Java Game Development: Grid Adventure

Java Programming Report

Assignment

CS

JC2002

Full Name

ZISHAN XU

Student ID

50087477

Contents

1 Introduction

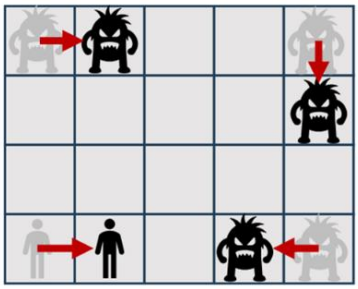
1.1 Project Objectives

This project aims to apply Object-Oriented Programming (OOP) to develop a simple text-based Java Role-Playing Game (RPG) named "Grid Adventure." The core objective of the game is to allow players to explore a 2D map, interact with monsters, and achieve the ultimate goal of defeating all monsters through strategy and combat. The game is characterized by its interactivity and strategic elements, requiring players to move flexibly and effectively respond to the random movements of monsters.

1.2 Game Mechanics

1、Character Movement

Player can command their character to move up,down,left,or right by one grid cell,while monsters move randomly by one step



2、Interaction and Combat

attack mechanism is triggered when the player attempts to a position occupied by a monster,and likewise when a monster moves to the player's position

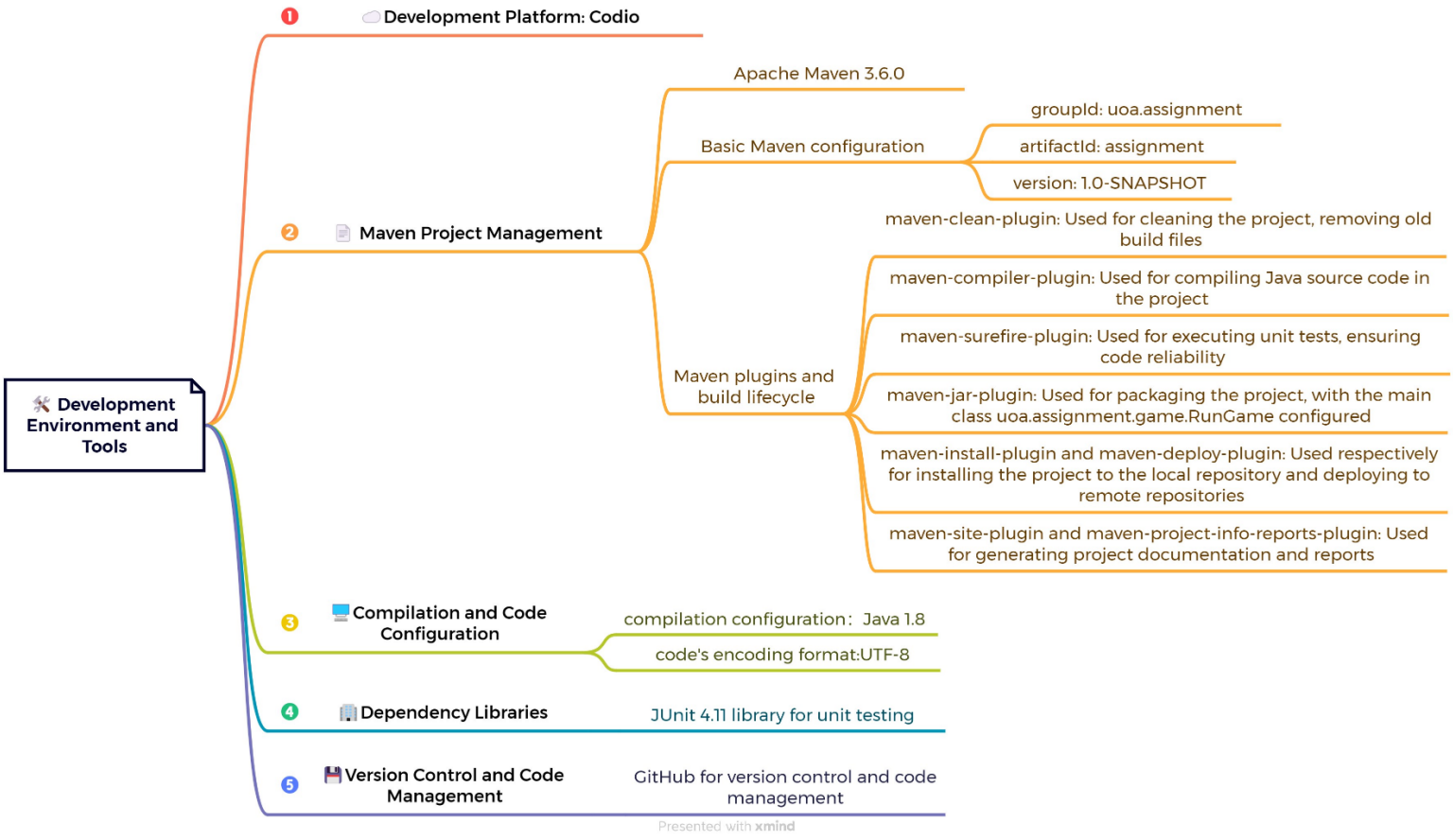
3、Health and Outcomes

start with 100 health points,a successful attack by the attacker will reduce the defender's health points (in the initial difficulty, player attacks reduce a monster's health by 50 points, while monster attacks reduce the player's health by 20 points). A character is considered "dead" when their health drops to zero.

