Project Report: shake_game_java

1. Project Objective

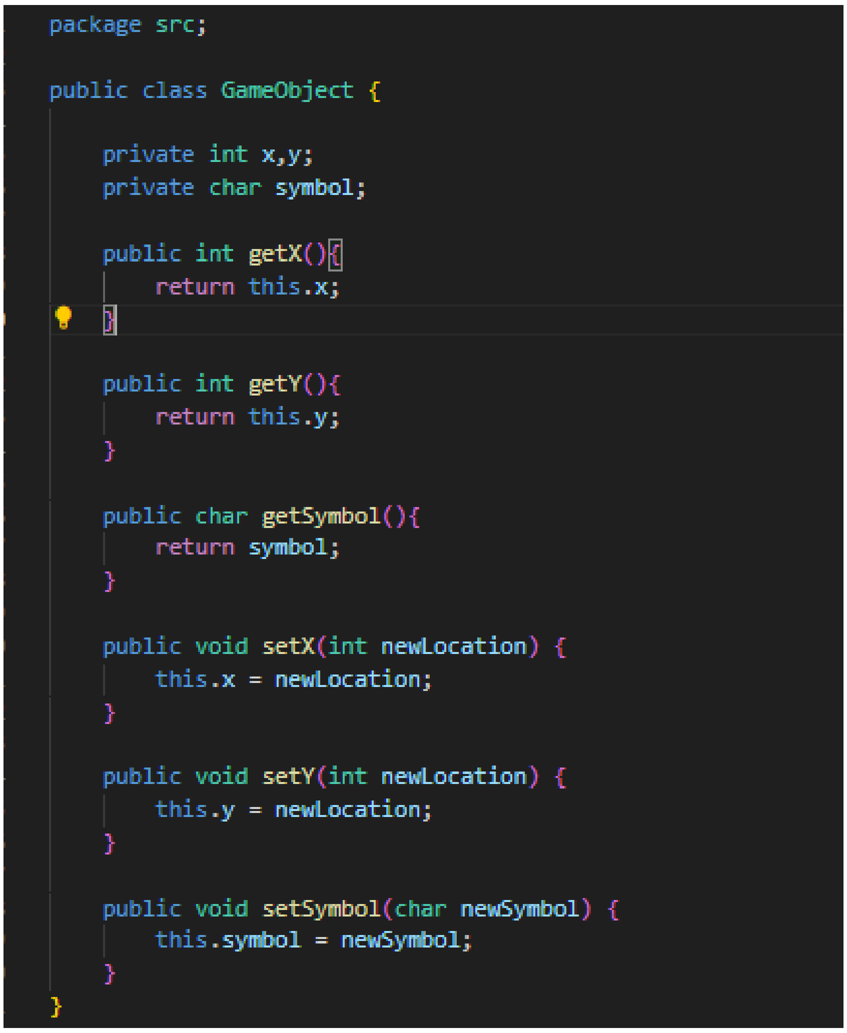
The objective of the project was to program a simple Snake game running in the console using the Java programming language and object-oriented programming. The player controls a snake that moves across the board and scores points by eating fruits that appear. The game ends when the snake hits a wall or the player chooses to quit.

2. Project Development Stages

- Designing the class structure: dividing the application into the following classes: GameObject, GameScreen, Snake, Fruit, Wall, Main.
- Creating the game board engine (GameScreen): implementing storage and display of objects on the board.
- Implementing game objects (Snake, Fruit, Wall): designing snake movement, fruit generation, and walls.
- Game control (Main): game loop, user input handling, game-over conditions.
- Testing and debugging: fixing errors, ensuring correct game behavior.

3. Description of Classes and Methods

3.1. GameObject Class



```java
package src;

public class GameObject {

    private int x,y;
    private char symbol;

    public int getX(){
        return this.x;
    }

    public int getY(){
        return this.y;
    }

    public char getSymbol(){
        return symbol;
    }

    public void setX(int newLocation) {
        this.x = newLocation;
    }

    public void setY(int newLocation) {
        this.y = newLocation;
    }

    public void setSymbol(char newSymbol) {
        this.symbol = newSymbol;
    }
}
```

Base class for all objects on the board.

