

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 a two hash sets that contains 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 corresponds 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  0 references
6  public class MapScript : MonoBehaviour
7  {
8      3 references
9      bool isPlayerNear = false;
10     3 references
11     public TextMeshProUGUI crosshair;
12     1 reference
13     public string AnimationName;
14     1 reference
15     public string interactStr;
16     3 references
17     GameObject hiddenObject;
18     2 references
19     bool interacted = false;
20
21     // Start is called once before the first execution of Update after the MonoBehaviour is created
22     0 references
23     void Start()
24     {
25         if (gameObject.name == "RecycleBin")
26         {
27             hiddenObject = GameObject.Find("Money");
28             hiddenObject.SetActive(false);
29         }
30     }
31
32     // Update is called once per frame
33     0 references
34     void Update()
35     {
36         if (isPlayerNear && Input.GetKeyDown(KeyCode.E))
37         {
38             GetComponent<Animator>().Play(AnimationName);
39             interacted = true;
40
41             if (gameObject.name == "RecycleBin"){
42                 hiddenObject.SetActive(true);
43             }
44         }
45     }
46
47     // Player looks toward door
48     0 references
49     private void OnTriggerEnter(Collider other)
50     {
51         if (other.CompareTag("Player") && !interacted)
52         {
53             isPlayerNear = true;
54             crosshair.gameObject.SetActive(true);
55             crosshair.text = "[E]\n" + interactStr;
56         }
57     }
58
59     // Player looks away
60     0 references
61     private void OnTriggerExit(Collider other)
62     {
63         if (other.CompareTag("Player"))
64         {
65             isPlayerNear = false;
66             crosshair.gameObject.SetActive(false);
67         }
68     }
69 }
```

```

1  using UnityEngine;
2  using TMPro;
3  using System.Collections;
4
5  0 references
6  public class CollectScript : MonoBehaviour
7  {
8      3 references
9      bool isPlayerNear = false;
10     4 references
11     TextMeshProUGUI crosshair;
12     4 references
13     public AudioSource collectAudio;
14     1 reference
15     public string objectType; // change in inspector
16
17     // Start is called once before the first execution of Update after the MonoBehaviour is created
18     0 references
19     void Start()
20     {
21         collectAudio.mute = false;
22
23         crosshair = GameObject.Find("InteractText").GetComponent<TextMeshProUGUI>();
24     }
25
26     // Update is called once per frame
27     0 references
28     void Update()
29     {
30         if (isPlayerNear && Input.GetKeyDown(KeyCode.E))
31         {
32             SharedData.inventory.Add(objectType);
33             collectAudio.mute = false;
34             collectAudio.Play();
35
36             transform.position = new Vector3(9999, 9999, 9999);
37             StartCoroutine(DisableAfterSound());
38         }
39     }
40
41     // Player looks toward Key
42     0 references
43     private void OnTriggerEnter(Collider other)
44     {
45         if (other.CompareTag("Player"))
46         {
47             isPlayerNear = true;
48             crosshair.gameObject.SetActive(true);
49             crosshair.text = "[E]\nCollect";
50         }
51     }
52
53     // Player looks away
54     0 references
55     private void OnTriggerExit(Collider other)
56     {
57         if (other.CompareTag("Player"))
58         {
59             isPlayerNear = false;
60             crosshair.gameObject.SetActive(false);
61         }
62     }
63
64     1 reference
65     private IEnumerator DisableAfterSound()
66     {
67         yield return new WaitForSeconds(collectAudio.clip.length);
68         gameObject.SetActive(false);
69     }
70
71

```

```

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  0 references
10 public class GhostControl : MonoBehaviour
11 {
12     // Set agent, player, and layer mask
13     10 references
14     public NavMeshAgent agent;
15     4 references
16     public Transform player;
17     0 references | 2 references
18     public LayerMask whatIsGround, whatIsPlayer;
19
20     // Patrol
21     0 references
22     public UnityEngine.Vector3 walkPoint;
23     0 references
24     public float walkPointRange;
25     6 references
26     public List<GameObject> waypoints = new List<GameObject>();
27     8 references
28     private int count = 0;
29
30     // Attacking
31     2 references | 2 references
32     public float sightRange, attackRange;
33     3 references | 4 references
34     public bool playerInSightRange, playerInAttackRange;
35
36     // Audio
37     3 references
38     public AudioSource endAudio;
39
40     2 references
41     private bool hasPlayed = false;
42     2 references
43     private bool hasAttacked = false;
44
45     1 reference
46     private float chaseUpdateRate = 0.2f;
47     2 references
48     private float nextChaseUpdateTime = 0f;
49
50     0 references
51     void Start()
52     {
53         endAudio.mute = true;
54     }
55
56     0 references
57     void Update()
58     {
59         // Check for sight and attack range
60         playerInSightRange = Physics.CheckSphere(transform.position, sightRange, whatIsPlayer);
61         playerInAttackRange = Physics.CheckSphere(transform.position, attackRange, whatIsPlayer);
62
63         // This gives us 4 states to work with
64         if (SharedData.isHidden)
65         {
66             Patrolling();
67             return; // this will skip the decision structures below
68         }
69
70         if (!playerInSightRange && !playerInAttackRange) {

```

