Group 13 - Dynamic VR Horror Game Coding Report

Prepared by

Asvin Ragunathan, Isaac Sanchez,

Jacob Mielczarek, and Sean Comben

at the

University of Illinois Chicago

April 2023

Table of Contents

List of Tables

I. Project Description

1. Project Overview

Dynamic VR Horror Game is a horror game that dynamically changes based on the player's responses during their playthrough. This is achieved by not relying on jumpscares, like many other games and movies of the genre, but by using the capabilities of technologies such as VR, microphones, and smart watches to create spine chilling playthrough experiences that are tailored to the user's fears and phobias which will be explained further in future slides. The game can be broken down into two parts: the exploration phase wherein the user explores a set of given environments while the game records their responses to the fear elements they encounter, and the survival phase where the game uses the data collected in the exploration phase to generate a custom environment designed to truly terrify the player. These two phases complete what the Dynamic VR Horror Game is about.

2. Project Domain

Version One of Dynamic VR Horror Game's main purpose is to implement a system that is able to track an individual's fear of different entities within the game. As the player progresses through each environment in the game, the end scene must show an accurate representation of what scared the player in the game. The environments must also contain some form of randomness to them to ensure replayability of the game.

3. Relationship to Other Documents

VR Dynamic Horror Game Report - Development Report By Aguilar, Chan, Egbesemirone, and Hashim.

Unity C# Documentation - Guide in creating scripts within Unity for the project

4. Naming Conventions and Definitions

4a. Definitions of Key Terms

Fear Factor - A percentage applied to each entity within the game that grades how scared the player was of that entity.

Phase One - The exploration phase, where the player interacts with different environments that learn about what scares the player the most.

Phase Two - The survival phase, where the player must face their fears and interact with the elements that scared them the most in the exploration phase.

4b. UML and Other Notation Used in This Document

This document generally follows the Version 2.0 OMG UML standard, as described by Fowler. UML documentation has been referenced in Section VIII, the Bibliography. Any exceptions are noted where used.

4c. Data Dictionary for Any Included Models

Scare Factor = Average HeartRate (Taken From SmartWatch) + Microphone Levels

Fear Factor = Individual Element's Scare Factor / Highest Element's Scare Factor

II. Project Deliverables

5. First Release

Date of first release: 2/24/23

In terms of the first release, group 13 had somewhat of a working game with each individual scene working for the most part with little to no bugs save for the fact that some of the transitions between the scenes were missing. Our scenes had doors added that allow the players to get from one scene to the next and the keys were also added throughout the scenes that were used to unlock the doors.

Pending: Intro, Outro, transition states between scenes

6. Second Release

Date of second release: 3/31/23

In the second release, group 13 made the game a smoother experience by improving the transitions between scenes as well as adding a beginning and an ending menu interface. In terms of intro, an intro dialog was added as well as a main menu. The player will be greeted with an introduction story followed by the choice to be able to select whether they would like to start or quit the game. The ending shows certain statistics that were recorded during the playthrough of the game.

Pending: N/A

7. Comparison with Original Project Design Document

In terms of consistency to the previous groups' VR Horror game design, the final prototype matches the original concept fairly well. The biggest difference is in the number of phases initially planned and the number of phases implemented. The original game idea was to have an exploration and survival phase, but the final prototype only contains an exploration phase and half of the survival phase (the fear factor component which calculates what the player is scared of the most, not the world generation portion).

However, the remainder of the prototype stays true to the original design. Every time a game begins, the environments, such as the monsters generated and the maze solutions, change. The game also includes an overarching story which assists in tying the different environments into a single narrative. The game ends once the player finds the door and key located in each scene. While they do this, the game records various player responses such as changes in heart rate, screams, and shaking, all of which combine to create the user's fear factor score for the fear elements they encounter.

