Report

Project Overview: This project is a Java based small game, the game includes players and monsters two kinds of roles. The player and the monster have basic behaviors such as movement, attack and defense, respectively. The victory condition of the game is to destroy all monsters, and the failure condition is that the player's health is reduced to zero.

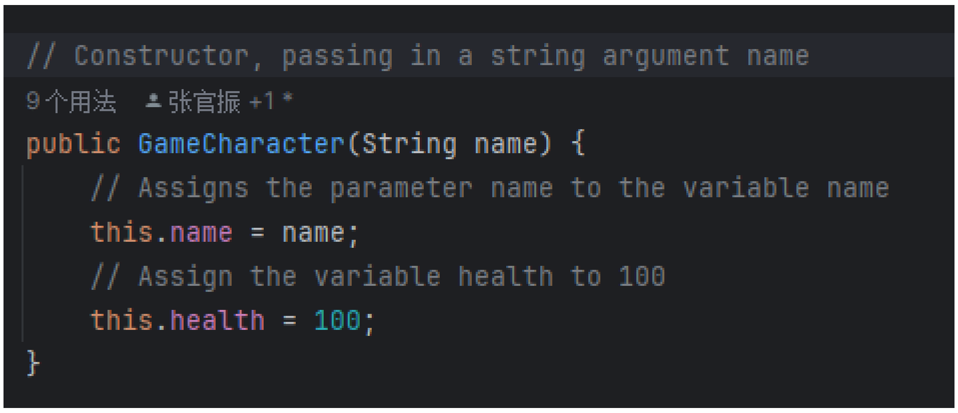
The basic behavior of the player and the monster is as follows:
Movement: Players and monsters can move in four directions on the map: up, down, left and right.
Attacks: Players and monsters can attack enemy characters, and a successful attack will cause the enemy character to lose a certain amount of health.
Defense: Players and monsters have a chance to defend themselves when attacked, and successful defense mitigates or avoids damage.

The specific process of the game is as follows:
1. At the beginning of the game, the player and the monster each start with 100 health and are placed in the corner of the map.
2. Each turn, the player can make one move, and all surviving monsters will also make one move. The result of the move will be displayed on the console.
3. Player and monster health changes will be displayed on the console at the end of each turn.
4. When all monster health drops to zero, the player wins, the game ends and "YOU HAVE WON!" is displayed on the console. .
5. If the player's health drops to zero, the player loses and the game ends with "YOU HAVE DIED!" displayed on the console. .

The entire game unfolds on the console, and players can enter movement commands according to the prompts, destroy monsters and protect their health through attack and defense. The game has a certain amount of randomness and strategy, and players need to act carefully to achieve victory.

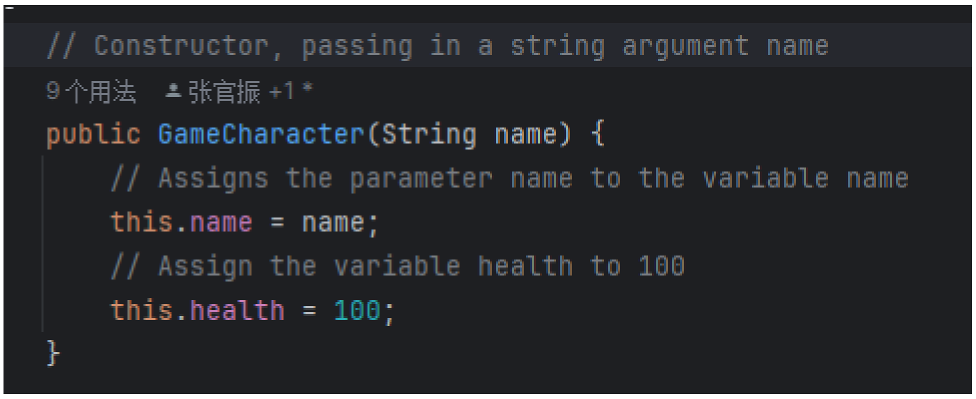
Here's how I implemented the game code:

Task 1: Implement basic methods for roles
1.1)
In the constructor of the GameCharacter class, the name field is initialized with the passed name argument.



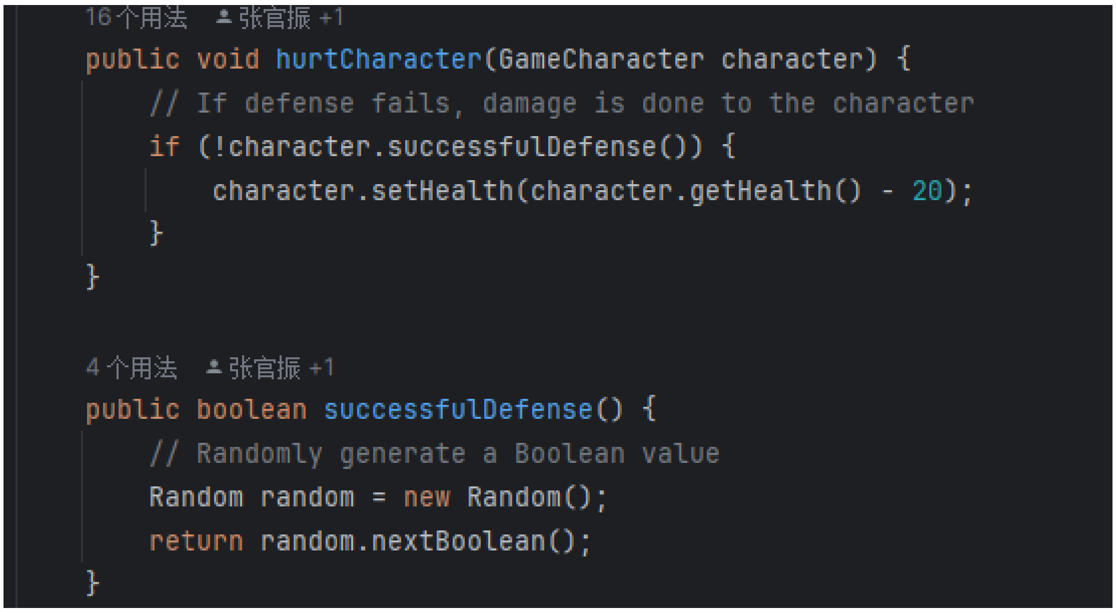
```
// Constructor, passing in a string argument name
9个用法  张官振 +1 *
public GameCharacter(String name) {
    // Assigns the parameter name to the variable name
    this.name = name;
    // Assign the variable health to 100
    this.health = 100;
}
```

1.2)
Implement the sayName() method of the GameCharacter class to return the character's name string.
1.3)
Modify the GameCharacter class so that the private integer variable health is initialized to the value 100 by default when an instance is created, and then complete the setter and getter methods for health.



