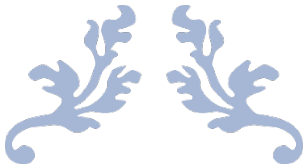


DOCUMENTATION OF THE 3RD ASSIGNMENT (EX: 2)

Programming Technology

DECEMBER 15, 2022

ELMOUMEN YOUSSEF | NEPTUN CODE: FVW745
ELTE

Table of Contents

I. INTRODUCTION

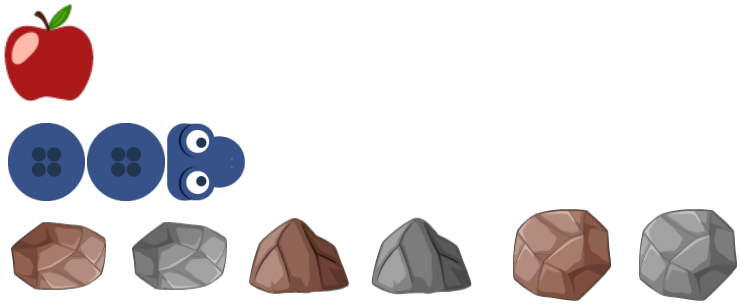
During the Programming Technology subject, three assignments should be written as a part of the final grade.
In this document, I'll be explaining the process of how I've implemented the third one and specifically the exercise number 2(Snake Game). I'll start with a brief description about it. Then, the UML class diagram, the project structure, the relationship between the action listener and their handlers, the classes and their special methods and finally a list of test cases I have done.

II. EXERSICE DESCRIPTION

The common requirements and goal behind the assignment as stated in the file is as follows:

- Friendly and easy to use User Interface.
- Oriented Object Implementation
- Use simple graphics for the game display.
- The user should be able to control the snake with WASD keyboard buttons.
- Generate the game levels with an algorithm, or simply load them from files.
- Create at least 10 predefined levels. If you generate the levels.
- The levels should be playable (player should be able to solve it).
- Each game needs to have a timer, which counts the elapsed time since the start of the game level.
- The documentation should contain the description of the task, its analysis, the structure of the program (UML class diagram), a chapter for the implementation, which describes the most interesting algorithms (e.g. that generates the level) etc. of the program. Also, the connections between the events and their handlers.
- The task description should contain the minimal requirements.

However, the exercise 2 of this assignment aims to build a snake game which can be played on a desert (board).
The main components of this snake game are the snake, the apple (food), the rock, and the desert which is simply the game board.

