

# **COMP 480: Final Project**

**Ghost Escape Room Game**

Adrian Cortez – April 30, 2025

## **Functional Specifications**

This game is a first-person horror puzzle escape room game. The concept of the game is a worker who overslept in the office, and finds themselves locked inside. To escape, the user must find different clues and tools they can use to escape the building, while also trying to escape a ghost. The user must utilize different objects round the map, and connect the dots in order to find a way out of the building. Throughout the game, the user can utilize computers, images on the wall, tools on the floor, and machines in the game. If the user can escape the building without running into the ghost, they win the game. If they get caught however, they lose. To the person grading this, it might take a while to figure the game out fully, so I added a list of instructions at the end.

## **Technical Specifications**

In order for this project to work, the following core scripts are needed: A shared data class, an animation script, an output script, a collect script, a ghost mover script, a player mover script. In our shared data class, we have two hash sets that contain the inventory of our player and the keys they have. Throughout the game, objects are added to these hash sets and used to determine if the player can use the tools in those hash sets on other objects. Within this class also contains a dictionary of all the clues in the game, which have the narration strings that correspond with those objects when the user interacts with them. Lastly, a function is created that will end the game when it is called. Any functions outside this class that are subscribed to this function will enact when invoked.

In our animation script, we play different animations utilizing the animator component of a game object. This includes using collider logic in order to determine if the player is looking at the game object that is to be animated. This is done by utilizing `onTriggerEnter` and `onTriggerExit` functions. In our output script, we instantiate prefab game objects that are custom made, and give a force to it in order to simulate the action of outputting something. This is done by getting the rigid body component of the prefab. In our collect script, the same collider logic is utilized to determine if the player is looking at the object or not. If they are, and they press the key E, the prefab is set inactive and a string is added to the players inventory to indicate that they have that item.

Next, the ghost mover script is a core script needed. This script utilizes Nav Mesh Agent, and creates a range of sight for the ghost and a range of attack. When the player enters one of these ranges, we can create three states: attacking, chasing, and patrolling. When the ghost is attacking, it ends the game by calling a function from the shared data class. If the ghost is chasing, the destination is repeatedly called toward the player. If it is patrolling, the ghost moves to preset waypoint objects throughout the map.

Lastly, there is a script that works for the player's camera movement. This includes the run speed, walk speed, jumping mechanics, and more. Moreover, there is a box collider that is attached to the player that extends from the "eye" of the player out. So, whenever the player is looking towards something, it will trigger code. There are more scripts involved, but these are the main ones.