Reference: CS440 Dynamic VR Horror Game, Spring 2020

III. Testing

8. Items to be Tested

I. Backrooms Scene:
   - A. Binary Space Partitioning
     1. Binary Space Partition Constructor - Test ID 1
     2. Binary Space Partition Initialization - Test ID 2
   - B. Room Nodes
     1. Room Node Constructor - Test ID 3
     2. Room Node Add Child - Test ID 4
     3. Room Node Remove Child - Test ID 5

II. Main Menu:
   - A. Play Button - Test ID 11
   - B. Options Button - Test ID 12
   - C. Exit Button - Test ID 13
   - D. Intro Scene - Test ID 14

III. Forest Scene (Generate Maze):
   - A. Generate Maze Edges Loaded - Test ID6
   - B. Generate Maze Nodes Listed - Test ID7
   - C. Generate Maze Mandatory Edges Loaded - Test ID8
   - D. Generate Maze Additional Edges Chosen - Test ID9
   - E. Generate Maze All Edge Loaded - Test ID10

IV. Movement of Objects:
   - A. Start of object movement - Test ID15
   - B. Checking for collision - Test ID16

9. Test Specifications

ID1 - Binary Space Partition (BSP) Constructor

Description: This test checks whether the BSP class instance is created properly.

Items covered by this test: Checks that the dimensions of the root room node match with the given dimensions and that the root node has no parent.

Environmental needs: This test requires a computer with the Unity Editor 2021.3.17f1 installed and the Testing Framework enabled.

Intercase Dependencies: NA

Test Procedures: Run the test using the Unity Testing Framework

Input Specification: Two, randomly generated integers which will specify the dimensions of the root room node.

Output Specifications: An BSP instance containing a root node with dimensions matching those specified in the input.

Pass/Fail Criteria: For the test to pass, a root node must exist for the new BSP, its dimensions should match those provided, and it must have no parent node.

<u>ID2 - Binary Space Partition (BSP) Initialization</u>

Description: This test ensures the division and generation of the room spaces matches its expected behavior.

Items covered by this test: This test ensures that the number of room spaces generated does not exceed the maximum number of allowable room spaces. It also checks that the dimensions of the final room spaces do not fall below the minimum width and length specified.

Environmental needs: This test requires a computer with the Unity Editor 2021.3.17f1 installed and the Testing Framework enabled.

Intercase Dependencies: Test ID1

Test Procedures: Run the test using the Unity Testing Framework.

Input Specification: Three, randomly generated integer values will be used to specify the maximum number of room spaces as well as the minimum width and length of a room space.

Output Specifications: The output will be a list of room spaces representing a binary tree of rooms nodes and the children room nodes it is split into.

Pass/Fail Criteria: For the test to pass, the number of leaf room nodes must be less than or equal to the provided maximum number of room nodes, and the dimensions of each room node must be greater than or equal to the provided minimum room node dimensions.

ID3 - Room Node Constructor

Description: This test ensures the creation of a root node object meets the expected behavior in any given case and with any given input.

Items covered by this test: This test checks that the boundaries of the room match the room corner coordinates it was given on construction, ensures the tree layer value matches the index, the nodes parent reference matches the one it is given, and that, in the case of a non-null parent reference, the parent adds the new node to its list of children.

Environmental needs: This test requires a computer with the Unity Editor 2021.3.17f1 installed and the Testing Framework enabled.

Intercase Dependencies: NA

Test Procedures: Run the test using the Unity Test Framework.

Input Specification: Two, vector coordinates used to define the lower left corner of the room and upper right corner of the room, one integer value to denote the room node index, and a reference to the room's parent.

Output Specifications: A new instance of a RoomNode object configured to match the given input specifications.

Pass/Fail Criteria: For the test to pass, the vector coordinates that define the boundaries of the room must align with the values in the coordinates it was given. The room node instance must also have the same index value and parent reference. If the parent reference is not null, that parent must also have the new room node in its list of child nodes.

ID4 - Room Node Add Child

Description: This test checks that child room nodes are added correctly.

Items covered by this test: This test checks that the room node's list of children contains the added child node when a child node is added.

Environmental needs: This test requires a computer with the Unity Editor 2021.3.17f1 installed and the Testing Framework enabled.

Intercase Dependencies: Test ID3

Test Procedures: Run the test using the Unity Testing Framework.

Input Specification: None

Output Specifications: None

Pass/Fail Criteria: For the test to pass, the number of children nodes must match the number of added children and that each of the children added must exist in the nodes list of children.

ID5 - Room Node Remove Child

Description: This test checks that child room nodes are removed correctly.

Items covered by this test: This test checks that the room node's list of children does not contain the removed child node when a child node is removed.

Environmental needs: This test requires a computer with the Unity Editor 2021.3.17f1 installed and the Testing Framework enabled.

