Sprawozdanie- projekt z programowania w języku Java
Jędrzej Cieślikiewicz grupa 2ID11B
Politechnika Świętokrzyska w Kielcach

Spis treści:

Założenia projektu

Tematem projektu jest klasyczna gra Snake, która polega na sterowaniu po planszy linią której długość zwiększa się po zjedzeniu “jedzenia”.
W mojej wersji wąż po kolizji ze ścianą lub z samym sobą zmniejsza się do rozmiaru startowego I natychmiast zaczyna od początku (w oryginale gra się po prostu kończyła).
Gra oferuje tryb jednoosobowy oraz dwuoosobowy (lokalnie lub klient-serwer).
Do wykonania projektu użyłem bibliotek graficznych awt oraz Swing.

Uruchomienie

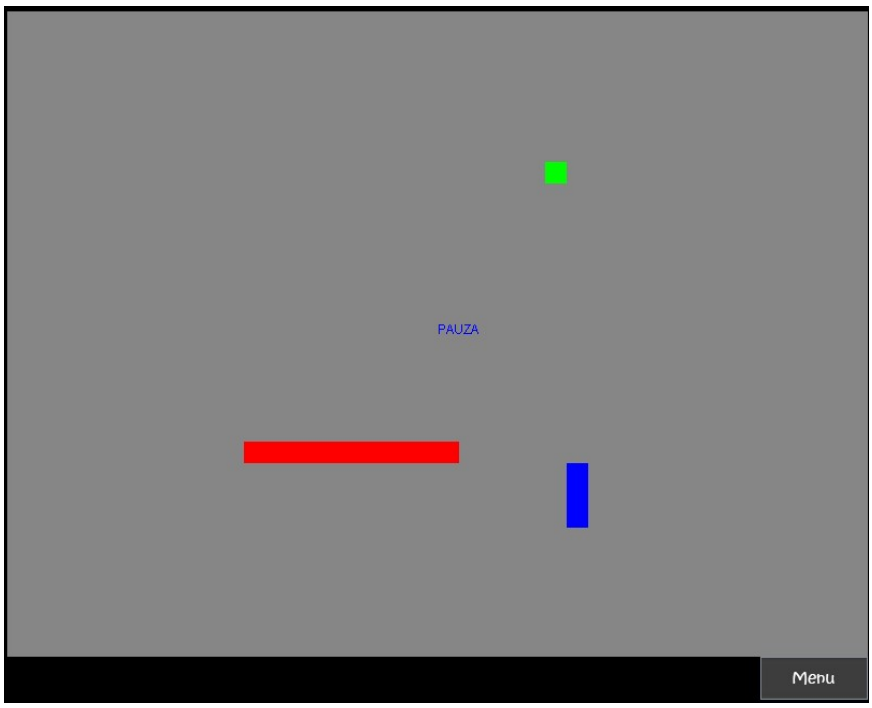
Do uruchomienia projektu używałem JDK w wersji 11

Aby zagrać w tryb jednoosobowy lub dwuosobowy lokalnie wystarczy uruchomić Main.java

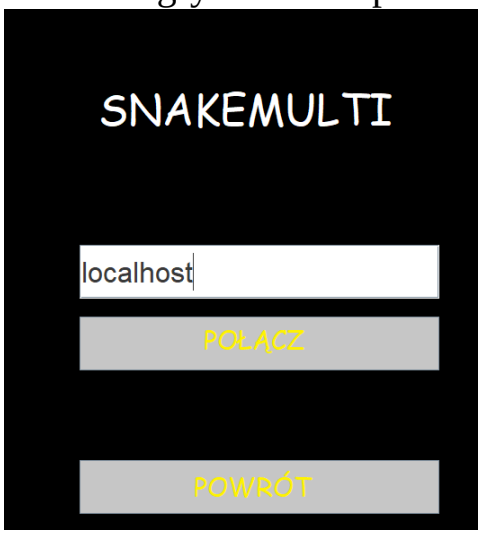
Aby zagrać w wersji klient-serwer należy:
1. Uruchomić SnakeServer.java
2. Uruchomić Main.java na pierwszym komputerze
3. Wybrać przycisk ONLINE I wpisać localhost w polu tekstowym
4. Uruchomić Main.java na drugim komputerze wybrać ONLINE I wpisać adres IP komputera z uruchomionym serwerem

Sterowanie:
Gracz1: WSAD
Gracz2: strzałki
Klawiszem Escape można zapauzować grę w trybie jednoosobowym I dwuosobowym lokalnym.