4、Game end Condition

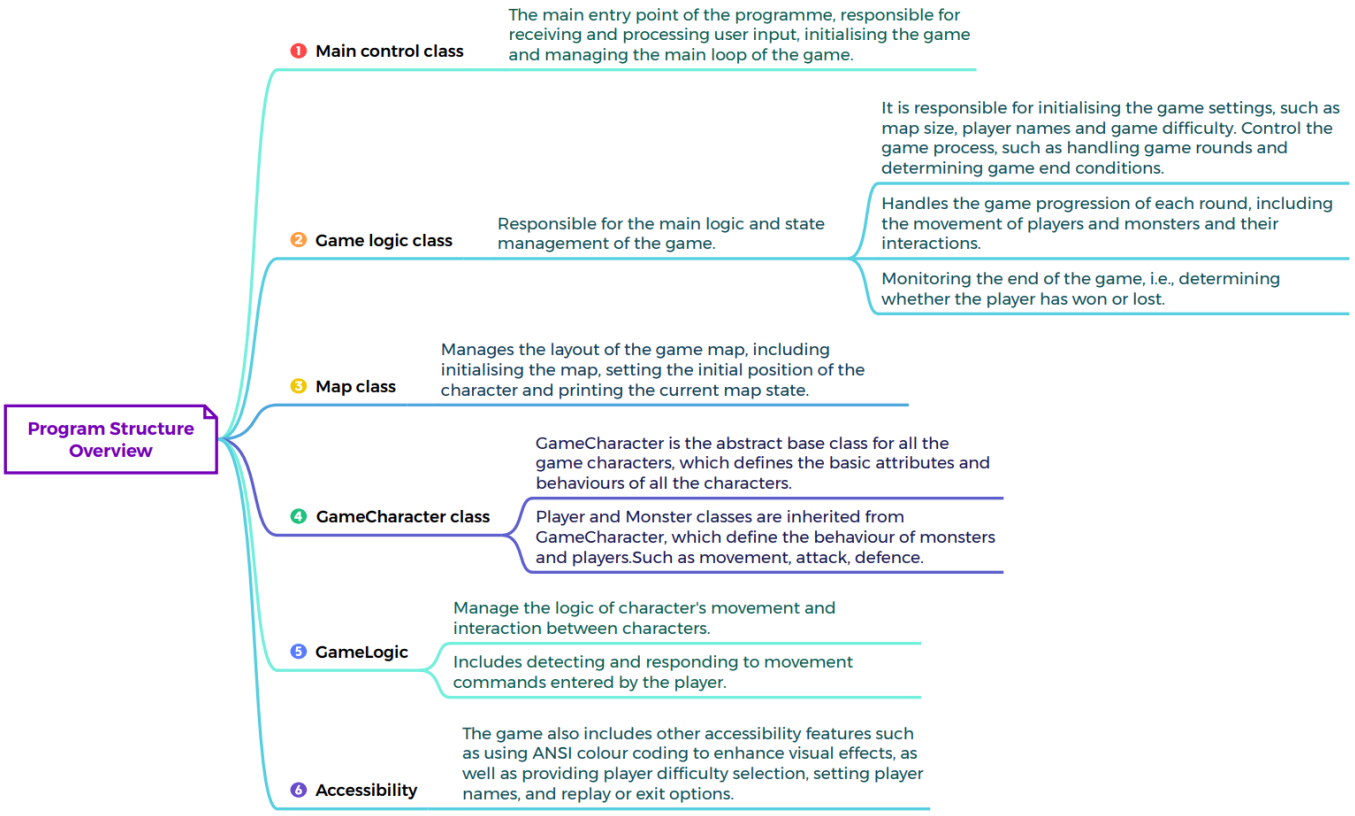
The player needs to eliminate all monsters before being defeated by them. The game ends based on the death of the player or all monsters.

1.3 Development Environment and Tools



2 Details of Program Implementation

2.1 Overview of Program Structure



2.2 Main Classes and Their Associations

2.2.1 Game Character Design

Step 1: GameCharacter Class

The GameCharacter class serves as the foundation for all game characters. As an abstract class, it defines basic attributes of game characters such as name (`name`), health value (`health`), and position on the game map (`row` and `column`). It includes two constructors: one with a string parameter `name` for creating named character instances with health initialized to 100, and a default constructor for creating unnamed character instances. Key methods like `sayName()` and `setName(String name)` are used for retrieving and modifying the character's name, while `getHealth()` and `setHealth(int health)` manage the character's health. Abstract methods `hurtCharacter(GameCharacter character)` and `successfulDefense()` support the game's combat and interaction mechanisms. These are concretely implemented in derived classes such as Player and Monster, enhancing the game's interactivity and challenge.

Step2:Player/Monster class

In our game design, the Monster and Player classes are key extensions of the GameCharacter class, representing the monster and player roles in the game. These classes inherit from GameCharacter, retaining core attributes such as name, health value, and position on the map,

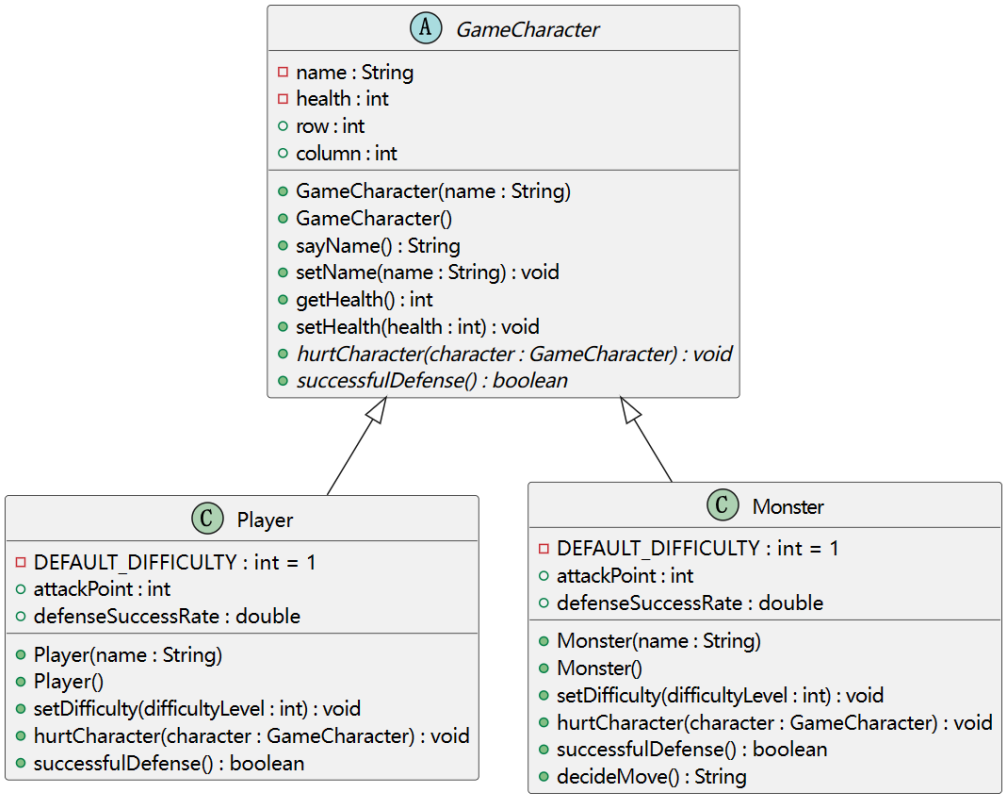
and they adopt the standard behaviors of GameCharacter while introducing unique functionalities and characteristics specific to each role.

