Snake

Adam Czerwiński

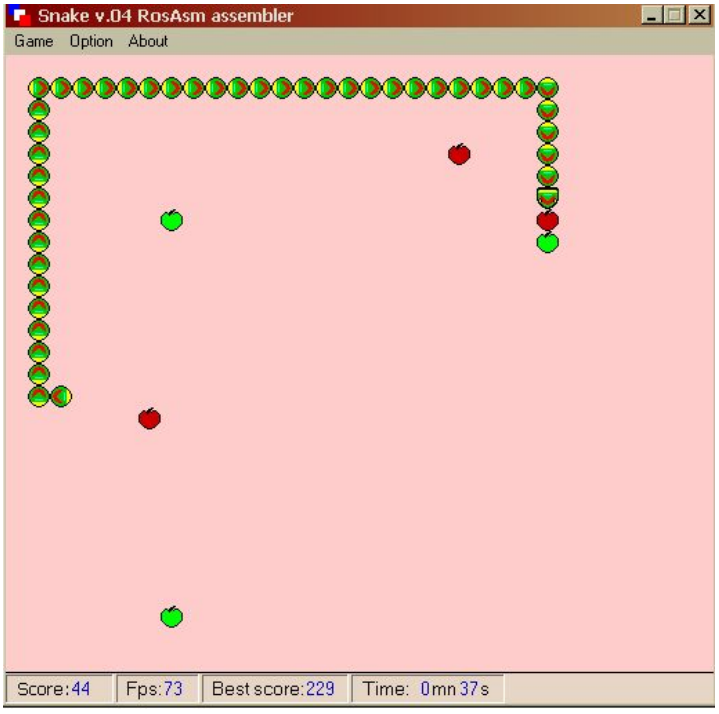
Projekt na lab. programowania wydział MS inf. sem. III 2018/19r