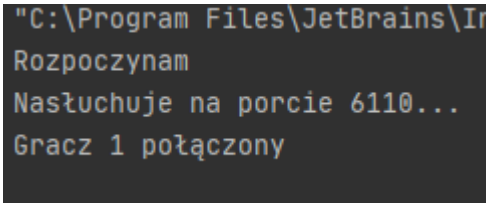
Działanie:



Tryb dwuosobowy lokalny:
Gracz1 czerwony
Gracz2 niebieski
Jedzenie zielone
w prawym dolnym rogu przycisk pozwalający na szybki powrót do menu
w trakcie gry można zapauzować przyciskiem Escape



po wybraniu trybu online musimy wpisać w pole tekstowe adres IP serwera



serwer w terminalu na bieżąco informuje czy gracze są połączeni

Klasa Point

```java
public class Point implements Serializable{

    private static final long serialVersionUID = 13801921310397422l5L;

    int x;
    int y;

    public Point(int x, int y) { setPoint(x, y); }

    public Point(Point p) { setPoint(p); }

    public boolean equals(Point p) { return (this.x == p.x && this.y == p.y); }

    public void setPoint(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public void setPoint(Point p) {
        this.x = p.x;
        this.y = p.y;
    }

    public int getX() { return x; }

    public int getY() { return y; }

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

Zawiera metodę porównywania współrzędnych punktów oraz ich ustawiania

Klasa Direction

```java
public enum Direction {
    up, right, down, left;
}
```

zawiera typ wyliczeniowy z kierunkami w których można się poruszać wężem

Klasa Constants

```java
public class Constants {
    public final static int size = 20;
}
```

zawiera pole size definiujące rozmiar boku kwadratów z których składa się wąż I jedzenie

Klasa Entity

```java
public abstract class Entity {
    protected int X;
    protected int Y;

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

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

    public int getX() { return X; }

    public int getY() { return Y; }

    protected void tick() {

    }

    protected void render(Graphics g, Color c, int offX, int offY) {
        g.setColor(c);
        g.fillRect( x: offX+X,  y: offY+Y, Constants.size, Constants.size);
    }
}
```

Klasa abstrakcyjna po której dziedziczą klasy Food oraz Snake zawierająca metody bez implementacji używane przez te klasy

Klasa Food

```java
public class Food extends Entity{

    public Food(int x, int y) {
        this.X = x;
        this.Y = y;
    }

    public Food() {
        this.X = new Random().nextInt( bound: 30) * Constants.size;
        this.Y = new Random().nextInt( bound: 30) * Constants.size;
    }

    public void reset() {
        this.X = new Random().nextInt( bound: 30) * Constants.size;
        this.Y = new Random().nextInt( bound: 30) * Constants.size;
    }

    @Override
    public void tick() {

    }

    @Override
    public void render(Graphics g, Color c, int offsetX, int offsetY) {
        g.setColor(c);
        g.fillRect( i: offsetX+X,  i1: offsetY+Y, Constants.size, Constants.size);
    }
```

zawiera metody odpowiadające za wylosowanie punktu na planszy w którym pojawi się jedzenie

Klasa Snake

```java
public class Snake extends Entity{

    private Direction direction;
    public ArrayList<Point> location;

    private int originX;
    private int originY;
    private Direction defaultDirection;

    public Snake() {
        location = new ArrayList<Point>();
        direction = Direction.right;
        location.add(new Point( x: 0,  y: 0));
        originX = 0;
        originY = 0;
        defaultDirection = Direction.right;
    }

    public Snake(int x, int y, Direction d) {
        location = new ArrayList<Point>();
        direction = d;
        location.add(new Point(x, y));
        defaultDirection = d;
        originX = x;
        originY = y;
    }
```

```java
public boolean checkCollisionWith(Entity e) {
    if(e instanceof Food) {
        if(location.get(0).getX() == e.getX() && location.get(0).getY() == e.getY()) {
            increaseLength();
            return true;
        }
    }

    if(e instanceof Snake) {

        Snake other = (Snake) e;

        for(int i = 0; i < other.location.size(); i++) {
            if(location.get(0).equals(other.location.get(i))) {

                if(i == 0) {
                    other.reset();
                }

                reset();
            }
        }
    }

    return false;
}
```

zawiera metody przydzielające pozycje początkową węża na planszy, metodę sprawdzającą kolizje węża z nim samym lub jedzeniem

```java
public void reset() {
    int size = location.size();

    for(int i = size - 1; i > 0; i--) {
        location.remove(i);
    }

    location.get(0).setPoint(originX, originY);
    setDirection(defaultDirection);
}
```

metoda resetująca rozmiar węża I jego położenie na planszy do stanu startowego