## Images

---

```
2  using UnityEngine;
3  using System.Collections;
4
5  public class MapScript : MonoBehaviour
6  {
7      bool isPlayerNear = false;
8      public TextMeshProUGUI crosshair;
9      public string AnimationName;
10     public string interactStr;
11     GameObject hiddenObject;
12     bool interacted = false;
13
14     // Start is called once before the first execution of Update after the MonoBehaviour is created
15     void Start()
16     {
17         if (gameObject.name == "RecycleBin")
18         {
19             hiddenObject = GameObject.Find("Money");
20             hiddenObject.SetActive(false);
21         }
22     }
23
24     // Update is called once per frame
25     void Update()
26     {
27         if (isPlayerNear && Input.GetKeyDown(KeyCode.E))
28         {
29             GetComponent<Animator>().Play(AnimationName);
30             interacted = true;
31
32             if (gameObject.name == "RecycleBin"){
33                 hiddenObject.SetActive(true);
34             }
35         }
36     }
37
38     // Player looks toward door
39     private void OnTriggerEnter(Collider other)
40     {
41         if (other.CompareTag("Player") && !interacted)
42         {
43             isPlayerNear = true;
44             crosshair.gameObject.SetActive(true);
45             crosshair.text = "[E]\n" + interactStr;
46         }
47     }
48
49     // Player looks away
50     private void OnTriggerExit(Collider other)
51     {
52         if (other.CompareTag("Player"))
53         {
54             isPlayerNear = false;
55             crosshair.gameObject.SetActive(false);
56         }
57     }
58 }
```

```
1  using UnityEngine;
2  using TMPro;
3  using System.Collections;
4
5  public class CollectScript : MonoBehaviour
6  {
7      bool isPlayerNear = false;
8      TextMeshProUGUI crosshair;
9      AudioSource collectAudio;
10     string objectType; // change in inspector
11
12     // Start is called once before the first execution of Update after the MonoBehaviour is created
13     void Start()
14     {
15         collectAudio.mute = false;
16
17         crosshair = GameObject.Find("InteractText").GetComponent<TextMeshProUGUI>();
18     }
19
20     // Update is called once per frame
21     void Update()
22     {
23         if (isPlayerNear && Input.GetKeyDown(KeyCode.E))
24         {
25             SharedData.inventory.Add(objectType);
26             collectAudio.mute = false;
27             collectAudio.Play();
28
29             transform.position = new Vector3(9999, 9999, 9999);
30             StartCoroutine(DisableAfterSound());
31         }
32     }
33
34     // Player looks toward Key
35     private void OnTriggerEnter(Collider other)
36     {
37         if (other.CompareTag("Player"))
38         {
39             isPlayerNear = true;
40             crosshair.gameObject.SetActive(true);
41             crosshair.text = "[E]\nCollect";
42         }
43     }
44
45     // Player looks away
46     private void OnTriggerExit(Collider other)
47     {
48         if (other.CompareTag("Player"))
49         {
50             isPlayerNear = false;
51             crosshair.gameObject.SetActive(false);
52         }
53     }
54
55     private IEnumerator DisableAfterSound()
56     {
57         yield return new WaitForSeconds(collectAudio.clip.length);
58         gameObject.SetActive(false);
59     }
60
```