Spis treści

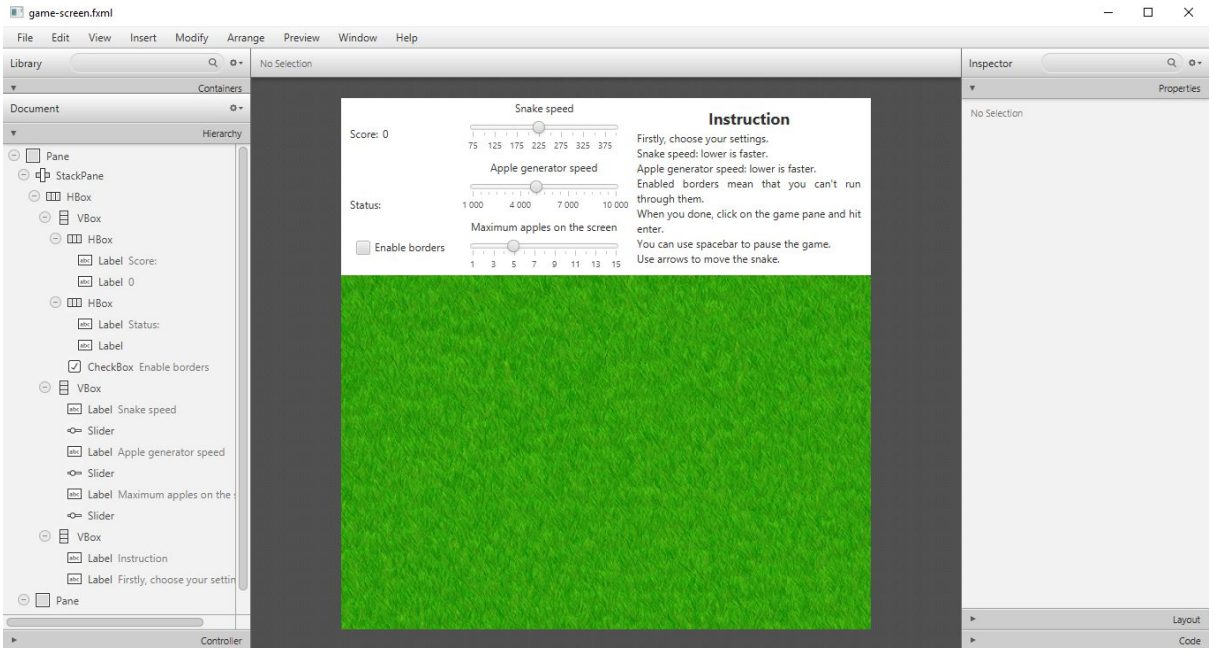
Wstęp

Projekt jest oparty na grze z roku 1976, wydaną pod nazwą Blockade oraz udoskonalony o własne pomysły. Program został napisany w języku JAVA w wersji jdk 11.0.1 wraz z wykorzystaniem biblioteki JavaFX, która jest używana do tworzenia GUI. Korzystano również z narzędzia Scene Builder do szybkiego stworzenia układu wizualnego. Użyto środowiska IDE IntelliJ IDEA.

