Snake Readme File
April 2018

Welcome to the Snake game!

This file contains information to help you install "The Snake".

The Snake game is a JavaFX implementation of the classic arcade game, Snake.

[CONTENTS]

[1.1 SYSTEM REQUIREMENTS]

The game requires Java SE 8 or higher to run. Java Runtime Environment 8 (JRE 8) can be downloaded at http://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html

[1.2 INSTALLATION]

Double-click the executable zip file, setup.exe, and select the destination folder. The game can be run by double-clicking the SnakeGame.jar file in the game folder. The scores.db file needs to be present in the same folder or the game will not run properly.

[1.3 MANIFEST]

/game
SnakeGame.jar
scores.db

[1.4 SOURCE CODE]

The source code is found in the source folder. It is run through the main function of the Main class which extends Application and launches JavaFX. Snake, Food and Direction classes provide logical support for the game objects and the Segment class handles the snake’s images. The introductory Scene is provided by an instance of the StartScreen class and the database for the highest scores is managed by static methods in the DBManager class. High scores are displayed by an instance of the HighScoreBox class. Input and validation are provided by instances of the InputBox and AlertBox classes.

Logically the snake is the Group of Nodes that make up its body. When the snake grows Nodes are added to the Snake's ObservableList. Once the snake has a tail, it moves by positioning the last Node in the snake's ObservableList to the front of the snake:

```
public class Directions {

    ...

    /** moveInDirection method
     * Moves the given Node to the head of the snake */
    public static void moveInDirection(Directions direction, Snake snake,
Node tempSegment) {
      switch (direction) {
          case UP:

      tempSegment.setTranslateX(snake.getSnake().get(0).getTranslateX());


      tempSegment.setTranslateY(snake.getSnake().get(0).getTranslateY() -
Main.BLOCK_SIZE); // Move up
            break;

          ...

      }
    }
```

The food is displayed by calling its ImageView's getter and is positioned by calling its randomMove method. The snake displayed to the player is handled in the Main class' KeyFrame by the Segment class' static changeSegmentView method based the Snake's Length, Node position in the Snake's ObservableList and the arrow key that the player pressed. The first Node in the list is always the head and the remaing Nodes are always body segments:

```
    /** Public changeSegmentView method
     * Returns the ImageView of the Snake's body for the Snake's
direction given as a char */
    public static ImageView changeSegmentView(char dir) {
        switch (dir) {

          ...

            case 'R':
                segmentView = new ImageView(headRight);
                return segmentView;
            case 'S':
                segmentView = new ImageView(segment);
                return segmentView;
            default:
                return null;
        }
    }
```

If the player presses the spacebar the pauseGame method is called to pause the Timeline, or the resume method is called to resume the TimeLine if the game is already paused. Collision detection is done in the Main's KeyFrame for both scoring and ending the game. When the player scores the Text message bar at the top of the game's BorderPane is updated. When the game ends the stopGame method is called. stopGame resets the game to its inital state and pauses the Timeline. The checkScore method is called to determine if the player beat a high score. checkScore gets a list of the high scores from the database using DBManager's getScores method:

```
    /** public getScores
     * returns arrays of the database's names and scores as a Pair */
    public static Pair<String[], int[]> getScores() {

        ...

            ResultSet rs = stmt.executeQuery("SELECT * FROM SCORES ORDER
BY SCORE DESC LIMIT 5;");
            int i = 0;
            while (rs.next()) {
                names[i] = rs.getString("name");
                scores[i] = rs.getInt("score");
                i++;
            }

      ...

        return new Pair<>(names, scores);
    }
```

If the player beat a high score, the Main class creates an instance of InputBox to get the player's initials and update the database. The input is validated to ensure that the user enters 3 letters. An instance of AlertBox is create to display an error validation message to the player. The InputBox calls DBManager's insertHighScore method which uses its private delete method to remove the lowest score and insert the new score using the removed entry's the primary key.

```
    /** public insertHighScore method
     * inserts a given score and name into the database
     * uses private insertScore and delete methods to so this */
    public static void insertHighScore(String name, int score) {
        insertScore(name, score, delete());
    }

    /** private insertScore method
     * inserts a new entry into the database given its primary key id,
     * name and score */
    private static void insertScore(String name, int score, int id) {

      ...

    }
```

```
    /** private delete method
     * removes the last entry from the database and returns
     * its primary key */
    private static int delete() {

          ...

            // Read the id of the lowest score
            ResultSet rs = stmt.executeQuery("SELECT * FROM SCORES ORDER
BY SCORE ASC LIMIT 1;");

          ...

            // Remove the entry with that primary key;
            String sql = "DELETE from SCORES where ID=" + id + ";";

          ...

        return id; // Return the primary key
    }
```

Whether the player has beat a high score or not, an instance of HighScoreBox is created to display the high scores to the player. When the HighScoreBox is closed it sets a static boolean in Main in order to communicate that the game can be reset. The game scene then switches to the intro scene and the introductory scene is displayed to the player. The player can play all over again.

[2.1 GAMEPLAY]

Upon running for the first time an introductory scene is displayed with the gameplay instructions, the scene switches to the game with a mouse click.

The player chooses the direction of the snake using the arrow keys on the keyboard. The player may pause the current game using the spacebar. An apple appears on at random positions in the game and the player’s score increases when the snake’s head is successfully navigated to the apple.

The length of the snake increases after scoring. If the snake runs past the edge of the screen or into itself, the game ends. There are 5 difficulty levels and the length that the snake increases per point increase with each level. When the game ends, the high scores are displayed and the game returns to the introductory scene. If the player has beat a high score they are prompted to enter their initials.

[3.1 AUTHORS]

(Svyatoslav Loboda, Fraser Raney, Shivam Parekh, and Jiatao Wen)

[3.2 LICENSE]

This program is free for use with modification. If it can help others learn more about JavaFX, JDBC, and programming in general, we are all happy.

[3.3 ACKNOWLEDGMENTS]

The graphics used in the game were provided under free-for-use-with-modification license. The original game algorithm by Almas Baimagambetov (https://www.youtube.com/watch?v=nK6l1uVlunc) was completely redesigned. Tutorials Point is a great resource for learning about JDBC (https://www.tutorialspoint.com/sqlite/sqlite_java.htm). There are great other great tutorials for JavaFX available on YouTube.com from thenewboston at https://www.youtube.com/channel/UCJbPGzawDH1njbqV-D5HqKw.