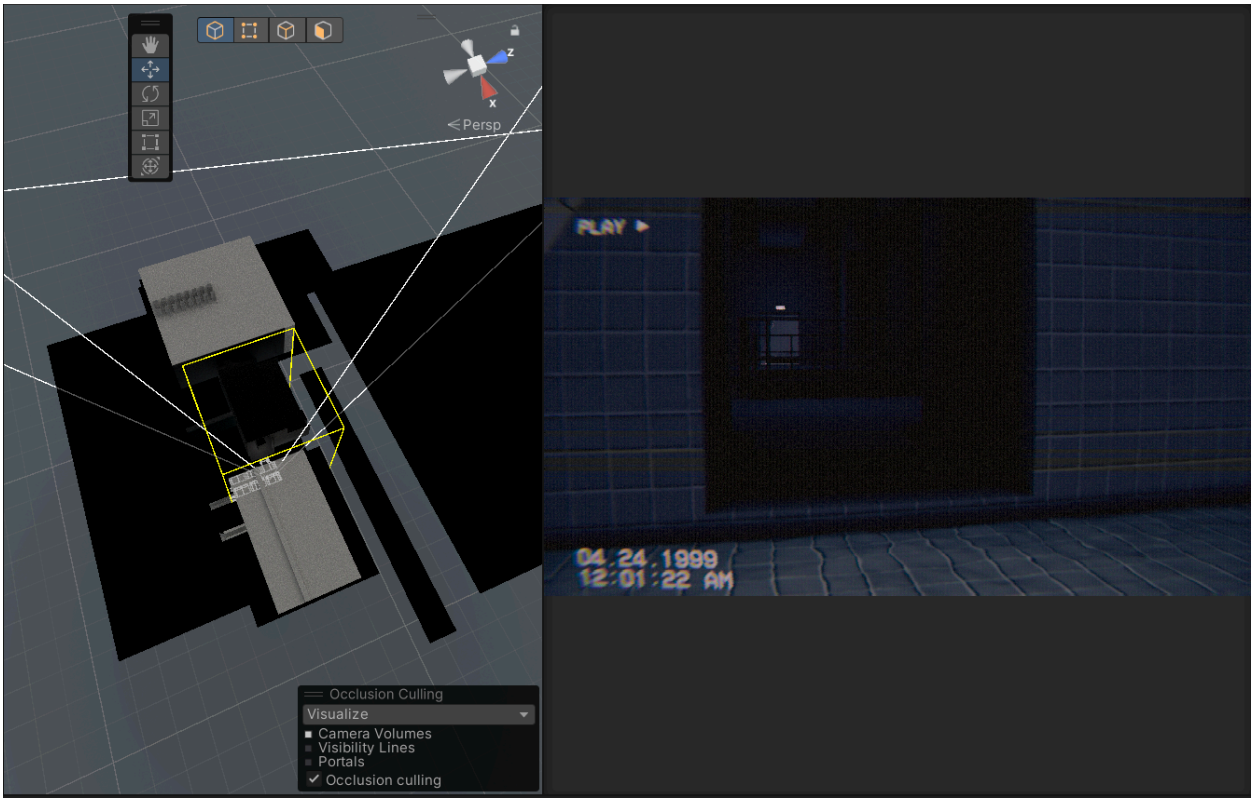

Figure 24: Scene 3 Occlusion Culling, September 29, 2024

AI Characters - 90% Complete

- In terms of AI, I have two separate enemies with their own distinct behavior. THer first enemy is my poolrooms antagonist, a silhouette made of water (Dubbed the waterman for development purposes). This character is based on my previously existing testAI script which utilizes manually placed patrol destinations and a simple state machine consisting of walking, chasing, and searching states. The AI will start off by randomly walking between each destination and idling in-between, with a chance to play a "searching" animation in which he looks around. If you get close enough to the AI, he will enter into the chasing state in which he will begin sprinting toward the player. If you get caught, the player's view is switched to a "jumpscare" camera that is focused on the enemy and an attack animation is played. Currently, additional effects and sounds are still needed for when the player is killed by an enemy. Lastly, a simpler version of the footstep method from the character controller was utilized in order to give audible sound cues to his movement.
- **Waterman AI Code:**