Intercase Dependencies: Test ID3, Test ID4

Test Procedures: Run the test using the Unity Testing Framework.

Input Specification: None

Output Specifications: None

Pass/Fail Criteria: For the test to pass, the number of children nodes must match the number of added children minus the number of children nodes removed. Also, the list of children nodes cannot contain a reference to the removed child room node after removal.

ID6 - Generate Maze Edges Loaded

Description: This test checks that the GenerateMaze.init() function creates a list of edge objects with an entry for each of the edges in the forest scene.

Items covered by this test: GenerateMaze.cs, Connection.cs

Requirements addressed by this test: None

Environmental needs: Latest version of the Unity Testing Framework Package and Version 2021.3.17f1 of the Unity Editor

Intercase Dependencies: Test ID7 (ID 7 depends on this test's success)

Test Procedures: Run TestScript in the “Test Runner” tab in the Unity Editor, in EditMode

Input Specification: Object from the GenerateMaze class

Output Specifications: Message in Test Runner

Pass/Fail Criteria: The test is passed if the size of the list of edges in the GenerateMaze object is equal to the total number of edges in the scene before the game is started (in this case, 16).

ID7 - Generate Maze Nodes Listed

Description: This test checks that the GenerateMaze.init() function creates a list of node objects with an entry for each of the nodes in the forest scene.

Items covered by this test: GenerateMaze.cs, Connection.cs

Requirements addressed by this test: None

Environmental needs: Latest version of the Unity Testing Framework Package and Version 2021.3.17f1 of the Unity Editor

Intercase Dependencies: Test ID6

Test Procedures: Run TestScript in the “Test Runner” tab in the Unity Editor, in EditMode

Input Specification: Object from the GenerateMaze class

Output Specifications: Message in Test Runner

Pass/Fail Criteria: The test is passed if the size of the list of nodes in the GenerateMaze object is equal to the total number of nodes in the scene before the game is started (in this case, 12).

ID8 - Generate Maze Mandatory Edges Loaded

Description: This test checks that the GenerateMaze.GeneratePath() function generates enough edges to create a path in the forest scene that the player can use to move across the crevice in the scene.

Items covered by this test: GenerateMaze.cs

Requirements addressed by this test: None

Environmental needs: Version 2021.3.17f1 of the Unity Editor

Intercase Dependencies: ID10 (ID 10 depends on this test's success)

Test Procedures: Run game in Unity Editor. Read the first Debug.Log() message in the Console tab ("Mandatory edges created: #")

Input Specification: checkVal1 (an int that starts at zero, and is incremented every time a mandatory edge is activated)

Output Specifications: Debug.Log() message containing value of checkVal1 after all mandatory edges have been activated

Pass/Fail Criteria: This test is passed if checkVal1's value in the Delog.Log() message is equal to the minimum number of edges needed to move across the crevice (in this case, 5).

<u>ID9 - Generate Maze Additional Edges Chosen</u>

Description: This test checks that the GenerateMaze.GeneratePath() function does not iterate through the loop that generates additional edges an excessive amount of times.

Items covered by this test: GenerateMaze.cs

Requirements addressed by this test: None

Environmental needs: Version 2021.3.17f1 of the Unity Editor

Intercase Dependencies: ID10 (ID 10 depends on this test's success)

Test Procedures: Run game in Unity Editor. Read the Debug.Log() messages in the Console tab ("new edge selected: #"). Identify the last message in the Console tab that contains the above string, and read the final integer value at the end of that string

Input Specification: checkVal2 (an int that starts at zero before the first iteration of the loop that generates additional edges, and is incremented every time that loop is iterated through)

Output Specifications: Debug.Log() message containing value of checkVal2 after all additional edges have been activated

Pass/Fail Criteria: The test is passed if the final value of checkVal2 in the final Debug.Log() message with the string "new edge selected: #" is less than 100 for 9 out of every 10 times that the game runs.

<u>ID10 - Generate Maze All Edge Loaded</u>

Description: This test checks that the GenerateMaze.GeneratePath() function activates enough mandatory and additional edges to create a maze that the player must complete to move across the crevice in the forest scene.