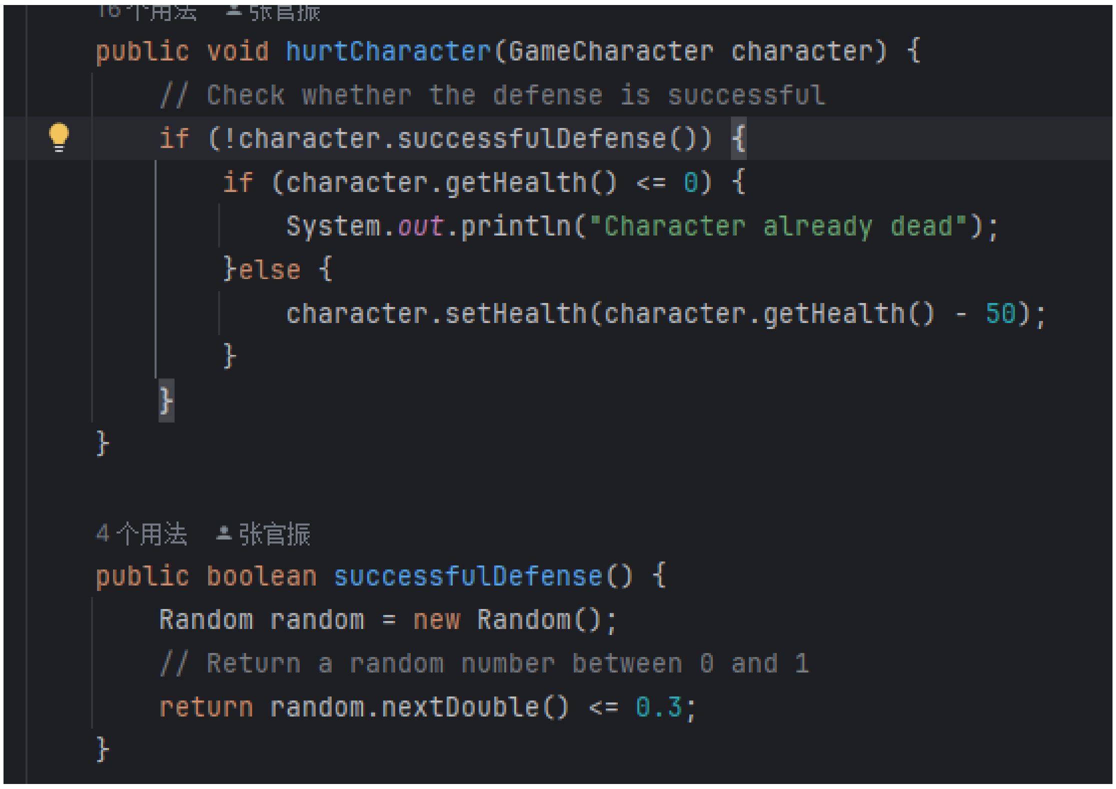
```
// Constructor, passing in a string argument name
9个用法  张官振 +1 *
public GameCharacter(String name) {
    // Assigns the parameter name to the variable name
    this.name = name;
    // Assign the variable health to 100
    this.health = 100;
}
```

1.4)
In the Monster class, the hurtCharacter(GameCharacter character) and successfulDefense() methods are implemented based on the abstract methods in the GameCharacter class to handle the logic of the character's injury and successful defense.



```
16个用法  ≗张官振 +1
public void hurtCharacter(GameCharacter character) {
    // If defense fails, damage is done to the character
    if (!character.successfulDefense()) {
        character.setHealth(character.getHealth() - 20);
    }
}

4个用法  ≗张官振 +1
public boolean successfulDefense() {
    // Randomly generate a Boolean value
    Random random = new Random();
    return random.nextBoolean();
}
```

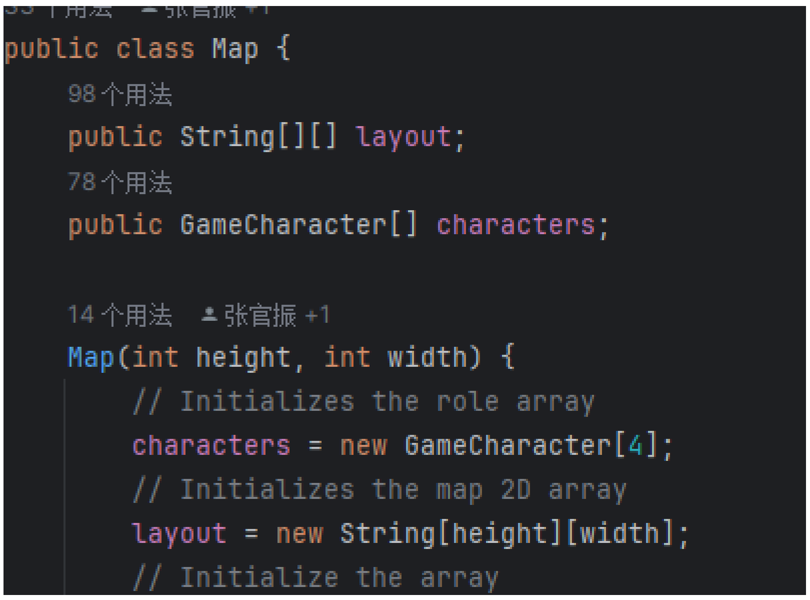
1.5)
In the Player class, the hurtCharacter(GameCharacter character) and successfulDefense() methods are implemented based on the abstract methods in the GameCharacter class to handle the logic of the character's injury and successful defense.



```
16个用法  ≗张官振
public void hurtCharacter(GameCharacter character) {
    // Check whether the defense is successful
    if (!character.successfulDefense()) {
        if (character.getHealth() <= 0) {
            System.out.println("Character already dead");
        }else {
            character.setHealth(character.getHealth() - 50);
        }
    }
}

4个用法  ≗张官振
public boolean successfulDefense() {
    Random random = new Random();
    // Return a random number between 0 and 1
    return random.nextDouble() <= 0.3;
}
```