Both the Player and Monster classes share the `DEFAULT_DIFFICULTY` attribute, set at a default difficulty level of 1. Their `setDifficulty` method allows adjusting attack power (`attackPoint`) and defense success rate (`defenseSuccessRate`) according to the game's difficulty level. Specifically, the default attack power for a player is set to 50, while for a monster, it is 20. In terms of defense success rate, the player's default rate is 30%, and the monster's is 50%.

These classes extend the GameCharacter class by overriding the `hurtCharacter` and `successfulDefense` methods, defining how to attack other characters and how to randomly determine the success of defense, respectively. This implementation in the game reflects proactive attack and passive defense strategies, enhancing the game's interactivity and challenge. Unlike players, the Monster class's unique method `decideMove` grants it the capability of random movement, adding unpredictability and challenge to the game.

Step3:Principles of Object-Oriented Programming

In the project, we cleverly applied the principles of Object-Oriented Programming (OOP) to the design of game characters. The `GameCharacter` serves as an abstract base class, setting common attributes and behaviors for all characters, such as health values and positions on the game map. The `Player` and `Monster` classes, inheriting from this base class, showcase the power of inheritance in OOP, possessing unique behaviors and attributes while sharing common features. The application of polymorphism is evident in the abstract methods defined in `GameCharacter` (such as `hurtCharacter()` and `successfulDefense()`), which have distinct implementations in `Player` and `Monster`, thereby presenting diverse behaviors under the same interface. This practice of OOP not only enhances code reusability and maintainability but also enriches the game's interactivity and strategic depth.



2.2.2 Map class

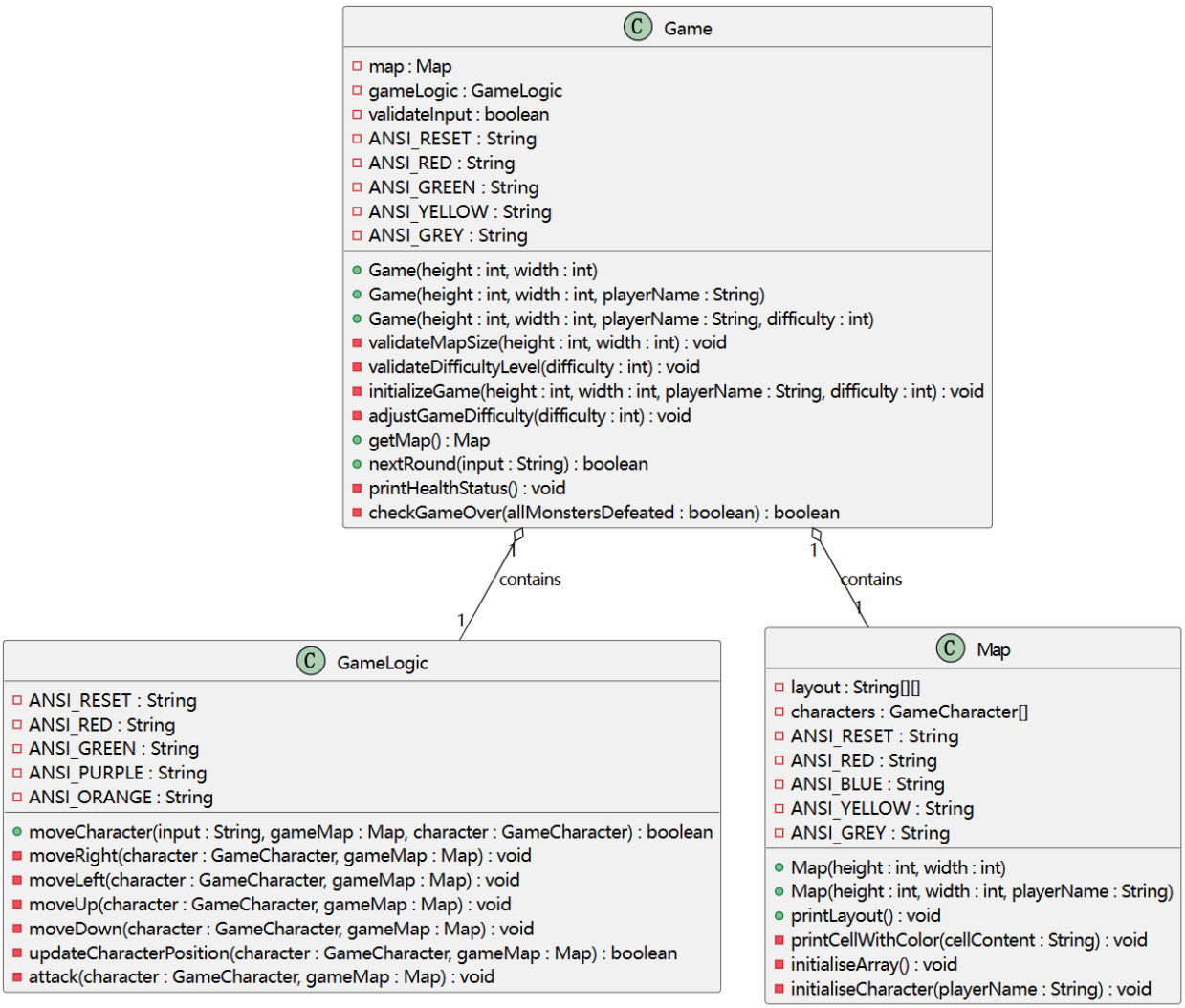
In our game design, the `Map` class plays a key role, managing the layout of the game map and the positions of characters. It uses a two-dimensional string array `layout` to represent each cell of the map and a `GameCharacter` array `characters` to store characters like players and monsters. The `Map` class has two types of constructors, one for initializing the map layout based on specified dimensions and another optionally setting the player's name. It initializes the grid cells and places characters through `initialiseArray()` and `initialiseCharacter(String playerName)` methods and offers a `printLayout()` method to display the map layout in different colors on the console, enhancing the visualization. This design ensures that the positions of players and monsters directly affect the game's strategy and interaction, providing essential visual and logical support for navigation and combat.