Items covered by this test: GenerateMaze.cs

Requirements addressed by this test: None

Environmental needs: Version 2021.3.17f1 of the Unity Editor

Intercase Dependencies: Test ID8, Test ID9

Test Procedures: Run game in Unity Editor. Read the Debug.Log() message in the Console tab (“All edges created: #”)

Input Specification: checkVal1 (an int that starts at zero, and is incremented every time a mandatory edge is activated). checkVal2 (an int that starts at zero before the first iteration of the loop that generates additional edges, and is incremented every time that loop is iterated through)

Output Specifications: Debug.Log() message containing sum of checkVal1 and checkVal2 after all additional and mandatory edges have been activated

Pass/Fail Criteria: This test is passed if the sum of checkVal1 and checkVal2 is greater than or equal to the sum of the number of mandatory edges activated in test ID8 and the number of additional edges activated in test ID9 (in this case, checkVal1 + checkVal2 must be greater than or equal to 5 mandatory edges + 5 additional edges = 10 total edges)

ID11 - Play Button

Description: This test checks that upon clicking the play button in the main menu scene, the main menu switches to the intro scene.

Items covered by this test: mainMenu.cs

Requirements addressed by this test: None

Environmental needs: Version 2021.3.17f1 of the Unity Editor

Intercase Dependencies: None

Test Procedures: Run game in Unity EditorClick “Play” in the Main Menu

Input Specification: playing (A boolean that checks for the start game signal)

Output Specifications: Debug.Log() message saying the current scene is the intro scene.

Pass/Fail Criteria: This test is passed if the playing boolean is checked upon clicking the Play Button, and the scene switches to the intro scene with the Debug.Log() function outputting such result.

ID12 - Options Button

Description: This test checks that upon clicking the options button in the main menu scene, the main menu switches to the options settings.

Items covered by this test: mainMenu.cs

Requirements addressed by this test: None

Environmental needs: Version 2021.3.17f1 of the Unity Editor

Intercase Dependencies: None

Test Procedures: Run game in Unity Editor, then click “Options” in the Main Menu

Input Specification: None

Output Specifications: Debug.Log() message saying the current scene is the options settings scene.

Pass/Fail Criteria: This test is passed if the playing boolean is checked upon clicking the Options Button, and the scene switches to the options settings scene with the Debug.Log() function outputting such result.

ID13 - Exit Button

Description: This test checks that upon clicking the Exit button in the main menu scene, the application quits.

Items covered by this test: mainMenu.cs

Requirements addressed by this test: None

Environmental needs: Version 2021.3.17f1 of the Unity Editor

Intercase Dependencies: None

Test Procedures: Run game in Unity Editor. Click “Exit” in the Main Menu

Input Specification: None

Output Specifications: The application closes.

Pass/Fail Criteria: This test is passed if the playing boolean is checked upon clicking the Exit button, and the application closes.

ID14 - Intro Scene

Description: This test checks if the intro scene runs and upon ending, enters the starting scene.

Items covered by this test: mainMenu.cs

Requirements addressed by this test: None

Environmental needs: Version 2021.3.17f1 of the Unity Editor

Intercase Dependencies: None

Test Procedures: Run game in Unity Editor. Click “Play” in the Main Menu. Wait for Intro scene to finish

Input Specification: None

Output Specifications: Debug.Log() message saying the current scene is the starting scene.

Pass/Fail Criteria: This test is passed the scene switches to the starting scene with the Debug.Log() function outputting such result when the intro animation is complete.

ID15 - Object Movement

Description: This test checks if the objects are moving when the player enters the scene

Items covered by this test: EnemyMovementScript.cs

Requirements addressed by this test: None

Environmental needs: Version 2021.3.17f1 of the Unity Editor

Intercase Dependencies: None

Test Procedures: Run game in Unity Editor, then click “Play” in the Main Menu, then go through the scenes until the hospital scene where the monsters are located and see if the monsters move.

Input Specification: None

Output Specifications: None.

Pass/Fail Criteria: This test passes when the monsters are clearly moving when the player enters the scene.

ID16 - Checking for Collision

Description: This test checks if the monsters change direction when colliding with another object in the game, for example a wall.

Items covered by this test: EnemyMovementScript.cs

Requirements addressed by this test: None

Environmental needs: Version 2021.3.17f1 of the Unity Editor

Intercase Dependencies: None

Test Procedures: Run game in Unity Editor, then click “Play” in the Main Menu, then wait for Intro scene to finish

Input Specification: None

Output Specifications: None

Pass/Fail Criteria: This test passes when we see the monsters switching directions from the initial path after colliding with an object.