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

public class enemyAI : MonoBehaviour
{
    public NavMeshAgent ai;
    public Camera killCam;
    public Camera mainCam;
    public List<Transform> destinations;
    public Animator aiAnim;
    public Transform player;
    public NewFPSController playerController;
    [Header("AI Settings")]
    public float walkSpeed, chaseSpeed, minIdleTime, maxIdleTime,
idleTime, detectionDistance, catchDistance, searchDistance,
minChaseTime, maxChaseTime, minSearchTime, maxSearchTime,
jumpscareTime;
    public bool walking, chasing, searching;

    Transform currentDest;
    Vector3 dest;
```

```
    public Vector3 rayCastOffset;
    public float aiDistance;
    private float footstepTimer;

    [Header("Footstep Sounds")]
    [SerializeField] private AudioClip[] waterSounds = default;
    [SerializeField] private float baseStepSpeed = 0.7f;
    [SerializeField] private float sprintStepSpeed = 0.3f;
    [SerializeField] private AudioSource footstepAudioSource =
default;
    //public GameObject hideText, stopHideText;

    private void Start()
    {
        walking = true;
        currentDest = destinations[Random.Range(0,
destinations.Count)];
        killCam.enabled = false;
    }
    private void Update()
    {
        Vector3 direction = (player.position -
transform.position).normalized;
        RaycastHit hit;
        aiDistance = Vector3.Distance(player.position,
this.transform.position);
        if (Physics.Raycast(transform.position + rayCastOffset,
direction, out hit, detectionDistance))
        {
            if (hit.collider.gameObject.tag == "Player")
            {
                walking = false;
                StopCoroutine("stayIdle");
                StopCoroutine("searchRoutine");
                StartCoroutine("searchRoutine");
                searching = true;
            }
        }
        if (searching == true)
```

```
        {
            ai.speed = 0;
            aiAnim.ResetTrigger("walk");
            aiAnim.ResetTrigger("idle");
            aiAnim.ResetTrigger("sprint");
            aiAnim.SetTrigger("search");
            if (aiDistance <= searchDistance)
            {
                StopCoroutine("stayIdle");
                StopCoroutine("searchRoutine");
                StopCoroutine("chaseRoutine");
                StartCoroutine("chaseRoutine");
                chasing = true;
                searching = false;
            }
        }
        if (chasing == true)
        {
            HandleFootsteps();
            dest = player.position;
            ai.destination = dest;
            ai.speed = chaseSpeed;
            aiAnim.ResetTrigger("walk");
            aiAnim.ResetTrigger("idle");
            aiAnim.ResetTrigger("search");
            aiAnim.SetTrigger("sprint");
            if (aiDistance <= catchDistance)
            {
                aiAnim.ResetTrigger("walk");
                aiAnim.ResetTrigger("idle");
                aiAnim.ResetTrigger("search");
                //hideText.SetActive(false);
                //stopHideText.SetActive(false);
                aiAnim.ResetTrigger("sprint");
                aiAnim.SetTrigger("jumpscare");
                ai.speed = 0;
                killCam.enabled = true;
                mainCam.enabled = false;
                StartCoroutine("deathRoutine");
```

```
                chasing = false;
            }
        }
        if (walking == true)
        {
            HandleFootsteps();
            dest = currentDest.position;
            ai.destination = dest;
            ai.speed = walkSpeed;
            aiAnim.ResetTrigger("sprint");
            aiAnim.ResetTrigger("idle");
            aiAnim.ResetTrigger("search");
            aiAnim.SetTrigger("walk");
            if (ai.remainingDistance <= ai.stoppingDistance)
            {
                aiAnim.ResetTrigger("sprint");
                aiAnim.ResetTrigger("walk");
                aiAnim.ResetTrigger("search");
                aiAnim.SetTrigger("idle");
                ai.speed = 0;
                StopCoroutine("stayIdle");
                StartCoroutine("stayIdle");
                walking = false;
            }
        }
    }
    public void stopChase()
    {
        walking = true;
        chasing = false;
        StopCoroutine("chaseRoutine");
        currentDest = destinations[Random.Range(0,
destinations.Count)];
    }
    private void HandleFootsteps()
    {
        footstepTimer -= Time.deltaTime;
        if (walking == true)
```