---

```
1  using System.Numerics;
2  using System.Runtime.CompilerServices;
3  using UnityEngine;
4  using UnityEngine.AI;
5  using System.Collections.Generic;
6  using System.Collections;
7  using TMPro;
8
9  public class GhostControl : MonoBehaviour
10 {
11     // Set agent, player, and layer mask
12     public NavMeshAgent agent;
13     public Transform player;
14     public LayerMask whatIsGround, whatIsPlayer;
15
16     // Patrol
17     public UnityEngine.Vector3 walkPoint;
18     public float walkPointRange;
19     public List<GameObject> waypoints = new List<GameObject>();
20     private int count = 0;
21
22     // Attacking
23     public float sightRange, attackRange;
24     public bool playerInSightRange, playerInAttackRange;
25
26     // Audio
27     public AudioSource endAudio;
28
29     private bool hasPlayed = false;
30     private bool hasAttacked = false;
31
32     private float chaseUpdateRate = 0.2f;
33     private float nextChaseUpdateTime = 0f;
34
35
36     void Start()
37     {
38         endAudio.mute = true;
39     }
34
40
41     void Update()
42     {
43         // Check for sight and attack range
44         playerInSightRange = Physics.CheckSphere(transform.position, sightRange, whatIsPlayer);
45         playerInAttackRange = Physics.CheckSphere(transform.position, attackRange, whatIsPlayer);
46
47         // This gives us 4 states to work with
48         if (SharedData.isHidden)
49         {
50             Patroling();
51             return; // this will skip the decision structures below
52         }
53
54         if (!playerInSightRange && !playerInAttackRange) {
```

```
55     Patroling();
56     Debug.Log("Patrolling");
57     hasPlayed = false;
58 }
59 if (playerInSightRange && !playerInAttackRange && SharedData.clearedFirstRoom) {
60     ChasePlayer();
61
62     Debug.Log("Chasing");
63 }
64 if (playerInAttackRange) {
65     hasPlayed = false;
66
67     if (!hasAttacked)
68     {
69         endAudio.mute = false;
70         endAudio.Play();
71
72         hasAttacked = true;
73     }
74
75     AttackPlayer();
76     agent.ResetPath();
77 }
78 }
79
0 references
80 private void Awake()
81 {
82     player = GameObject.Find("PlayerBody").transform;
83     agent = GetComponent<NavMeshAgent>();
84 }
85
2 references
86 private void Patroling()
87 {
88     // Check if the agent has finished its current path and reached the target
89     if (!agent.pathPending && agent.remainingDistance < 0.5f)
90     {
91         // Attempt to set the next waypoint as the target
92         if (CanReachWaypoint(waypoints[count].transform.position))
93         {
94             // If the waypoint is reachable, move to the next one
95             count = (count + 1) % waypoints.Count;
96             agent.SetDestination(waypoints[count].transform.position);
97             //Debug.Log("Going to " + waypoints[count].name + " which is at " + count);
98         }
99         else
100         {
101             // If the waypoint is not reachable, skip it and go to the next
102             count = (count + 1) % waypoints.Count;
103             agent.SetDestination(waypoints[count].transform.position);
104             //Debug.Log("Waypoint is unreachable, moving to the next one: " + waypoints[count].name);
105         }
106     }
107
108     // Ensure the agent keeps heading to the current target
109     if (!agent.hasPath)
110     {
111         agent.SetDestination(waypoints[count].transform.position);
112         //Debug.Log("Going to " + waypoints[count].name + " which is at " + count);
113     }
114 }
115
116 // Check if the agent can find a path to the given position
1 reference
117 private bool CanReachWaypoint(UnityEngine.Vector3 targetPosition)
```

```
118     {
119         NavMeshPath path = new NavMeshPath();
120         // Calculate a path to the target
121         agent.CalculatePath(targetPosition, path);
122         // If the path has been calculated successfully and is not empty, the waypoint is reachable
123         return path.status == NavMeshPathStatus.PathComplete;
124     }
125
126     1 reference
127     private void ChasePlayer()
128     {
129         if (Time.time >= nextChaseUpdateTime)
130         {
131             // Adjust the destination slightly before the player to avoid walking over them
132             UnityEngine.Vector3 offsetPosition = player.position - transform.forward * 4f;
133
134             // Set the new destination with offset
135             agent.SetDestination(offsetPosition);
136             nextChaseUpdateTime = Time.time + chaseUpdateRate;
137         }
138     }
139
140     1 reference
141     private void AttackPlayer()
142     {
143         // Only rotate horizontally toward the player (no tipping)
144         UnityEngine.Vector3 targetPosition = new UnityEngine.Vector3(player.position.x, transform.position.y, player.position.z);
145         transform.LookAt(targetPosition);
146
147         // player looks at ghost
148         SharedData.SetGameOver(false);
149     }
150
151     // Debug
152     0 references
153     private void OnDrawGizmosSelected()
154     {
155         Gizmos.color = Color.red;
156         Gizmos.DrawWireSphere(transform.position, attackRange);
157         Gizmos.color = Color.yellow;
158         Gizmos.DrawWireSphere(transform.position, sightRange);
159     }
160 }
```

```
1  using UnityEngine;
2  using TMPro;
3  using System.Collections;
4
5  public class PrinterScript : MonoBehaviour
6  {
7      bool isPlayerNear = false;
8      TextMeshProUGUI crosshair;
9      TextMeshProUGUI narration;
10     GameObject outputObject;
11     AudioSource outputSound;
12     bool isOutputted = false;
13     string trigger;
14     string interactText;
15
16     // Start is called once before the first execution of Update after the MonoBehaviour is created
17     void Start()
18     {
19         outputSound.mute = true;
20     }
21
22
23     // Update is called once per frame
24     void Update()
25     {
26         if (isPlayerNear && Input.GetKeyDown(KeyCode.E) && SharedData.inventory.Contains(trigger))
27         {
28             // Instantiate our output object and 'spit' it out
29             GameObject spawnedObject = Instantiate(outputObject, transform.position, transform.rotation);
30
31             if (spawnedObject != null)
32             {
33                 Debug.Log("Working");
34                 Rigidbody rb = spawnedObject.GetComponent<Rigidbody>();
35                 if (rb != null)
36                 {
37                     rb.AddForce(transform.forward * 1, ForceMode.Impulse);
38                 }
39
40                 outputSound.mute = false;
41                 outputSound.Play();
42             }
43
44             // play some narration
45             string message;
46             if (SharedData.objectMessages.TryGetValue(gameObject.name, out message))
47             {
48                 StartCoroutine(EnableTemporarily(message));
49             }
50             else
51             {
52                 Debug.Log("No Message found.");
53             }
54
55             // remove input from inventory
56             SharedData.inventory.Remove(trigger);
57
58             // disable option to click
59             isOutputted = true;
60         }
61     }
62 }
```

```
-- 60    }
61    else if (isPlayerNear && Input.GetKeyDown(KeyCode.E)) {
62        // play some narration
63        string message;
64        if (SharedData.objectMessages.TryGetValue(gameObject.name + "Not", out message))
65        {
66            StartCoroutine(EnableTemporarily(message));
67            Debug.Log(message);
68        }
69        else
70        {
71            Debug.Log("No Message found.");
72        }
73    }
74}
75
76 // Player looks toward object
77 // 0 references
77 private void OnTriggerEnter(Collider other)
78 {
79     if (other.CompareTag("Player") && !isOutputted)
80     {
81         isPlayerNear = true;
82         crosshair.gameObject.SetActive(true);
83         crosshair.text = "[E]\n" + interactText;
84     }
85 }
86
87 // Player looks away
87 // 0 references
88 private void OnTriggerExit(Collider other)
89 {
90     if (other.CompareTag("Player"))
91     {
92         isPlayerNear = false;
93         crosshair.gameObject.SetActive(false);
94     }
95 }
96
97 // Temporarily display narration
97 // 2 references
98 private IEnumerator EnableTemporarily(string text)
99 {
100     narration.gameObject.SetActive(true);
101     narration.text = text;
102     yield return new WaitForSeconds(5f);
103     narration.gameObject.SetActive(false);
104 }
105 }
```