```
public void increaseLength() { location.add(new Point(location.get(location.size()-1))); }
```

metoda wydłużająca rozmiar węża

```
@Override
public void tick() {

    for(int i = (location.size() - 1); i > 0; i--) {
        location.get(i).setPoint(location.get(i-1));
    }

    switch(direction) {
    case up:
        location.get(0).setY(location.get(0).getY() - Constants.size);
        break;
    case right:
        location.get(0).setX(location.get(0).getX() + Constants.size);
        break;
    case down:
        location.get(0).setY(location.get(0).getY() + Constants.size);
        break;
    case left:
        location.get(0).setX(location.get(0).getX() - Constants.size);
        break;
    }
```

```
    // Sprawdzanie kolizji
    for(int i = 1; i < location.size(); i++) {
        if(location.get(0).equals(location.get(i)))
            reset();
    }

    // Sprawdzanie czy poza granica
    if(location.get(0).getX() >= 800|| location.get(0).getX() < 0 || location.get(0).getY() >= 600 || location.get(0).getY() < 0) {
        reset();
    }


}
```

metoda odpowiadająca za ruch węża po planszy a także sprawdzająca czy nie wyjechał poza plansze

```
    @Override
    public void render(Graphics g, Color c, int offX, int offY) {

        g.setColor(c);

        for(int i = 0; i < location.size(); i++) {
            g.fillRect( i: location.get(i).getX() + offX,  i1: location.get(i).getY() + offY, Constants.size, Constants.size);
        }


    }

}
```

metoda renderująca ciało węża

Klasa GameMode

```java
public class GameMode {

    Snake snake;
    Food food;
    boolean paused = false;
    Input input;

    public Snake getSnake() { return snake; }

    public Food getFood() {
        return food;
    }
}
```

klasa po której dziedziczą klasy SinglePlayer, LocalPlayer, OnlineMode zawiera metody zwracające instancje obiektów Snake I Food

Klasa SinglePlayer

```java
public class SinglePlayer extends GameMode implements Runnable{

    // Właściwości gry
    public static boolean running = false;
    public static boolean paused = false;
    static int gameSpeed = 100000000;

    Snake snake;
    Food food;

    public SinglePlayer() {
        snake = new Snake();
        food = new Food();
    }

    public void reset() {
        snake.reset();
        food.reset();
        running = false;
        paused = false;
    }
```

```java
@Override
public void run() {
    running = true;

    long last = System.nanoTime();
    while(running) {
        if(!paused) {
            if(System.nanoTime() - last > gameSpeed) {
                last = System.nanoTime();
                tick();
            }
        }
    }
}

public void tick() {

    snake.tick();
    food.tick();

    if(snake.checkCollisionWith(food)) {
        food = new Food();
    }

}
```

zawiera pola odpowiadająca za szybkość gry oraz jej stan (działająca/pauza)

Klasa LocalPlayer

```java
public class LocalPlayer extends GameMode implements Runnable{

    ArrayList<Snake> snakes;
    Food food;

    // Game Properties
    public static boolean running = false;
    public static boolean paused = false;
    static int gameSpeed = 100000000;

    public LocalPlayer() {
        snakes = new ArrayList<Snake>();
        snakes.add(new Snake());
        snakes.add(new Snake( x: 780,  y: 580, Direction.left));
        food = new Food();
    }

    public void reset() {
        snakes.get(0).reset();
        snakes.get(1).reset();
        food.reset();
        running = false;
        paused = false;
    }
```

```java
    private void tick() {
        snakes.get(0).tick();
        snakes.get(1).tick();
        food.tick();

        if(snakes.get(0).checkCollisionWith(food)) {
            food = new Food();
        }

        if(snakes.get(1).checkCollisionWith(food)) {
            food = new Food();
        }

        if(snakes.get(0).checkCollisionWith(snakes.get(1))) {
            food = new Food();
        }

        if(snakes.get(1).checkCollisionWith(snakes.get(0))) {
            food = new Food();
        }
    }


    public Snake getSnake(int x) { return snakes.get(x); }

    public Food getFood() { return food; }
}
```

jako że potrzebne są nam dwa węże są one przechowywane w formie listy

Klasa OnlineMode

```java
public class OnlineMode extends GameMode implements Runnable{

    ArrayList<Snake> snakes;
    Food food;