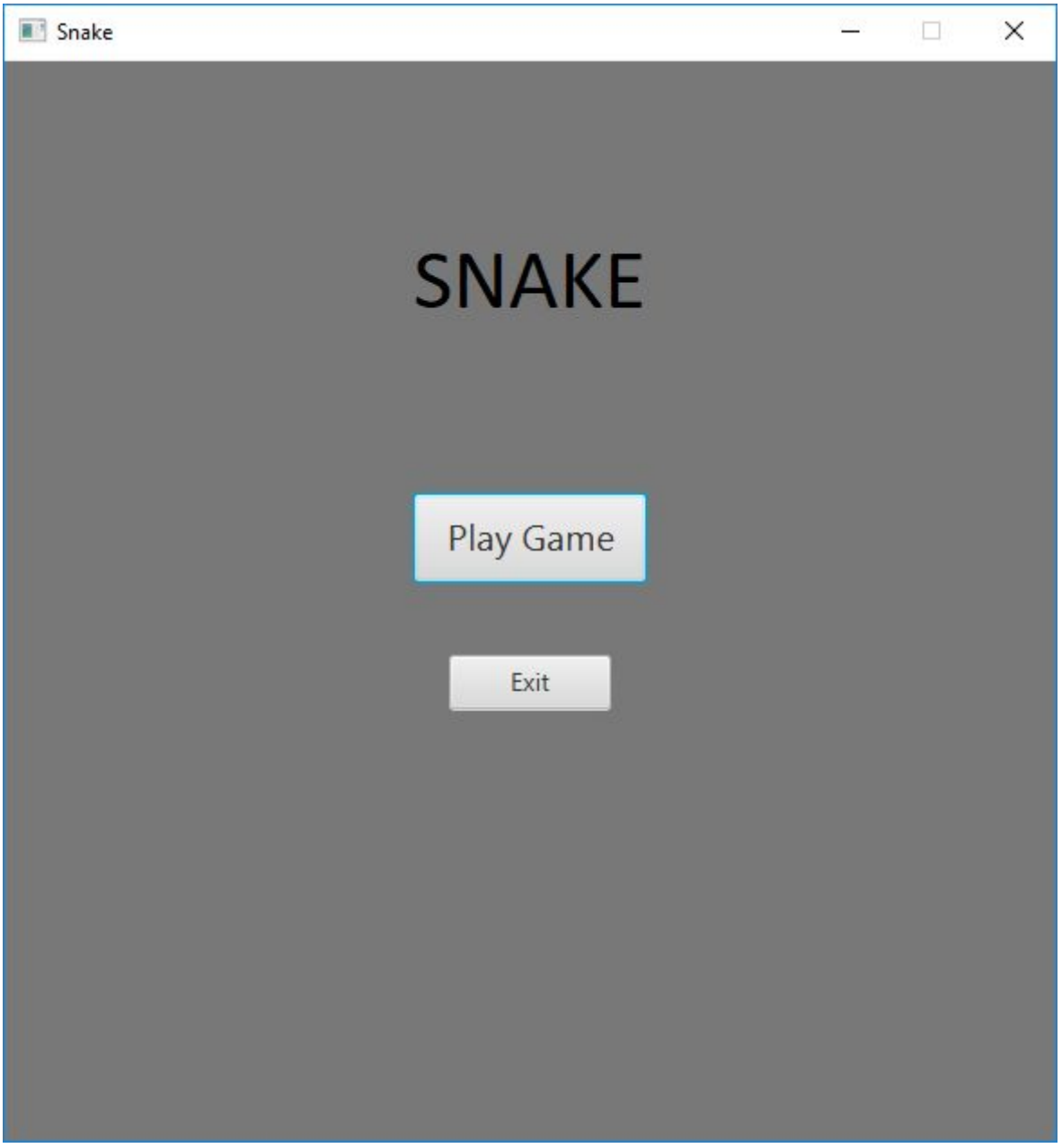
Blockade



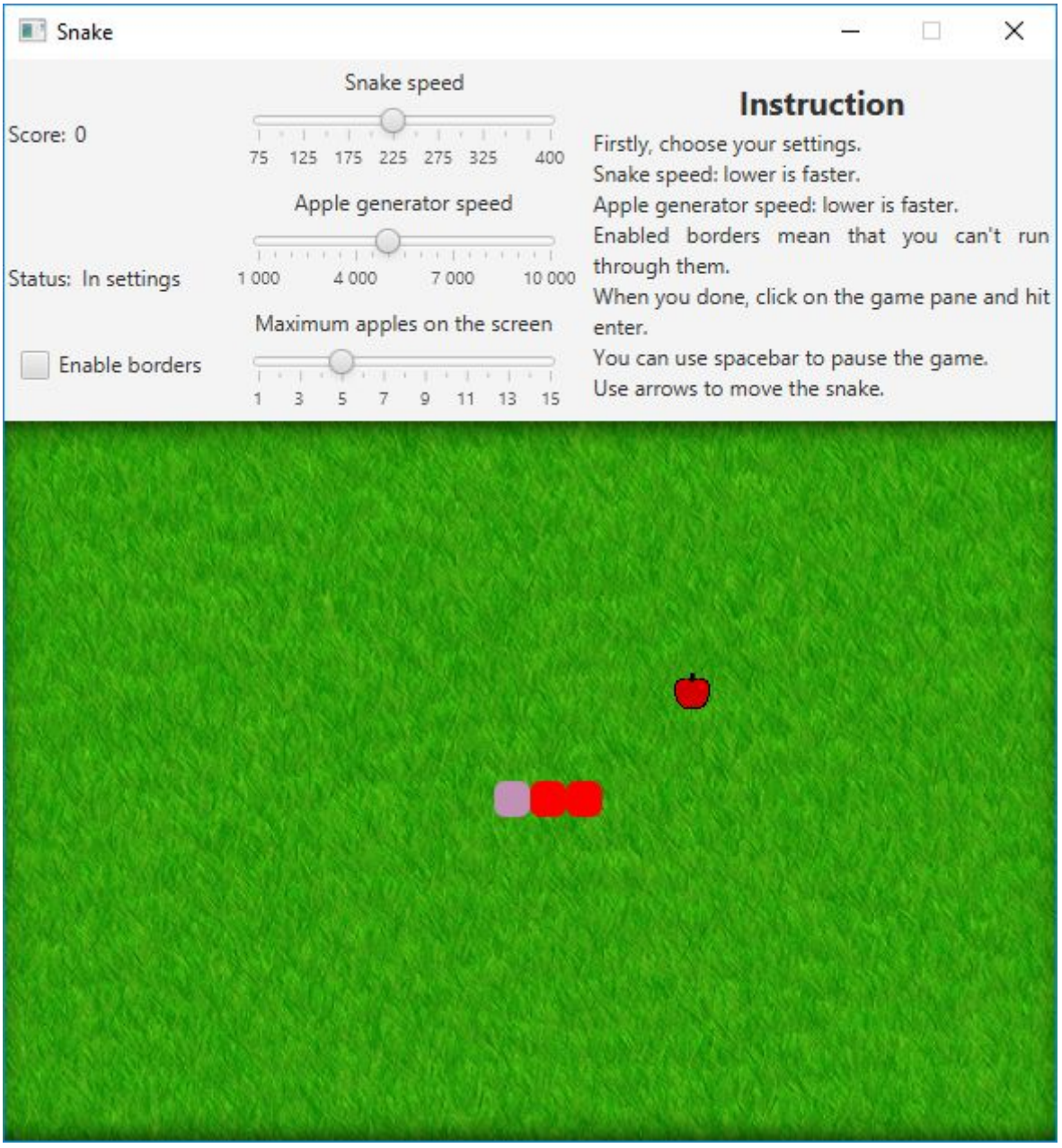
Scene builder



Opis interfejsu



Po uruchomieniu gry mamy dwie opcje - “Play Game” oraz “Exit”. Ostatnia z nich pyta użytkownika o potwierdzenie wyjścia z gry. Pierwsza z opcji przekierowuje nas do panelu z grą.