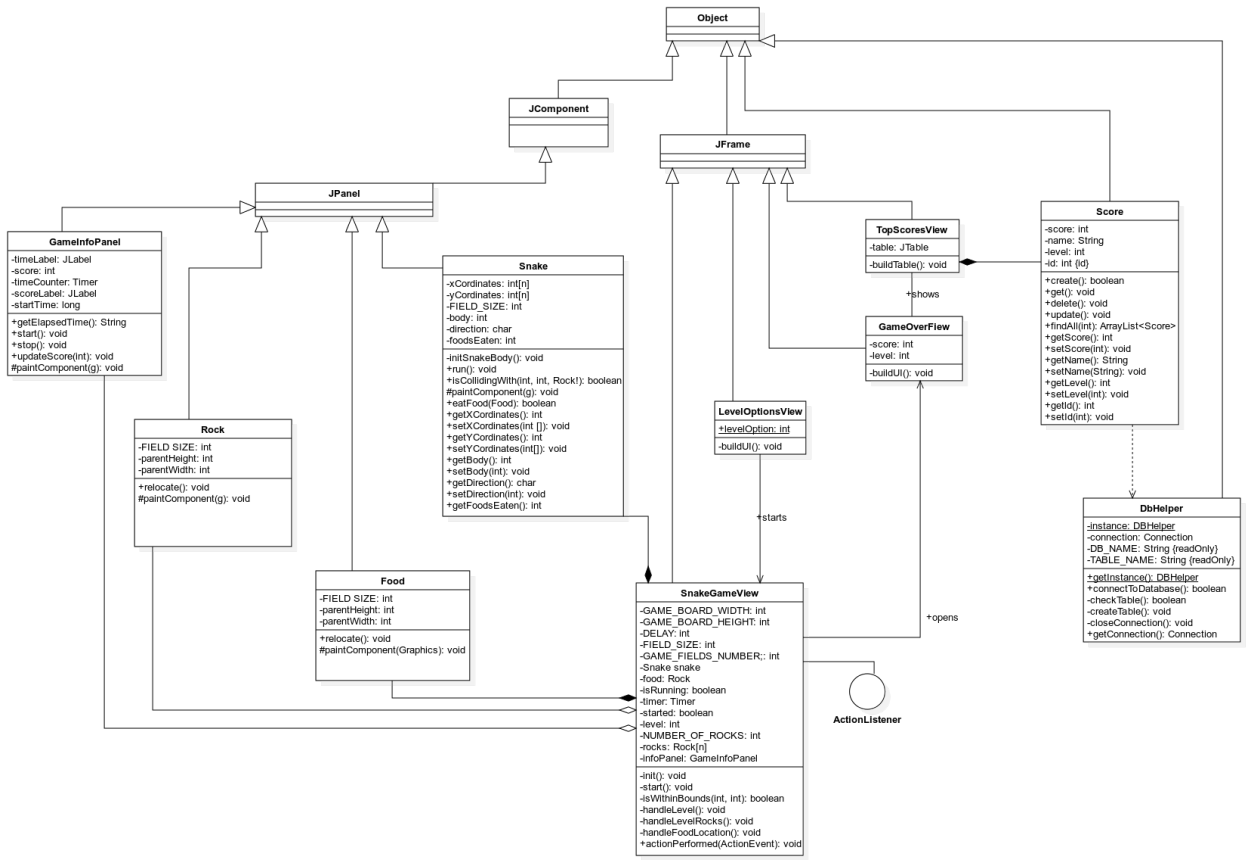


Initially, the snake is two units long. It starts off from the center in a random direction and then collects the apple that appears and is placed randomly at a time.
The player can control the movement of the snake's head with keyboard buttons to eat and collect the apples as mentioned before, he should also avoid hitting the rocks which make the game harder.

The game is over whenever the snake hits itself, the walls, or one of the rocks that we placed randomly as well.
After that, a window shows that the game is over and the score achieved (number of apples eaten) the user can choose to save that score, restart the game or consult the top scores table.

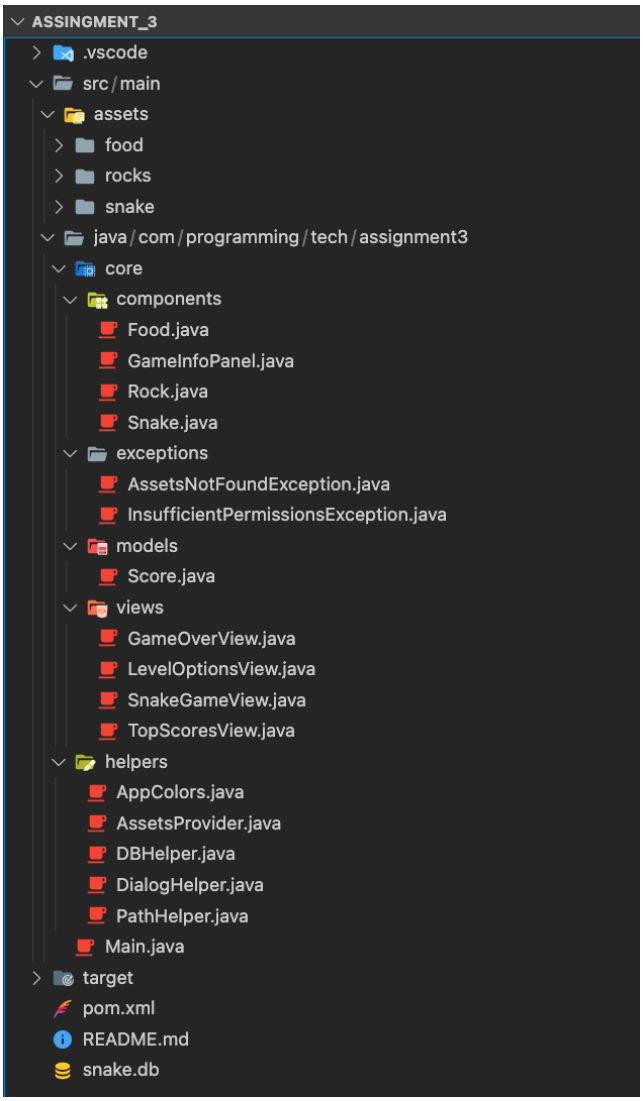
III. UML CLASS DIAGRAM

The following is the class diagram of our exercise:



IV. PROJECT STRUCTURE

The following image illustrates the project structure.



src/*: a folder contains all the building blocks of this program.

/assets/: a folder that contains the resources including apple icon, snake parts, and also the rocks images.

***/components/*:** contains the building blocks of the game: Food, Rock, GameInfoPanel and the most import the Snake component.

