**Devlog - PAPA**

**Player Mechanics:**

* **Goal:** Player Movement, Lookout, Breathing effect.
* **Process:** A player controller script was created and the mechanics like movement, lookout, and breathing were achieved using the code in the script, for the movement the Vertical Axis was mapped to W, and S. The horizontal axis was mapped to A, and D. Mouse axes were mapped for lookout mechanics, and we used a Sin wave for the breathing mechanics.

**Interaction Mechanics:**

* **Goal:** Interaction with object.
* **Process:** For the interaction mechanics, we have designed an interactable script for the objects and have attached it to the inspector. In the script the “OnTriggerEnter” checks for trigger events if the player has entered the collision box which then sets the Boolean “caninteract”. In the update function, we check if the player has pressed the “E” key and call the “DealInteraction” function. We had subclasses using this interactive class for different objects present in the game like doors or any readable objects and have overwritten this function accordingly.

**Ghost Follow Mechanics:**

* **Goal:** Make the Ghost Object follow the player.
* **Process:** The plan was to make the ghost follow the player with vector subtraction.
* **Problem:** With this approach, we could make the ghost follow the player, but the ghost was getting stuck behind the obstacles if the player went straight.
* **Solution:** After some research, we found that Unity has NavMesh features that calculate the shortest path to the player and can go around the obstacles to follow the player. Initially, baking is required but after the game runs smoothly. (See Fig1 & Fig2). For the in-game application, I have made some triggers so that when the player enters those triggers, the ghost starts to follow the player.
* **Additional Information on how it works:** The NavMesh Agent checks for the static objects in the scene according to that it calculates the path on which the NavMesh agent(ghost) can move avoiding the obstacles.

A white objects on a checkered surface

Description automatically generated A white objects on a checkered surface

Description automatically generated

Fig 1: With Vector subtraction Fig 2: With NavMeshAgent

**Fear Level Mechanics:**

* **Goal:** If the player is close to the ghost the fear level is high and vice versa.
* **Process:** For this mechanics, We used Vector subtraction between the ghost and the player to determine the distance and then subtracted with 100 to get a variable value where 100 is like when the ghost is almost close to the player and 80 is the farthest they can be, this range can be adjusted by adjusting the speed of the ghost and the player.(See Fig3& Fig4).

A white object on a checkered surface

Description automatically generatedA white tube on a checkered surface

Description automatically generated

Fig 3: When the player is far Fig 4: When the player is near

**Updating Priority list**

* **Goal:** Optimize Priority List.
* **Plan:** As some of our mechanics were completed quickly and some took more time, we broke the mechanics into parts so we could work together and organized some of the mechanics because we wanted to test all the mechanics together and with the UI so I moved the great boxing of the haunted house before we could test all the mechanics in the game.
* **Note:** There were no additional tasks added to the list.

**Haunted House Grey boxing**

* **Goal:** Grey boxing for testing the mechanics.
* **Process:** Used Unity Terrain tools and 3d objects to make the house layout. (See Fig 5 – Fig 8).

A transparent object with a square object on it

Description automatically generated with medium confidenceA white cube on a grey and white surface

Description automatically generated

Fig 5: Top View Fig 6: Perspective view

A white house plan on a checkered surface

Description automatically generated A white rectangular object with a hole in the ground

Description automatically generated

Fig 7: First Floor Fig 8: Ground Floor and Basement

**Checkpoint system**

* **Goal:** Checkpoint system to save the progress of the game
* **Process:** a game manager script was first created, and that component was added to the empty game object inside the script needed a function called save last checkpoint in which it checks through all the triggers in the game object to match the tags of the checkpointand return which one last triggered collision with the player**.**

**Mechanics**

* **Checkpoint System:** The checkpoint system is designed to save the player's progress. When the player reaches a checkpoint, it is marked as the last save checkpoint. If the player loses between checkpoints, they will respawn at the last saved checkpoint. This is managed by a checkpoint script that checks if the player is inside the trigger and saves the checkpoint.
* **Interaction System:** The interaction system is controlled by a trigger box. When the player is near an object and presses 'E,' they can interact with the object.