    // Konfiguracja klienta
    String host;
    Socket socket;
    ObjectInputStream ois;
    ObjectOutputStream oos;

    public OnlineMode(String ip) {
        snakes = new ArrayList<Snake>();
        snakes.add(new Snake());
        snakes.add(new Snake( x: 780,  y: 580, Direction.left));
        food = new Food();
        host = ip;
        try {
            socket = new Socket(ip,  port: 6110);
            oos = new ObjectOutputStream(socket.getOutputStream());
            ois = new ObjectInputStream(socket.getInputStream());
        } catch (UnknownHostException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

```java
public void setIP(String ip) {
    host = ip;
    try {
        socket = new Socket(ip,  port: 6110);
        oos = new ObjectOutputStream(socket.getOutputStream());
        ois = new ObjectInputStream(socket.getInputStream());
    } catch (UnknownHostException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

metoda ustawiająca adres IP

Klasa Screen

```
public enum Screen {
    Menu, PlayMenu, Game, LocalGame, OnlineMenu, OnlineGame;
}
```

zawiera typ wyliczeniowy z elementami okna

Klasa SnakePanel

```
// Rozmiar okna
public int w = 800;
public int h = 600;

Screen screenState;

// Komponenty panelu
Image snakeTitleIMG;
Image playButtonIMG;
Image menuButtonIMG;
Image singleButtonIMG;
Image localButtonIMG;
Image onlineButtonIMG;
Image backButtonIMG;
Image connectButtonIMG;
Image exitButtonIMG;
```

w tej klasie są zdefiniowane wymiary planszy oraz komponenty menu (przyciski, obrazy, pole tekstowe do wprowadzenia adresu IP) oraz metody renderujące

Klasa ButtonListener

```
public class ButtonListener implements ActionListener{

    SnakePanel panel;

    public ButtonListener(SnakePanel panel) { this.panel = panel; }
```

```
public void actionPerformed(ActionEvent e) {

    if(panel.playButton == e.getSource()) {
        panel.setScreenMode(Screen.PlayMenu);
    }else if(panel.menuButton == e.getSource()) {
        SinglePlayer.running = false;
        LocalPlayer.running = false;
        panel.setScreenMode(Screen.Menu);
    }else if(panel.singleButton == e.getSource()) {
        panel.setScreenMode(Screen.Game);
        Main.startSinglePlayer();
    }else if(panel.localButton == e.getSource()) {
        panel.setScreenMode(Screen.LocalGame);
        Main.startLocalPlayer();
    }else if(panel.onlineButton == e.getSource()) {
        panel.setScreenMode(Screen.OnlineMenu);
    }else if(panel.connectButton == e.getSource()){
        panel.setScreenMode(Screen.OnlineGame);
        Main.startOnlineGame(panel.ipTextField.getText());
    }else if(panel.backButton == e.getSource()) {
        panel.setScreenMode(Screen.Menu);
    }else if(panel.returnButton == e.getSource()) {
        panel.setScreenMode(Screen.PlayMenu);
    }else if(panel.exitButton == e.getSource()) {
        System.exit( status: 0);
    }

}
```

zawiera metodę odpowiadającą za działanie przycisków menu

Klasa Input

```
public void keyPressed(KeyEvent e) {

    if(Main.singleThread.isAlive()) {
        if(e.getKeyCode() == KeyEvent.VK_W) {
            if(Main.singleGame.getSnake().getDirection() != Direction.down)
                Main.singleGame.getSnake().setDirection(Direction.up);
        }else if(e.getKeyCode() == KeyEvent.VK_D) {
            if(Main.singleGame.getSnake().getDirection() != Direction.left)
                Main.singleGame.getSnake().setDirection(Direction.right);
        }else if(e.getKeyCode() == KeyEvent.VK_S) {
            if(Main.singleGame.getSnake().getDirection() != Direction.up)
                Main.singleGame.getSnake().setDirection(Direction.down);
        }else if(e.getKeyCode() == KeyEvent.VK_A) {
            if(Main.singleGame.getSnake().getDirection() != Direction.right)
                Main.singleGame.getSnake().setDirection(Direction.left);
        }else if(e.getKeyCode() == KeyEvent.VK_ESCAPE) {
            SinglePlayer.paused = !SinglePlayer.paused;
        }
    }
```