2.2.3 GameLogic class

In our game design, the `GameLogic` class is a key component for managing character movement and interactions. It controls the movements of players and monsters on the map and manages interactions such as attacks and defenses through a series of static methods. `GameLogic` enhances console output readability using ANSI color codes to highlight different game events. Key methods like `moveCharacter` determine the movement direction based on player input, while `moveRight`, `moveLeft`, `moveUp`, `moveDown` implement the specific movement logic. The `updateCharacterPosition` method updates positions on the map and handles interaction logic, such as encounters between players and monsters. The `attack` method is responsible for handling combat interactions. These methods simplify the code structure and provide a clear logical center for the game, ensuring that behaviors are closely aligned with game rules and map states, enhancing strategic and dynamic aspects of the game.

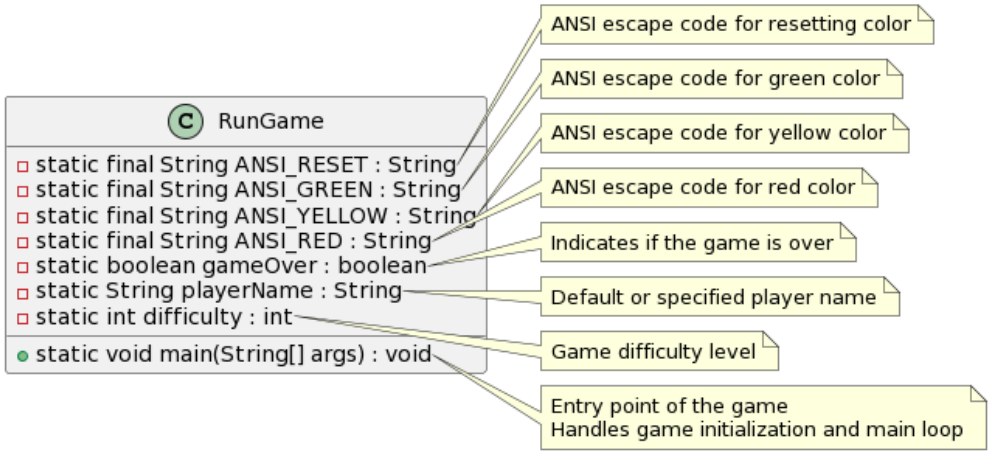
2.2.4 Game class

In our game design, the Game class is the core of the game process, responsible for initializing the environment, managing rounds, and determining the end conditions. Its constructor method flexibly sets key parameters like map size, player name, and game difficulty. Core methods include nextRound for handling movement each round, printHealthStatus for displaying health status, and checkGameOver to check if the game has ended. The Game class centrally manages character movement and interaction, providing clear game logic management.



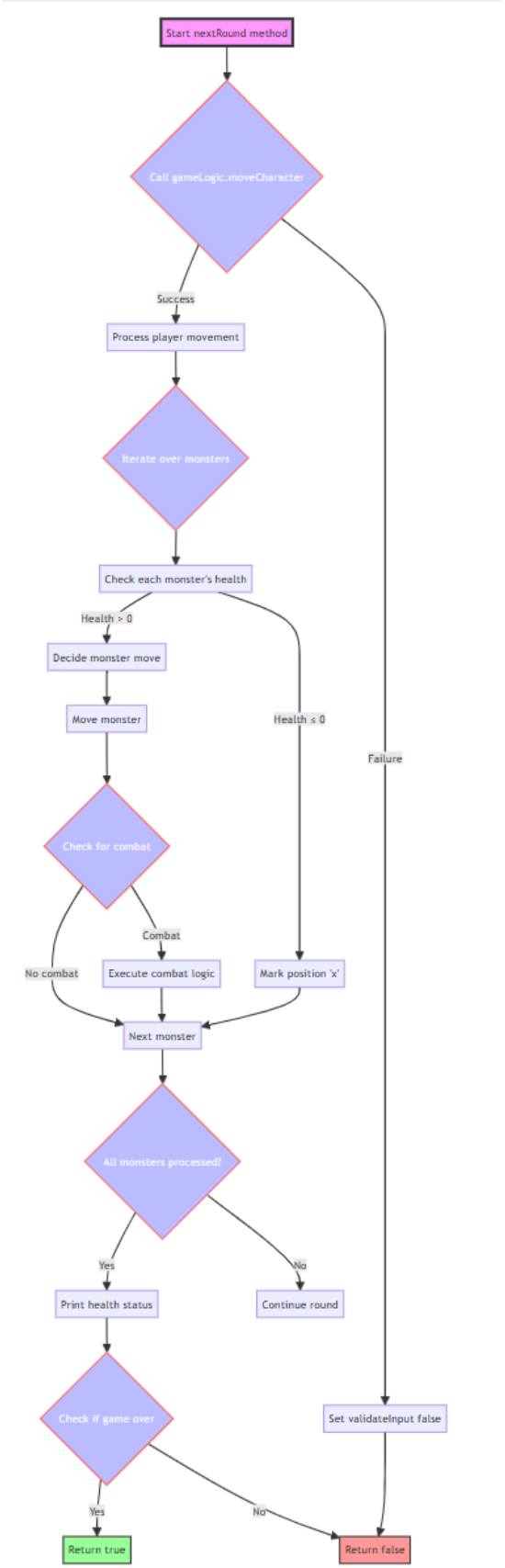
2.2.5 Main Control Class

In our game design, the RunGame class serves as the main entry point and plays a crucial role in managing the entire game process. It uses the gameOver boolean to monitor whether the game should end or continue and allows players to personalize their character names with the playerName string (default is "Player"). The difficulty integer provides a choice of different difficulty levels, increasing the challenge. The RunGame class parses command-line arguments to set initial conditions like map size and player name and receives input through a Scanner object. Its key game loop collects player inputs and calls the Game class's nextRound method to process the logic. This design ensures the coherence of the game process and provides an intuitive interaction method. At the end of the game, RunGame inquires if players wish to re-start, reflecting the game's replayability and user-friendliness.