```

55     Patrolling();
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
80 0 references
81 private void Awake()
82 {
83     player = GameObject.Find("PlayerBody").transform;
84     agent = GetComponent<NavMeshAgent>();
85 }
86
87 2 references
88 private void Patrolling()
89 {
90     // Check if the agent has finished its current path and reached the target
91     if (!agent.pathPending && agent.remainingDistance < 0.5f)
92     {
93         // Attempt to set the next waypoint as the target
94         if (CanReachWaypoint(waypoints[count].transform.position))
95         {
96             // If the waypoint is reachable, move to the next one
97             count = (count + 1) % waypoints.Count;
98             agent.SetDestination(waypoints[count].transform.position);
99             //Debug.Log("Going to " + waypoints[count].name + " which is at " + count);
100         }
101         else
102         {
103             // If the waypoint is not reachable, skip it and go to the next
104             count = (count + 1) % waypoints.Count;
105             agent.SetDestination(waypoints[count].transform.position);
106             //Debug.Log("Waypoint is unreachable, moving to the next one: " + waypoints[count].name);
107         }
108     }
109
110     // Ensure the agent keeps heading to the current target
111     if (!agent.hasPath)
112     {
113         agent.SetDestination(waypoints[count].transform.position);
114         //Debug.Log("Going to " + waypoints[count].name + " which is at " + count);
115     }
116 }
117
118 // Check if the agent can find a path to the given position
119 1 reference
120 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     1 reference
140     private void AttackPlayer()
141     {
142         // Only rotate horizontally toward the player (no tipping)
143         UnityEngine.Vector3 targetPosition = new UnityEngine.Vector3(player.position.x, transform.position.y, player.position.z);
144         transform.LookAt(targetPosition);
145
146         // player looks at ghost
147         SharedData.SetGameOver(false);
148     }
149
150     // Debug
151     0 references
152     private void OnDrawGizmosSelected()
153     {
154         Gizmos.color = Color.red;
155         Gizmos.DrawWireSphere(transform.position, attackRange);
156         Gizmos.color = Color.yellow;
157         Gizmos.DrawWireSphere(transform.position, sightRange);
158     }

```

```

1  using UnityEngine;
2  using TMPro;
3  using System.Collections;
4
5  0 references
6  public class PrinterScript : MonoBehaviour
7  {
8      4 references
9      | bool isPlayerNear = false;
10     | 3 references
11     | public TextMeshProUGUI crosshair;
12     | 3 references
13     | public TextMeshProUGUI narration;
14     | 1 reference
15     | public GameObject outputObject;
16     | 3 references
17     | public AudioSource outputSound;
18     | 2 references
19     | bool isOutputted = false;
20     | 2 references
21     | public string trigger;
22     | 1 reference
23     | public string interactText;
24
25     // Start is called once before the first execution of Update after the MonoBehaviour is created
26     0 references
27     void Start()
28     {
29         | outputSound.mute = true;
30     }
31
32     // Update is called once per frame
33     0 references
34     void Update()
35     {
36         | if (isPlayerNear && Input.GetKeyDown(KeyCode.E) && SharedData.inventory.Contains(trigger))
37         | {
38             | // Instantiate our output object and 'spit' it out
39             | GameObject spawnedObject = Instantiate(outputObject, transform.position, transform.rotation);
40
41             | if (spawnedObject != null)
42             | {
43                 | Debug.Log("Working");
44                 | Rigidbody rb = spawnedObject.GetComponent<Rigidbody>();
45                 | if (rb != null)
46                 | {
47                     | rb.AddForce(transform.forward * 1, ForceMode.Impulse);
48                 | }
49
50                 | outputSound.mute = false;
51                 | outputSound.Play();
52             | }
53
54             | // play some narration
55             | string message;
56             | if (SharedData.objectMessages.TryGetValue(gameObject.name, out message))
57             | {
58                 | StartCoroutine(EnableTemporarily(message));
59             | }
60             | else
61             | {
62                 | Debug.Log("No Message found.");
63             | }
64
65             | // remove input from inventory
66             | SharedData.inventory.Remove(trigger);
67
68             | // disable option to click
69             | isOutputted = true;
70         | }
71     }

```