***/models/*:** contains the Models of the game such as **Score** model.

***/views/*:** contains the views of the game

LevelOptionsView: to select a level form 1 to 10

SnakeGameView: the play board view

GameOverView: A view that appears after the game is over with save score, restart, and show top scores options

TopScoresView: a view that displays a table of the top 10 scores

***/exceptions/*:** custom exception file like

InsufficientPermissionsException: that is thrown if the app doesn't have the sufficient permissions to read or write to the device.

AssetsNotFoundException that we throw when the ***/assets/*** folder or one of the game assets is missing.

***/helpers/*:** this folder contains a set of helpful classes :

AppColors: class that has colors of the game.

AssetsProvider: this class is the handler of this project's assets

DBHelper: manages the database connections as well as its configurations

DialogHelper: handles the dialogs of success, error and confirmation states

PathHelper: this class will convert the path to the current OS path syntax to make sure that all the assets and the access to program's files requests works properly because UNIX based operating systems and Windows have different path syntaxes

V. METHODS DESCRIPTION:

In this section, I'm providing a short description of each method of the classes of the exercise:

class Food

- public void relocate(): Relocates the food to a random position and sets its bounds
- @Override protected void paintComponent(Graphics g) Paints the food on the game board using the food image, if the asset is not found it will be painted as a red square

class Rock

- public void relocate(): Relocates the food to a random position and sets its bounds
- @Override protected void paintComponent(Graphics g) Paints the food on the game board using the food image, if the asset is not found it will be painted as a brown square

class GameInfoPanel

- public String getElapsedTime() This method returns the elapsed time in the format hh:mm:ss
- public void start(): This method starts the timer and sets the start time
- public void stop(): This method stops the timer
- public void updateScore(int score) This method updates the score
- @Override protected void paintComponent(Graphics g) Refreshes the time label using the getElapsedTime method
- public Timer getTimeCounter() Time counter getter

class Snake

- public Snake(int totalFieldUnits, int unit) constructor for the snake class, it takes the total number of field units and the size of each unit as parameters then inits the snake's body
- private void initSnakeBody() inits the snake's body by randomizing its head position and setting the
- public void run() runs the snake's body by moving each part of the body to the position of the part in front of it then moves the head to the next position based on the direction the snake is moving in
- public boolean isCollidingWith(int verticalWall, int horizontalWall, Rock rocks[]) checks if the snake is colliding with itself or the walls or the rocks

- @Override protected void paintComponent(Graphics g) paints and refreshes the snake's body
- public boolean eatFood(Food food) checks if the snake is eating the food, if it is, it increases the snake's body and the number of foods eaten
- public void setDirection(int key) handles the direction of the snake based on the key pressed by the user. Arrows or SWAD keys are used to control the snake

class Score
- public boolean create() throws SQLException Saves this score to the database
- public void get() throws SQLException fetchs a score from the database using this instance's id
- public void delete() throws SQLException removes this score from the database
- public void update() throws SQLException updates this score in the database
- public static ArrayList<Score> findAll(int limit) throws SQLException fetchs a list of the top scores from the database

class GameOverView
- private void buildUI() builds the UI of the GameOver View and handles the logic of the restart, save score and top scores buttons.

class LevelOptionsView
- private void buildUI() builds the UI of the LevelOptions View and manages the user selection after the user selects a level and clicks on the start button, this view disposes and the game starts

public class SnakeGameView
- public SnakeGameView() Constructor for the SnakeGameView class, it reads the LevelOption from the LevelOptionsView then inits the game, the game starts after the user presses on a key
- private void init() Initializes the game, by setting the size of the window, the background color, the number of fields in the game board, the snake, the food, the rocks, the info panel, and the key listener..etc
- public void start() Starts the game by initializing the timer, the snake and setting the isRunning to true
- private boolean isWithinBounds(int x, int y) Handles the proper location of the rocks & food
- private void handleLevel()
  Handles the logic of the levels of the game
  delays are calculated with the following formula: 170-(level-1) *10
  which produces the following delays (slowest to fastest)=> 170, 160, 150,140,130, 120, 110, 100,90, 80
  Rocks by the fomula: level + 2; => Results: 3,4,5,6,7,8,9,10,11,12

  private void handleLevelRocks() Handles the rocks in the game based on the Number of rocks in the level.
  It creates an array of rocks and adds them to the game board on ramdon & fixed locations