---

---

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using TMPro;
4  using Unity.VisualScripting;
5  using UnityEngine;
6
7  [RequireComponent(typeof(CharacterController))]
8
9  public class PlayerController : MonoBehaviour
10 {
11     // Mouse look sensitivity and vertical rotation limit
12     public float speedLook = 2.0f;
13     public float lookXLimit = 45.0f;
14
15     // Speed for user movement
16     public float walkingSpeed = 8f;
17     public float runningSpeed = 12f;
18     public float jumpingSpeed = 2f;
19
20     // Gravity force applied to the user
21     public float gravity = 30.0f;
22
23     // Camera
24     public Camera playerCamera;
25
26     // Character controller component
27     CharacterController characterController;
28     Vector3 directionMove = Vector3.zero;
29     float xRotation = 0;
30
31     // Crosshair indicator
32     public TextMeshProUGUI crosshairText;
33     // Inventory UI
34     public TextMeshProUGUI inventory;
35
36     [HideInInspector]
37     public bool move = true; // used to stop or allow player movement
38
39     // Narration Text and Win Text
40     public TextMeshProUGUI narration;
41     public TextMeshProUGUI winTextObject;
42     // boolean to stop player from moving
43     private bool gameEnded = false;
44     // Get parent, child, and ghost objects
45     GameObject parentObject;
46     Transform firstChild;
47     public GameObject ghost;
48
49
50     void Start()
51 {
```

---

