Maze of Lost Souls

## Software Engineering for Gaming 2024 Project

### **Group Members**

* Padraig Mc Cormack
* Benjamin Mills
* Liam Moore
* Michael Doherty

*Members of the group did not work specifically on one class, group worked together on code with multiple members contributing to certain classes and members helping each other on issues with code*

**Group Member Contributions**

1. **Benjamin:**
   * Designed and implemented the level system, ensuring that a new map is generated when the player uses the key on the door, creating a seamless progression through levels.
   * Made sure the player’s stats (health, level, inventory) are saved and persist across different maps, ensuring that player progression is not lost when transitioning between levels.
   * Handled the placement of the key and door on the map, ensuring that each map contains one of each item, which the player can interact with, and that the key is displayed in the inventory when collected.
   * Cleaned code and added some more comments, improving readability and maintainability of the system for future development and easier debugging.
2. **Liam:**
   * Designed and implemented the save and load functionality, allowing the game state to persist between sessions.
   * Contributed to the inventory system by adding methods for managing items.
   * Enhanced code maintainability by applying encapsulation principles to improve the structure and clarity of key classes.
   * Created Game Manager Class’s
3. **Michael:**
   * Designed and implemented the enemy system by creating the Enemy classes.
   * Used encapsulation for enemy attributes such as health and damage, ensuring proper data hiding.
   * Created a damagePlayer() method to reduce player health, integrated into the main file. This method simulates a scenario where the enemy is disguised as a health potion (p), tricking the player when collected.
   * The enemy was placed on the map using placeItemOnMap(), a method provided by another team member.
4. **Padraig:**
   * Designed and implemented the core game mechanics, ensuring smooth interaction between the various game components.
   * Developed the player movement system, allowing players to move around the map using directional inputs.
   * Created the map generation logic, including randomized maze layouts and the placement of key elements like walls, potions, and open spaces.
   * Handled item interactions, enabling players to pick up and use items during gameplay.
   * Organized the game flow, ensuring all systems worked together to provide a complete gameplay experience.

**User Manual**

**How to play the game:**

**1.Start the game:**

* Launch the game from the console
* Choose from the menu one of the options:
  + [1] Start Game – this begins the game
  + [2] Load Game – this will load from the saved file
  + [3] Quit – this exits the game

**2.Moving the Player**

* W – move up
* S – move down
* A – move left
* D – move right

Avoid walls which are displayed with a “#” throughout the game.

**3. Interacting with items:**

* When standing on the location of an item press the ‘space’ key to collect it.
* To view your inventory, you can press ‘I’.
* To pick up or interact with an item you press ‘H’.

**4.Saving and quit:**

* Press the ‘q’ key when in game to save and quit.

**5.Objective:**

* Collect the keys ‘K’ for the door to unlock ‘D’.
* Avoid running into the enemy’s ‘E’.
* Make it through the level and escape.

**Code Explanations**

1. **Player Class:**
   * Manages the player’s in-game attributes:
     + position: Tracks the player’s location on the map, represented by x and y coordinates.
     + health: Represents the player’s health, starting at a specific value and decreasing when attacked or increasing when healed.
     + level: Indicates the player’s progression in the game, tied to experience points increasing when XP exceeds value
     + inventory: A collection of items the player picks up during gameplay.
   * Includes methods for:
     + move(): Updates the player’s position based on directional input (e.g., WASD keys).
     + collectItem(): Adds items to the inventory and checks for specific conditions, like gaining a key.
     + hasItem(): Checks if a particular item exists in the player’s inventory.
     + discardItem(): Removes an item from the inventory after use.
     + setPosition(): Explicitly sets the player’s location on the map.
   * How it works:
     + The player interacts with the game world by moving on the map grid, collecting items, and managing health and inventory.
2. **Map Class:**
   * Represents the game world as a grid of tiles (2D array):
     + Each tile contains information about walls, open spaces, and items like keys or doors.
     + width and height: Define the dimensions of the map.
   * Includes methods for:
     + generateMaze(): Randomly creates the maze by placing walls and open spaces, ensuring a playable environment.
     + setTile(): Updates a specific tile with a new value (e.g., placing a player or item).
     + isItem(): Checks if a tile contains an interactable item.
     + getItem(): Retrieves and removes the item from the map when the player interacts with it.
   * How it works:
     + The map provides the environment for gameplay, allowing dynamic interactions like moving players and picking up items.
3. **Item Class:**
   * Represents collectible or interactable objects in the game:
   * **Attributes:**
     + **name:** Identifies the item (e.g., "Key").
     + **description:** Explains what the item does or represents.
   * **Methods:**
     + **getName():** Returns the item's name.
     + **getDescription():** Provides details about the item.
     + **printDetails():** Displays the item’s name and description in the console for the player to see.
   * **How it works:**
     + Items are initialized with a name and description, placed on the map, and collected by the player. Their details can be accessed through these methods when needed in gameplay interactions.
4. **Enemy Class:**
   * Manages non-player character that challenge the player:
     + position: Tracks the enemy’s location on the map.
     + behaviour: Determines how the enemy interacts with the player (e.g., attacking, moving).
   * Includes methods for:
     + damagePlayer(): Reduces the player’s health when an enemy is encountered.
   * How it works:
     + Enemies are represented on the map and interact with the player through attacking
5. **SaveLoadManager Class:**
   * Handles persistence of game state between sessions:
     + saveGame(): Writes player and map data to a file.
     + loadGame(): Restores game state from a file.
   * How it works:
     + The save and load functionality ensures players can continue their progress without starting over.

**UML Diagram**

A screenshot of a computer screen

Description automatically generated