- public void handleFoodLocation() Handles the food location it relocates the food on a random location then checks if the location is good if not it relocates it again
- public class SnakeGameKeyAdapter extends KeyAdapter: Handles the key events. if the game is not started, it starts it & the timer of the game information panel
- @Override public void actionPerformed(ActionEvent e)
  Handles the timer of the game
  if the game is not started, it does nothing
  if the game is running, it runs the snake, and checks if it eats food, if it does it updates the score and handles the food location
  if the snake collides with the rocks, the walls or itself it stops the timer and displays the game over screen

public class TopScoresView
- private void buildTable() Builds the table that shows the top scores by fetching the top 10 scores from the database and displaying them in that table

public class AssetsProvider
- public static Image getSnakeHead(char direction) returns the snake head image based on the direction of the snake
- public static Image getRockImage(int rockId) returns the rock image based on the rock id
- public static void handleAppAssetsExistance() throws AssetsNotFoundException checks if the project has the required assets to run
- public static void handleFilePermissions() throws InsufficientPermissionsException checks if the project has the required permissions to run

public class DBHelper
- public static DBHelper getInstance() returns the singleton instance of this class
- public boolean connectToDatabase() connects to the database, it creates the database if it doesn't exist and as well as the table scores if it also doesn't exist
- public boolean checkTable() checks if the table exists and returns a boolean value accordingly
- public void createTable() creates the table if it doesn't exist
- public void closeConnection() throws SQLException closes the connection to the database
- public Connection getConnection() returns the instance of the connection to the database to be used by other classes

public class DialogHelper
- public static void showErrorMessage(Component parent, String message) shows an error message

- `public static void showErrorMessage(Component parent, String title, String message)` shows an error message but with a custom title
- `public static void showInfoMessage(Component parent, String message)` shows an information/success message
- `public static int showConfirmationDialog(Component parent, String message)` shows a confirmation dialog

`public class PathHelper`

- `public static String toCurrentOSPathSyntax(String unixPathSyntax)` converts the path to the current OS path syntax
- `public static boolean isWindows()` if the current OS is windows this function will return true
- `public static String getProjectLocation()` returns the current project directory [the location of Main.java file]