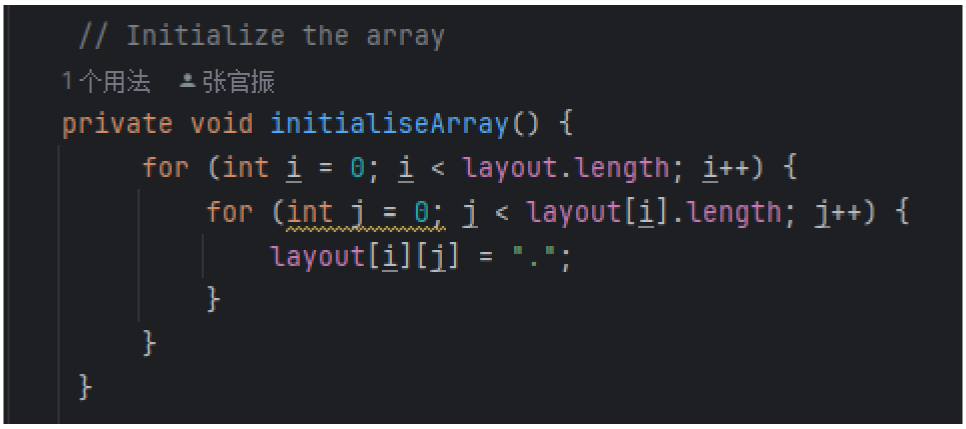
Task 2: Implement the game map
2.1)
Modify the characters variable of the Map class to accommodate four GameCharacter

objects. We also modify the constructor of the Map class to initialize a 2D array layout based on the input parameters (the height and width of the map).



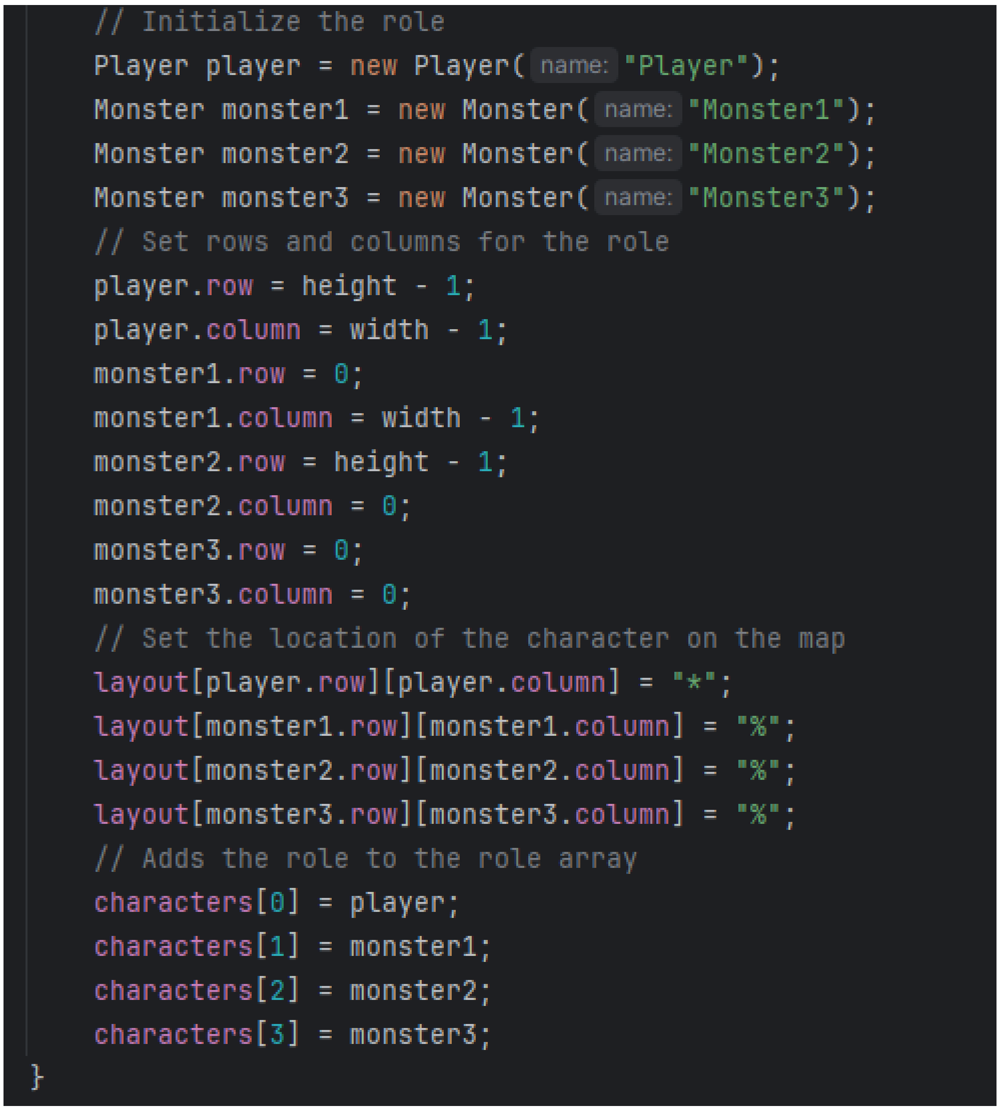
```
33 个用法  张官振 +1
public class Map {
    98个用法
    public String[][] layout;
    78个用法
    public GameCharacter[] characters;

    14个用法  张官振 +1
    Map(int height, int width) {
        // Initializes the role array
        characters = new GameCharacter[4];
        // Initializes the map 2D array
        layout = new String[height][width];
        // Initialize the array
```

2.2)
Create a private method initializeArray() in the Map class to populate the 2D array with the all-dot character '.' when creating an instance of the Map class. Also complete the printLayout() method, which prints the 2D layout array to the console.



```
    // Initialize the array
    1个用法  张官振
    private void initialiseArray() {
        for (int i = 0; i < layout.length; i++) {
            for (int j = 0; j < layout[i].length; j++) {
                layout[i][j] = ".";
            }
        }
    }
```

2.3)
Create a Player object and three Monster objects in the Map class and add them to the characters array based on their location information.
2.4)
Update the code so that monsters and players on the map are represented by symbols '%' in the layout array (monsters) and '' (players), and print the layout after initializing the map.