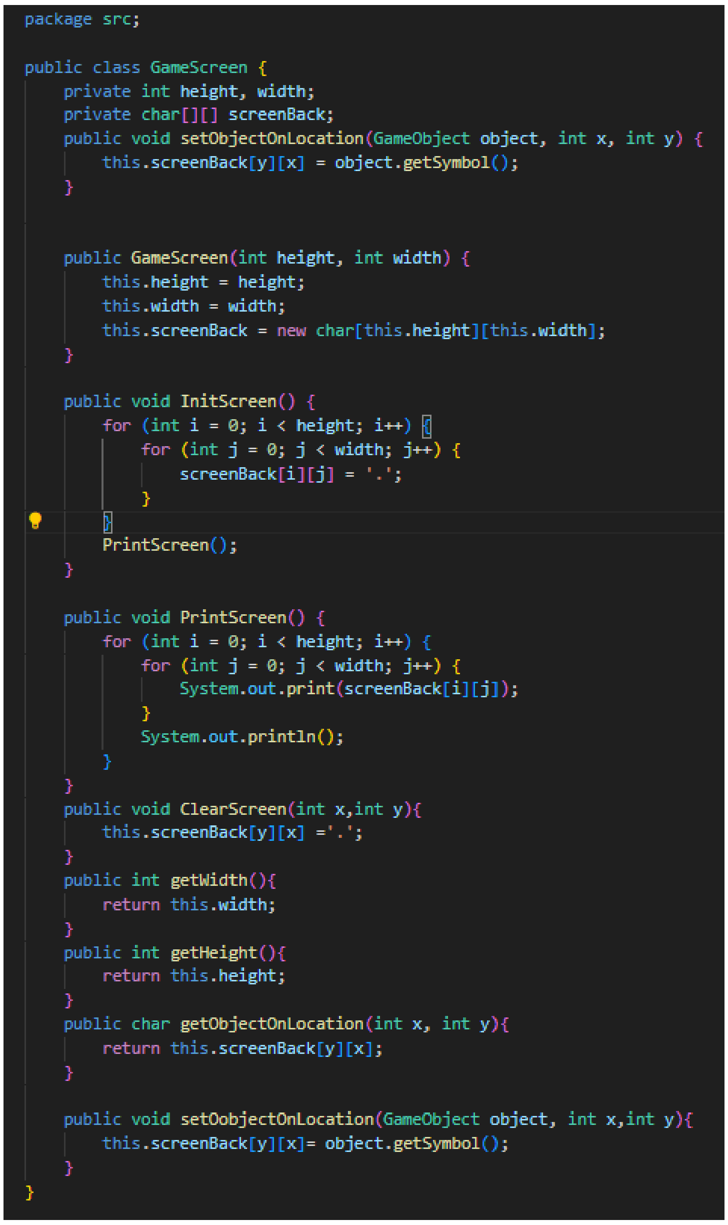
Fields:

- x, y: coordinates of the object
- symbol: character representing the object

Methods:

- getX, getY: return coordinates
- setX, setY: set new coordinates
- getSymbol: returns the object's symbol
- setSymbol: sets the symbol

3.2. GameScreen Class



```java
package src;

public class GameScreen {
    private int height, width;
    private char[][] screenBack;
    public void setObjectOnLocation(GameObject object, int x, int y) {
        this.screenBack[y][x] = object.getSymbol();
    }


    public GameScreen(int height, int width) {
        this.height = height;
        this.width = width;
        this.screenBack = new char[this.height][this.width];
    }

    public void InitScreen() {
        for (int i = 0; i < height; i++) {
            for (int j = 0; j < width; j++) {
                screenBack[i][j] = '.';
            }
        }
        PrintScreen();
    }

    public void PrintScreen() {
        for (int i = 0; i < height; i++) {
            for (int j = 0; j < width; j++) {
                System.out.print(screenBack[i][j]);
            }
            System.out.println();
        }
    }
    public void ClearScreen(int x,int y){
        this.screenBack[y][x] ='.';
    }
    public int getWidth(){
        return this.width;
    }
    public int getHeight(){
        return this.height;
    }
    public char getObjectOnLocation(int x, int y){
        return this.screenBack[y][x];
    }

    public void setOobjectOnLocation(GameObject object, int x,int y){
        this.screenBack[y][x]= object.getSymbol();
    }
}
```

Represents the game board as a character array.

