Name

Fahima Afrin Anuva (2321049042)

MD. Minhaz Ahamed Rohan (2232065642)

MD. Tushar Khan (2232344642)

Dungeon Quest Game Report

Abstract

The Dungeon Quest Game is a text-based adventure game built using Java Swing, which integrates Object-Oriented Programming (OOP) principles and GUI design. The game offers an interactive dungeon exploration experience where the player navigates through rooms filled with randomized events like treasures, traps, and enemies. The project is designed to demonstrate a variety of programming concepts while ensuring a user-friendly, engaging gameplay experience.

Introduction

The **Dungeon Quest Game** was created with the primary goal of providing an engaging, interactive adventure experience. By leveraging Java Swing for the graphical user interface (GUI) and incorporating Object-Oriented Programming (OOP) principles, the game allows the user to explore a dungeon, encountering random events and challenges along the way. Players interact with the game by selecting directions to move, with their health being affected by encounters like treasures, traps, and enemies.

The project not only serves as an enjoyable game but also provides a solid foundation for implementing and demonstrating key concepts in OOP, GUI design, and exception handling.

System Design

The system was designed using a modular architecture that focuses on key components such as:

1. **Player**: The player class holds the player's attributes, such as health, and includes methods to handle player actions like moving, attacking, and healing.
2. **GameWindow**: This class is responsible for the graphical user interface, using Swing components like `JTextArea`, `JTextField`, and `JButton`. It facilitates user interaction, displaying messages, and receiving input for player actions.
3. **RandomEvent**: This class generates random events such as treasure discovery, enemy encounters, and traps, which significantly impact the player's health and progress.
4. **Event Handler**: The event handler is responsible for processing player input and invoking the corresponding actions based on the current room the player is in.

The game operates on a simple **game loop** where each turn the player chooses one of the four directions to explore, leading them to encounter a random event. The architecture allows for easy extension of the game by adding new types of events or features.

Key OOP Principles

The design leverages several key OOP principles:

- **Encapsulation**: The player's attributes (such as health) and behavior (such as attacking or healing) are encapsulated within the `Player` class. This ensures that the player's state is managed within a single object and access to it is controlled through methods.
- **Inheritance**: The `RandomEvent` class serves as a parent class for various types of events like `Treasure`, `Trap`, and `Enemy`. This allows for code reuse and the addition of new event types in the future without modifying existing functionality.
- **Polymorphism**: The game handles different event outcomes, such as healing from treasure or taking damage from enemies or traps, using polymorphic methods. For example, the `RandomEvent` class defines an abstract method `triggerEvent()` that is implemented differently by subclasses like `Treasure`, `Trap`, and `Enemy`.

Implementation

Key Features

1. **Randomized Events**: The core of the game revolves around randomized events that the player encounters as they choose directions to move. These include:
    a. **Treasure**: Increases player health.
    b. **Trap**: Decreases player health.
    c. **Enemy**: Triggers a battle where the player and enemy attack each other until one is defeated.
2. **Graphical User Interface (GUI)**: The game uses Java Swing to create a simple yet functional GUI that allows users to interact with the game. The GUI includes:
    a. A `JTextArea` for displaying game messages and player status.
    b. A `JTextField` for user input.
    c. Buttons for submitting player choices.
3. **Exception Handling**: The game anticipates and handles user input errors, such as invalid direction choices or non-numeric inputs, to prevent crashes and ensure a smooth user experience.
4. **Health Management**: The game tracks the player's health, which is affected by events such as finding treasure, falling into traps, or battling enemies. If the player's health drops below 40, the game ends with a "Game Over" message. Similarly, if the player accumulates enough health, the game ends with a "Win" message.
5. **Player Interaction**: The game is turn-based, with the player making choices by selecting one of four directions (North, South, East, West). Each direction leads to a random event.

Output Examples

When the game starts, the following output appears on the screen:

Welcome to Dungeon Quest Game!

Your current health: 100
Select any direction to move into a room:
1) North
2) South
3) East
4) West
Type the number of your choice and press Enter.

Depending on the player's input, the game displays the result of the random event:

- If the player finds **treasure**:

Congratulations! You found a treasure. Health increased by 40.

- If the player encounters a **trap**:

You fell into a trap and died! Thank you for playing. The game ended.

- If the player faces an **enemy**:

Goblins attacked you! Health decreased by 20.

Testing

Testing was performed manually to ensure all features worked as expected. The key focus was on validating the functionality of randomized events and verifying that the game correctly handles user input. Additionally, edge cases such as invalid inputs (non-numeric values or out-of-range choices) were tested to ensure that the exception handling mechanisms were functioning as intended. No crashes or unexpected behavior were observed during testing.

Challenges

A few challenges encountered during development included:

- **Designing an intuitive GUI**: Creating a user interface that was both functional and engaging was a challenge, as it had to display a lot of information while remaining easy to navigate.
- **Handling Game Logic Integration**: Ensuring that the different game mechanics (health management, event handling, and user interaction) integrated smoothly took several iterations of testing and debugging.

Future Enhancements

Future improvements for the game include:

1. **Multiplayer Support**: Allowing multiple players to play in the same game session, either cooperatively or competitively, would add depth to the gameplay.
2. **Enhanced Graphics**: Incorporating more advanced graphical elements such as images for rooms, characters, and items would significantly improve the user experience.

Expanded Game Mechanics: Adding more event types, complex enemy AI, and additional story elements would increase the game's depth and replay ability.

Conclusion

The Dungeon Quest Game project allowed for the practical application of fundamental programming concepts like Object-Oriented Programming, exception handling, and GUI design. It provided a hands-on opportunity to explore the principles of game development while creating an engaging and fun interactive experience. While simple, the game lays the groundwork for future, more complex game development projects.