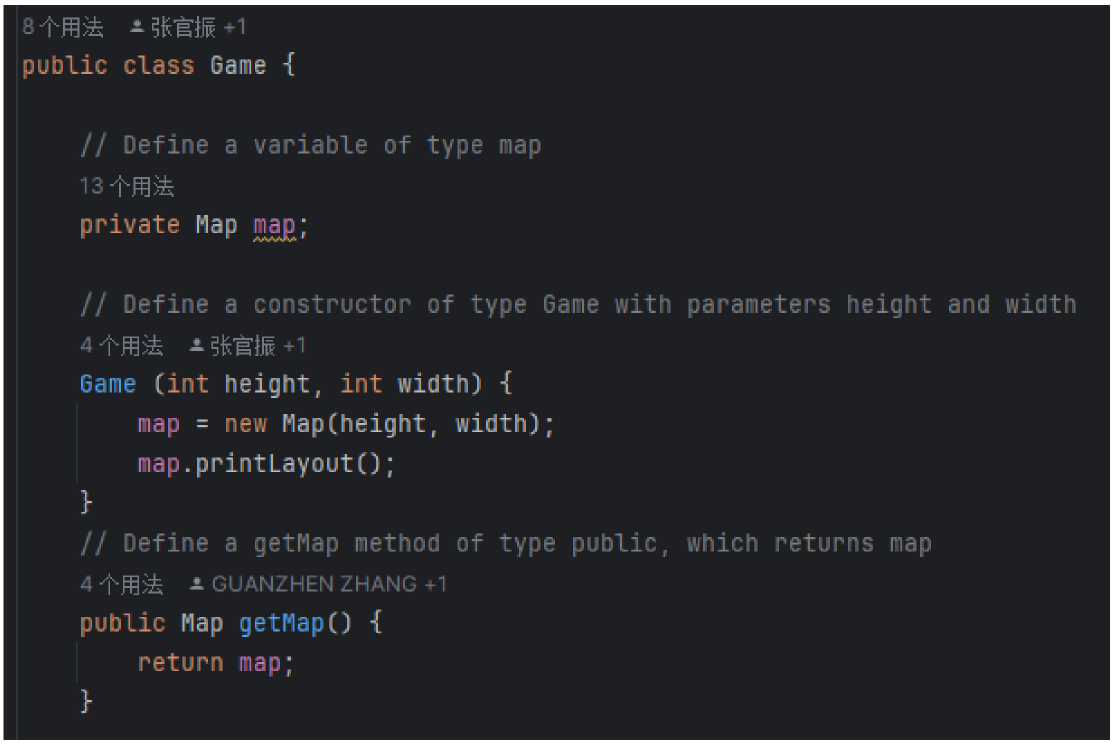


```
    // Initialize the role
    Player player = new Player( name: "Player");
    Monster monster1 = new Monster( name: "Monster1");
    Monster monster2 = new Monster( name: "Monster2");
    Monster monster3 = new Monster( name: "Monster3");
    // Set rows and columns for the role
    player.row = height - 1;
    player.column = width - 1;
    monster1.row = 0;
    monster1.column = width - 1;
    monster2.row = height - 1;
    monster2.column = 0;
    monster3.row = 0;
    monster3.column = 0;
    // Set the location of the character on the map
    layout[player.row][player.column] = "*";
    layout[monster1.row][monster1.column] = "%";
    layout[monster2.row][monster2.column] = "%";
    layout[monster3.row][monster3.column] = "%";
    // Adds the role to the role array
    characters[0] = player;
    characters[1] = monster1;
    characters[2] = monster2;
    characters[3] = monster3;
}
```

Task 3: Implement basic game features

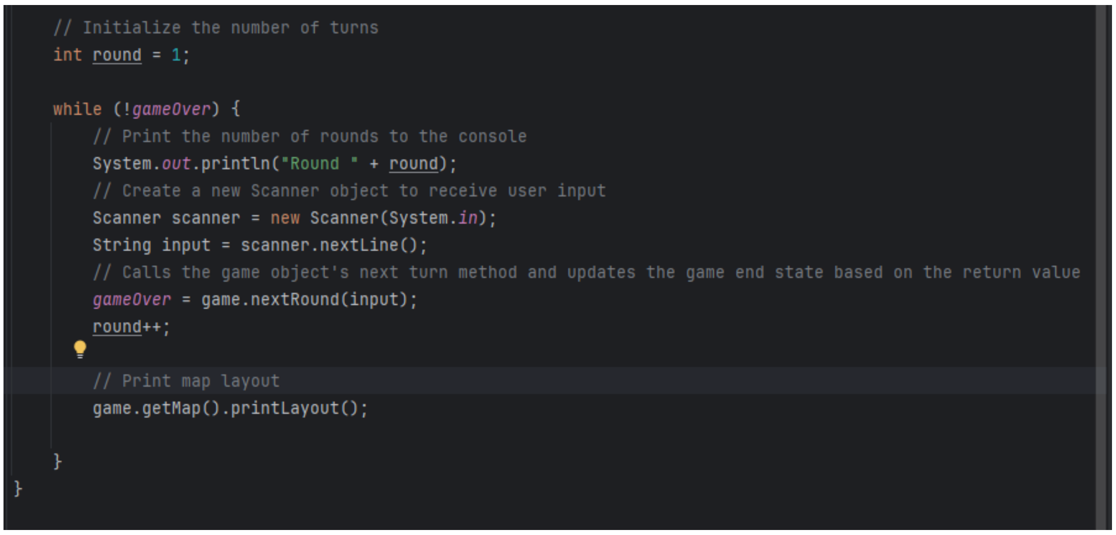
3.1)

Modify the constructor of the Game class, instantiate and initialize a Map object, and print the map layout on the console.



