Sprawozdanie z projektu: **shake_game_java**

1. Cel projektu

Celem projektu bylo zaprogramowanie prostej gry Snake dzialajacej w konsoli w jezyku Java z wykorzystaniem programowania obiektowego. Gracz steruje wezem, ktory porusza sie po planszy i zdobywa punkty poprzez zjadanie pojawiajacych sie owocow. Gra konczy sie w momencie uderzenia w sciane lub zakonczenia przez uzytkownika.

2. Etapy realizacji projektu

- Projektowanie struktury klas: podzial aplikacji na klasy: GameObject, GameScreen, Snake, Fruit, wall, Main.
- Tworzenie silnika planszy (GameScreen): implementacja przechowywania i wyswietlania obiektow na planszy.
- Implementacja obiektow gry (Snake, Fruit, wall): zaprojektowanie poruszania sie weza, generowania owocow oraz scian.
- Sterowanie gra (Main): petla gry, obsluga wejscia uzytkownika, warunki zakonczenia.
- Testowanie i debugowanie: poprawa bledow, sprawdzenie dzialania gry.

3. Opis klas i metod

3.1. Klasa GameObject

```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;
    }
}
```

Klasa bazowa dla wszystkich obiektow na planszy.

Pola:

- x, y: wspolrzedne obiektu
- symbol: znak reprezentujacy obiekt

Metody:

- getX, getY: zwracaja wspolrzedne
- setX, setY: ustawiaja nowe wspolrzedne
- getSymbol: zwraca symbol obiektu
- setSymbol: ustawia symbol

3.2. Klasa GameScreen

```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();
    }
}
```

Reprezentuje plansze gry jako tablice znakow.

Pola:

- height, width: wysokosc i szerokosc planszy
- screenBack: dwuwymiarowa tablica znakow

Metody:

- konstruktor: inicjalizuje plansze
- InitScreen: wypelnia plansze pustymi znakami
- PrintScreen: wyswietla plansze w konsoli
- setObjectOnLocation: ustawia obiekt na danej pozycji
- ClearScreen: czyści dane pole
- getObjectOnLocation: zwraca symbol z danego pola
- getWidth, getHeight: zwracaja rozmiary planszy

3.3. Klasa Snake

```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);
        }
    }

}
```

Reprezentuje gracza sterujacego wezem.

Konstruktor:

- ustawia symbol i poczatkowa pozycje

Metody:

- moveLeft, moveRight, moveUp, moveDown: zmieniaja pozycje weza, ustawiajac nowa pozycje na planszy i czyszczac poprzednia

3.4. Klasa Fruit

```
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]);
    }

}
```

Reprezentuje owoc, ktory wonsz moze zjesc.

Konstruktor:

- ustawia symbol owocu

Metody:

- getRandomEmptyPos: losuje puste pole
- spawnFruit: ustawia owoc na planszy

3.5. Klasa wall

```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);
        }
    }
}
```

Reprezentuje sciany ograniczajace plansze.

Konstruktory:

- domyslny symbol sciany
- alternatywny z mozliwoscia ustawienia innego symbolu

Metody:

- addWallsRow: ustawia rzad scian
- addWallsColumn: ustawia kolumne scian

3.6. Klasa Main

Zawiera metode main, odpowiadajaca za logike gry.

Elementy metody main:

- Inicjalizacja planszy, weza, owocu i scian

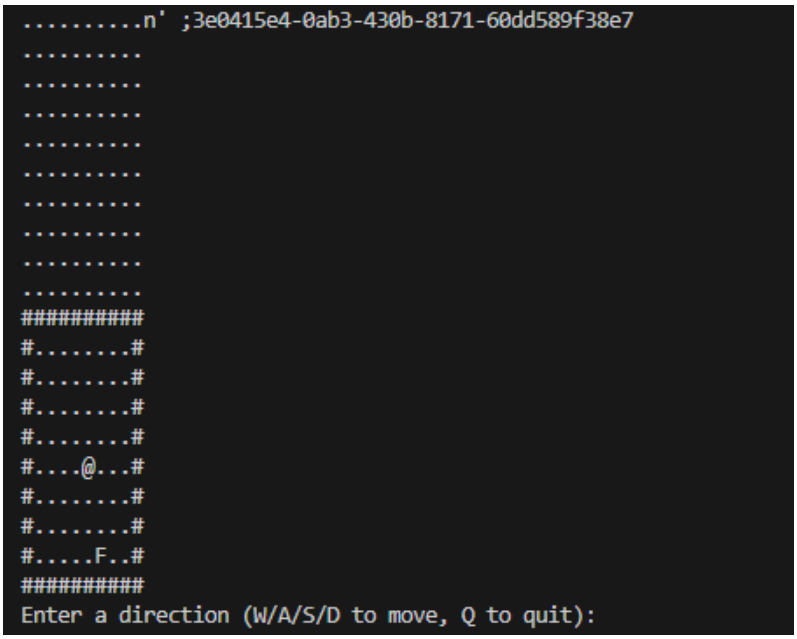
```java
package src;

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);
```

- Petla gry: pobieranie wejscia (W, A, S, D, Q), poruszanie wezem, sprawdzanie kolizji, wyswietlanie planszy

```java
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();
```

- Zakonczenie gry i wyswietlenie wyniku

4. Wynik koncowy



Gra dziala zgodnie z zalozeniami. Uzytkownik moze sterowac wezem, zjesc owoc i zakonczyc gre przy kolizji. Kod jest modularny, obiektowy i latwy do rozbudowy.