Assignment 3

Programming Technology

Tron Game

Md Rakibul Hasan

JR9RXP

Task:

Create a game, with we can play the light-motorcycle battle (known from the Tron movie) in a top view. Two players play against each other with two motors, where each motor leaves a light trace behind of itself on the display. The motor goes in each seconds toward the direction, that the player has set recently. The first player can use the WASD keyboard buttons, while the second one can use the cursor buttons for steering. A player loses if its motor goes to the boundary of the game level, or it goes to the light trace of the other player. Ask the name of the players before the game starts, and let them choose the color their light traces. Increase the counter of the winner by one in the database at the end of the game. If the player does not exit in the database yet, then insert a record for him. 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.

Game Mechanism:

Controls:

First Player: Uses the WASD keys for navigation.

Second Player: Uses the arrow keys (cursor buttons) for steering.

Objective: Each player controls a motor that leaves a light trail behind. The goal is to avoid crashing into the game boundary or the opponent's light trail.

Losing Conditions: A player loses if their motor hits the game level boundary or intersects with the light trace of the other player.

1. Game Initialization

Main Window (MainWindow): Serves as the primary interface for the game. It initializes the game environment, including the start menu, game board, and database connection for leaderboards.

Start Menu: Allows players to input their names and select colors before starting the game. It also provides options to view the leaderboard.

Database Connection (DataBaseSQL): Establishes a connection to a SQL database for storing and retrieving game scores.

2. Game Components

Game Logic (Game): Manages the game state, including loading levels, handling player movements, and keeping track of the game board.

Game Levels (GameLevel): Represents individual levels in the game, including player positions, obstacles, and the layout of the level.

Game Board (Board): A graphical component that displays the current state of the game, including the game level, player positions, and obstacles.

Player Positions (Position): Represents the coordinates of players in the game.

3. Gameplay

Key Events: The MainWindow listens for key events to control player movements and other game actions, such as pausing the game or restarting a level.

Player Movement: Player positions are updated based on key inputs. The Game class handles the logic for moving players and checking for collisions or invalid moves.

Board Refreshing: The game board is refreshed to reflect the updated positions of players and changes in the game state.

4. Level Progression

Level Loading: The game loads levels with specific challenges and layouts from the **''Level.txt".** Players progress through these levels as they complete each one.

Scoring and Leaderboard: Scores are updated based on game outcomes. High scores are stored in the database and can be viewed through the leaderboard.

5. Game Scaling and Graphics

Image Rendering: The Board class uses images to represent players and level items, which are scaled and rendered according to the current game state.

Dynamic Scaling: Players can adjust the scale of the game board for a customized view.

6. Database and Leaderboard

Score Insertion: After each game, scores are inserted into the database.

Leaderboard Retrieval: The leaderboard displays the top scores retrieved from the database.

7. Game Termination

Game Over: The game concludes after reaching the final level or upon the player's choice.

Database Closure: The database connection is closed properly when the game ends.

Method Descriptions:

1. Game.Java:

Game(): Initializes the game, setting up levels and mappings.

load(GameID gameID): Loads a game level based on a given GameID.

printGameLvl(): Prints the current game level, if loaded.

ColorToField(int x, int y, String stg): Maps a color string to a field on the game level.

isValidPos1(): Checks if the position for player 1 is valid in the current level.

isValidPos2(): Checks if the position for player 2 is valid in the current level.

move1(Direction d1): Moves player 1 in the specified direction.

move2(Direction d2): Moves player 2 in the specified direction.

getDifficulties(): Returns a collection of all difficulty levels.

isLvlLoaded(): Checks if a level is currently loaded.

readLvl(): Reads level data from a resource file.

addNewLvl(GameLevel gameLevel): Adds a new level to the game.

Map<String, LevelItem> initializeColorMappings(): Initializes mappings between color strings and LevelItem objects.

2. GameID:

int hashCode(): Overrides hashCode() to generate a hash code based on mode and lvl.

boolean equals(Object obj): Overrides equals() to compare this GameID object with another object for equality, based on mode and lvl.

3. GameLevel:

initializeLevel(ArrayList<String> gameLevelRows): Initializes the level layout based on input strings.

setItem(int i, int j, char c): Sets an item at a specific position in the level.

fillRow(int row, int startCol): Fills a row with default items from a specified column.

calcMaxCol(ArrayList<String> gameLevelRows): Calculates the maximum number of columns in the level.

isValidPos1(): Checks if the position of player 1 is valid.

isValidPos2(): Checks if the position of player 2 is valid.

player1step(Direction d1): Moves player 1 in the specified direction.

player2step(Direction d2): Moves player 2 in the specified direction.

printLvl(): Prints the current state of the game level.