```

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
78 private void OnTriggerEnter(Collider other)
79 {
80     if (other.CompareTag("Player") && !isOutputted)
81     {
82         isPlayerNear = true;
83         crosshair.gameObject.SetActive(true);
84         crosshair.text = "[E]\n" + interactText;
85     }
86 }
87
88 // Player looks away
89 0 references
90 private void OnTriggerExit(Collider other)
91 {
92     if (other.CompareTag("Player"))
93     {
94         isPlayerNear = false;
95         crosshair.gameObject.SetActive(false);
96     }
97 }
98
99 // Temporarily display narration
100 2 references
101 private IEnumerator EnableTemporarily(string text)
102 {
103     narration.gameObject.SetActive(true);
104     narration.text = text;
105     yield return new WaitForSeconds(5f);
106     narration.gameObject.SetActive(false);
107 }

```

```

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  0 references
10 public class PlayerController: MonoBehaviour
11 {
12     // Mouse look sensitivity and vertical rotation limit
13     2 references
14     public float speedLook = 2.0f;
15     2 references
16     public float lookXLimit = 45.0f;
17
18     // Speed for user movement
19     2 references
20     public float walkingSpeed = 8f;
21     2 references
22     public float runningSpeed = 12f;
23     1 reference
24     public float jumpingSpeed = 2f;
25
26     // Gravity force applied to the user
27     1 reference
28     public float gravity = 30.0f;
29
30     // Camera
31     1 reference
32     public Camera playerCamera;
33
34     // Character controller component
35     4 references
36     CharacterController characterController;
37     6 references
38     Vector3 directionMove = Vector3.zero;
39     4 references
40     float xRotation = 0;
41
42     // Crosshair indicator
43     1 reference
44     public TextMeshProUGUI crosshairText;
45     // Inventory UI
46     1 reference
47     public TextMeshProUGUI inventory;
48
49     [HideInInspector]
50     4 references
51     public bool move = true; // used to stop or allow player movement
52
53     // Narration Text and Win Text
54     3 references
55     public TextMeshProUGUI narration;
56     1 reference
57     public TextMeshProUGUI winTextObject;
58     // boolean to stop player from moving
59     2 references
60     private bool gameEnded = false;
61     // Get parent, child, and ghost objects
62     2 references
63     GameObject parentObject;
64     2 references
65     Transform firstChild;
66     1 reference
67     public GameObject ghost;
68
69     0 references
70     void Start()
71     {

```

```

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         // Determine movement directions based on player's facing direction
85         Vector3 forward = transform.TransformDirection(Vector3.forward);
86         Vector3 right = transform.TransformDirection(Vector3.right);
87
88         // Determine if the player is sprinting
89         bool isRunning = Input.GetKey(KeyCode.LeftShift);
90
91         // Calculate speed based on if the player is sprinting or walking
92         float curSpeedX = move ? (isRunning ? runningSpeed : walkingSpeed) * Input.GetAxis("Vertical") : 0;
93         float curSpeedY = move ? (isRunning ? runningSpeed : walkingSpeed) * Input.GetAxis("Horizontal") : 0;
94
95         // Keep Y-axis movement
96         float movementDirectionY = directionMove.y;
97
98         // Move direction, which is based on input
99         directionMove = (forward * curSpeedX) + (right * curSpeedY);
100
101         // Handle jump
102         if (Input.GetButton("Jump") && move && characterController.isGrounded)
103         {
104             directionMove.y = jumpingSpeed;
105         }
106         else
107         {
108             directionMove.y = movementDirectionY;
109         }
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
132 1 reference
133 private IEnumerator EnableTemporarily()
134 {
135     narration.gameObject.SetActive(true);
136     narration.text = "Oh no. Looks like I overslept again. I better get out of here before boss shows up";
137     yield return new WaitForSeconds(5f);
138     narration.gameObject.SetActive(false);
139 }
140
141 1 reference
142 void EndGame()
143 {
144     // Stop the player from moving
145     gameEnded = true;
146
147     // Stop player's rigid body
148     Rigidbody playerRb = parentObject.GetComponent<Rigidbody>();
149     if (playerRb != null)
150     {
151         playerRb.linearVelocity = Vector3.zero; // stop all motion
152         playerRb.angularVelocity = Vector3.zero; // stop spinning
153         playerRb.isKinematic = true; // turn off physics
154     }
155     firstChild.LookAt(ghost.transform);
156 }
157 }

```