10. Test Results

ID1 - Binary Space Partition (BPS) Constructor

Date(s) of Execution: 4/11/2023

Staff conducting tests: Jacob Mielczarek

Expected Results: Upon construction, a root room node is created with the given dimensions.

Actual Results: A root room node was created with the given dimensions.

Test Status: Pass

ID2 - Binary Space Partition (BSP) Initialization

Date(s) of Execution: 4/11/2023

Staff conducting tests: Jacob Mielczarek

Expected Results: A tree of room nodes is generated and all room nodes have dimensions greater than or equal to the specified minimum dimensions.

Actual Results: A tree of room nodes was generated. All room nodes had dimensions greater than or equal to the specified minimum dimensions.

Test Status: Pass

ID3 - Room Node Constructor

Date(s) of Execution: 4/11/2023

Staff conducting tests: Jacob Mielczarek

Expected Results: The room node has the given dimensions, tree layer index, and parent reference. If the parent reference is not null, the parent has the new room node added to its list of children.

Actual Results: The created room node has the correct dimensions, tree layer index, and parent reference. In the case where the parent reference given is not null, the parent reference has the new node within its list of children.

Test Status: Pass

ID4 - Room Node Add Child

Date(s) of Execution: 4/11/2023

Staff conducting tests: Jacob Mielczarek

Expected Results: The room node has the added node within its list of child nodes after it is added.

Actual Results: The room node had the added node within its list of child nodes after it was added.

Test Status: Pass

ID5 - Room Node Remove Child

Date(s) of Execution: 4/11/2023

Staff conducting tests: Jacob Mielczarek

Expected Results: The room node no longer has the child node within its list of children after it is removed.

Actual Results: The room node no longer had the child node within its list of children after it was removed.

Test Status: Pass

ID6 - Generate Maze Edges Loaded

Date(s) of Execution: April 10th, 2023

Staff conducting tests: Isaac

Expected Results: 16

Actual Results: 16

Test Status: Pass

ID7 - Generate Maze Nodes Listed

Date(s) of Execution: April 10th, 2023

Staff conducting tests: Isaac

Expected Results: 12

Actual Results: 12

Test Status: Pass

ID8 - Generate Maze Mandatory Edges Loaded

Date(s) of Execution: April 10th, 2023

Staff conducting tests: Isaac

Expected Results: 5

Actual Results: 5

Test Status: Pass

ID9 - Generate Maze Additional Edges Chosen

Date(s) of Execution: April 12th, 2023

Staff conducting tests: Isaac

Expected Results: 10 integers less than or equal to 50

Actual Results: 11, 12, 9, 18, 9, 8, 8, 10, 6, 13

Test Status: Pass

ID10 - Generate Maze All Edge Loaded

Date(s) of Execution: April 10th, 2023

Staff conducting tests: Isaac

Expected Results: an integer greater than or equal to 10

Actual Results: 18

Test Status: Pass

ID11 - Play Button

Date(s) of Execution: 4/12/2023

Staff conducting tests: Asvin Ragunathan

Expected Results: Upon pressing the play button, the intro scene starts.

Actual Results: Upon pressing the play button, the intro scene starts.

Test Status: Pass

ID12 - Options Button

Date(s) of Execution: 4/12/2023

Staff conducting tests: Asvin Ragunathan

Expected Results: Upon pressing the options button, the option settings opens.

Actual Results: Upon pressing the options button, the option settings opens.

Test Status: Pass

ID13 - Exit Button

Date(s) of Execution: 4/12/2023

Staff conducting tests: Asvin Ragunathan

Expected Results: Upon pressing the exit button, the application closes

Actual Results: Upon pressing the options button, the application closes

Test Status: Pass

<u>ID14 - Intro Scene</u>

Date(s) of Execution: 4/12/2023

Staff conducting tests: Asvin Ragunathan

Expected Results: Intro scene ends and continues to the starting scene.

Actual Results: Intro scene ends and continues to the starting scene.

Test Status: Pass

<u>ID15 - Object Movement</u>

Date(s) of Execution: 4/12/2023

Staff conducting tests: Sean Comben

Expected Results: Monsters moving in the hospital scene.

Actual Results: Monsters moving in the hospital scene.