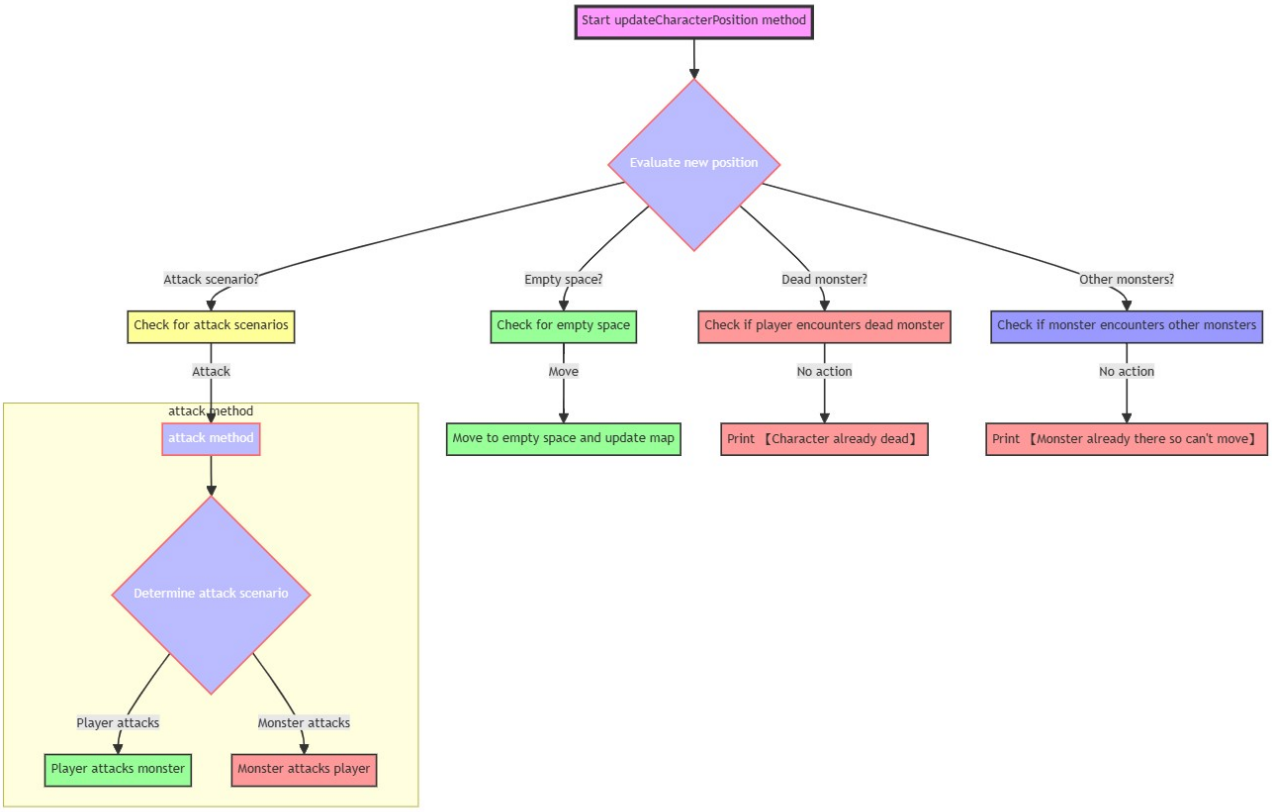
2.3 Key Methods

2.3.1 The `nextRound()` Method in the Game Class



2.3.2 The `moveCharacter` Method in the GameLogic Class

See **Appendix 4: The moveCharacter method in GameLogic** and the diagram below:

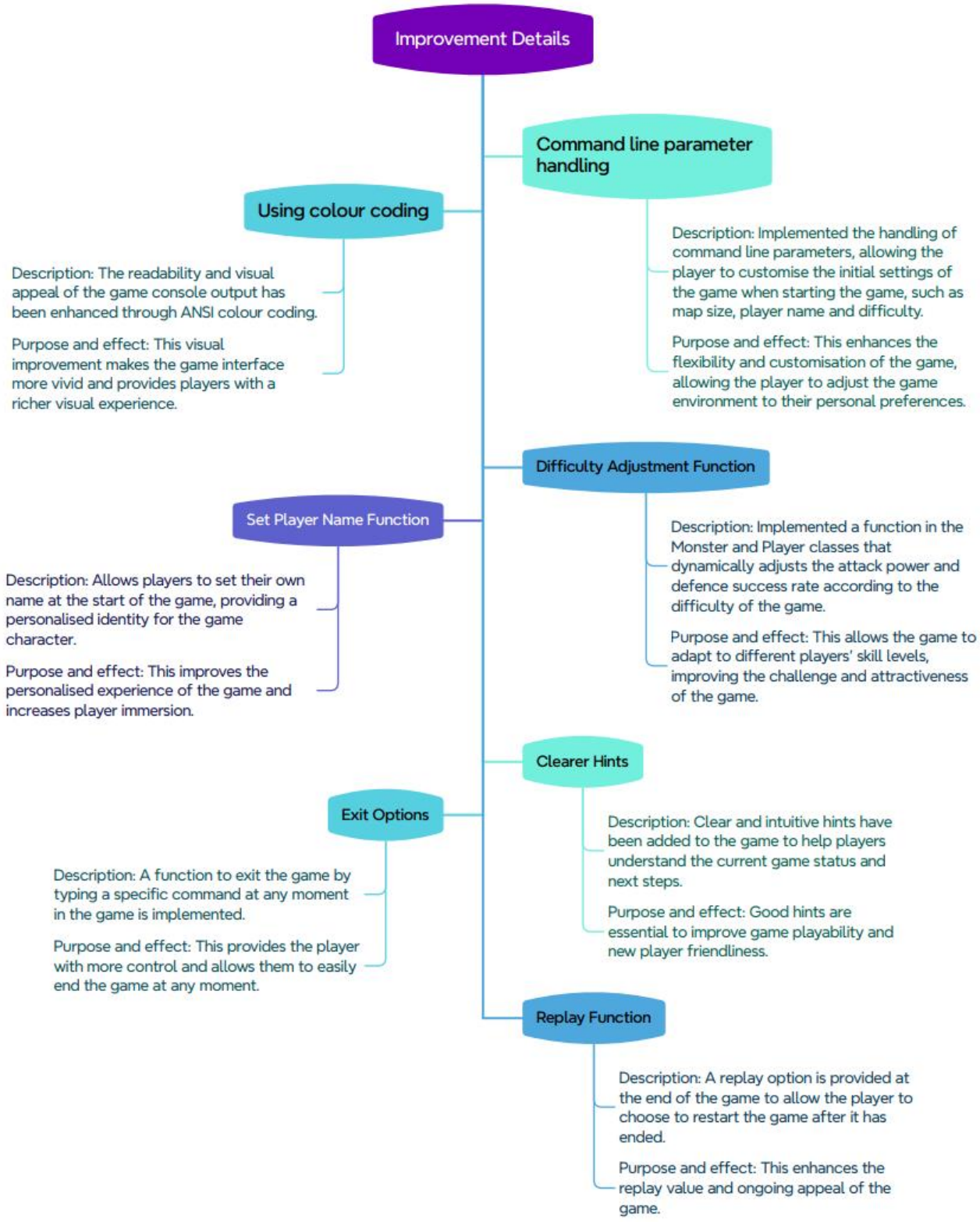


2.4 Game Lifecycle

See **Appendix 5: Sequence Diagram of the 【Grid Adventure】 Game**

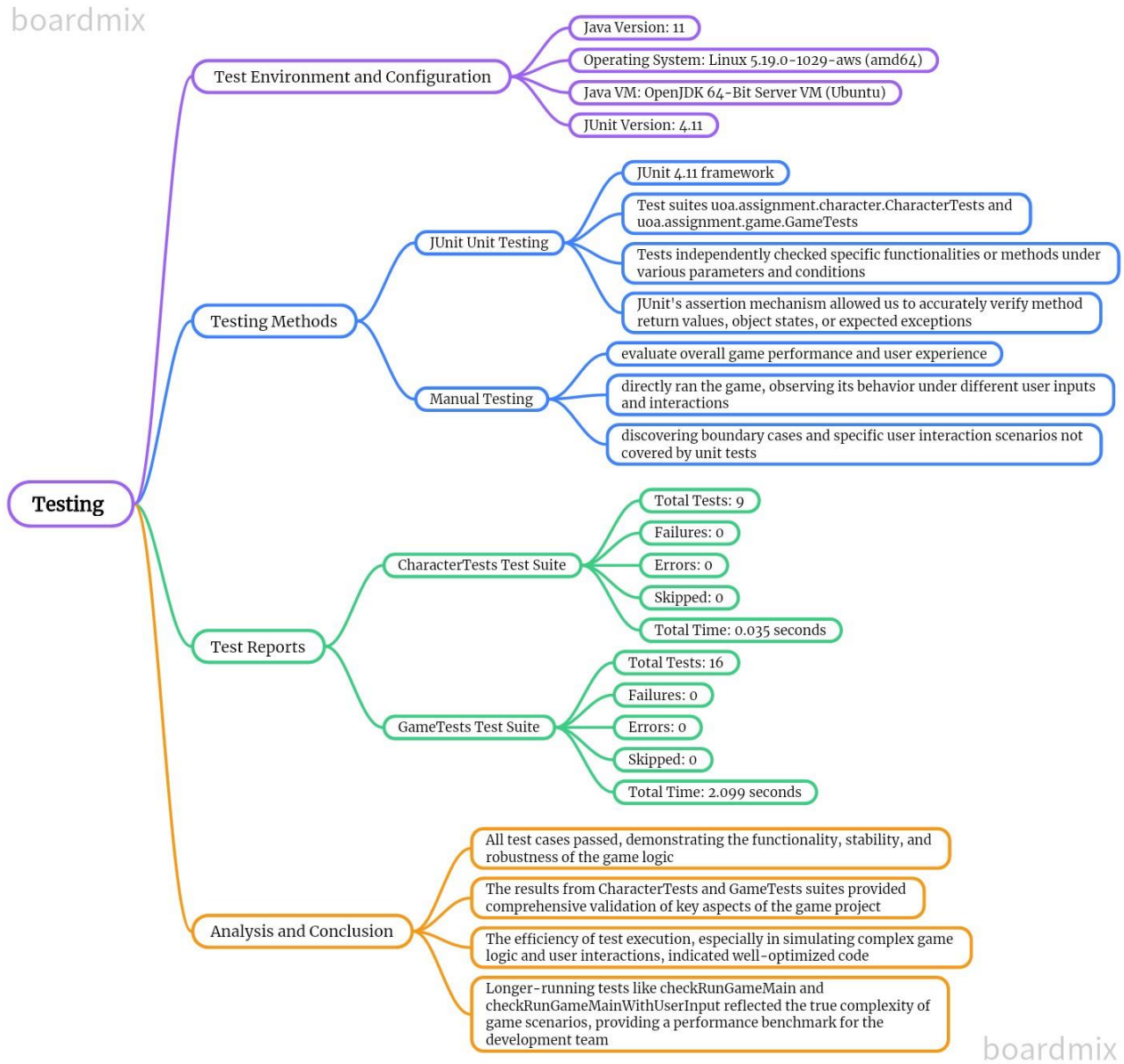
2.5 Program Improvements

In this Java programming course project, I designed and implemented a text-based 2D grid game where players interact with multiple monsters. Beyond meeting the basic course requirements, I made several key improvements to the game to enhance the gaming experience and interactivity, as illustrated below:


