Third Assignment

Description of the exercise:

2. Snake

We have a rattlesnake in a desert, and our snake is initially two units long (head and rattler). We have to collect with our snake the foods on the level, that appears randomly. Only one food piece is placed randomly at a time on the level (on a field, where there is no snake).

The snake starts off from the center of the level in a random direction. The player can control the movement of the snake's head with keyboard buttons. If the snake eats a food piece, then its length grow by one unit.

It makes the game harder that there are rocks in the desert. If the snake collides with a rock, then the game ends. We also lose the game, if the snake goes into itself, or into the boundary of the game level.

In these situations show a popup messagebox, where the player can type his name and save it together with the amount of food eaten to the database. Create a menu item, which displays a highscore table of the players for the 10 best scores. Also, create a menu item which restarts the game.

Short description how to use my program (user doc)

Use WASD keyboard buttons to control the movement of the snake's head.

Red circle is food. Snake can eat food and grow up.

Black circle is rocks, If the snake collides with a rock, then the game ends.

We also lose the game, if the snake goes into itself, or into the boundary of the game level.

When the game field is full, you can get to the next level, which has 5 more rocks.

The most interesting algorithms:

```
/**
   * check if we can create new food and rock.
   *
   * @return false if the snake is too long and no place to create food or
   * rocks. then game go to next level
   */
public boolean createFoodAndRock() {
```

```
    points = new ArrayList<Point>();
    for (int i = 0; i < 40; ++i) {
        for (int j = 0; j < 30; ++j) {
            points.add(new Point(i, j));
        }
    }
    Collections.shuffle(points);

    //Only one food piece is placed randomly at a time on the level
    //(on a field, where there is no snake).

    for (int i = 0; i < 1;) {
        Point point = points.remove(points.size() - 1);
        if (!mySnake.collideSnake(level.get(point))) {
            food = new Food((int) (20 * point.getX()), (int) (20 * point.getY()), 20, 20, foodImage);
            i++;
        }
    }

    //create random rocks
    //(on a field, where there is no snake and food).
    rocks = new ArrayList<Rock>();
    for (int i = 0; i < levelNum;) {
        Point point = points.remove(points.size() - 1);
        if (!mySnake.collideSnake(level.get(point))) {
            Rock rock = new Rock((int) (20 * point.getX()), (int) (20 * point.getY()), 20, 20, rockImage);
            rocks.add(rock);
            i++;
        }
    }

    //we check if  snake is too long and no place to create food and rocks , then game end.
    if (points.isEmpty()) {
        return false;
    }
    return true;
}


/**generate the movement of the snake
 *meanwhile  check if the snake head move to next position is also food there.
 *
 * @param food
 * @return true if head position equal food. false if not
 */
public boolean eat(Sprite food) {
    //head of snake:
    int firstX = snake.get(0).getX() + velx;
    int firstY = snake.get(0).getY() + vely;

    //The origin position of the tail of the snake:
    int tempx = snake.get(snake.size() - 1).getX();
    int tempy = snake.get(snake.size() - 1).getY();

    //every node of the snake equal previous node position:
    for (int i = snake.size() - 1; i > 0; i--) {
        snake.get(i).setX(snake.get(i - 1).getX());
        snake.get(i).setY(snake.get(i - 1).getY());
    }
    snake.get(0).plusX(velx);
```

```
        snake.get(0).plusY(vely);

        //If the snake eats a food piece, then its length grow by one unit.
        //if collide food ,add new tail of snake equal to old one
        if (firstX == food.getX() && firstY == food.getY()) {
           snake.add(new Sprite(tempx, tempy, 20, 20, snakeImage));
           return true;
        }
        //If the snake doesn't eats a food piece, then its length not change.
        return false;
     }
```

The connections between the events and event handlers.

```
this.getInputMap().put(KeyStroke.getKeyStroke("A"), "pressed left");
    this.getActionMap().put("pressed left", new AbstractAction() {
       @Override
       public void actionPerformed(ActionEvent ae) {
          mySnake.setVelx(-SNAKE_MOVEMENT);
          mySnake.setVely(0);
       }
    });
```

when you press on the button”A”, the head of snake will go left.

...Same for “WSD”.

```
class NewFrameListener implements ActionListener {


   @Override
   public void actionPerformed(ActionEvent ae) {
     if (!paused) {
       //First, we check next step if our snake eat food,
       if (mySnake.eat(food)) {  //if ture, we need to create new food and rock
          eatenFood++;
          if (!createFoodAndRock()) {   //to create new food and rock
             // if the board is full, you need to get to next level


             playername = JOptionPane.showInputDialog(null, "You won this level! You had " +
eatenFood
                   + " foods in " + elapsedTime() + " ms.\nPlease, enter your name: ", "Congrats!",
JOptionPane.PLAIN_MESSAGE);
```

```
                try {
                    highScores.putHighScore(playername, eatenFood);
                } catch (SQLException ex) {
                    Logger.getLogger(GameEngine.class.getName()).log(java.util.logging.Level.SEVERE, null,
ex);
                }
                restart();
            }
        } //if snake doesn't eat food, we need to check if the snake goes into itself,
        //or into the boundary of the game level.
        if (!mySnake.move()) {
            // not movable, means the snake collodes itself or the wall, game over.
            playername = JOptionPane.showInputDialog(null, "Game Over ! You had " + eatenFood
                    + " foods in " + elapsedTime() + " ms.\nPlease, enter your name: ", "Game_Over!",
JOptionPane.PLAIN_MESSAGE);

            try {
                highScores.putHighScore(playername, eatenFood);
            } catch (SQLException ex) {
                Logger.getLogger(GameEngine.class.getName()).log(java.util.logging.Level.SEVERE, null,
ex);
            }
            levelNum = 5;
            eatenFood = 0;
            restart();
        } else if (collideRocks()) {  //  if the snake collides rocks , game over

            playername = JOptionPane.showInputDialog(null, "Game Over ! You had " + eatenFood
                    + " foods in " + elapsedTime() + " ms.\nPlease, enter your name: ", "Game_Over!",
JOptionPane.PLAIN_MESSAGE);

            try {
                highScores.putHighScore(playername, eatenFood);
            } catch (SQLException ex) {
                Logger.getLogger(GameEngine.class.getName()).log(java.util.logging.Level.SEVERE, null,
ex);
            }
            levelNum = 5;
            eatenFood = 0;
```

```
            restart();
          }
        }
        repaint();
    }
  }
```

First, we check if our snake eat food, if ture, we need to create new food and rock.

When the board is full, you need to get to next level

If snake doesn't eat food, we need to check if the snake goes into itself, or into the boundary of the game level. If the snake collodes itself or the wall, game over.

if the snake collides rocks , game over.

When game is over, we ask for player name and add to database.

Class Diagram