<u>ID16 - Checking for Collision</u>

Date(s) of Execution: 4/12/2023

Staff conducting tests: Sean Comben

Expected Results: Monsters changing direction when colliding with an object

Actual Results: Monsters changing direction when colliding with an object.

11. Regression Testing

- ID1 - Binary Space Partition (BSP) Constructor
- ID2 - Binary Space Partition (BSP) Initialization
- ID3 - Room Node Constructor
- ID4 - Room Node Add Child
- ID5 - Room Node Remove Child
- ID8 - Generate Maze Mandatory Edges Loaded
- ID9 - Generate Maze Additional Edges Chosen
- ID10 - Generate Maze All Edge Loaded

IV. Inspection

12. Items to be Inspected

I. Backrooms Scene
   A. Map Generation
II. Main Menu Scene
   A. Main Menu Scripting
III. Forest Scene
   A. Collision Detection
IV. Hospital Scene
   A. Entity Collision

13. Inspection Procedures

Group 13 used checklists based on Java checklists found in *Software Testing And Analysis: Process, Principles, and Techniques*, by Mauro Pezze and Michal Young. The checklist used here checks basic script formatting, class/function/parameter visibility, and requirements specific to each feature inspected.

Group 13 manually read through the most critical code files and worked through their logic. As they read, they noted the various classes and methods used, as well as their functionalities. They also noted what the code does well, any problems in the code, and potential improvements to the code.

Collision elements and interactions between objects within the scenes were checked, such as the key-door interactions, the collisions between entities within the Hospital scene, and collisions between platforms within maze generation in the Forest scene. Elements that collided together were recorded and their collision modifiers were checked to ensure that no odd collisions occurred within each of the scenes.

Enemy movement code files were checked and followed the logic as well as going through the Unity project to make sure that the code is working as intended. The code does indeed generate random movement in any one direction for the monsters and when colliding with an object turns away to go towards the opposite direction.

Inspection results were shared in a group meeting where any issues with the code and its functionality were presented and resolved with the help of the code's original authors. We had 5 meetings for both testing and inspection, with the final meeting being where we showed our results for inspections in-person and resolved some encountered issues.

Checklist for Inspection:

Script Format: Does the following script follow these rules?	Yes	No	Comment
- Header comment explaining the purpose of the code			
- In-line comments containing complex lines of code			
- Properly indented lines (for loop bodies)			
- Short functions (50 lines of code or less, with helper functions if necessary)			

Classes, Functions, and Parameters Format: Does the following script follow these rules?	Yes	No	Comment
- Internal components are hidden from the rest of the program when applicable			
- Getters/Setters used to access private variables when necessary			
- No unnecessary parameters			
- Public functions have a purpose for public viewing			

Custom Section (Dependent on code inspection)	Yes	No	Comment
- Custom Criteria #1			
- Custom Criteria #2			
- Custom Criteria #3			

14. Inspection Results

Backrooms Scene (Inspected by Jacob Mielczarek, Isaac Sanchez, and Sean Comben):

Script Format: Does the following script follow these rules?	Yes	No	Comment

- Header comment explaining the purpose of the code		N	
- In-line comments containing complex lines of code		N	
- Properly indented lines (for loop bodies)	Y		
- Short functions (50 lines of code or less, with helper functions if necessary)	Y		

Classes, Functions, and Parameters Format: Does the following script follow these rules?	Yes	No	Comment
- Internal components are hidden from the rest of the program when applicable	Y		
- Getters/Setters used to access private variables when necessary	Y		
- No unnecessary parameters		N	There are a handful of private member variables that exist but are never used.
- Public functions have a purpose for public viewing	Y		

Custom Section (Dependent on code inspection)	Yes	No	Comment
- The design and structure of the program is sound	Y		The author demonstrates a good use of design patterns and algorithms to accomplish their goal. The breakdown of work between different classes is well done.
- There are no errors in the code.		N	One error was spotted upon inspection of the logic. However, it was easily fixed.
- Could the code be improved?	Y		The inspectors noted a lack of documentation in the form of comments to explain the code. They recommend adding more comments to assist in

			readability.