**Scripts and Coding Workflow**

**Player Controller:**

* The player controller script defines all the methods and the variables of the player, there are components like Rigid Body, and Capsule Collider attached in the game to the player which is called in the script to use in the various methods (defined below). All the variables have “private” keywords and there are getter and setter methods defined to get and set the variables in the code.
* As Unity has the input “W, A, S, D” already mapped to the Axis the rigid body component is called and used to move the player by changing the transform of the player by mapping to X, Y direction.
* For the Mouse the X and Y are already mapped in the unity input system, a Camera variable is defined that is attached to the player, and the transform of the camera is mapped to the mouse X and Y
* A slight Breathing effect has been added to the player by altering the camera y rotation and using a sin wave with the angle of time and the breathing speed (which the user can change).
* Additional variables like walking speed, rotation speed, and breathing speed are defined so that they can be altered in the unity inspector and the keyword “[SerializeField] protected” is used for encapsulation.
* In the Update Method, the if statement check is the Boolean “can move” (See Fig 10), and if it is true the player can move. This Boolean is set to true when the game is not paused, and the player is not dead.

**A computer screen shot of a program code

Description automatically generated**

**Fig 9: Player Controller Move Function: Controls movement of the player**

**A screen shot of a computer

Description automatically generated**

Fig 10: Player Condition to Move

**Ghost Controller:**

* The idea is to make the ghost follow the player inside the haunted house avoiding the obstacles, so for this, we used NavMesh mechanics that Unity has, which calculates the area avoiding the static objects in the scene with the agent component that is attached to the ghost. This agent component triggers the “followplayer” Boolean variable. When the player enters these trigger boxes, the ghost is activated and follows the player until they reach a safe place.
* The ghost also has animation which is controlled by the animator component attached to the ghost game object, The animation and the ghost follow only triggers when the player enters some trigger boxes that are placed in the level
* The game object also has an audio component that plays the audio of scary sounds when the following player is true

A screen shot of a computer

Description automatically generated

Fig 11: Move Method for the ghost

**Game Manager:**

* The game manager is responsible for the progress of the game, and communication between scripts which have been defined using different methods and variables inside the script, other scripts use the game manager as a reference to call the function and the variables inside the game manager to save the progress in the game, trigger events, update media, update UI.
* There are many instances of reference created like the player controller, ghost controller, and other objects in the game, all the variables defined here are encapsulated and their getter(Accessors) and setter(Mutators) methods are created for the other scripts to access them. (See Fig 13)
* In the Update method, the game manager checks if the game is paused then certain code is executed as shown in the code snippet below (See Fig 12), and if the game is paused then performs another set of code.
* If the game is not paused then the player can move the “CheckpointReached()”, and “TimeEvents()” methods will be called, timer events will be called, and another if statement here that if the player is closer to the coast will respawn, the code for that is defined in “ReloadlastCheckpoint”.
* If the game is paused then the pause menu is shown on the screen, the player and the ghost are not allowed to move and there are no interactions between the objects in the game.

A screen shot of a computer program

Description automatically generated A screen shot of a computer program

Description automatically generated

Fig 12: Game Paused / Resume Fig 13: Getter and Setter Methods

* Timer Events Method: this method has a Boolean count-down start which when true starts the countdown from 1 and inside the if statement, there are different if statements that are activated with two conditions, the first one checking the modulus of the countdown and a constant to be 0, and the second which defines the state of the game. (See Fig 14)
* This timer event is used in the game to trigger events/methods after a certain point of time or completing an objective. After the event is completed, the countdown starts Boolean is against it too falls and the countdown is set to 1 again.
* Initially the countdown was set to 0 after the completion of the event but while debugging it was clear that the 0 modulus of anything is 0, with that code the logic would not work properly so it was decided that the countdown should be set to 1.

**A screenshot of a computer program

Description automatically generated**

Fig 14: Timer Events

* SaveCheckpoint Method: this method is defined to save the checkpoint then the player reaches a particular place in the game.
* This method is called in the “Checkpoint” script which is attached to a game object with a trigger and when the player enters that trigger with the arguments the game manager saves the position and the game objects tag which is later used in are there methods in the script. (See Fig 15)

**A black screen with white text

Description automatically generated**

Fig 15: Save Checkpoint

* CheckpointReached Method: This method checks for the last checkpoint switches between the different cases which checks last the checkpoints and executes different codes, this method is also responsible for updating UI, updating media, and saving progress in the game. (See Fig 16)
* ReloadLastCheckpoint Method: When this method is called, it checks for the last checkpoint switches between the different cases and progress accordingly in the game. (See Fig 17)
* The “ReloadLastCheckpoint” method is called when the player encounters the ghost before he reaches the next checkpoint, the code logic for that is defined in the update method so if the player yes closer to the course and the distance between them is 97 units then “ReloadLastCheckpoint” is triggered.

