Product Documentation

GitHub repository: https://github.bath.ac.uk/grey-team/DungeonGame/tree/delivery

1. User Manual

You will take control of a grave keeper whose job is to collect the ghosts of the dead people that are still trapped in the graveyard. These ghosts are still roaming the living world, and your job is to send them to the world of the dead. There are two places that you can send these ghost to, and these places are heaven (reserved to people who were good during their lifetime) and hell (reserved to people that were bad during their lifetime), but you have to be careful because if you get spotted by any of the ghosts you will die immediately, and you will have to restart the game.
In order to beat the game, the player has to be stealthy, and the player must remember the stories of the ghosts that he or she collected throughout the gameplay. Basically, the player starts the game in a graveyard, and his job is to roam and collect all the ghosts of dead people that he or she encounters. In order to capture the ghost, the player has to sneak behind the ghost and press space bar, but if the player gets spotted by the ghost, he or she will have to restart the game. The ghost's vision is displayed as a cone on the screen. However, capturing the ghost is not your only objective in the game. When, the player in in close proximity with the ghost, dialogues will start popping and going away. Each ghost will have three pieces of dialogue that will be displayed randomly while the player is in close proximity. So ideally, the player should stay close and to the ghost and read all the dialogues that the ghost has to offer while avoiding getting spotted. After reading the dialogue, he or she should be able to sort the ghost ahead as good ghost or bad ghost. After extracting this information, the player will be free to capture the ghost and proceed to the next room where he or she will encounter another ghost which will require repeating the same process that was mentioned before. Each ghost that is captured will be added to the player's inventory which is represented at the bottom left of the screen.

By the time the player reaches the last room, he or she should have collected all the ghosts from the previous rooms. Now, the player's job is to sort these ghosts into bad ghosts and good ghosts. There will be two boxes in the final room, a white box and a red box. Depositing a ghost in the white box will send it straight to heaven. On the other hand, depositing a ghost in the red box will send it straight to hell. Your job is to send the good ghosts to heaven (white box) and send the bad ghosts to hell (red box). If you fail to sort the ghosts correctly, you will lose the game and return to the main menu, but if you manage to sort them correctly, we can congratulate you because you managed to beat the game. You need to remember that the ghosts will be deposited in reversed order which means that the ghost that you captured last will be deposited first.

Player's key bindings:

- **Up/W** arrow moves upward
- **Down/S** arrow moves down
- **Left/A** arrow moves left
- **Right/D** arrow moves right
- **Space bar** to capture a ghost
- '**E**' to deposit the ghost into the box

Gameplay skills required:

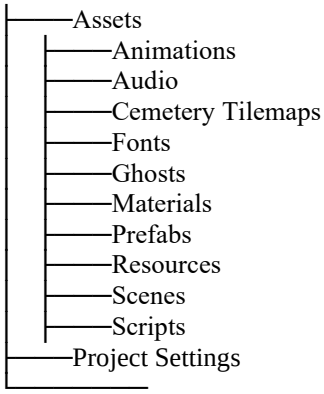
- Stealth and Sneakiness
- Good memory

2. Installation Guide

1. Pull the "delivery" branch from the provided repository
2. Open the "build" folder
3. Run the "Ghostcatchers.exe" file
4. Enjoy the game

3. Maintenance Guide

Folder structure:



Assets – contains all assets used in this project, organized in sub-folders
- **Animations** – contains animations and animation controllers
- **Audio** – contains the soundtrack of the game
- **Cemetery Tilemaps** – contains organizes tilemaps that can be used to create new levels
- **Fonts** – contains the fonts used in this project
- **Ghosts** – contains Scriptable Objects of type ghosts
- **Materials** – contains materials used in this project
- **Prefabs** – contains game object with saved preferences that can be re-used in any scene
- **Resources** – contains .json files used by the Ghost Scriptable Objects
- **Scenes** – contains all the scenes used in the project, new scenes should be added here
- **Scripts** – contains all the C# files used in the project, new scripts should be added here

Project Settings – contains settings used internally by the Unity Game Engine

All the files contained in these folders are ready to use and open sourced.