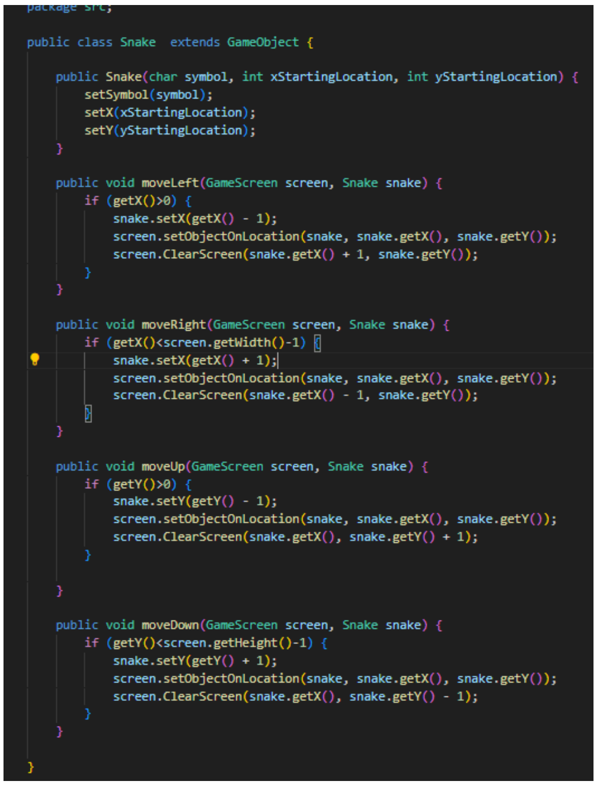
Fields:

- height, width: height and width of the board
- screenBack: two-dimensional array of characters

Methods:

- constructor: initializes the board
- InitScreen: fills the board with empty characters
- PrintScreen: displays the board in the console
- setObjectOnLocation: places an object on a given position
- ClearScreen: clears a field
- getObjectOnLocation: returns the symbol from a field
- getWidth, getHeight: return the board dimensions

3.3. Snake Class



```java
package src;

public class Snake  extends GameObject {

    public Snake(char symbol, int xStartingLocation, int yStartingLocation) {
        setSymbol(symbol);
        setX(xStartingLocation);
        setY(yStartingLocation);
    }

    public void moveLeft(GameScreen screen, Snake snake) {
        if (getX()>0) {
            snake.setX(getX() - 1);
            screen.setObjectOnLocation(snake, snake.getX(), snake.getY());
            screen.ClearScreen(snake.getX() + 1, snake.getY());
        }
    }

    public void moveRight(GameScreen screen, Snake snake) {
        if (getX()<screen.getWidth()-1) {
            snake.setX(getX() + 1);
            screen.setObjectOnLocation(snake, snake.getX(), snake.getY());
            screen.ClearScreen(snake.getX() - 1, snake.getY());
        }
    }

    public void moveUp(GameScreen screen, Snake snake) {
        if (getY()>0) {
            snake.setY(getY() - 1);
            screen.setObjectOnLocation(snake, snake.getX(), snake.getY());
            screen.ClearScreen(snake.getX(), snake.getY() + 1);
        }

    }

    public void moveDown(GameScreen screen, Snake snake) {
        if (getY()<screen.getHeight()-1) {
            snake.setY(getY() + 1);
            screen.setObjectOnLocation(snake, snake.getX(), snake.getY());
            screen.ClearScreen(snake.getX(), snake.getY() - 1);
        }
    }

}
```

Represents the player-controlled snake.