```
if(Main.localThread.isAlive()) {
    if(e.getKeyCode() == KeyEvent.VK_W) {
        if(Main.localGame.getSnake( x: 0).getDirection() != Direction.down)
            Main.localGame.getSnake( x: 0).setDirection(Direction.up);
    }else if(e.getKeyCode() == KeyEvent.VK_D) {
        if(Main.localGame.getSnake( x: 0).getDirection() != Direction.left)
            Main.localGame.getSnake( x: 0).setDirection(Direction.right);
    }else if(e.getKeyCode() == KeyEvent.VK_S) {
        if(Main.localGame.getSnake( x: 0).getDirection() != Direction.up)
            Main.localGame.getSnake( x: 0).setDirection(Direction.down);
    }else if(e.getKeyCode() == KeyEvent.VK_A) {
        if(Main.localGame.getSnake( x: 0).getDirection() != Direction.right)
            Main.localGame.getSnake( x: 0).setDirection(Direction.left);
    }else if(e.getKeyCode() == KeyEvent.VK_ESCAPE) {
        LocalPlayer.paused = !LocalPlayer.paused;
        if(!LocalPlayer.running) {
            LocalPlayer.running = true;
        }
    }
```

```
    if(e.getKeyCode() == KeyEvent.VK_UP) {
        if(Main.localGame.getSnake( x: 1).getDirection() != Direction.down)
            Main.localGame.getSnake( x: 1).setDirection(Direction.up);
    }else if(e.getKeyCode() == KeyEvent.VK_RIGHT) {
        if(Main.localGame.getSnake( x: 1).getDirection() != Direction.left)
            Main.localGame.getSnake( x: 1).setDirection(Direction.right);
    }else if(e.getKeyCode() == KeyEvent.VK_DOWN) {
        if(Main.localGame.getSnake( x: 1).getDirection() != Direction.up)
            Main.localGame.getSnake( x: 1).setDirection(Direction.down);
    }else if(e.getKeyCode() == KeyEvent.VK_LEFT) {
        if(Main.localGame.getSnake( x: 1).getDirection() != Direction.right)
            Main.localGame.getSnake( x: 1).setDirection(Direction.left);
    }
}
```

zawiera metodę keyPressed odpowiadającą za przypisanie pod konkretne klawisze akcji związanych z poruszaniem wężem I zapauzowanie

Klasa SnakeServer

```
public class SnakeServer {

    static SnakeServer server;

    // Właściwości serwera
    int port = 6110;
    ServerSocket socket;

    // Gracze
    Player player1;
    Player player2;
    Input player1Input;
    Input player2Input;
    Output player1Output;
    Output player2Output;


    Food food;

    // Właściwości gry
    boolean running = false;
    int gameSpeed = 100000000;
```

```
private void init() {
    try {
        System.out.println("Rozpoczynam");
        socket = new ServerSocket(port);
        player1 = new Player(1);
        player2 = new Player(2);
        food = new Food();
        player1Input = new Input(player1);
        player2Input = new Input(player2);

        player1Output = new Output(player1, player2, food);
        player2Output = new Output(player2, player1, food);

        player1InputThread = new Thread(player1Input);
        player2InputThread = new Thread(player2Input);
        player1OutputThread = new Thread(player1Output);
        player2OutputThread = new Thread(player2Output);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

metoda inicjująca serwer

```
private void listenForPlayers() throws IOException {
    System.out.println("Nasłuchuje na porcie " + port + "...");
    while(!player1.isAssigned() || !player2.isAssigned()) {
        Socket socket = server.socket.accept();
        if(!player1.isAssigned()) {
            System.out.println("Gracz 1 połączony");
            player1.setSocket(socket);
            player1.setObjectInputStream();
            player1.setObjectOutputStream();
            player1.setAssigned(true);
        }else if(!player2.isAssigned()) {
            System.out.println("Gracz 2 połączony");
            player2.setSocket(socket);
            player2.setObjectInputStream();
            player2.setObjectOutputStream();
            player2.setAssigned(true);
        }
    }

}
```

metoda nasłuchująca na porcie, sprawdzająca czy gracze są już połączeni

```
private void sendData() {
    try {

        player1.getObjectOutputStream().writeBoolean(true);
        player1.getObjectOutputStream().writeObject(player1.getSnake().getLocation());
        player1.getObjectOutputStream().writeObject(player2.getSnake().getLocation());
        player1.getObjectOutputStream().writeInt(food.getX());
        player1.getObjectOutputStream().writeInt(food.getY());
        player1.getObjectOutputStream().reset();

        player2.getObjectOutputStream().writeBoolean(true);
        player2.getObjectOutputStream().writeObject(player2.getSnake().getLocation());
        player2.getObjectOutputStream().writeObject(player1.getSnake().getLocation());
        player2.getObjectOutputStream().writeInt(food.getX());
        player2.getObjectOutputStream().writeInt(food.getY());
        player2.getObjectOutputStream().reset();

    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

metoda wysyłająca dane o położeniu graczy

Wnioski:

Wykonując projekt poznałem dokładniej praktyki programowania obiektowego oraz zapoznałem się z systemem kontroli wersji Git