```

1  using UnityEngine;
2  using System.Collections.Generic;
3  using TMPro;
4
5  40 references
6  public static class SharedData
7  {
8      5 references
9      public static bool isHidden = false;
10     5 references
11     public static bool clearedFirstRoom = false;
12     3 references
13     public static bool viewedPostcard = false;
14     4 references
15     public static bool compCorrect = false;
16     4 references
17     public static bool newCompCorrect = false;
18     4 references
19     public static bool wifiUp = false;
20     4 references
21     public static HashSet<string> keys = new HashSet<string>();
22     10 references
23     public static HashSet<string> inventory = new HashSet<string>();
24     5 references
25     public static TextMeshProUGUI winTextObject;
26
27     // set win conditions
28     1 reference
29     public static bool gameEnded = false;
30     1 reference
31     public static bool win = false;
32
33     // create an event that will notify the game objects that the game is over
34     1 reference
35     public delegate void GameOver();
36     3 references
37     public static event GameOver OnGameOver;
38
39     0 references
40     public static bool WifiUp {
41         get { return wifiUp; }
42         set { wifiUp = value; }
43     }
44
45     0 references
46     public static bool IsHidden {
47         get { return isHidden; }
48         set { isHidden = value; }
49     }
50
51     0 references
52     public static TextMeshProUGUI WinTextObject
53     {
54         get { return winTextObject; }
55         set { winTextObject = value; }
56     }
57
58     0 references
59     public static bool ClearedFirstRoom {
60         get { return clearedFirstRoom; }
61         set { clearedFirstRoom = value; }
62     }
63
64     0 references
65     public static bool CompCorrect {
66         get { return compCorrect; }
67         set { compCorrect = value; }
68     }

```

```

53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109

```

```

0 references
public static bool NewCompCorrect {
    get { return newCompCorrect; }
    set { newCompCorrect = value; }
}

0 references
public static bool ViewedPostcard {
    get { return viewedPostcard; }
    set { viewedPostcard = value; }
}

// Messages tied to specific doors
1 reference
public static Dictionary<string, string> doorMessages = new Dictionary<string, string>()
{
    { "OfficeDoor", "Huh. Seems like the janitors locked the doors. I swear there's a spare key somewhere..." },
    { "StorageDoor", "The custodians probably have a key for this one." },
    { "BossDoor", "If boss saw me trying to get in... I'd lose my job. No, I'd lose my life." },
    { "FileDoor", "I've always wanted to see the company archives." }
};

// Messages tied to specific objects
8 references
public static Dictionary<string, string> objectMessages = new Dictionary<string, string>()
{
    { "Postcard", "Lee did say he was taking vacation in New York." },
    { "Books", "Good thing ghosts aren't real." },
    { "Flashlight", "Looks like someone left a flashlight." },
    { "Clock", "Oh boy, it's almost midnight. I better get going. I can't get fired in this economy." },
    { "Note", "I have great coworkers." },
    { "Printer", "That's an odd spot to hide a key." },
    { "PrinterNot", "I have to turn on the printer to use it. Maybe if I can login to a computer I can turn it on." },
    { "RecycleFlyer", "Reduce, Reuse, Recycle." },
    { "Vending", "This will be useful if I need patch something. Or if I have bad breath." },
    { "VendingNot", "Looks like I forgot to bring my wallet." },
    { "OfficeKey", "I can finally get out of here!" },
    { "BossKey", "If boss catches me with this... Hello Walmart job." },
    { "VentPivotNot", "If I can find a tool of some sort, I can definitely get through this vent." },
    { "VentPivot", "Hopefully something useful is in here." },
    { "Gum", "Hopefully the Wifi can start working. IT can never fix something themselves." },
    { "GumNot", "Looks like I need to patch this up with something in order to get the Wifi up." }
};

// Static method to end the game
2 references
public static void SetGameOver(bool isWin)
{
    gameEnded = true;

    winTextObject.gameObject.SetActive(true);
    winTextObject.text = isWin ? "You Win!" : "You Lose!";
    win = isWin ? true : false;

    if (OnGameOver != null)
    {
        OnGameOver.Invoke(); // invoke all functions subscribed to this
    }
}

```

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