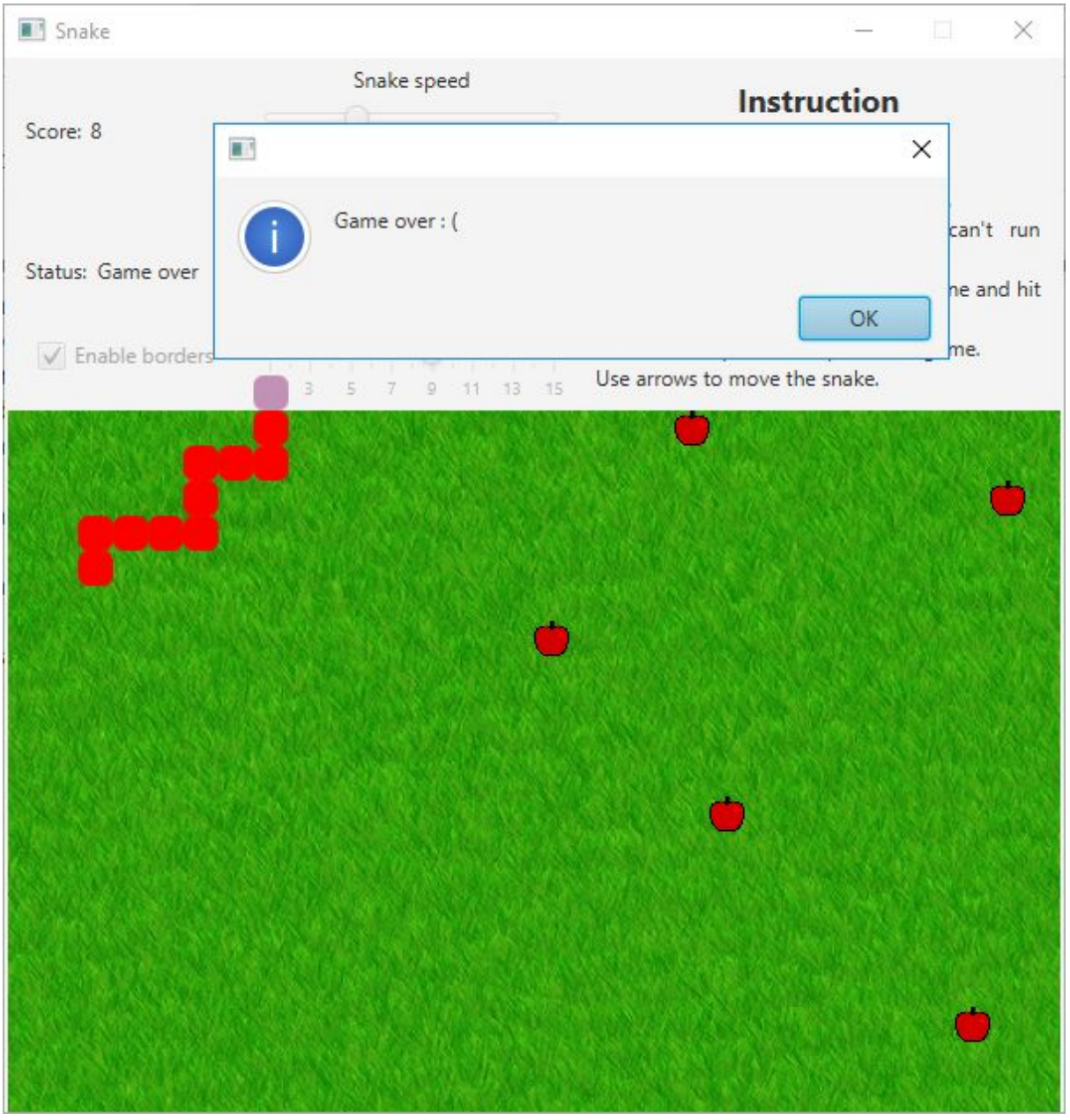


Na początku możemy ustawić poszczególne opcje gry używając kontrolek Slider oraz jednej kontrolki CheckBox. Po wybraniu naszych opcji możemy przejść do gry naciskając przycisk Enter. Do poruszania wężem używamy strzałek. Możemy zatrzymać grę, wznowić oraz wrócić do menu głównego. Gdy gra jest wstrzymana, nie można zmieniać opcji, jedynie przed rozpoczęciem nowej gry.