```
        {
            if (footstepTimer <= 0)
            {
                footstepAudioSource.pitch =
UnityEngine.Random.Range(0.9f, 1.1f);

footstepAudioSource.PlayOneShot(waterSounds[UnityEngine.Random.Range(
0, waterSounds.Length - 1)]);
                footstepTimer = baseStepSpeed;
            }
        }
        else if (chasing == true)
        {
            if (footstepTimer <= 0)
            {
                footstepAudioSource.pitch =
UnityEngine.Random.Range(0.9f, 1.1f);

footstepAudioSource.PlayOneShot(waterSounds[UnityEngine.Random.Range(
0, waterSounds.Length - 1)]);
                footstepTimer = sprintStepSpeed;
            }
        }
    }
    IEnumerator stayIdle()
    {
        idleTime = Random.Range(minIdleTime, maxIdleTime);
        yield return new WaitForSeconds(idleTime);
        walking = true;
        currentDest = destinations[Random.Range(0,
destinations.Count)];
    }
    IEnumerator searchRoutine()
    {
        yield return new WaitForSeconds(Random.Range(minSearchTime,
maxSearchTime));
        searching = false;
        walking = true;
        currentDest = destinations[Random.Range(0,
```

```
destinations.Count)];
    }
    IEnumerator chaseRoutine()
    {
        yield return new WaitForSeconds(Random.Range(minChaseTime,
maxChaseTime));
        stopChase();
    }
    IEnumerator deathRoutine()
    {
        yield return new WaitForSeconds(jumpscareTime);
        playerController.KillPlayer();
    }
}
```

- The other enemy type is a "Weeping Angel" type enemy that will only move when the player is looking at them. There are two implementations of this behavior right now: a hostile mannequin with walking animations and sounds that start/stop and a non-hostile ghost enemy that remains static with no animations or sounds. This script is fairly simple as it sets the AI agent's destination to be the player and simply sets their movement and animation speeds to 0 when they are within the player's sight.
- **Mannequin AI Code:**

```
using System.Collections;
using System.Collections.Generic;
using Unity.VisualScripting;
using UnityEngine;
using UnityEngine.AI;

public class mannequinAI : MonoBehaviour
{
    public GameObject deathStatic;
    public Animator aiAnimator;
    public NewFPSController playerController;
    public NavMeshAgent mannequin;
    public Transform player;
    Vector3 destination;
    public Camera playerCam;
    public Camera killCam;
```

```
    public float aiSpeed;
    public float catchDistance;
    public float jumpscareTime;
    private float footstepTimer;

    [Header("Footstep Sounds")]
    [SerializeField] private float baseStepSpeed = 0.7f;
    [SerializeField] private AudioSource footstepAudioSource =
default;
    [SerializeField] private AudioClip[] grassSounds = default;
    [SerializeField] private AudioClip[] dirtSounds = default;
    [SerializeField] private AudioClip[] tileSounds = default;
    [SerializeField] private AudioClip[] waterSounds = default;

    private void Start()
    {
        killCam.enabled = false;
    }
    private void Update()
    {
        Plane[] planes =
GeometryUtility.CalculateFrustumPlanes(playerCam);

        float distance = Vector3.Distance(transform.position,
player.position);

        if (GeometryUtility.TestPlanesAABB(planes,
this.gameObject.GetComponent<Renderer>().bounds))
        {
            mannequin.speed = 0;
            aiAnimator.speed = 0;
            mannequin.SetDestination(transform.position);
        }

        if (!GeometryUtility.TestPlanesAABB(planes,
this.gameObject.GetComponent<Renderer>().bounds))
        {
            HandleFootsteps();
            mannequin.speed = aiSpeed;
```

```
            aiAnimator.speed = 1;
            destination = player.position;
            mannequin.destination = destination;

            if (distance <= catchDistance)
            {
                aiAnimator.ResetTrigger("walking");
                aiAnimator.SetTrigger("idle");
                playerCam.enabled = false;
                killCam.enabled = true;
                aiAnimator.speed = 0;
                deathStatic.SetActive(true);
                StartCoroutine("deathRoutine");
            }

        }
    }
    IEnumerator deathRoutine()
    {
        yield return new WaitForSeconds(jumpscareTime);
        playerController.KillPlayer();
    }
    private void HandleFootsteps()
    {

        footstepTimer -= Time.deltaTime;

        if (footstepTimer <= 0)
        {
            footstepAudioSource.pitch =
UnityEngine.Random.Range(0.9f, 1.1f);
            if (Physics.Raycast(this.transform.position,
Vector3.down, out RaycastHit hit, 3))
            {
                switch (hit.collider.tag)
                {

                    case "Grass":
                        {
```

```
footstepAudioSource.PlayOneShot(grassSounds[UnityEngine.Random.Range(
0, grassSounds.Length - 1)]);
                        }
                        break;
                    case "Dirt":
                        {

