Report

Project overview and purpose

The project aims to develop a simple role-playing minigame where players can move around the map and interact with monsters. The game includes two different types of characters, player characters and monster characters, as well as components such as map layout and game logic. The purpose is to demonstrate the basic concepts and implementation of object-oriented programming.

Development environment

Programming language: Java
Development tool: Codio
External library: None

Main classes and variables and their relationships

Projects mainly include the following categories:
1.GameCharacter: Defines the basic attributes and behaviors of a game character, such as name, health, location, etc. Abstract methods for damage and defense are specified, as well as methods for obtaining names and health points.
2.Player: Inherited from the GameCharacter, representing the player character. The concrete methods of injury and defense are realized, as well as the random logic of successful defense.
3.Monster: Inherited from GameCharacter, monster character. Similar to the Player, specific methods of injury and defense are implemented, as well as the logic of deciding to move.
4.Map: Represents the game map, including the map layout and character array. It provides methods for initializing maps and printing map layouts.
5.Game: Game class, responsible for the game initialization and turn processing logic.
6.GameLogic: A game logic class that deals with character movement and interaction logic.
7.RunGame: The game running class, which is responsible for running the game's rounds.

Here are some of their relationships:

1.The Game class contains a Map object and calls the methods of the GameLogic class to process the game logic.
2.The Map class contains a map layout and an array of roles. You can print the map layout to the console by calling the printLayout() method.
3.The GameCharacter class is an abstract class for characters. The Player and Monster classes inherit from the GameCharacter class and implement their own methods of damage and defense.
4.The GameLogic class contains methods to move the character, calling the corresponding move method according to the input command to update the character's location and map layout.

5.The RunGame class is the entry to the entire Game and is responsible for creating game objects and continuing the game's rounds until the game is over.

Core method function

1.moveCharacter(String input, Map gameMap, GameCharacter character):
Description: This method is used to move the position of the game character on the map.
input parameter: input: Indicates the movement direction entered by the player, which can be one of "up", "down", "left", or "right".
gameMap: Indicates the current game map object.
character: represents the game character object to be moved.
Function: Based on the movement direction entered by the player, determine whether the character can move to the target position.
If it can be moved, the map layout is updated to move the character to the target location.
If there are monsters in the target location, attack the monsters and update the map layout.
If the target location is obstructed or out of map range, the player is told that they cannot move.
2.hurtCharacter(GameCharacter character):
Description: This method is used to attack characters and reduce the target character's health.
Input parameter: character: indicates the target role object to be attacked.
Function: Determine whether the attack is successful, and reduce the target character's health according to the success or not.
If the target character's health drops to 0 or below, the target character is deemed dead.
3.successfulDefense():
Description: This method is used to determine whether the player's defense is successful.
Function: According to a certain probability, judge whether the player's defense is successful.
Return true if defense succeeds. Otherwise return false.
4.nextRound(String input):
Description: This method is used for logical processing of the next round of the game.
input parameter: input: Indicates the movement direction entered by the player, which can be one of "up", "down", "left", or "right".
Return value: Boolean value indicating whether the game is over.
Function: First print the current turn count to the console.
Based on the movement direction entered by the player, the moveCharacter() method is called to move the player character and update the map layout.
Loop through all monster characters, make monster moves if the monster character's health is greater than 0, and determine whether to fight the player character.
Print all characters' health to the console.

Condition to determine whether the game is over: If all monster characters' health drops to 0 or below, the player wins, prints a message and returns true.
If the player character's health drops to 0 or below, the player fails, prints a message and returns true.
If the game is not over, return false.

Test method

The test code uses the JUnit framework to write the unit tests. In each test case, a scenario is created and the related functions are called to perform the test. The test covers everything from character movements, health changes, the end of the game, etc. Verify that the program performs as expected by examining program output and object properties.
The test code also simulates user input and the game loop to ensure that the program handles user input correctly and runs properly in the game loop. In addition, randomness in the game was tested, such as whether the monster's defense behaved randomly, and the randomness of the monster's movement.
In general, testing covers key parts of the program, including character behavior, game flow, and randomness, to ensure that the program is functioning properly.

Difficulties and challenges encountered in the development process and solutions:

1. In the process of writing code, I do not understand the code logic, and some methods do not know how to write; Solution: Summarize experience by asking teachers and classmates, and then write code after understanding.
2. In this program, there are multiple classes that need to cooperate with each other, and the relationship between classes is not well understood when the code is originally written; The solution is to use object-oriented programming principles, ensure that each class has a clear responsibility, and use interfaces or abstract classes to define the relationships between them.
3. There are many abnormal situations in the test of the program, and there are often cases when the test is not passed after writing the code test, such as the number of groups out of bounds, null pointer reference, etc.; The solution is to set breakpoints in the program and then debug to find errors in the code or use try-catch statements to catch and handle exceptions.
4. In the program, there are many repeated code segments, such as a large number of similar codes in moveUp(), moveDown(), moveLeft() and moveRight() methods; Workaround: Encapsulate duplicate code segments into separate methods and call them when needed. This can reduce the amount of code and improve the maintainability and extensibility of the code

Sum up

The above is a technical report on the project, describing the project's purpose,

development environment, implementation details, testing methods, challenges, and solutions. Through this project, I have deepened my understanding of object-oriented programming, improved my coding ability and problem solving ability.