Assignment 4

Team: Alyssa Barrientos

Project Requirements

Your project must include the following functionality at minimum:

- ☑ ~~Player Input that is explained to the player~~
- ☐ Saving and Loading used for settings and gameplay
- ☐ Multiple sounds
- ☐ Settings (including volume)
- ☑ ~~Multiple animated objects~~
- ☐ Main Menu, Gameplay Scene(s), Pausing
    - ☐ Ability to close application from main menu
    - ☐ Unpause
    - ☐ Exit to main menu
- ☑ ~~A more complicated scope of gameplay features than Assignment 3~~
- ☑ ~~Some form of emergent gameplay or progression is provided~~
    - ☑ ~~This may appear as multiple levels that get completed in succession~~
    - ☑ ~~It could also mean having a variety of players/weapons and enemies to populate the game with~~
- ☑ ~~Win and/or Lose conditions that are explained to the player~~

Executable

Completed work may be delivered to target Windows (preferred) or HTML 5. If HTML is used, a public URL may be shared. For Windows, build into an empty folder, then zip the folder produced after completion and submit that zip file. **Game builds for this project will be submitted publicly, so that your classmates may play your games.** More information about sharing game builds will be available soon.

Documentation

The document outlining what has been done should include:

- Introduction - Brief explanation of game, controls, and goals.
- Brief summary of written scripts
- Important Game Objects - What are some important pieces of your scene, and how do they work/interact with other pieces?
- **Identify how all listed Project Requirements are satisfied**
- **If** you are building on from a previous game:
    - Outline the changes and additions that have been made since the previous submission, including at minimum five improvements (ranging from small to

significant) over your previous submission. These can be graphical (like animations, lighting, or particle effects) or directly relevant to gameplay (like new mechanics or enemies)

- If you are **not** building on from a previous game, provide a mixture of three:
    - Items that have been reused from previous projects
    - Items made for this project that could be reused in future projects
- Referenced Material
    - Identify any assets, packages, guides, or tutorials that contributed significantly to your project, and the degree to which they were used (ie as-is, slightly modified, light usage)

Source Code

Source code for all C# scripts must be provided by one of the following methods:

- Github: https://github.com/AlyBarr/Rougelike

Concept for "Rougelike"

A dungeon-crawling adventure where players control a character navigating a perilous dungeon. The objective is to survive the dungeon's many dangers, solve puzzles, collect items, and ultimately find the potion needed to escape. The game features procedurally generated levels, various types of monsters, and a range of items and abilities to collect. The player's ultimate goal is to navigate through the dungeon and survive long enough to find the potion and escape.

Controls

- **Arrow Keys**: Move your character in the dungeon.
- **G**: Pick up items.
- **D**: Drop items from your inventory.
- **I**: Open your inventory to view and equip items.
- **V**: View past messages, which include important information and story progress.
- **ESC**: Pause the game and access the pause menu.

Game Mechanics

The game provides a series of messages that guide the player through the dungeon, inform them about gameplay, and give instructions on how to survive.

Key Features:

1. **Procedural Generation**: The dungeon layout changes with every game, creating a new adventure each time.
2. **Monsters**: Monsters like Flying Eyes and Fire Spitters lurk in the dungeon. If a monster touches the player, they die. Players must carefully navigate around these threats to survive.

3. **Item Collection**: Players can collect items (e.g., potions, weapons) that help them survive and progress.
4. **Inventory System**: Items can be picked up, dropped, and used. The inventory is accessed by pressing 'I'.
5. **Messages**: Important messages appear throughout the game to guide the player and convey crucial information.

Important Game Objects

1. **Dungeon Layout**: The dungeon is procedurally generated, consisting of rooms, corridors, and doors. Each room may contain monsters, items, or challenges.
2. **Player Character**: The player's avatar, controlled using the arrow keys, can pick up items, interact with objects, and avoid monsters.
3. **Monsters**: Creatures that roam the dungeon and must be avoided. Contact with monsters results in death.
4. **Items**: Objects that can be found, picked up, and used in the game. These may include potions, weapons, or keys to unlock doors.

Game Mechanics

The game provides a series of messages that guide the player through the dungeon, inform them about gameplay, and give instructions on how to survive.