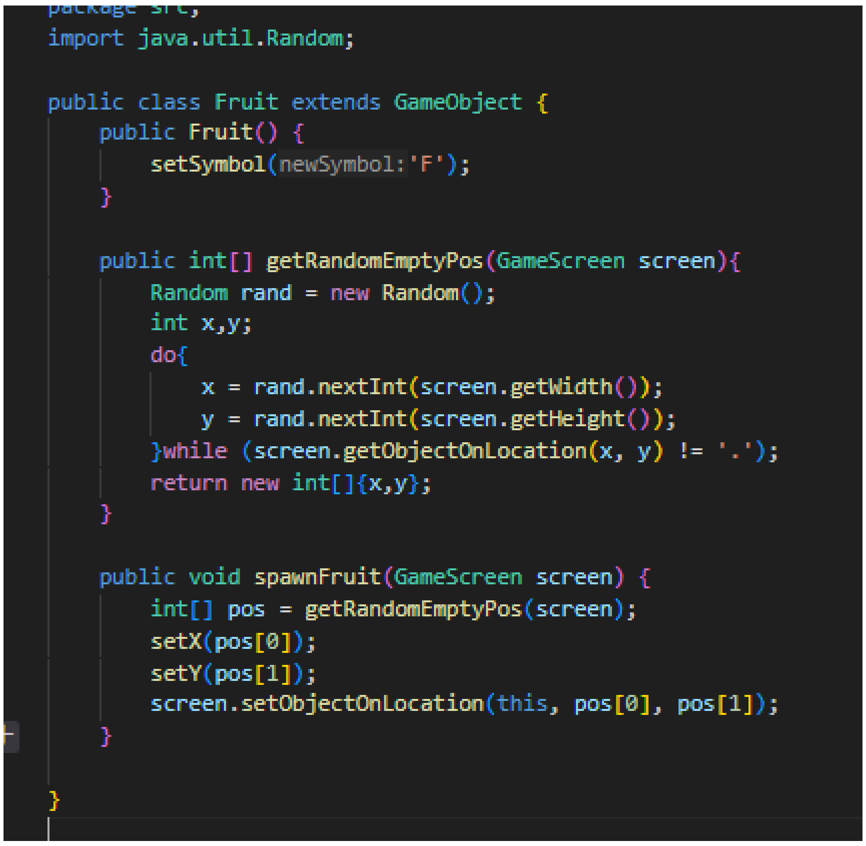
Constructor:

- sets the symbol and initial position

Methods:

- moveLeft, moveRight, moveUp, moveDown: change the snake's position, update the board, and clear the previous position

3.4. Fruit Class



```java
package src;
import java.util.Random;

public class Fruit extends GameObject {
    public Fruit() {
        setSymbol(newSymbol:'F');
    }

    public int[] getRandomEmptyPos(GameScreen screen){
        Random rand = new Random();
        int x,y;
        do{
            x = rand.nextInt(screen.getWidth());
            y = rand.nextInt(screen.getHeight());
        }while (screen.getObjectOnLocation(x, y) != '.');
        return new int[]{x,y};
    }

    public void spawnFruit(GameScreen screen) {
        int[] pos = getRandomEmptyPos(screen);
        setX(pos[0]);
        setY(pos[1]);
        screen.setObjectOnLocation(this, pos[0], pos[1]);
    }

}
```

Represents a fruit that the snake can eat.