```
-- 52 characterController = GetComponent<CharacterController>();
53
54 // Lock cursor and make it visible
55 Cursor.lockState = CursorLockMode.Locked;
56 Cursor.visible = false;
57
58 // Disable crosshair text
59 crosshairText.gameObject.SetActive(false);
60
61 // Enable start narration
62 StartCoroutine(EnableTemporarily());
63
64 // Initialize win text
65 SharedData.winTextObject = winTextObject;
66
67 // Subscribe to end game function
68 SharedData.OnGameOver += EndGame;
69
70 // Get parent object (has the rigid body) and child object (has camera)
71 parentObject = gameObject.transform.parent.gameObject;
72 firstChild = transform.GetChild(0);
73
74
0 references
75 void Update()
76 {
77     if (!gameEnded) {
78         // Update inventory
79         string result = string.Join("\n", SharedData.inventory);
80         inventory.text = "Inventory and Known Clues:\n" + result;
81
82         Vector3 direction = Vector3.forward;
83         -----
84
85         // Determine movement directions based on player's facing direction
86         Vector3 forward = transform.TransformDirection(Vector3.forward);
87         Vector3 right = transform.TransformDirection(Vector3.right);
88
89         // Determine if the player is sprinting
90         bool isRunning = Input.GetKey(KeyCode.LeftShift);
91
92         // Calculate speed based on if the player is sprinting or walking
93         float curSpeedX = move ? (isRunning ? runningSpeed : walkingSpeed) * Input.GetAxis("Vertical") : 0;
94         float curSpeedY = move ? (isRunning ? runningSpeed : walkingSpeed) * Input.GetAxis("Horizontal") : 0;
95
96         // Keep Y-axis movement
97         float movementDirectionY = directionMove.y;
98
99         // Move direction, which is based on input
100        directionMove = (forward * curSpeedX) + (right * curSpeedY);
101
102        // Handle jump
103        if (Input.GetButton("Jump") && move && characterController.isGrounded)
104        {
105            directionMove.y = jumpingSpeed;
106        }
107        else
108        {
109            directionMove.y = movementDirectionY;
110        }
111
112        if (!characterController.isGrounded)
113        {
114            directionMove.y -= gravity * Time.deltaTime;
115        }
116    }
}
```

```
117     // // Handle mouse look if movement is allowed
118     characterController.Move(directionMove * Time.deltaTime);
119
120     // Player and Camera rotation
121     if (move)
122     {
123         xRotation += -Input.GetAxis("Mouse Y") * speedLook;
124         xRotation = Mathf.Clamp(xRotation, -lookXLimit, lookXLimit);
125         playerCamera.transform.localRotation = Quaternion.Euler(xRotation, 0, 0);
126         transform.rotation *= Quaternion.Euler(0, Input.GetAxis("Mouse X") * speedLook, 0);
127     }
128 }
129
130
131 // Narration text
1 reference
132 private IEnumerator EnableTemporarily()
133 {
134     narration.gameObject.SetActive(true);
135     narration.text = "Oh no. Looks like I overslept again. I better get out of here before boss shows up";
136     yield return new WaitForSeconds(5f);
137     narration.gameObject.SetActive(false);
138 }
139
1 reference
140 void EndGame()
141 {
142     // Stop the player from moving
143     gameEnded = true;
144
145     // Stop player's rigid body
146     Rigidbody playerRb = parentObject.GetComponent< Rigidbody >();
147     if (playerRb != null)
148     {
149         playerRb.linearVelocity = Vector3.zero; // stop all motion
150         playerRb.angularVelocity = Vector3.zero; // stop spinning
151         playerRb.isKinematic = true; // turn off physics
152     }
153
154     firstChild.LookAt(ghost.transform);
155 }
156
157 }
```

```
1  using UnityEngine;
2  using System.Collections.Generic;
3  using TMPro;
4
5  public static class SharedData
6  {
7      public static bool isHidden = false;
8      public static bool clearedFirstRoom = false;
9      public static bool viewedPostcard = false;
10     public static bool compCorrect = false;
11     public static bool newCompCorrect = false;
12     public static bool wifiUp = false;
13     public static HashSet<string> keys = new HashSet<string>();
14     public static HashSet<string> inventory = new HashSet<string>();
15     public static TextMeshProUGUI winTextObject;
16
17     // set win conditions
18     public static bool gameEnded = false;
19     public static bool win = false;
20
21     // create an event that will notify the game objects that the game is over
22     public delegate void GameOver();
23     public static event GameOver OnGameOver;
24
25
26
27     public static bool WifiUp {
28         get { return wifiUp; }
29         set { wifiUp = value; }
30     }
31
32     public static bool IsHidden {
33         get { return isHidden; }
34         set { isHidden = value; }
35     }
36
37     public static TextMeshProUGUI WinTextObject
38     {
39         get { return winTextObject; }
40         set { winTextObject = value; }
41     }
42
43
44     public static bool ClearedFirstRoom {
45         get { return clearedFirstRoom; }
46         set { clearedFirstRoom = value; }
47     }
48
49     public static bool CompCorrect {
50         get { return compCorrect; }
51         set { compCorrect = value; }
52     }
```