Key Features:

1. **Procedural Generation**: The dungeon layout changes with every game, creating a new adventure each time.
2. **Monsters**: Monsters like Flying Eyes and Fire Spitters lurk in the dungeon. If a monster touches the player, they die. Players must carefully navigate around these threats to survive.
3. **Item Collection**: Players can collect items (e.g., potions, weapons) that help them survive and progress.
4. **Inventory System**: Items can be picked up, dropped, and used. The inventory is accessed by pressing 'I'.
5. **Messages**: Important messages appear throughout the game to guide the player and convey crucial information.

Important Game Objects

1. **Dungeon Layout**: The dungeon is procedurally generated, consisting of rooms, corridors, and doors. Each room may contain monsters, items, or challenges.
2. **Player Character**: The player's avatar, controlled using the arrow keys, can pick up items, interact with objects, and avoid monsters.
3. **Monsters**: Creatures that roam the dungeon and must be avoided. Contact with monsters results in death.

4. **Items**: Objects that can be found, picked up, and used in the game. These may include potions, weapons, or keys to unlock doors.

Messages are presented to the player as soon as they begin their adventure. They explain:

1. **Introduction to the Dungeon**: A lighthearted greeting that sets the stage for the game.
2. **Objective**: The player must find a potion to escape the dungeon.
3. **Dangers**: A brief explanation of the monsters that inhabit the dungeon, emphasizing the risk of death upon contact.
4. **Movement**: Instructions on how to move around and interact with the environment using the arrow keys, and how to pick up and drop items.
5. **Game Management**: Information on how to view past messages and access the pause menu.

Scripts

1. UIManager.cs

The **UIManager** script is responsible for managing the user interface elements of the game, including displaying messages, health status, inventory, and more. It also controls the game's pause screen and the message log that provides feedback to the player.

Key Functions:

- **UpdateHealthDisplay()**:
  - Updates the player's health bar to visually represent their current health status.
- **ShowInventory()**:
  - Displays the player's inventory UI, allowing them to view collected items, use or drop them.
- **DisplayMessages()**:
  - Displays all messages in the message queue (using AddMessage), which can be used to notify the player about game events, instructions, or warnings.
- **ShowPauseMenu()**:
  - Activates the pause menu when the ESC key is pressed. The game is paused, and the player can either resume or quit the game.
- **AddMessage(string message, string color)**:
  - Adds messages to the message log. These messages are shown on the screen to give feedback to the player. They can be displayed in different colors to signify different types of information (e.g., blue for general messages, red for warnings).

This function conveys crucial game instructions to the player, like how to move, how to interact with items, and what the goal is.

---

2. GameManager.cs

The **GameManager** script is responsible for controlling the overall flow of the game. It handles initialization, the game's main loop, and the game's states (e.g., running, paused, game over).

Key Functions:

- **Start()**:
  - Initializes the game, setting up the dungeon, the player, and any other necessary elements (e.g., monsters, items).
- **Update()**:
  - Runs every frame to manage input (movement, item usage) and check game states (e.g., whether the game should be paused or if the player is dead).
- **PauseGame()**:
  - Pauses the game when the player presses the ESC key. It also brings up the pause menu with options to resume or quit the game.
- **ResumeGame()**:
  - Resumes the game from the pause menu.
- **GameOver()**:
  - Triggered when the player dies or when they meet a losing condition. Displays a game over screen.

The AddMessage function could be used here to show warning messages or tutorial information at the start of the game, which is handled by the GameManager and UIManager.

---

3. ProcGen.cs

The **ProcGen** (Procedural Generation) script is responsible for creating the dungeon layout randomly, ensuring that each new playthrough is unique. This script creates rooms, corridors, walls, and places items and monsters.

Key Functions:

- **GenerateDungeon()**:
  - This function generates the dungeon using a procedural algorithm. It creates rooms, corridors, walls, and places objects (e.g., items, enemies) within the generated layout.
- **CreateRoom()**:
  - Creates individual rooms that are connected via corridors. Rooms could have different sizes and shapes and may contain special items or enemies.
- **PlaceMonsters()**:
  - Randomly places monsters in the dungeon after it is generated. Monsters could be placed in specific rooms or wandering between corridors.