**A screenshot of a computer program

Description automatically generated A computer screen shot of a program

Description automatically generated**

Fig 16: Checkpoint Reached Method Fig 17: Reload Last Checkpoint

**Checkpoint:**

* This script is attached to a game object that has a trigger box Collider and when the player enters the trigger the last checkpoint variable in the game manager is set to this checkpoint with a game tag of the checkpoint name so for example if the last checkpoint is checkpoint 0 the game object tag is “Checkpoint0”. Using this, a game manager can track the last checkpoint which helps in tracking the progress of the game.
* This script has a reference for the game manager from which it saves the checkpoint using the method “SaveCheckpoint” which is defined in the game manager-script. (See Fig 18)

**A screen shot of a computer program

Description automatically generated**

Fig 18: Save Checkpoint on Trigger Active

**Dissolve Controller**

This script creates a dissolved animation of the material attached to a skinned mesh object. The base material has a parameter called dissolver amount which controls the animation, and, in the code, a while loop has been used to decrease this dissolve amount until the value is less than 1. (See Fig 19)

A computer screen shot of a program code

Description automatically generated

Fig 19: Dissolve Controller

**Interactable**

* This script is a parent class end defines the interaction between the player and the game object, The script attached to the game object has a box Collider, in the script checks for the box Collider trigger and when the player enters it enables the text for interaction and then when the player presses “E” the Deal interaction method is called. (See Fig 20 & 21)
* The deal interaction method in the parent class is empty and abstract. This method is overridden in the subclasses according to the animation and the requirements.

A computer screen shot of a program

Description automatically generated A screen shot of a computer

Description automatically generated

Fig 20: TriggerEnter and TriggerExit Fig 21: Calling Deal Interaction Method

**Interactable Subclasses**

* These subclasses derive from the intractable class and are used for different interactions in the game, like opening doors, opening a drawer, and examining clues.
* Beyond the images of the deal interaction method overriding for different subclass scripts like “OpenDoor” (Fig 22), “OpenDrawer” (Fig 23), and Read (Fig 24).

A screen shot of a computer program

Description automatically generated A screen shot of a computer program

Description automatically generated

Fig 22: Deal Interaction – Open Door Fig 23: Deal Interaction – OpenDrawer

A screen shot of a computer

Description automatically generated

Fig 24: Deal Interaction – Read

**Level UI Manager**

* Level manager is responsible for all the UI elements like the fear meter, and clues found, and displays images and instructions on the screen. These are triggered by Booleans or trigger boxes. The script also checks for the game paused and enables and disables the pause menu.
* This script also has the same checkpoint setup where it checks for the last checkpoint in updates the UI accordingly using the tag, once it knows the checkpoint it executes the code accordingly.
* The buttons like the resume button, settings button, and main menu button designed on the canvas have their functionality defined in this script. (See Fig 25)

A computer screen with white and blue text

Description automatically generated A screen shot of a computer program

Description automatically generated

Fig 25: Button Functionality Fig 26: UpdateUI at Checkpoint

**Media Manager**

* The media manager is responsible for managing and updating playing audio and video when required. (See Fig 27)
* There are separate methods defined if the audio must be played once or any number of times. (See Fig 28)
* The Media Manager also checks for the checkpoint and updates the media accordingly. Through the reference of the game manager, it checks if the last checkpoint game tag and executes a different code. (See Fig 29)

A screen shot of a computer program

Description automatically generated

Fig 27: Play Video

A computer screen shot of text

Description automatically generated A computer screen shot of a program code

Description automatically generated

Fig 28: Audio play once Fig 29: Update Media at Checkpoint

**Objectives**

* These scripts show the objective at different checkpoints using the “Sprite renderer” component attached to the game object. (See Fig 30)
* A list of the positions stored in the “position list” which is then used by the sprite rendered to project the objective in 3d space at the position's particular position. (See Fig 30)
* LookAtCameraMethod: This method uses the Look at method to orient the game object to the direction of the Camera, here this is used to show the objective to the player no matter where the player turns the objective will always be facing the player. (See Fig 31)

**A computer screen shot of a program

Description automatically generated**

Fig 30: Update Objective at Checkpoint

**A computer screen with text

Description automatically generated**

Fig 31: Look At Camera

**Trigger Follow Player**

* This script is attached to a game object with a box collider, the “OnTriggerEnter” Method checks if the Player has entered the collision box the “followplayer” Boolean is triggered in the ghost controller script which then starts to follow the player. (See Fig 32)