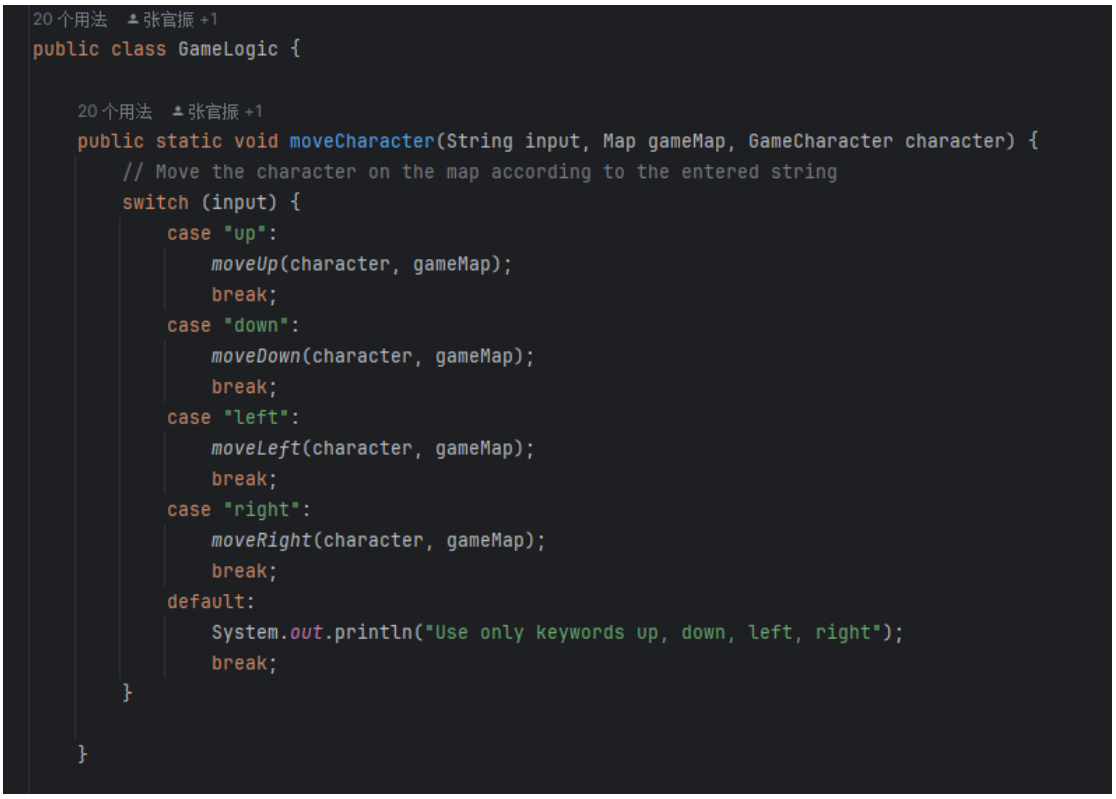
```java
8个用法  ≗张官振 +1
public class Game {

    // Define a variable of type map
    13个用法
    private Map map;

    // Define a constructor of type Game with parameters height and width
    4个用法  ≗张官振 +1
    Game (int height, int width) {
        map = new Map(height, width);
        map.printLayout();
    }
    // Define a getMap method of type public, which returns map
    4个用法  ≗GUANZHEN ZHANG +1
    public Map getMap() {
        return map;
    }
```

3.2)
Adds a loop to the RunGame's main() method that receives user console input and prints the number of turns at the beginning of each iteration.



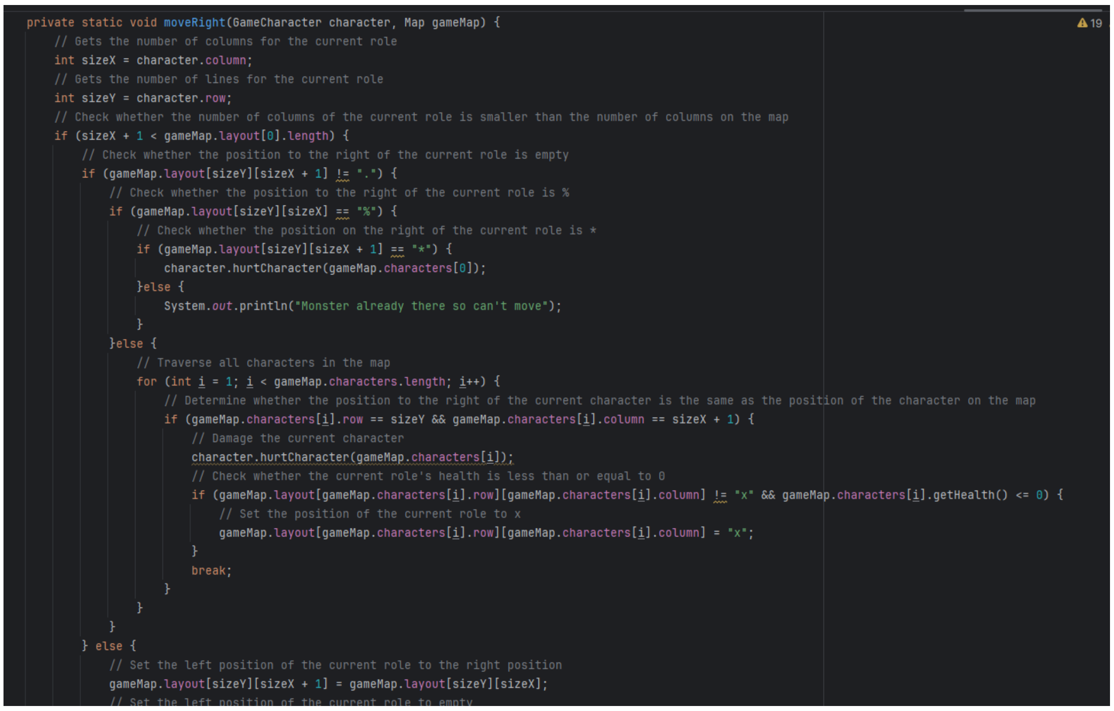
```java
    // Initialize the number of turns
    int round = 1;

    while (!gameOver) {
        // Print the number of rounds to the console
        System.out.println("Round " + round);
        // Create a new Scanner object to receive user input
        Scanner scanner = new Scanner(System.in);
        String input = scanner.nextLine();
        // Calls the game object's next turn method and updates the game end state based on the return value
        gameOver = game.nextRound(input);
        round++;

        // Print map layout
        game.getMap().printLayout();

    }
}
```

3.3)
Complete the moveCharacter(String input, Map gameMap, GameCharacter character) method in the GameLogic class to move the character based on user input.