4. Board:

initializeImages(): Loads images for each LevelItem and player images.

fixScale(double scale): Updates the scaling factor of the board and refreshes it.

getNew_size(): Returns the current scaled size of the tiles.

isValidPosP1(): Checks if Player 1's position is valid.

isValidPosP2(): Checks if Player 2's position is valid.

refresh(): Refreshes the board's dimensions and repaints it.

getPlayerIMG(Direction direction): Loads the image for a player based on their direction.

setP1IMG(Direction d1): Sets the image for player 1 based on their direction.

setP2IMG(Direction d2): Sets the image for player 2 based on their direction.

paintComponent(Graphics g): Custom painting method for the board, draws the level and players.

5. MainWindow:

initializeDatabase(): Initializes the database connection.

gameStartingMenu(): Sets up the starting menu interface.

initializeGame(): Initializes the game logic.

initializeWindow(): Configures the main window settings.

initializeMenuBar(): Sets up the menu bar.

lvlMenuItems(JMenu menu): Creates menu items for level selection.

scalingItems(JMenu menu, double from, double to, double by): Adds scaling options to the menu.

initializeGameStatLabel(): Initializes the game status label.

initializeBoard(): Initializes the game board.

elapsedTimeM(): Returns the elapsed time in milliseconds.

startGameTimer(): Starts a timer for the game.

startGame(): Handles logic for starting the game.

ActionListener taskPerformer: ActionListener for game update events.

updateGame(): Updates game state and player positions.

updatePlayerPositions(): Updates positions of players on the board.

declareWinner(String winner, String loser): Declares a winner and updates scores.

showLeaderboard(): Displays the leaderboard.

updatePlayerDirectionAndColor(Direction oldDirection, Direction newDirection, Position position, Color color): Updates player direction and color on the board.

getColorDirection(Direction oldDir, Direction newDir, Color color): Returns string representation of the direction based on color.

getColorPrefix(Color color): Returns prefix for a given color.

6. Pair:

Public T getFirst(): Returns the first element of the pair.

public U getSecond(): Returns the second element of the pair.

public boolean equals(Object o): Overrides the equals method to compare this pair with another object for equality.

public int hashCode(): Overrides the hashCode method to generate a hash code based on both elements of the pair.

7. DataBaseSQL:

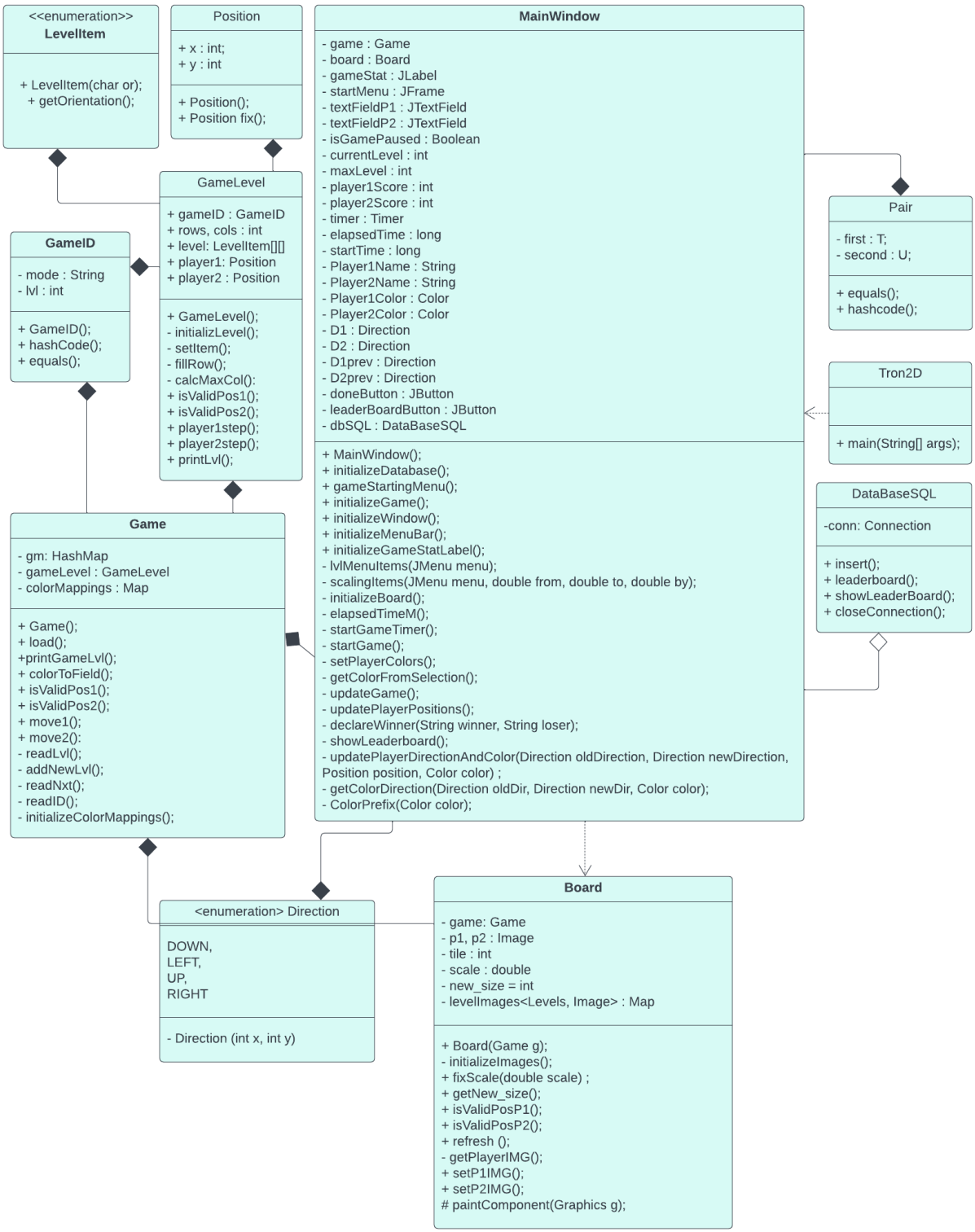
insert(String winnerName, int score): Inserts the winner's name and score into the database. If the insertion fails, it shows an error message.