footstepAudioSource.PlayOneShot(dirtSounds[UnityEngine.Random.Range(0
, dirtSounds.Length - 1)]);
                        }
                        break;
                    case "Tile":
                        {

footstepAudioSource.PlayOneShot(tileSounds[UnityEngine.Random.Range(0
, tileSounds.Length - 1)]);
                        }
                        break;
                    case "Water":
                        {

footstepAudioSource.PlayOneShot(waterSounds[UnityEngine.Random.Range(
0, waterSounds.Length - 1)]);
                        }
                        break;
                    default:
                        footstepTimer = baseStepSpeed;
                        break;
                }
            }
            footstepTimer = baseStepSpeed;
        }
    }
}
```

- **Ghost AI Script:**

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

public class stalkerAI : MonoBehaviour
{
    public NavMeshAgent stalker;
    public Transform player;
    Vector3 destination;
    public Camera playerCam;
    public float aiSpeed;

    private void Update()
    {
        Plane[] planes =
GeometryUtility.CalculateFrustumPlanes(playerCam);

        if (GeometryUtility.TestPlanesAABB(planes,
this.gameObject.GetComponent<Renderer>().bounds))
        {
            stalker.speed = 0;
            stalker.SetDestination(transform.position);
        }

        if (!GeometryUtility.TestPlanesAABB(planes,
this.gameObject.GetComponent<Renderer>().bounds))
        {
            stalker.speed = aiSpeed;
            destination = player.position;
            stalker.destination = destination;
        }
    }
}
```

Bugs & Challenges:

- **Character Controller**
    - The Camera zoom only works in the movement controller test scene at the moment. The reason behind this is that it was originally written with the standard camera component in mind, however, the camera component in the main scene was eventually replaced with a Cinemachine virtual camera setup for the handheld camera shake feature and to provide the option to create scripted cutscenes in the future. On paper, fixing this issue simply requires replacing the script reference to the standard camera with a reference to the Cinemachine camera, however I originally did not have any luck in finding the proper way to achieve this.
    - Due to the rigidbody that was added to the character for the handling of fall damage, the player can now sometimes fall through the map if walking into a vertex gap or corner while repeatedly crouching and standing up. Surprisingly the fall damage system can sometimes manage to kill the player when this happens, which somewhat fixes the problem. As a more robust countermeasure, I have encased the levels in a box collider and created a simple script that uses OnTriggerExit to call the level manager and restart the scene in the event that the player falls out of bounds.
    - When approaching the exit door at the end of the underground tunnel in scene 1, it was found that running through the tunnel can apply fall damage and actually kill the player. I have no idea why this happens and this is the only spot where I have seen this happen consistently. There may be a correlation between this and getting stuck on level geometry. Fixing this will require further investigation of the fall damage code and more extensive testing.
- **Environments**
    - At some point during the mapping process some rooms were accidentally cloned leading to duplicate vertices and faces being layered on top of each other. This wasn't noticed until well into the texturing process, during which these duplicate faces were deleted without full awareness of the issue. Upon investigation, it was found that deleting one of the duplicates and repairing the butchered geometry of the original room would be a needlessly time consuming process. So it was decided that the best course of action within the scope of this term would be to leave the duplicate rooms as is, since the issue has no impact on gameplay.
    - Certain pools in scenes 2 and 3 currently play the wrong footstep sound effects. Originally thought to be a quick fix using water triggers, it was found that properly fixing this would require more extensive modification of the character controller script so that footstep surface states can be modified and set from other scripts.