Improvement Details
Command line parameter handling
Description: Implemented the handling of command line parameters, allowing the player to customise the initial settings of the game when starting the game, such as map size, player name and difficulty.
Purpose and effect: This enhances the flexibility and customisation of the game, allowing the player to adjust the game environment to their personal preferences.
Using colour coding
Description: The readability and visual appeal of the game console output has been enhanced through ANSI colour coding.
Purpose and effect: This visual improvement makes the game interface more vivid and provides players with a richer visual experience.
Difficulty Adjustment Function
Description: Implemented a function in the Monster and Player classes that dynamically adjusts the attack power and defence success rate according to the difficulty of the game.
Purpose and effect: This allows the game to adapt to different players' skill levels, improving the challenge and attractiveness of the game.
Set Player Name Function
Description: Allows players to set their own name at the start of the game, providing a personalised identity for the game character.
Purpose and effect: This improves the personalised experience of the game and increases player immersion.
Clearer Hints
Description: Clear and intuitive hints have been added to the game to help players understand the current game status and next steps.
Purpose and effect: Good hints are essential to improve game playability and new player friendliness.
Exit Options
Description: A function to exit the game by typing a specific command at any moment in the game is implemented.
Purpose and effect: This provides the player with more control and allows them to easily end the game at any moment.
Replay Function
Description: A replay option is provided at the end of the game to allow the player to choose to restart the game after it has ended.
Purpose and effect: This enhances the replay value and ongoing appeal of the game.

3 Testing

Through unit testing and manual testing, the program passed all tests and successfully **completed all the tasks.**



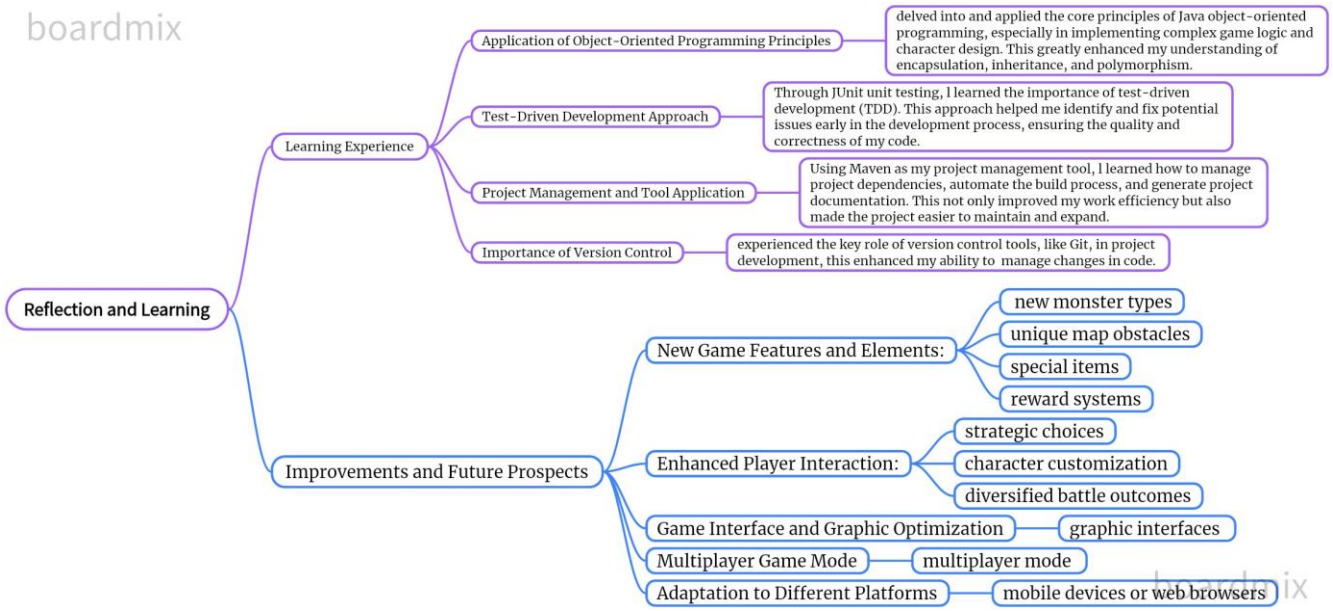
The unit result is as shown in the following picture:

```
[INFO] Running uoa.assignment.game.GameTests
[INFO] Tests run: 16, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 2.099 s - in uoa.assignment.game.GameTests
[INFO] Running uoa.assignment.character.CharacterTests
[INFO] Tests run: 9, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.035 s - in uoa.assignment.character.CharacterTests
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 25, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  4.481 s
```

4 Challenges Faced and Their Solutions

Challenge	Solution	Description
Complex game logic	Modularity and encapsulation	Tackle complex game logic through a modular approach. Encapsulating different functions (e.g. character movement, attack logic, map rendering) in different classes and methods makes the code more manageable and simplifies the testing and debugging process.
Code readability and maintainability	Code Refactoring and Continuous Integration	To improve the readability and maintainability of the code, a strategy of code refactoring was adopted. This included removing redundant code, optimising the design of classes and methods, and providing clear comments and documentation. In addition, code stability and quality were ensured by implementing continuous integration and automated testing (e.g. JUnit testing) so that new changes did not break existing functionality.

5 Reflection and Learning



Appendices

Appendix 1

Introduce:

The currently supported formats:

1. `height width`
2. `height width playername`
3. `height width playername difficulty`
4. `height,width`
5. `height,width,playername`
6. `height,width,playername,difficulty`

```
PS D:\game2\game2> java -jar target/assignment-1.0-SNAPSHOT.jar 3 3
Welcome to the Game, Player!
You choose the difficulty level:1
Are you ready to challenge the monsters?
```
(1)

```
PS D:\game2\game2> java -jar target/assignment-1.0-SNAPSHOT.jar 3 3 zs
Welcome to the Game, zs!
You choose the difficulty level:1
Are you ready to challenge the monsters?
```
(2)

```
PS D:\game2\game2> java -jar target/assignment-1.0-SNAPSHOT.jar 3 3 zs 3
Welcome to the Game, zs!
You choose the difficulty level:3
Are you ready to challenge the monsters?
```
(3)

```
PS D:\game2\game2> java -jar target/assignment-1.0-SNAPSHOT.jar 3,3
Welcome to the Game, Player!
You choose the difficulty level:1
Are you ready to challenge the monsters?
```
(4)

```
PS D:\game2\game2> java -jar target/assignment-1.0-SNAPSHOT.jar 3,3,zs
Welcome to the Game, zs!
You choose the difficulty level:1
Are you ready to challenge the monsters?
```
(5)

```
PS D:\game2\game2> java -jar target/assignment-1.0-SNAPSHOT.jar 3,3,zs,3
Welcome to the Game, zs!
You choose the difficulty level:3
Are you ready to challenge the monsters?
```
(6)