- **PlaceItems()**:
    - Randomly places collectible items throughout the dungeon, such as healing potions, weapons, keys, etc.

The dungeon generation process could ensure that a potion is placed somewhere within the dungeon, which the player needs to find.

---

4. MapManager.cs

The **MapManager** script controls the layout of the dungeon and the display of the map. This script could be responsible for managing the visibility of rooms, corridors, and ensuring the player's position is updated as they explore.

Key Functions:

- **UpdateMapDisplay()**:
    - This function updates the visual map to show the player's current location in the dungeon. It could reveal or mark explored rooms and corridors as the player navigates.
- **RevealRoom()**:
    - When the player enters a new room, this function ensures the room is revealed on the map display.
- **HideRoom()**:
    - This function hides rooms that the player has not yet discovered.

The UpdateMapDisplay method could be used to update the map as the player moves around and reveals new areas. As the player navigates, the system could update the map display to show areas they have visited or need to explore.

Project Requirements Checklist

1. Player Input:

- Players interact with the game using the arrow keys to move and the G, D, I, V, and ESC keys to manage their inventory and pause the game.

2. Saving/Loading:

- Not yet implemented. Future updates can include a save/load system to track session progress.

3. Sound:

- The game does not yet have a dedicated sound system. Future iterations may include sound effects for picking up items, monster encounters, and background music.

4. Settings:

- The settings menu can be added to allow for customizable volume and gameplay options in future updates.

5. Animated Objects:

- The Firespitter has a walking animation when it sees and chases after the player.

6. Main Menu, Gameplay Scene(s), Pausing:

- The game includes a main menu for starting the game, and players can pause the game using the ESC key.

7. Win/Lose Conditions:

- The game ends when the player dies (in contact with a monster) or successfully escapes the dungeon by finding the potion as I was not able to implement the win feature.

8. Emergent Gameplay or Progression:

- The procedurally generated dungeon ensures each gameplay session offers a unique experience with different challenges and paths.

Changes and Additions from Previous Submissions

If this game is being developed further, improvements could include:

1. **Procedural Generation Enhancements**: Further refinement of the dungeon generation system to create more varied and challenging levels.
2. **Enemy AI**: Implementing smarter enemies that pursue the player or react to environmental triggers.
3. **Item Variety**: Adding more types of items, such as keys to unlock special rooms, health-restoring potions, or weapons to defend against monsters.
4. **Sound and Music**: Introducing sound effects and background music to increase immersion.
5. **Save/Load Feature**: Adding the ability for players to save their progress and load it in future sessions.

Concept Art and Design Inspiration

The game's style is inspired by spooky and cartoon-like due to the dungeon-crawling twist. The characters and environment should feel slightly dangerous, with visual cues indicating threats and rewards.

- **Characters**: Almost ghostly maybe an ode that will be developed further.
- **Environment**: Dark, eerie dungeons filled and empty alongside lurking enemies like flying eyes and fire-spitting monsters.

Source Code

All C# scripts are available on the project's GitHub page:

Roguelike GitHub - https://github.com/AlyBarr/Rougelike

Referenced Material

Patrik Arts	Art	https://patrik-arts.itch.io/
ChatGPT	Debugging Help Script	https://chatgpt.com/
	Navigation Help and Refinement	
Youtube	Title Menu Inspiration	https://www.youtube.com/watch?v=-GWjA6dixV4
	Character Animation Unity Tutorial + Dungeon Gens	https://www.youtube.com/watch?v=tTIlDb71t2s&t=191s
		https://www.youtube.com/watch?v=FPenosY_MEU&list=PLzbRW-gm6o9ZxxDcx2u2Oj-Vap4HHQdwz&index=5
	Guide to Unity2D - Tilemaps	https://www.youtube.com/watch?v=sTKw-svEScQ&list=PLzbRW-gm6o9ZRam-sK4zIKjdxQPb8ZGv1&index=1

The following resources were referenced or used during development:

1. **Unity Documentation**: For learning about game object management, input handling, and procedural generation techniques.
2. **Tutorials**: Various YouTube tutorials on creating roguelike games and procedural dungeon generation.