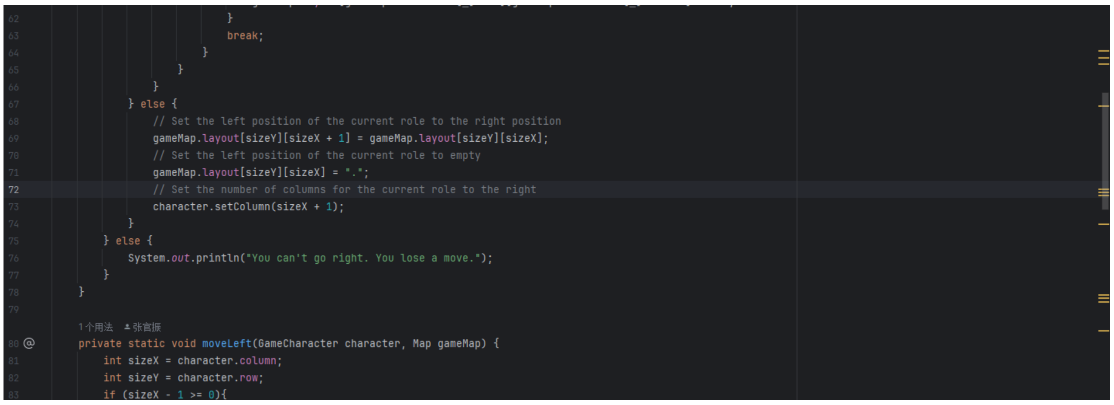


```
20 个用法   张宫振 +1
public class GameLogic {

    20 个用法   张宫振 +1
    public static void moveCharacter(String input, Map gameMap, GameCharacter character) {
        // Move the character on the map according to the entered string
        switch (input) {
            case "up":
                moveUp(character, gameMap);
                break;
            case "down":
                moveDown(character, gameMap);
                break;
            case "left":
                moveLeft(character, gameMap);
                break;
            case "right":
                moveRight(character, gameMap);
                break;
            default:
                System.out.println("Use only keywords up, down, left, right");
                break;
        }

    }
```

3.4)
Create moveRight(GameCharacter character, Map gameMap), moveLeft(GameCharacter character, Map gameMap), and Moveleft (Gamecharacter character, map Gamemap) in the GameLogic class. The moveUp(GameCharacter character, Map gameMap) and moveDown(GameCharacter character, Map gameMap) methods are used to move characters and update location information.



```
private static void moveRight(GameCharacter character, Map gameMap) {
    // Gets the number of columns for the current role
    int sizeX = character.column;
    // Gets the number of lines for the current role
    int sizeY = character.row;
    // Check whether the number of columns of the current role is smaller than the number of columns on the map
    if (sizeX + 1 < gameMap.layout[0].length) {
        // Check whether the position to the right of the current role is empty
        if (gameMap.layout[sizeY][sizeX + 1] != ".") {
            // Check whether the position to the right of the current role is %
            if (gameMap.layout[sizeY][sizeX] == "%") {
                // Check whether the position on the right of the current role is *
                if (gameMap.layout[sizeY][sizeX + 1] == "*") {
                    character.hurtCharacter(gameMap.characters[0]);
                }else {
                    System.out.println("Monster already there so can't move");
                }
            }else {
                // Traverse all characters in the map
                for (int i = 1; i < gameMap.characters.length; i++) {
                    // Determine whether the position to the right of the current character is the same as the position of the character on the map
                    if (gameMap.characters[i].row == sizeY && gameMap.characters[i].column == sizeX + 1) {
                        // Damage the current character
                        character.hurtCharacter(gameMap.characters[i]);
                        // Check whether the current role's health is less than or equal to 0
                        if (gameMap.layout[gameMap.characters[i].row][gameMap.characters[i].column] != "x" && gameMap.characters[i].getHealth() <= 0) {
                            // Set the position of the current role to x
                            gameMap.layout[gameMap.characters[i].row][gameMap.characters[i].column] = "x";
                        }
                        break;
                    }
                }
            }
        } else {
            // Set the left position of the current role to the right position
            gameMap.layout[sizeY][sizeX + 1] = gameMap.layout[sizeY][sizeX];
            // Set the left position of the current role to empty
```

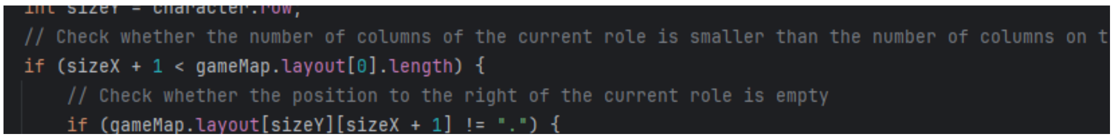


```java
                            }
                            break;
                        }
                    }
                }
            } else {
                // Set the left position of the current role to the right position
                gameMap.layout[sizeY][sizeX + 1] = gameMap.layout[sizeY][sizeX];
                // Set the left position of the current role to empty
                gameMap.layout[sizeY][sizeX] = ".";
                // Set the number of columns for the current role to the right
                character.setColumn(sizeX + 1);
            }
        } else {
            System.out.println("You can't go right. You lose a move.");
        }
    }

    1个用法  张官振
    private static void moveLeft(GameCharacter character, Map gameMap) {
        int sizeX = character.column;
        int sizeY = character.row;
        if (sizeX - 1 >= 0){
```

Task 4: Implement the game controls and basic logic

4.1)

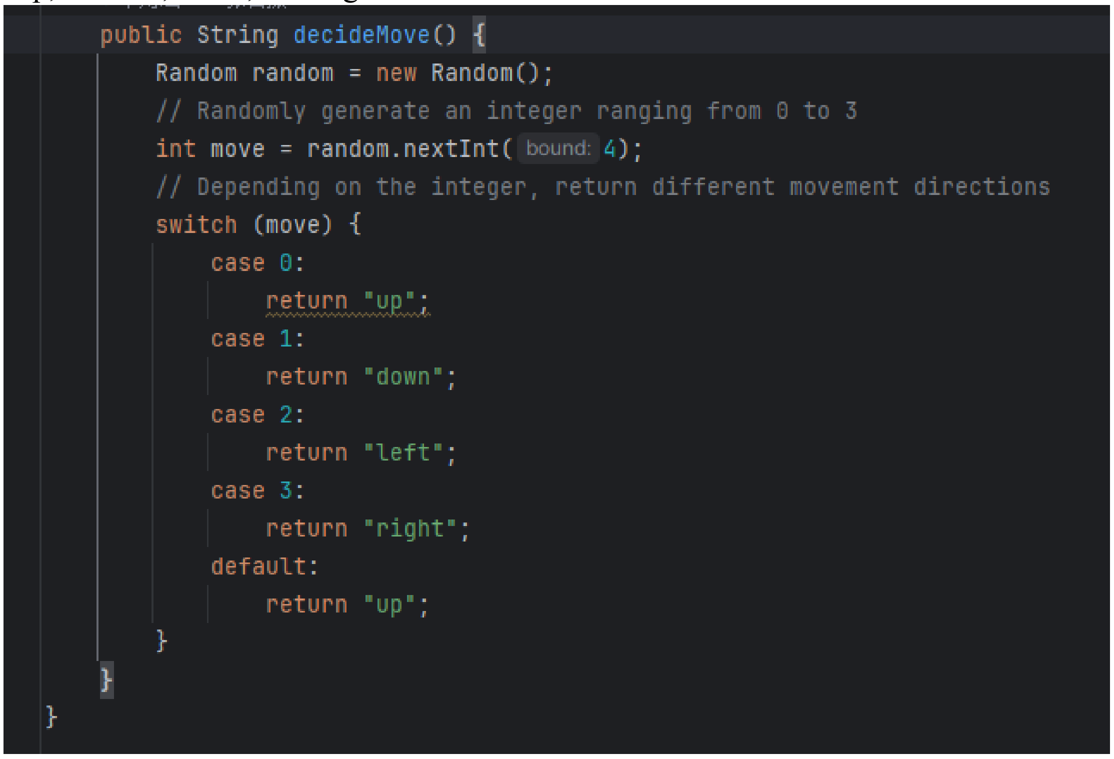
Update the implemented method so that the character cannot move to the wall, and print the corresponding information if it cannot be moved.



```java
int sizeY = character.row;
// Check whether the number of columns of the current role is smaller than the number of columns on t
if (sizeX + 1 < gameMap.layout[0].length) {
    // Check whether the position to the right of the current role is empty
    if (gameMap.layout[sizeY][sizeX + 1] != ".") {
```