- Setting up the Unity project
    1. Download Unity Hub from https://unity3d.com/get-unity/download
    2. Install Unity Hub and create a Unity Account
    3. Install Unity version 2018.4.8f1 (LTS)
    4. Add a new Unity project:
        - Create a new folder that will contain the game project
        - Initialize a new git repository
        - Pull the project from the provided GitHub repository
    5. In Unity Hub, add a new project and select the pulled GitHub repository
    6. Open the project in Unity

Examples of usage and possible game expansion ideas:

- Adding a new Ghost
    1. In the Ghost folder, right click: Create -> Scriptable Objects -> New Ghost
    2. In the Inspector, assign a sprite as the ghost Icon variable to the newly created Ghost object
    3. In the Resources folder, add a new .json file with the following format:

```
{
        "ghostName": "Casper",
        "good": true,
        "range": 10,
        "angle": 40,
        "story_1": "I can’t believe that my wife’s boyfriend shot me, what an embarrassing
        way to go.",
        "story_2": "If only I had pulled it from the beginning, I would still be alive.",
        "story_3": "But I was so slow to present to her that engagement ring."
}
```

4. In the Inspector, assign a the .json file to the Json File variable to the newly created Ghost object
5. Open the StartRoom scene
6. Select the GhostManager game object and add the new ghost to the ghost list in the Inspector

- Changing the number of colliders
    1. Open the SpawnColliders.cs file located in the Scripts folder
    2. Change the range of the "numberOfColliders" variable

- Adding a player teleport feature
    1. Open the PlayerMovement.cs file located in the Scripts folder
    2. Create a new public OnTriggerEnter method
    3. Check if the collider's tag is "Grave"
    4. If true, select a random grave on the made and set the player's transform position equal to it

Class descriptions:
Box.cs
- Checks for user input in order to remove a ghost from the inventory
- If the user sorts the ghost in the wrong box, the Death scene is loaded
- Should be assigned to a Box prefab

CameraMovement.cs
- Follows the player around the map
- Should be assigned to the main camera

CollectEnemy.cs
- Checks for user input in order to store the ghost in the ghost inventory

DialogueManager.cs - DialoguePanel.cs - DialogueUI.cs - IconPanel.cs
- These three classes are designed with the MVC pattern in mid
- DialogueManager:
    - Represents the controller
    - Is designed as a Singleton, meaning only one instance of this class can exists at runtime
    - When a ghost is rendered on screen, the DialogueManager updates the DialoguePanel
- DialoguePanel:
    - Represents the Model
    - Stores the current dialogue to display
- DialogueUI:
    - Represenst a View
    - Disaplays the current dialogue
- IconPanel
    - Represents a View
    - Display the current ghosts Icon

EnemyWanderRandom.cs
- Controls the ghost's movements around the map
- Instantiates a new Vision object for each ghost

ExitGame.cs
- Exits the game

FieldOfView.cs
- Creates and updates the ghost Vision
- If the player position is inside the vision, the Death scene is loaded

GhostInventory.cs - InventorySlot.cs - InventoryUI.cs

- These three classes are designed with the MVC pattern in mid
- Ghost Inventory
    - Represents the controller
    - Designed as a Singleton
    - Handles the process of adding and removing ghosts from the inventory
- InventorySlot
    - Represents the model
    - Stores information about a ghost
    - Stores: icon
- InventoryUI
    - Represents the view
    - Display a list of InvetorySlots

GhostObjectManager.cs - Ghost.cs - GhostSpawner.cs

- These three classes are designed with the MVC pattern in mid
- GhostObjectManager
    - Represent the controller
    - Generates new list of ghosts to be displayed each gameplay
    - Returns a random ghost
- Ghost
    - Represents the model
    - It is a Scriptable Object that allows for new ghosts to be created easily
- GhostSpawner
    - Represents the view
    - Gets a ghost based on its index
    - Instantiates the ghost on the map

SpawnDestinations.cs

- Spawns random destinations that ghosts travel to
- Each game spawns a different combination

Graves.cs

- It is a Scriptable Object
- Holds sprites for colliders

SpawnColliders.cs

- Spawns colliders at the beginning of each game
- Selects random colliders sprites from a Graves object
- Number of colliders is a random number between a pre-set range

MainMenu.cs

- Loads the main menu scene

OrderInLayer.cs

- Changes a colliders order in layer depending on the player position
- If the players transorm.position.y is greater thant the colliders, changes the colliders layer to a higher ranking one than the players one
- Creates a 2.5 illusion

Pause.cs

- Loads the pause UI

PlayerMovement.cs

- Handles the players movement based on user input
- Checks for arrow keys input or WASD input

Restart.cs

- Loads the Restart scene

StartGame.cs

- Loads the StartRoom scene