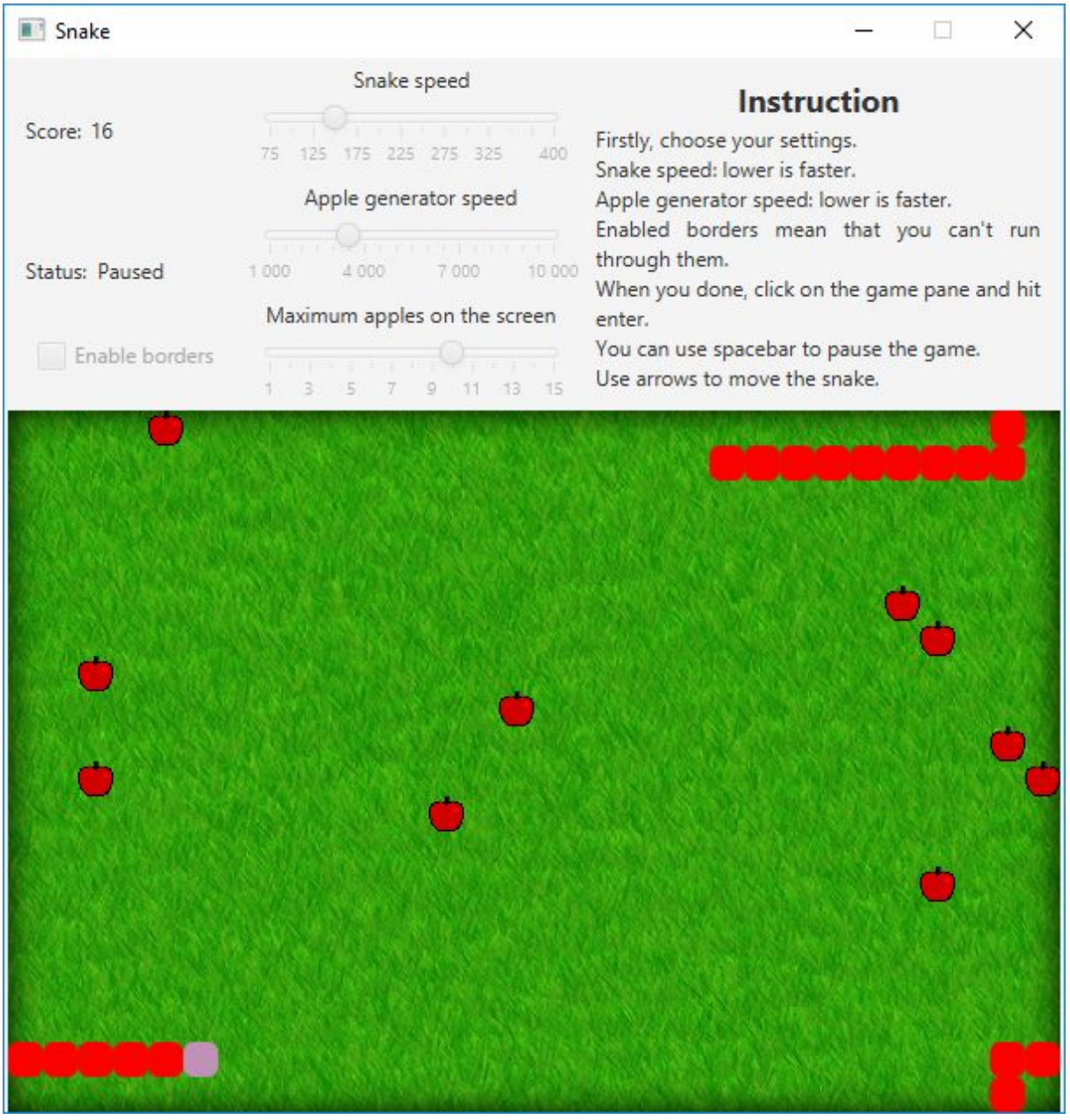
Działanie gry

Gra polega na jak największej ilości zebranych punktów. Punkty otrzymujemy za zjedzenie jabłek przez węża, wtedy również wąż rośnie. Koniec gry następuje po ugryzieniu samego siebie albo po uderzeniu w krawędzie (w zależności od opcji). Prędkość poruszania się węża jest zależna od wartości *Snake Speed*. Jest to różnica czasu pomiędzy poszczególnymi klatkami, dlatego im mniejsza wartość, tym szybciej porusza się nasz wąż. *Apple Generator* Speed jest to opcja odpowiadająca za częstotliwość pojawiania się nowych jabłek. Działa tak samo jak opcja *Snake Speed*. Możemy również ustawić ilość maksymalnie wyświetlanych jabłek w jednym momencie na panelu gry za pomocą opcji *Maximum apples on the screen*. CheckBox *Enable Borders* włącza/wyłącza kolizję z krawędziami. Gra jest odzwierciedlona na dwuwymiarowej tablicy 30 na 20.

CheckBox zaznaczony



CheckBox odznaczony



Kod źródłowy

Istnieją tylko dwie klasy przedstawiające obiekty w grze. Klasa Snake oraz klasa Apple, obie dziedziczą po klasie GameObject, która rozszerza klasę Point2D.

```
public class Point2D {

    int x;
    int y;
    int getX() { return x; }
    void setX(int x) { this.x = x; }
    int getY() { return y; }
    void setY(int y) { this.y = y; }
    public Point2D(){}
    public Point2D(int x, int y){this.x=x; this.y=y;}

}
```

```
public class GameObject extends Point2D{


    Rectangle getRectangle() { return this.rectangle; }


    Rectangle rectangle;
    final double width = 20.0;
    final double height = 20.0;

    GameObject(){}

}
```

```java
public class Snake extends GameObject{

    private Directions direction = Directions.UP;

    public void setDirection(Directions direction) { this.direction = direction; }

    public Directions getDirection() { return direction; }

    public void setRectangleX() { this.rectangle.setLayoutX(getX() * width); }

    public void setRectangleY() { this.rectangle.setLayoutY(getY() * height); }


    public Snake() {
        rectangle = new Rectangle(width, height);
        rectangle.setFill(Color.RED);
        rectangle.setArcHeight(12);
        rectangle.setArcWidth(12);
    }

}
```

```java
public class Apple extends GameObject{