4.2)

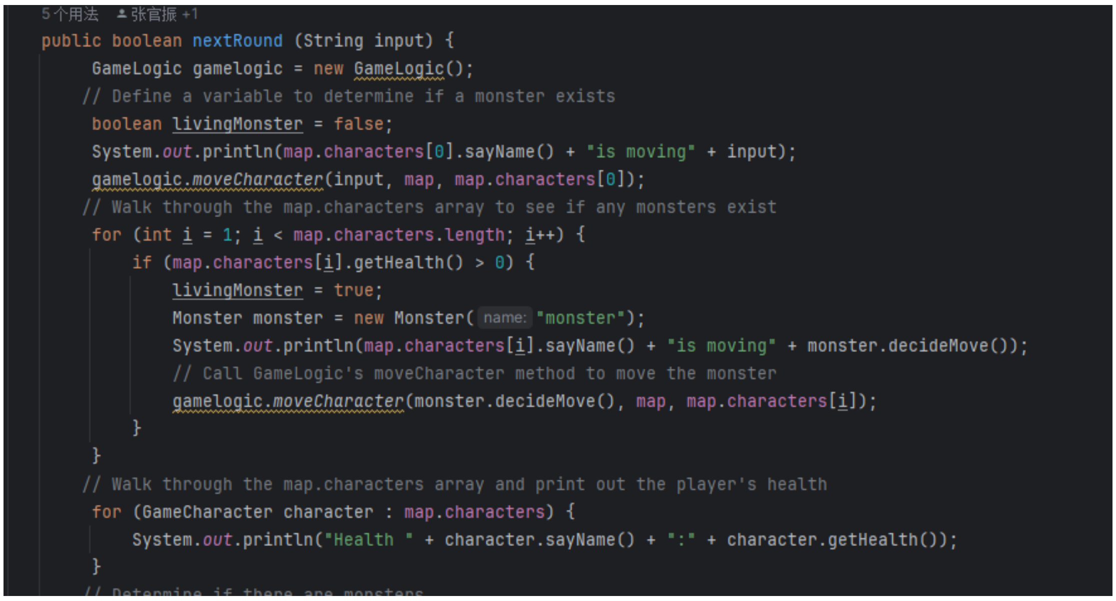
Complete the decideMove() method of the Monster class to return a random string of "up," "down," "left," or "right."



```java
    public String decideMove() {
        Random random = new Random();
        // Randomly generate an integer ranging from 0 to 3
        int move = random.nextInt( bound: 4);
        // Depending on the integer, return different movement directions
        switch (move) {
            case 0:
                return "up";
            case 1:
                return "down";
            case 2:
                return "left";
            case 3:
                return "right";
            default:
                return "up";
        }
    }
}
```

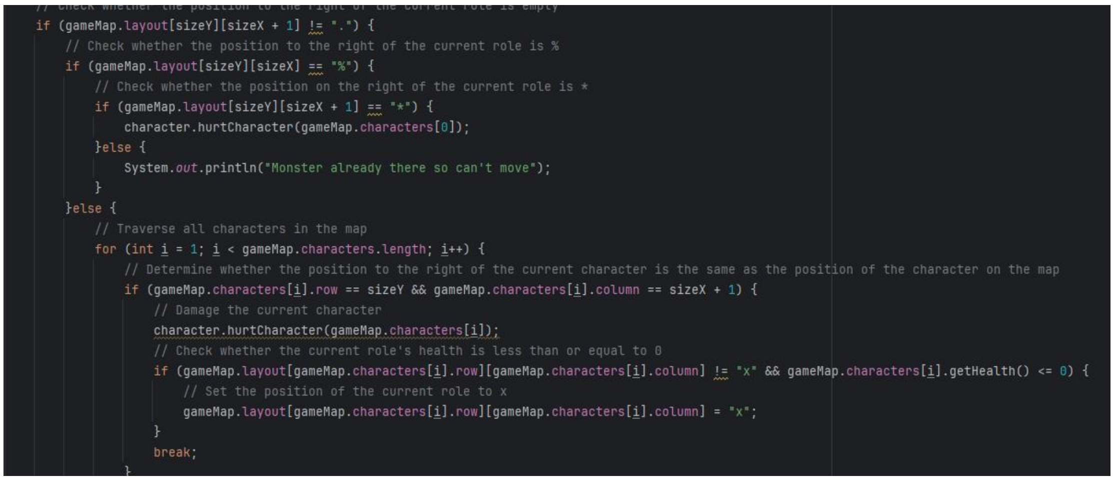
4.3)