    - Some areas of the three levels aren't scaled perfectly and will sometimes cause friction when entering doorways. In less intrusive cases, scaling issues mean that some areas are just slightly bigger than originally intended. A band aid fix for the doorway bug was simply to make the character collider skinnier and shorter although this was done sparingly as not to cause further scaling discrepancies.
    - Certain objects and areas may have slight gaps between vertices or noticeable z-fighting. Most of the z-fighting issues and texture misalignments were unavoidable due to the shortcomings of using Probuilder for all of my environmental modeling.
    - Doors will only open from one side after a botched attempt to make a door that only opens in one direction. Furthermore, doors will sometimes open in a direction that the door model wasn't intended to, causing some clipping into the door frame and or wall.
- **AI:**
    - Due to the more simple behavior of the waterman AI script, the player can be detected from any direction so long as they are close enough to the enemy. A long term plan is to entirely write this character's AI to feature a more complex vision cone that actually follows the character model's head.
    - When reaching a destination point the waterman will sometimes continue walking in place before going into his idle animation. Attempts to fix this were unsuccessful and will likely require a more extensive rewrite of the AI state system.
    - Footstep sounds will continue to play during the AI's jumpscare/attack animation
    - The AI seems to be very inconsistent with entering the search state. To be honest, the tutorial which I followed for this script was kind of lackluster but was closest to what I wanted, so I may just use this as the basis to design a new AI system from the ground up in the future.
    - AI animations lack proper blending and don't look very smooth when turning around or switching animation states.
    - Mannequin AI will sometimes move when just barely in the frame of the camera.
    - Mannequin AI doesn't properly display the static/death screen but plays the audio for it. This is likely due to an issue with the timing and the UI layer visibility on the jumpscare camera.
    - AI enemies will continue moving during the jumpscare animation/sequence.
    - Certain animation will make AI models float off the ground slightly. This was fixed for the waterman by simply changing the AI agent offset, but attempting to do this for the mannequin seems to prevent the footstep sounds from playing. I suspect that this has to do with the surface detection raycast being out of place after the offset is changed.
    - Mannequin AI remains in the walking pose during jumpscare sequence.

- **UI:**
    - Dropdown menus in the settings menu have a scaling error that causes all of the options in the dropdown to overlap each other. Seems like a simple fix, but I was unable to get this to scale properly in the time that I spent messing with it.
- **Audio:**
    - When the flashlight flicker sound effect plays, it will cause the background ambience to briefly fade out until the flicker sound has ended.
    - There is a chance that some sound effects weren't added to the audio mixer group and will lack the applied audio effects and be unaffected by the settings menu volume slider. Not entirely sure this is even an issue but it is worth reporting in case one slipped through the cracks for this submission.
    - Sometimes footsteps will only play briefly then stop working entirely. I have no idea what causes this. Currently, this is an issue in scene 3 after changing nothing.

Required Module Progress

- **Assets:**
    - **Needed:**
        - UI Icons and Indicators
    - **Completed:**
        - Environmental Textures and Materials
        - Asylum Props
        - Wall Signage and misc decals
        - Locker Room 3D Models
        - Pool Railing 3D Models
        - Pool Ladder 3D Models
        - UI Text and Fonts
        - Camera Filter and Effects
- **Environments:**
    - **Completed:**
        - AI Testing Ground Scene (Used to test enemy chase and patrol AI)
        - AI Testing Ground 2 Scene (Used to "Weeping Angel" style AI Enemies)
        - FPS Character Controller Test Level (Used to test custom character controller features and imported assets)
        - Main Menu Screen Environment (110% Done)
        - Chapter 1- Asylum Environment (90% Done)
        - Chapter 2 - Poolrooms Environment (90% Done)
        - Chapter 3 - Dark Poolrooms Environment (90% Done)
- **Characters:**
    - **Completed:**
        - Character Animations
        - Placeholder Models for Test AI

        - Character Models for Monsters
- **User Interface:**
    - **Needed:**
        - Camera Battery Tracker
        - Camera Zoom Indicator
        - Flashlight Indicator Icon
    - **Completed:**
        - HUD Text (No Functionality)
        - Main Menu Interface
        - Pause Menu Interface
        - Settings Menu Interface
        - Scripted Functionality for HUD Text
        - Loading Screen
- **Sounds:**
    - **Needed:**
        - Player Breathing/Running Sounds
        - Monster Sounds
    - **Completed:**
        - Footstep Sounds
        - Water Sounds
        - Ambience Tracks
        - Flashlight Toggle Sound Effect
        - Flashlight Flicker Sound Effect
        - Door Sounds
        - Light Bulb Buzz Sounds
- **Scripts:**
    - **Needed:**
        - Sound Effect Trigger Script
        - Door Trigger Script (Door opens when player enters trigger)
        - Lighting Trigger Script (Lighting toggles when player enters trigger)
        - Toggle Camera On/Off Script (Optional)
        - Oxygen Timer Script (Optional)
    - **Completed:**
        - Flashlight + Battery Level Script
        - Movement Controller Script
        - Test AI Script
        - Waterman AI Script
        - Stalker/Ghost AI Script
        - Mannequin AI Script
        - Hiding Place Script (Prototype)