    public void generateX() { this.x = (int) (Math.random() * 30); }

    public void generateY() { this.y = (int) (Math.random() * 20); }

    public void setRectangleX() { this.rectangle.setLayoutX(getX() * width); }

    public void setRectangleY() { this.rectangle.setLayoutY(getY() * height); }


    public Apple() {
        rectangle = new Rectangle(width, height);
        rectangle.setFill(new ImagePattern(new Image( s: "pic/apple.png")));
    }
}
```

Main loop

Metoda *GameLoop* odpowiada za aktualizowanie informacji o obiektach co określony czas. Wykorzystano dwie pętle, jedna do aktualizowania informacji o wężu, a druga do aktualizowania informacji o jabłkach, ponieważ mogą mieć różne wartości czasu.

```
private void GameLoop() {
    //---------------GAME LOOP
    gameLoop = new Timeline(
            new KeyFrame(
                    Duration.millis(snakePace.getValue()),
                    event -> {
                        if (Settings.inGame && !Settings.isPause) {
                            Settings.isPressed = false;
                            if (snake.get(0).getDirection() == Directions.UP) {
                                moveUp();
                            } else if (snake.get(0).getDirection() == Directions.RIGHT) {
                                moveRight();
                            } else if (snake.get(0).getDirection() == Directions.DOWN) {
                                moveDown();
                            } else if (snake.get(0).getDirection() == Directions.LEFT) {
                                moveLeft();
                            }
                            moveRest();
                            update();
                            updateStatus();
                        }
                    }
            )
    );//////END OF GAME LOOP
    gameLoop.setCycleCount(Animation.INDEFINITE);
    gameLoop.play();

    appleGenerator = new Timeline(
            new KeyFrame(
                    Duration.millis(appleGeneratorPace.getValue()),
                    event -> {
                        if (Settings.inGame && !Settings.isPause && apples.size() < maximumApplesOnScreen.getValue()) {
                            generateNewApple();
                        }
                    }
            )
```