Appendix 2

Introduce: A demonstration of the game (one round)

```
PS D:\game2\game2> java -jar target/assignment-1.0-SNAPSHOT.jar 3,3,zs,3
Welcome to the Game, zs!
You choose the difficulty level:3
Are you ready to challenge the monsters?
Current game map:
% . %
. . .
% . *

Round 1
up
Starting interaction...

zs is moving up
Monster1 is moving down
!!HIT!! Monster1 successfully attacked zs
Monster2 is moving left
You can't go left. You lose a move.
Monster3 is moving down

Current Health Status:
Health zs: 65
Health Monster1: 100
Health Monster2: 100
Health Monster3: 100


Current game map:
. . %
% . *
% . .
```

Appendix 3

Introduce: Asking if the player wants to replay/ Exiting the game (input “0”)

```
Current Health Status:
Health zs: 0
Health Monster1: 0
Health Monster2: 75
Health Monster3: 100

YOU HAVE DIED!

Current game map:
. % x
. % *
. . .
Game over! Do you want to play again? (yes/no): yes
Please choose the difficulty (1-4): 2
Welcome to the Game, zs!
You choose the difficulty level:2
Are you ready to challenge the monsters?
Current game map:
% . %
. . .
% . *

Round 1
up
Starting interaction...

zs is moving up
Monster1 is moving right
You can't go right. You lose a move.
Monster2 is moving right
```

（replay）

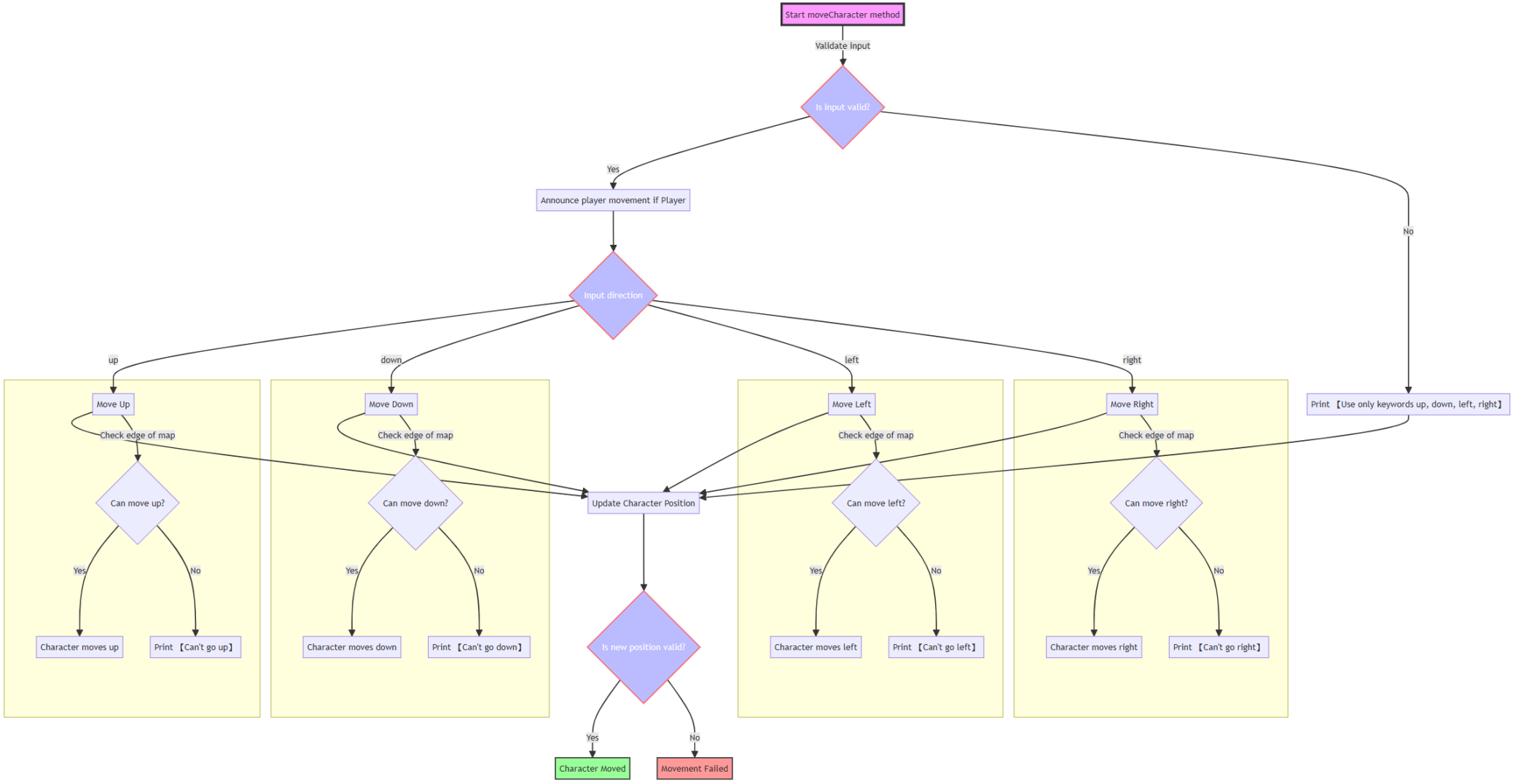
```
Current Health Status:
Health zs: 60
Health Monster1: 0
Health Monster2: 0
Health Monster3: 0

YOU HAVE WON!

Current game map:
. . x
. . x
. * x
Game over! Do you want to play again? (yes/no): no
Thanks for playing!
```

（not replay）

```
Round 3
0
Exiting the game.
Thanks for playing!
```

(Exiting the game)

Appendix 4: The moveCharacter method in GameLogic



Appendix 5: Sequence Diagram of the 【Grid Adventure】 Game