    - Level Transition Trigger Script
    - Door Scripts
    - Interactable Framework
    - Main Menu Script (90% Done)
    - Pause Menu Script (100% Done)
    - Settings Menu Script
    - Door Trigger Script
    - Level Manager Script
    - IDataService Script
    - JSONDataService Script

Overall Time Dedicated:

- Rate of Work: 3-5 hours each day, 5-7 days a week
- Total Estimated Hours: 200-250 hours
- Hours Spent on Level Design: 150+ Hours
- Hours Spent on UI Development: 25+ Hours
- Hours Spent on AI Development: 25+ Hours
- Hours Spent on Player/Interaction Scripting: 25+ Hours

References:

Assets

3D Everything. (2023, October 9). *Ceiling lamp: 3D interior*. Unity Asset Store. https://assetstore.unity.com/packages/3d/props/interior/ceiling-lamp-153376

ANDRE. (2021a, November 8). *Cloth ghost - download free 3D model by Andre (@andrey.sk)*. Sketchfab. https://sketchfab.com/3d-models/cloth-ghost-f4fc1cd448c04a809d106975cfa7011b

ATD-LondonFollow, A.-L. (2023, April 17). *Stairs & Railings - Download Free 3D model by ATD-London*. Sketchfab. https://sketchfab.com/3d-models/stairs-railings-a419edddce534968937313dcff8429e7

Audio, G. (2017, April 4). *Footstep and Foley Sounds: Foley Sound FX*. Unity Asset Store. https://assetstore.unity.com/packages/audio/sound-fx/foley/footstep-and-foley-sounds-85360

Bobor, L. (2020, July 17). *Door free pack AFERAR: 3D interior*. Unity Asset Store. https://assetstore.unity.com/packages/3d/props/interior/door-free-pack-aferar-148411

Cheese Animal Productions. (2023, January 25). *Door texture pack: 2d textures & materials*. Unity Asset Store. https://assetstore.unity.com/packages/2d/textures-materials/door-texture-pack-223425

ESsplashkid. (2018, February 15). *Old "USSR" LAMP: 3D Electronics*. Unity Asset Store. https://assetstore.unity.com/packages/3d/props/electronics/old-ussr-lamp-110400

Ferar, A. (2019, July 3). *Door free pack AFERAR: 3D interior*. Unity Asset Store. https://assetstore.unity.com/packages/3d/props/interior/door-free-pack-aferar-148411

Fidler, J. (2018, May 31). *Classic interior door pack 1: 3D interior*. Unity Asset Store. https://assetstore.unity.com/packages/3d/props/interior/classic-interior-door-pack-1-118744

Kobra Game Studios. (2014, November 18). *Bathroom props: 3D furniture*. Unity Asset Store. https://assetstore.unity.com/packages/3d/props/furniture/bathroom-props-25255

VIS Games. (2022, September 4). *Locker: 3D props*. Unity Asset Store. https://assetstore.unity.com/packages/3d/props/locker-56405

Kanistra Studio. (2015, December 29). *Modular railing set: 3D*. Unity Asset Store. https://assetstore.unity.com/packages/3d/modular-railing-set-17090

Mikalsen, R. (2022, February 11). *Blobb Man - Download free 3D model by robin.mikalsen*. Sketchfab. https://sketchfab.com/3d-models/blobb-man-c65bfc754c93428b9b9aecbddaa8275e

ShaxerTakkuY. (2023, March 29). *Experiment - twin peaks 3 - download free 3D model by Shaxertakkuy*. Sketchfab. https://sketchfab.com/3d-models/experiment-twin-peaks-3-a2935eb0f56b44849afe3477c0d0f3f2

Storm, V. (2023, January 5). *VHS pro: Fullscreen & camera effects*. Unity Asset Store. https://assetstore.unity.com/packages/vfx/shaders/fullscreen-camera-effects/vhs-pro-4495

Surpent. (2014, November 12). *Free steel ladder pack: 3D exterior*. Unity Asset Store. https://assetstore.unity.com/packages/3d/props/exterior/free-steel-ladder-pack-24892

tboiston. (2020, November 9). *Locker room bench*. Sketchfab. https://sketchfab.com/3d-models/locker-room-bench-14a60c19ae654f7aaf75902fb636309

Videos
https://www.youtube.com/playlist?list=PLCiMP35N5s8rXDOWyac56-NakDATjXZg4

Images
https://drive.google.com/drive/folders/1EpL_Kk5vPzuxkTi1kspSB4oJj05ZqHeP?usp=drive_link