Poruszanie wężem

Metody *moveUp*, *moveRight*, *moveDown*, *moveLeft* służą do zmiany kierunku głowy węża, a metoda *moveRest* służy do poruszania reszty ciała węża. Poruszanie węża polega na przypisaniu obiektowi wartości z obiektu go poprzedzającego, które te są przechowywane w pomocniczej tablicy obiektów klasy Point2D.

```
private void moveRight() {
    saveLastPositionSnake();
    snake.get(0).setX(snake.get(0).getX() + 1);
    snake.get(0).setRectangleX();
}

private void moveUp() {
    saveLastPositionSnake();
    snake.get(0).setY(snake.get(0).getY() - 1);
    snake.get(0).setRectangleY();
}

private void saveLastPositionSnake() {
    snakeCoordinates.get(0).setX(snake.get(0).getX());
    snakeCoordinates.get(0).setY(snake.get(0).getY());
}

private void moveRest() {
    for (int i = 1; i < snake.size(); i++)
    {
        snakeCoordinates.get(i).setX(snake.get(i).getX());
        snakeCoordinates.get(i).setY(snake.get(i).getY());
        snake.get(i).setX(snakeCoordinates.get(i - 1).getX());
        snake.get(i).setY(snakeCoordinates.get(i - 1).getY());
        snake.get(i).setRectangleX();
        snake.get(i).setRectangleY();
    }
}
```

Sprawdzanie kolizji

```
private void update() {
    if (collisionSnake() || (collisionBorder() && isBorders.get())) {
        quitGame();
        Alert alert = new Alert(Alert.AlertType.INFORMATION);
        alert.setTitle(null);
        alert.setHeaderText(null);
        alert.setContentText("Game over : (");

        alert.setOnHidden(evt -> mainController.loadMenu());

        alert.show();
    }

    if (collisionBorder() && !isBorders.get()) {
        outOfBorders();
    }

    int index = collisionApple();
    if (index >= 0) {
        increaseScore();
        addSnake();
        generateAppleCoordinates(index);
    }
}
```

Pierwszy If:

W warunkach jest sprawdzana kolizja z samym sobą (*collisionSnake*) oraz kolizja z krawędziami w zależności czy jest zaznaczona opcja z włączonymi obramówkami (*isBorders*). Jeżeli warunki zostaną spełnione, tworzony jest alert informujący gracza o przegranej grze, a następnie zostaje załadowane menu główne gry.

Drugi If:

Jeżeli występuje kolizja z krawędziami, a krawędzie są wyłączone, to wywołuje się metoda przemieszczająca węża.

Trzeci If:

Wartość całkowitoliczbowa *index* przechowuje indeks jabłka z którym występuje kolizja i jest on przesyłany dalej do metody *generateAppleCoordinates*, która ma za zadanie wygenerować nową pozycję dla zjedzonego jabłka. Metoda *increaseScore* zwiększa punkty, a metoda *addSnake* powiększa węża.