<u>Forest Scene - Collision Inspection (Inspected by Asvin Ragunathan, Isaac Sanchez, and Sean Comben)</u>:

Script Format: Does the following script follow these rules?	Yes	No	Comment
- Header comment explaining the purpose of the code		N	
- In-line comments containing complex lines of code	Y		Could use more inline comments
- Properly indented lines (for loop bodies)	Y		
- Short functions (50 lines of code or less, with helper functions if necessary)	Y		

Classes, Functions, and Parameters Format: Does the following script follow these rules?	Yes	No	Comment
- Internal components are hidden from the rest of the program when applicable	Y		Edge and Node subclasses are hidden from other classes
- Getters/Setters used to access private variables when necessary	Y		No need for getters and setters (getters were added to the code for testing)
- No unnecessary parameters	Y		
- Public functions have a purpose for public viewing	Y		The “init” and “GeneratePath” functions are used by the SceneLogic script

Maze Form: Does the maze follow these rules?	Yes	No	Comment
- There’s always a path generated that the player can use to fully cross the chasm	Y		
- There’s always extra rock edges generated to purposely confuse the player while crossing the chasm	Y		

- Some unnecessary edges are always removed when the maze is generated	Y		

Main Menu (Inspected by Isaac Sanchez, Asvin Ragunathan, Jacob Mielczarek):

Script Format: Does the following script follow these rules?	Yes	No	Comment
- Header comment explaining the purpose of the code		N	
- In-line comments containing complex lines of code	Y		Code is straightforward, so no inline comments are necessary
- Properly indented lines (for loop bodies)	Y		
- Short functions (50 lines of code or less, with helper functions if necessary)	Y		

Classes, Functions, and Parameters Format: Does the following script follow these rules?	Yes	No	Comment
- Internal components are hidden from the rest of the program when applicable	Y		Since the functions are called by player, they are public
- Getters/Setters used to access private variables when necessary	Y		No need for getters and setters
- No unnecessary parameters	Y		
- Public functions have a purpose for public viewing	Y		

Menu Format: Does the menu follow these rules?	Yes	No	Comment
- Menu is clear, with visible text for buttons	Y		
- “Play” button in Menu takes player to the starting scene	Y		Resolved in previous meeting
- “Quit” button in Menu stops the application	Y		Resolved in previous meeting

Hospital Scene - Entity Collision **<u>(Inspected by Sean Comben, Asvin Ragunathan, Jacob Mielczarek):</u>**

Script Format: Does the following script follow these rules?	Yes	No	Comment
- Header comment explaining the purpose of the code	Y		
- In-line comments containing complex lines of code	Y		
- Properly indented lines (for loop bodies)	Y		
- Short functions (50 lines of code or less, with helper functions if necessary)	Y		

Classes, Functions, and Parameters Format: Does the following script follow these rules?	Yes	No	Comment
- Internal components are hidden from the rest of the program when applicable	Y		
- Getters/Setters used to access private variables when necessary	Y		
- No unnecessary parameters	Y		
- Public functions have a purpose for public viewing	Y		

Custom Section (Dependent on code inspection)	Yes	No	Comment
- Entities always collide against walls		N	Current bug in the code that sometimes allows entities to pass through some walls.
- Entities make sound upon collision	Y		
- Entities move to a random direction upon impact	Y		

V. Recommendations and Conclusions

The codes inspecting have passed and there are minimal/no problems indicated during the testing phase. This concludes that the next course of action would be presenting the project to the audience to show what has been done throughout the semester.

VI. Project Issues

15. Open Issues

In terms of bugs, there's been a glitch in the Backrooms scene in which some hallways do not lead to rooms, but rather to dead-ends. In addition, there is still the issue of some creatures in the Hospital scene leaving the hospital building instead of moving back-and-forth whenever they hit the hospital walls.

In terms of technology, we are unsure if the API that we use for the smart watch's heartbeat detection will be available in the future; if this API is removed by the owner, then our game will lose a major portion of its fear-recording mechanic.

Finally, while VR safety is vital to developers and publishers due to many users being vulnerable to headaches or stomach aches during VR experiences, laws may be enacted in the future that add additional restrictions or rules for VR games and/or accessories. In this case, strong documentation would be required so that future developers could edit the game so that it abides by new VR industry standards.