Completing the nextRound(String input) method in the Game class causes the player to first move according to console input, and then all surviving monsters move automatically.



```
5个用法  张官振 +1
public boolean nextRound (String input) {
    GameLogic gamelogic = new GameLogic();
    // Define a variable to determine if a monster exists
    boolean livingMonster = false;
    System.out.println(map.characters[0].sayName() + "is moving" + input);
    gamelogic.moveCharacter(input, map, map.characters[0]);
    // Walk through the map.characters array to see if any monsters exist
    for (int i = 1; i < map.characters.length; i++) {
        if (map.characters[i].getHealth() > 0) {
            livingMonster = true;
            Monster monster = new Monster( name: "monster");
            System.out.println(map.characters[i].sayName() + "is moving" + monster.decideMove());
            // Call GameLogic's moveCharacter method to move the monster
            gamelogic.moveCharacter(monster.decideMove(), map, map.characters[i]);
        }
    }
    // Walk through the map.characters array and print out the player's health
    for (GameCharacter character : map.characters) {
        System.out.println("Health " + character.sayName() + ":" + character.getHealth());
    }
    // Determine if there are monsters
```

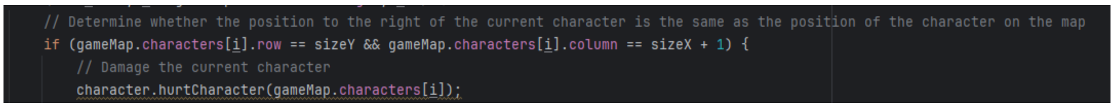
4.4)
Update the code, if the character tries to move to another character's location, print the corresponding information, and keep the character in its original location.



```
// Check whether the position to the right of the current role is empty
if (gameMap.layout[sizeY][sizeX + 1] != ".") {
    // Check whether the position to the right of the current role is %
    if (gameMap.layout[sizeY][sizeX] == "%") {
        // Check whether the position on the right of the current role is *
        if (gameMap.layout[sizeY][sizeX + 1] == "*") {
            character.hurtCharacter(gameMap.characters[0]);
        }else {
            System.out.println("Monster already there so can't move");
        }
    }else {
        // Traverse all characters in the map
        for (int i = 1; i < gameMap.characters.length; i++) {
            // Determine whether the position to the right of the current character is the same as the position of the character on the map
            if (gameMap.characters[i].row == sizeY && gameMap.characters[i].column == sizeX + 1) {
                // Damage the current character
                character.hurtCharacter(gameMap.characters[i]);
                // Check whether the current role's health is less than or equal to 0
                if (gameMap.layout[gameMap.characters[i].row][gameMap.characters[i].column] != "x" && gameMap.characters[i].getHealth() <= 0) {
                    // Set the position of the current role to x
                    gameMap.layout[gameMap.characters[i].row][gameMap.characters[i].column] = "x";
                }
                break;
            }
```

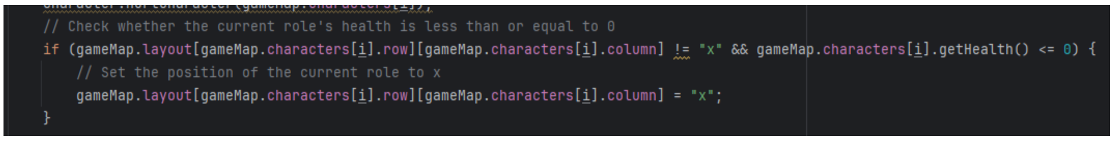
Task 5: Complete the rest of the game logic

5.1)
Update the code to allow the player character to attack when attempting to move to the monster's location, depending on the outcome of the monster's successfulDefense() method.
5.2)
Update the code so that monster characters attack when they try to move to the player's location, depending on the success of the attack as a result of the player's successfulDefense() method.



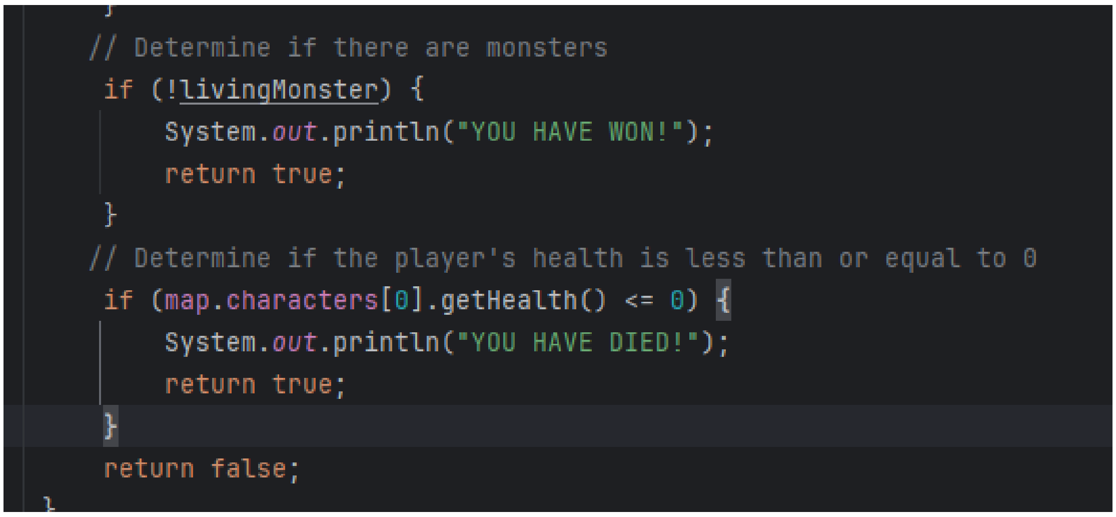
```java
// Determine whether the position to the right of the current character is the same as the position of the character on the map
if (gameMap.characters[i].row == sizeY && gameMap.characters[i].column == sizeX + 1) {
    // Damage the current character
    character.hurtCharacter(gameMap.characters[i]);
```

5.3)
Update the code to change the monster's position on the map to 'x' when the monster's health drops to zero, marking dead monsters.



```java
// Check whether the current role's health is less than or equal to 0
if (gameMap.layout[gameMap.characters[i].row][gameMap.characters[i].column] != "x" && gameMap.characters[i].getHealth() <= 0) {
    // Set the position of the current role to x
    gameMap.layout[gameMap.characters[i].row][gameMap.characters[i].column] = "x";
}
```

5.4)
Update the code to return true when there are no surviving monsters, and print the corresponding victory or defeat message on the console.



```java
    // Determine if there are monsters
    if (!livingMonster) {
        System.out.println("YOU HAVE WON!");
        return true;
    }
    // Determine if the player's health is less than or equal to 0
    if (map.characters[0].getHealth() <= 0) {
        System.out.println("YOU HAVE DIED!");
        return true;
    }
    return false;
}
```

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