```

-- 53
      |
      | 0 references
54  public static bool NewCompCorrect {
55    get { return newCompCorrect; }
56    set { newCompCorrect = value; }
57  }
58
      | 0 references
59  public static bool ViewedPostcard {
60    get { return viewedPostcard; }
61    set { viewedPostcard = value; }
62  }
63
64 // Messages tied to specific doors
65 // 1 reference
66 public static Dictionary<string, string> doorMessages = new Dictionary<string, string>()
67 {
68    { "OfficeDoor", "Huh. Seems like the janitors locked the doors. I swear there's a spare key somewhere..." },
69    { "StorageDoor", "The custodians probably have a key for this one." },
70    { "BossDoor", "If boss saw me trying to get in... I'd lose my job. No, I'd lose my life." },
71    { "FileDoor", "I've always wanted to see the company archives." }
72 };
73
74 // Messages tied to specific objects
75 // 8 references
76 public static Dictionary<string, string> objectMessages = new Dictionary<string, string>()
77 {
78    { "Postcard", "Lee did say he was taking vacation in New York." },
79    { "Books", "Good thing ghosts aren't real." },
80    { "Flashlight", "Looks like someone left a flashlight." },
81    { "Clock", "Oh boy, it's almost midnight. I better get going. I can't get fired in this economy." },
82    { "Note", "I have great coworkers." },
83    { "Printer", "That's an odd spot to hide a key." },
84    { "PrinterNot", "I have to turn on the printer to use it. Maybe if I can login to a computer I can turn it on." },
85    { "RecycleFlyer", "Reduce, Reuse, Recycle." },
86    { "Vending", "This will be useful if I need patch something. Or if I have bad breath." },
87    { "VendingNot", "Looks like I forgot to bring my wallet." },
88    { "OfficeKey", "I can finally get out of here!" },
89    { "BossKey", "If boss catches me with this... Hello Walmart job." },
90    { "VentPivotNot", "If I can find a tool of some sort, I can definitely get through this vent." },
91    { "VentPivot", "Hopefully something useful is in here." },
92    { "Gum", "Hopefully the Wifi can start working. IT can never fix something themselves." },
93    { "GumNot", "Looks like I need to patch this up with something in order to get the Wifi up." }
94 };
95
96 // Static method to end the game
97 // 2 references
98 public static void SetGameOver(bool isWin)
99 {
100    gameEnded = true;
101
102    winTextObject.gameObject.SetActive(true);
103    winTextObject.text = isWin ? "You Win!" : "You Lose!";
104    win = isWin ? true : false;
105
106    if (OnGameOver != null)
107    {
108        OnGameOver.Invoke(); // invoke all functions subscribed to this
109    }
110}

```

**Instructions to beat the game:**

First Room: Office

1. Look at postcard on ground in the starting area (offices)
  - a. Indicates New York
2. Look at the map, where New York is pinned
3. Remove the pin, which reveals a clue
4. The clue corresponds to the poster on the wall opposite to it, gives a clue to the password of the computer
  - a. Indicates the word “luck”
5. Type that into the computer, and upon logging in, the printer becomes enabled
6. Interact with the printer to spit out the key to get out of the starting office

Second Rooms: Surveillance Room, Storage Room, Boss Room, File Room

1. Go to surveillance room
2. Find Boss key
3. Go to storage room and collect wrench
4. Use boss key to get into CEO’s office
5. Move poster in CEO’s office
6. Use Wrench to open vent into Storage Room
7. Collect File keys from Storage Room
8. Go to File Room, find file that indicates that the password to the desk computer is “comp 480”
9. Sign into desk computer, indicates no internet connection

Third Rooms: Conference Room, Server Room, Lobby

1. Go into conference room
2. Inspect Trashcan, find money
3. Use money on vending machine, receive gum
4. Use gum to patch broken wire in the server room
5. Wi-fi now works, so go back to desk computer
6. Desk computer indicates that the front door is now open
7. Escape through front door