16. Waiting Room

A major mechanic that still needs to be added to the final release of the game is the rest of the survival phase. While we have implemented the exploration phase of the game in the prototype, as well as the fear factor mechanic and results screen that shows what the player was most scared of while playing, the full survival phase is required to elevate both the horror and the gameplay. The exploration phase, while having randomized environmental layouts and entities, may lack replayability when players become accustomed to the game loop of this phase. The survival phase adds immense amounts of depth to the game, due to its use of combat and potential for game-overs. Of course, given that this is a substantial portion of the game, with several new mechanics tied to it, it was unfeasible to fully implement this phase in our current release of the game. However, completing this phase should definitely be priority #1 in the next release.

As for future versions, the requirements from the original group's document that we still wish to implement/test are, in order of highest priority to lowest priority, ID9, ID10, ID11, ID17, ID15, ID19, ID16, ID18, ID35, ID2, ID36, ID6, ID26, ID25, ID24, ID29, ID21, ID22, ID33, ID5, ID27, ID4, ID32, ID31, and ID30.

Release 3 will contain ID9 (SafetyTest), ID10 (maintenanceTest), ID11 (supportabilityTest), ID17 (privacyTest), ID15 (AccessTest), ID19 (immunityTest), ID16 (integrityTest), ID18 (auditTest), ID35 (Compliance), and the survival phase of the game. Most of these requirements are vital for allowing the release of the game while ensuring revenue and following laws related to video games, privacy, and VR experiences.

Release 4 will contain ID2 (Save Game State), ID36 (Standards), ID6 (FailSafes), ID26 (Training), ID25 (User Documentation), ID24 (AccessibilityTest), ID29 (ExpectedEnvironment), ID21 (Personalization), ID22 (LearningTest), and ID33 (Cultural). Many of these requirements are important for ensuring players can learn, understand, and enjoy both the game and the VR environment.

Release 5 will contain ID5 (Large Enemy), ID27 (Appearance), ID4 (Aiming Accuracy), ID32 (Release), ID31 (Productization), ID30 (InterfacingAdjacentSystems). These requirements focus on either the gameplay experience, or on the game's actual release on digital stores when it's near completion.

17. Ideas for Solutions

When presenting our first release to one of the Teaching Assistants for the class, they recommended more cohesion between the scenes of the game. While this has been implemented with both a story and traveling between environments in-game, more connection could be built within the scenes, such as with a main enemy character or with landmarks in each environment that give a sense that the locations are in the same world.

18. Project Retrospective

Our use of both in-person and online meetings was great in ensuring that group members were on the same page when it came to project due dates and objectives. The in-person meetings were great when we wanted to tackle large portions of projects together, and the online meetings were excellent for quicker review sessions and when we couldn't meet in-person during weekends or breaks. The in-person meeting minutes were also useful for checking in with each member on any achievements they accomplished or struggles they had encountered.

However, our use of Unity with Git caused some issues, as it was difficult to share and update full versions of the project in the online repository when we also had to ensure that Git ignored any downloaded assets that were in our scene. We used a document that contained links where each downloaded asset in a scene could be found, but this clashed with our goal of ensuring that users could easily access our game (they would now have to download all of the outside assets used in the game before playing). For next time, we will try an approach that allows players to merely download the game to start playing, while also keeping unneeded downloaded assets out of the repository (potentially, a new folder named “used downloaded assets” could contain all assets from downloaded packages that are actually used in the game’s scenes, while all other downloaded assets are ignored).

In addition, it was difficult for some members to test the project due to the lack of accessible VR devices. In the future, most, if not all, members should have a VR device for testing the game in VR.

VII. Glossary

Asset: Any object used in the creation of an environment in a game. Could be anything from the audio, to the textures, to the 3D models.

Environment/Scene: A virtual setting in which a game takes place.

Git: A program that assists in version control of software development projects.

Unity Editor Version 2021.3.17f1: A specific version of the Unity game engine that includes development tools for Virtual Reality games.

Unity: A popular game engine that allows users to create a wide variety of games for a variety of platforms through its suite of game development tools.

VR: Virtual Reality.

VIII. References / Bibliography

[1] Aguilar, Chan, Egbesemirone, and Hashim. *VR Dynamic Horror Game Report*. Group 13, May 2022.

[2] M. Fowler, UML Distilled, Third Edition, Boston: Pearson Education, 2004.

[3] Pezzè Mauro, and Michael J. Young. *Software Testing and Analysis: Process, Principles and Techniques*. J. Wiley, 2008.

[4] Technologies, Unity. “Unity Scripting Reference.” *Unity*, https://docs.unity3d.com/ScriptReference/.

IX. Index