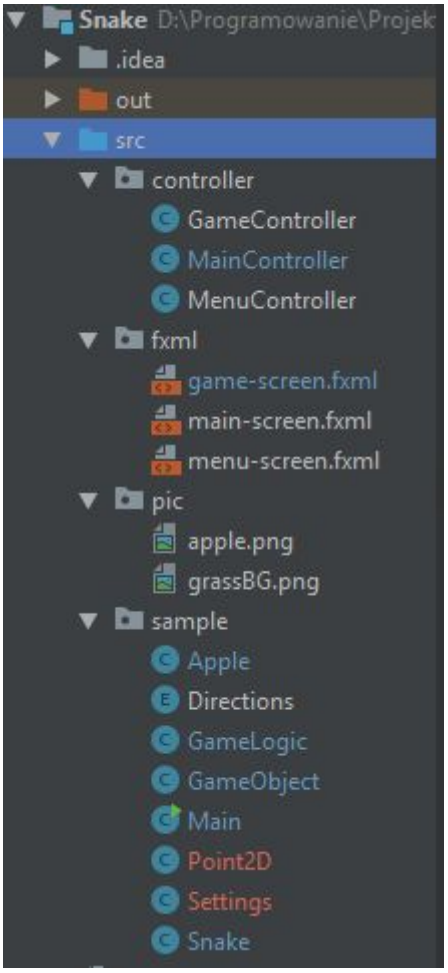
Generowanie nowego jabłka

```
private void generateNewApple() {
    apples.add(new Apple());
    int pom;
    do {
        pom = 0;
        apples.get(apples.size() - 1).generateX();
        apples.get(apples.size() - 1).generateY();
        for (Point2D appleCoord : applesCoordinates) {
            for (Point2D snakeCoord : snakeCoordinates) {
                if ((appleCoord.getX() == apples.get(apples.size() - 1).getX() &&
                        appleCoord.getY() == apples.get(apples.size() - 1).getY())
                        || (apples.get(apples.size() - 1).getX() == snakeCoord.getX() &&
                        apples.get(apples.size() - 1).getY() == snakeCoord.getY())) {
                    pom = 1;
                    break;
                }
            }
            if (pom == 1)
                break;
        }
    } while (pom >= 1);

    apples.get(apples.size() - 1).setRectangleX();
    apples.get(apples.size() - 1).setRectangleY();
    applesCoordinates.add(new Point2D(apples.get(apples.size() - 1).getX(), apples.get(apples.size() - 1).getY()));

    gameController.paneGameAdd(apples.get(apples.size() - 1).getRectangle());
}
```

Nowe jabłko jest generowane i wyświetlane dopiero wtedy, kiedy zostaną wybrane wartości losowe, takie które nie będą kolidowały z innymi obiektami na planszy gry.

Podział na pliki

JavaFX korzysta z plików o rozszerzeniu *FXML*, bazowane na języku *XML*. Służą one do odseparowania struktury logicznej ze strukturą wizualną programu. Do modyfikacji wyglądu m.in. kontrolek można wykorzystać język *CSS*, z którego w tym projekcie nie korzystałem. Pliki *FXML* są tworzone automatycznie przy użyciu programu Scene Builder, nie są one obowiązkowe w programie i można osiągnąć taki sam efekt wyłączenie za pomocą samych klas.



- Pakiet o nazwie *controller* zawiera w sobie klasy, które są powiązane z plikami FXML i mają bezpośredni dostęp do kontrolek stworzonych w odpowiadających im plikach FXML.
- Pakiet *fxml* zawiera pliki fxml.
- Pakiet *pic* zawiera pliki z rozszerzeniem .png.
- Pakiet *sample* zawiera klasy odpowiadające za logikę programu.

Przykładowy plik fxml

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.text.*?>
<?import javafx.scene.effect.*?>
<?import javafx.geometry.*?>
<?import java.lang.*?>
<?import java.util.*?>
<?import javafx.scene.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>

<Pane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity" prefHeight="600.0"
      prefWidth="600.0" xmlns="http://javafx.com/javafx/8" xmlns:fx="http://javafx.com/fxml/1" fx:controller="controller.MenuController">
   <children>
      <VBox alignment="CENTER" layoutX="235.0" layoutY="240.0" spacing="40.0">
         <children>
            <Button mnemonicParsing="false" onAction="#onActionPlayButton" prefHeight="50.0" prefWidth="130.0" text="Play Game">
               <font>
                  <Font size="20.0" />
               </font>
            </Button>
            <Button mnemonicParsing="false" onAction="#onActionExitButton" prefHeight="30.0" prefWidth="90.0" text="Exit">
               <font>
                  <Font size="14.0" />
               </font>
            </Button>
         </children>
      </VBox>
   </children>
</Pane>
```

Schemat blokowy