A computer screen shot of a game

Description automatically generated A screen shot of a computer program

Description automatically generated

Fig 32: Ghost Follow Trigger Fig 33: Trigger Play Audio

**Trigger Play Audio**

* This script is attached to a game object with a box collider, the “OnTriggerEnter” Method checks if the “canplay” Boolean is true which is a serialized public variable and is set in the unity inspector. When the player enters the trigger, the audio plays. (See Fig 33)

**Trigger Show Instructions**

* This script is attached to a game object with a box collider, the “OnTriggerEnter” Method checks if the “canshow” Boolean is true, In the script there are some variables, the “showonce” is a serialized public variable which is set in the inspector in the unity, and they can show variable is set to true and false based on different conditions through the code in the game manager. (See Fig 34)

A screen shot of a computer program

Description automatically generated

Fig 34: Trigger Show Instructions

**Communicating between the scripts**

* All the game objects in Unity have tags defined which have been used in the script to initiate the managers, game objects, and instances. (See Fig 35)
* All the instances of the script are attached to the game object and are called encapsulated variables in the game manager script, separate getter, and setter methods are defined for the ease of calling the variables from the game manager script. (See Fig 36)
* In Fig 37, I have defined how the script communicates with each other and the game manager, basically they use the game manager to check/get the data and manipulate it to support the mechanics of the game.

A screen shot of a computer program

Description automatically generated

Fig 35: Variables Initialization in Game Manager Script

A screen shot of a computer program

Description automatically generated

Fig 36: Example of calling variables from the Game Manager Script in other scripts

Save Checkpoint

Media Manager

Trigger Show instruction

Player Controller

Level UI Manager

Game Manager

Ghost Controller

Trigger Play Audio

Interactable

Trigger Follow Player

Fig 37: Code Workflow

**Game Objects**

* **Player:** The player game object includes the Player controller script, Rigid body, Capsule Collider for the trigger, Another capsule collider for collision
* **Ghost:** The ghost game object includes a ghost controller script, a Capsule Collider for the trigger, Another capsule collider for collision, a Rigid body, a Navmesh Agent component
* **Managers:** Separate game objects exist for the game manager, media manager, and level UI manager. Each has the respective script attached.
* **UI Elements:** UI elements include the fear meter, instructions, and images under the canvas. Trigger scripts enable or disable these elements according to the game's progress.

**Sequence of Events**

**Overview**

* The player starts in Suzie's room, finding clues.
* They move to Amanda's room for more clues, then find a secret door to Richard's room.
* After finding more clues in Richard's room, they go to the ground floor and escape through the back door.
* They proceed through the basement, discovering that the ghost is Suzie's father, and then escape the house.
* Currently, I'm still working on the clues and the escape sequence in the basement, but everything else is done.

**Coding**

* The coding involves defining various Boolean variables and tracking the player's progress through the checkpoint system we created earlier.
* If the player dies, they respawn at the last checkpoint.
* Throughout the game, voiceovers and instructions help the player understand the game mechanics.
* So far, all mechanics are working properly, and we conduct playtests every time we implement new mechanics, which are performing well.

**Room Descriptions**

**Suzie's Room**

* Suzie's room is designed with a floral pattern and pink shades on the walls, reflecting that she is a small child.
* We plan to add more photos of Suzie and her dad, along with some gameplay mechanics, if time permits.
* This room is considered a safe room where the ghost cannot follow Suzie.
* We plan to introduce some game mechanics here, so the player understands them before the main game starts.

**A room with a bed and a dresser

Description automatically generated**

Fig 10: Suzie’s Room

**Amanda's Room**

* Amanda's room is almost complete with all the main clues in place.
* We may add additional items like clothes and extra interactable items that do not lead to clues.
* This allows the player to spend some time exploring the environment to find the main clues.

**A room with a bed and a wood floor

Description automatically generated**

Fig 11: Amanda’s room

**Richard's Room**

* Richard's room is also complete with the main clues that need to be found.
* We might add extra interactable items that do not lead to clues but can be explored.
* This adds suspense, making the player excited about finding the main clue.

**A room with purple boxes and a white spot

Description automatically generated**

Fig 12: Richard’s room

**Basement**

* In the basement, we have added some of the main clues.
* More clues and the sequence of events still need to be coded.
* We have planned a ghost evaporating animation using particle effects. If time permits, we will implement it; otherwise, the ghost will simply vanish.
* After the ghost disappears, the player will try to escape the basement, which will end the game.

**A video game of a video game

Description automatically generated**

Fig 13: Basement

**Prof Feedback**

* **Suggestion:** To make the ending with some text.
* **Plan:** Additional audio was added “I am Never coming Back” to support the end showing that this is the end of the game and then there is an end game screen.