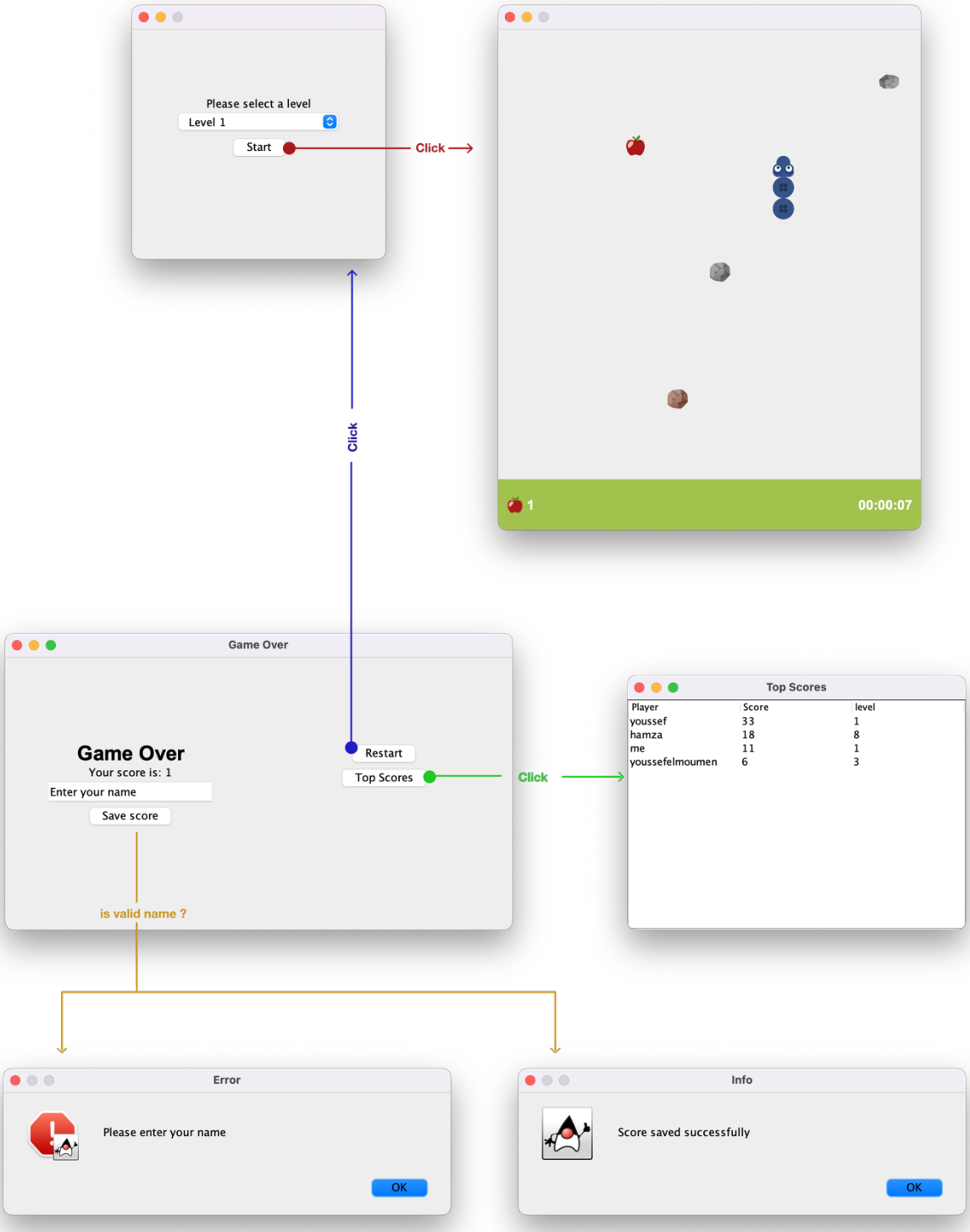
VI. LEVELS LOGIC

The logic is handled by the following method `SnakeGameView().handleLevel()` that simply uses:

- The formula: $delay = 170 - (level - 1) * 10$ to calculate the delays and this produces the following ones (from slowest to fastest: 170, 160, 150,140,130, 120, 110, 100,90, 80 for the levels 1,2,3,4,5,6,7,8,9,10

- Regarding the number of the rocks that increases on each upper level the formula is as follows: $nRocks = level + 1$ then Results will 3,4,5,6,7,8,9,10,11,12 for the levels 1,2,3,4,5,6,7,8,9,10

VII. CONNECTIONS BETWEEN THE EVENTS AND EVENT HANDLERS

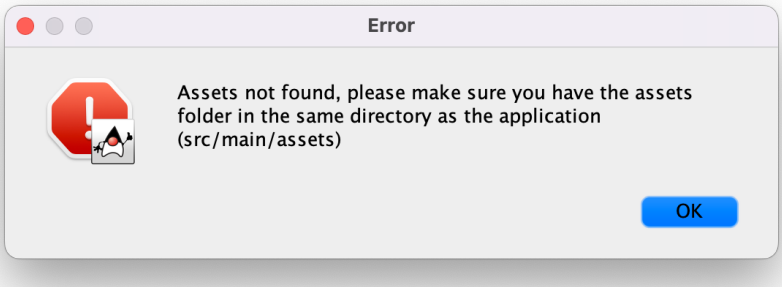


VIII. LIST OF TEST CASES

1. Remove/Rename a resource

In this case the program catches the AssetsNotFoundException, shows an error then quits the game

I tried to **remove** the image **rock_1.png** as an example and the following dialog message appears:



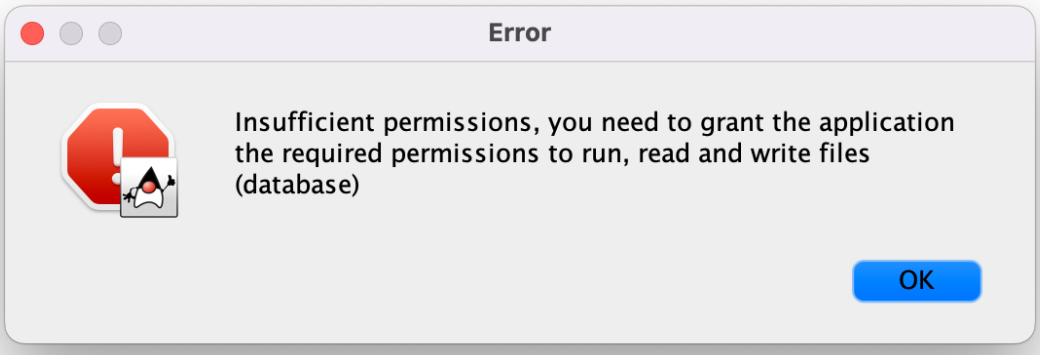
2. Removing the read/write permission

In this case the program catches the InsufficientPermissionsException, shows an error and quits the game.

I tried to remove the write permission by executing the command

chmod -w assignment_3

The following dialog message appears



3. Hitting the boundary, a rock or the snake itself

The game ends and shows the GameOverView

4. Removing the snake.db database during the runtime

In this case, a new database will be created once the user calls the DbHelper

5. Removing the snake.db database before the startup

A database with the name snake.db created and the game starts working properly