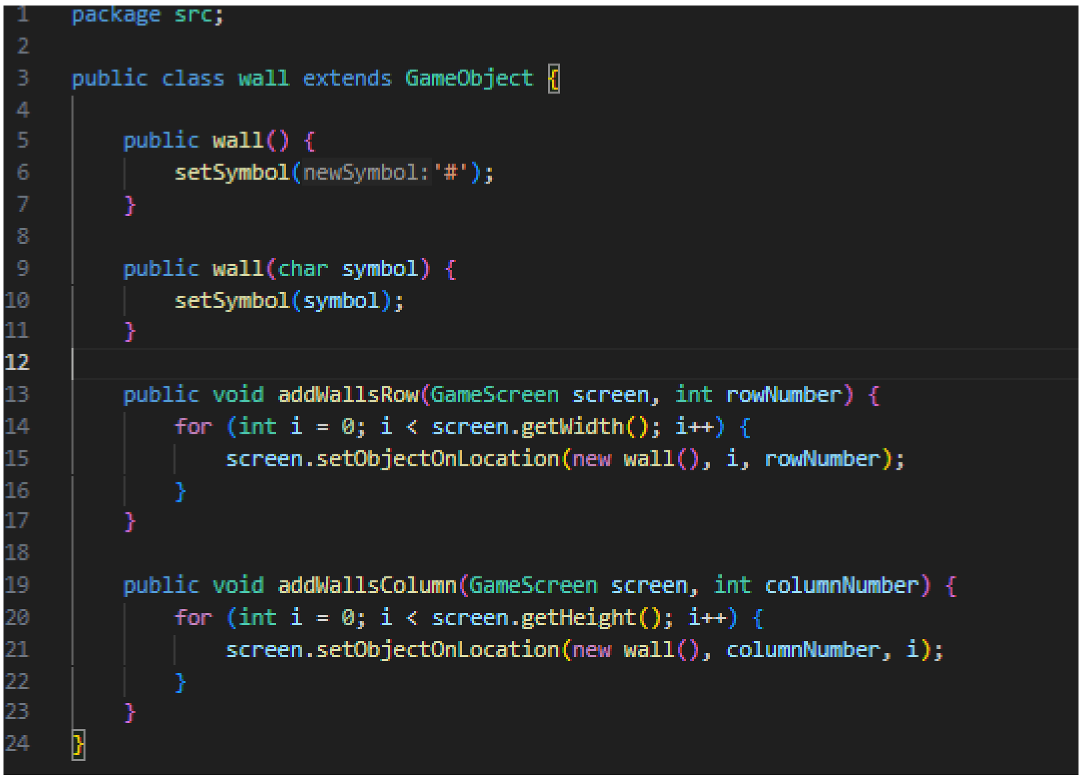
Constructor:

- sets the fruit symbol

Methods:

- getRandomEmptyPos: randomly selects an empty field
- spawnFruit: places the fruit on the board

3.5. Wall Class



```java
package src;

public class wall extends GameObject {

    public wall() {
        setSymbol(newSymbol:'#');
    }

    public wall(char symbol) {
        setSymbol(symbol);
    }

    public void addWallsRow(GameScreen screen, int rowNumber) {
        for (int i = 0; i < screen.getWidth(); i++) {
            screen.setObjectOnLocation(new wall(), i, rowNumber);
        }
    }

    public void addWallsColumn(GameScreen screen, int columnNumber) {
        for (int i = 0; i < screen.getHeight(); i++) {
            screen.setObjectOnLocation(new wall(), columnNumber, i);
        }
    }
}
```

Represents the walls that limit the board.

Constructors:

- default wall symbol
- alternate constructor allows setting a custom symbol

Methods:

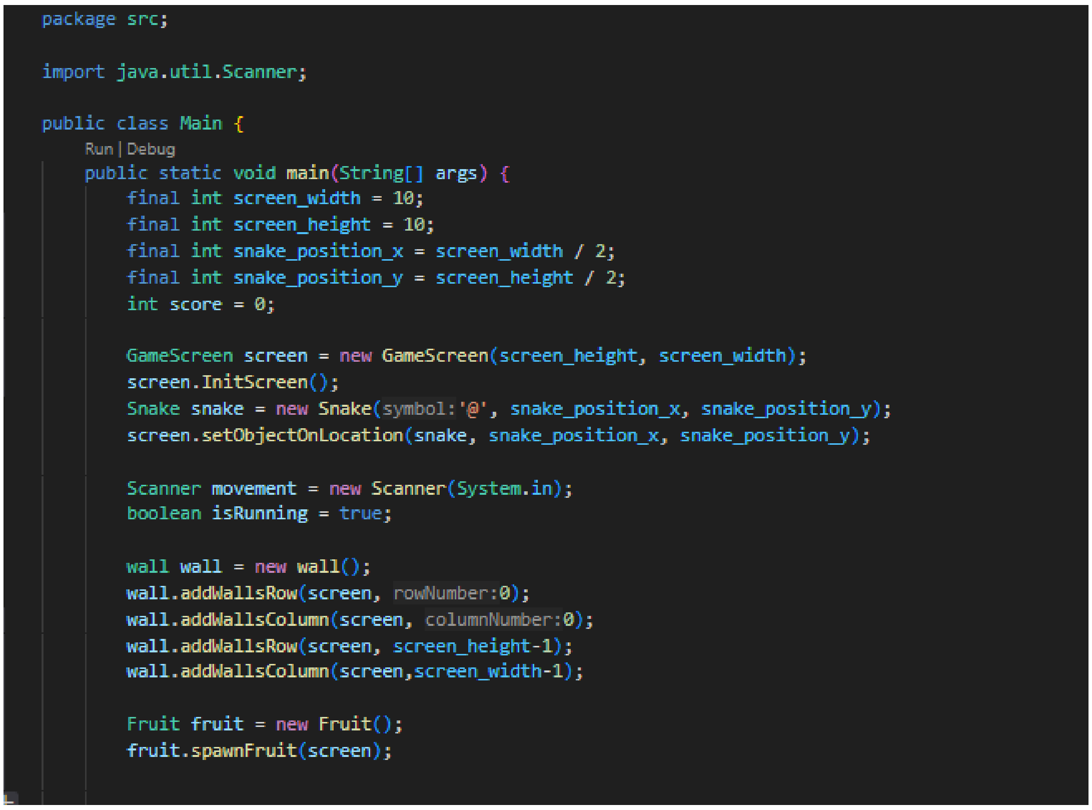
- addWallsRow: places a row of walls
- addWallsColumn: places a column of walls

3.6. Main Class

Contains the main method responsible for the game logic.

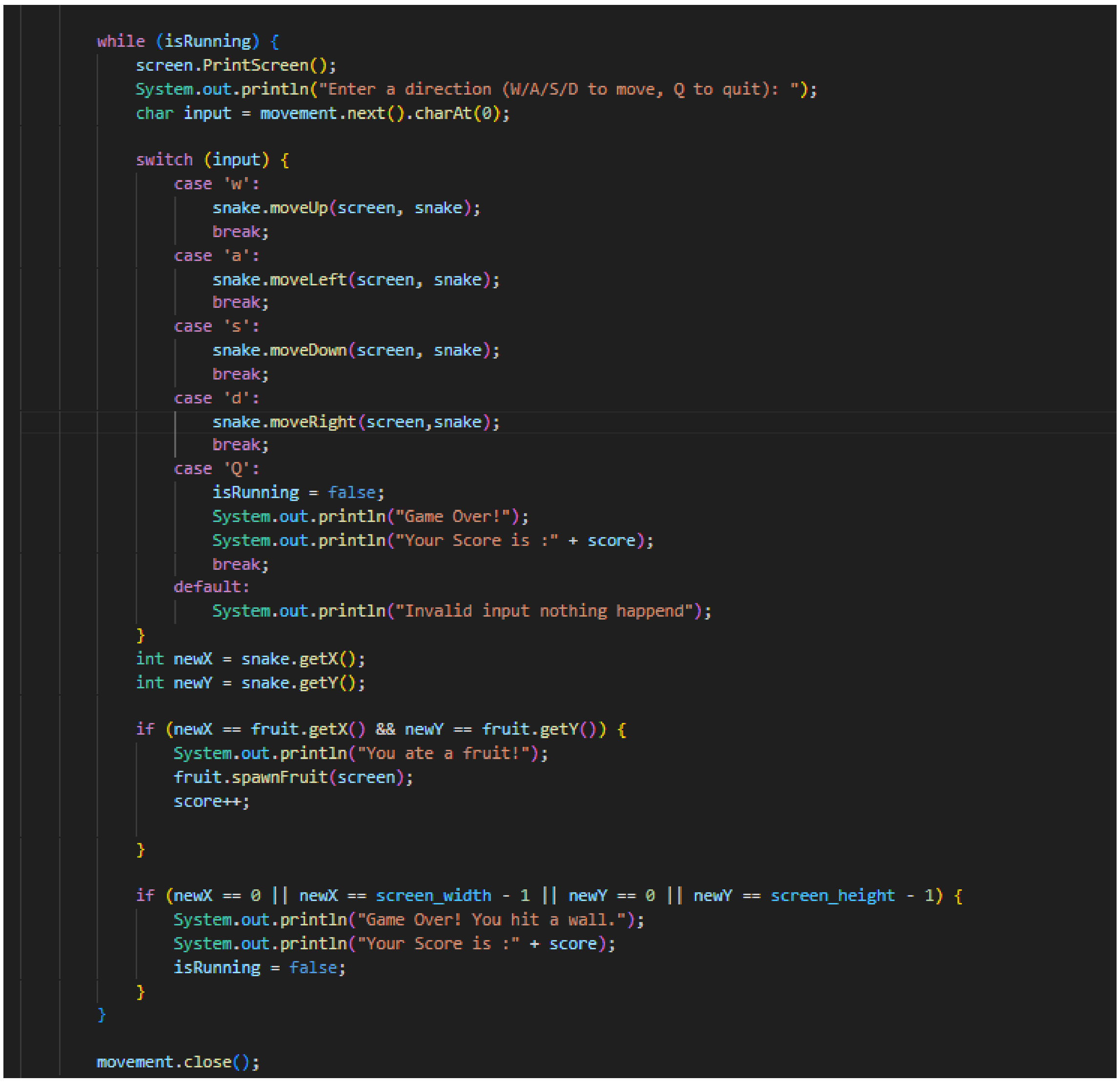
Elements of the main method:

- Initialization of the board, snake, fruit, and walls



```
package src;

import java.util.Scanner;

public class Main {
    Run | Debug
    public static void main(String[] args) {
        final int screen_width = 10;
        final int screen_height = 10;
        final int snake_position_x = screen_width / 2;
        final int snake_position_y = screen_height / 2;
        int score = 0;

        GameScreen screen = new GameScreen(screen_height, screen_width);
        screen.InitScreen();
        Snake snake = new Snake(symbol:'@', snake_position_x, snake_position_y);
        screen.setObjectOnLocation(snake, snake_position_x, snake_position_y);

        Scanner movement = new Scanner(System.in);
        boolean isRunning = true;

        wall wall = new wall();
        wall.addWallsRow(screen, rowNumber:0);
        wall.addWallsColumn(screen, columnNumber:0);
        wall.addWallsRow(screen, screen_height-1);
        wall.addWallsColumn(screen,screen_width-1);

        Fruit fruit = new Fruit();
        fruit.spawnFruit(screen);
```

- Game loop: receiving user input (W, A, S, D, Q), moving the snake, checking for collisions, printing the board



```
        while (isRunning) {
            screen.PrintScreen();
            System.out.println("Enter a direction (W/A/S/D to move, Q to quit): ");
            char input = movement.next().charAt(0);

            switch (input) {
                case 'w':
                    snake.moveUp(screen, snake);
                    break;
                case 'a':
                    snake.moveLeft(screen, snake);
                    break;
                case 's':
                    snake.moveDown(screen, snake);
                    break;
                case 'd':
                    snake.moveRight(screen,snake);
                    break;
                case 'Q':
                    isRunning = false;
                    System.out.println("Game Over!");
                    System.out.println("Your Score is :" + score);
                    break;
                default:
                    System.out.println("Invalid input nothing happend");
            }
            int newX = snake.getX();
            int newY = snake.getY();

            if (newX == fruit.getX() && newY == fruit.getY()) {
                System.out.println("You ate a fruit!");
                fruit.spawnFruit(screen);
                score++;

            }

            if (newX == 0 || newX == screen_width - 1 || newY == 0 || newY == screen_height - 1) {
                System.out.println("Game Over! You hit a wall.");
                System.out.println("Your Score is :" + score);
                isRunning = false;
            }
        }

        movement.close();
```

- Ending the game and displaying the score

4. Final Result

The game works as intended. The user can control the snake, eat fruit, and end the game upon collision. The code is modular, object-oriented, and easy to expand.