leaderboard(): Retrieves the leaderboard from the database, ordered by player score in descending order. It then calls showLeaderBoard to display the results. If there's an error, it shows an error message.

showLeaderBoard(String[][] scores): Displays the leaderboard in a JTable within a message dialog.

closeConnection(): Closes the database connection. If there's an error during closure, it shows an error message.

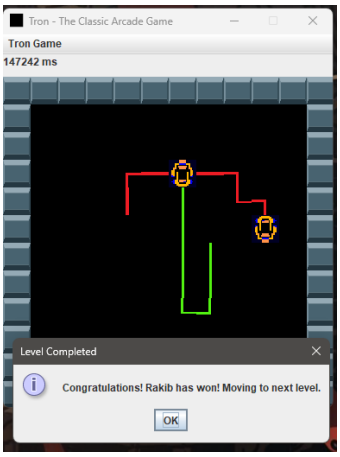
UML:


<<enumeration>>
LevelItem
+ LevelItem(char or);
+ getOrientation();
Position
+ x : int;
+ y : int
+ Position();
+ Position fix();
MainWindow
- game : Game
- board : Board
- gameStat : JLabel
- startMenu : JFrame
- textFieldP1 : JTextField
- textFieldP2 : JTextField
- isGamePaused : Boolean
- currentLevel : int
- maxLevel : int
- player1Score : int
- player2Score : int
- timer : Timer
- elapsedTime : long
- startTime : long
- Player1Name : String
- Player2Name : String
- Player1Color : Color
- Player2Color : Color
- D1 : Direction
- D2 : Direction
- D1prev : Direction
- D2prev : Direction
- doneButton : JButton
- leaderBoardButton : JButton
- dbSQL : DataBaseSQL
+ MainWindow();
+ initializeDatabase();
+ gameStartingMenu();
+ initializeGame();
+ initializeWindow();
+ initializeMenuBar();
+ initializeGameStatLabel();
- lvlMenuItems(JMenu menu);
- scalingItems(JMenu menu, double from, double to, double by);
- initializeBoard();
- elapsedTimeM();
- startGameTimer();
- startGame();
- setPlayerColors();
- getColorFromSelection();
- updateGame();
- updatePlayerPositions();
- declareWinner(String winner, String loser);
- showLeaderboard();
- updatePlayerDirectionAndColor(Direction oldDirection, Direction newDirection, Position position, Color color) ;
- getColorDirection(Direction oldDir, Direction newDir, Color color);
- ColorPrefix(Color color);
Pair
- first : T;
- second : U;
+ equals();
+ hashcode();
GameLevel
+ gameID : GameID
+ rows, cols : int
+ level: LevelItem[][]
+ player1: Position
+ player2 : Position
+ GameLevel();
- initializLevel();
- setItem();
- fillRow();
- calcMaxCol():
+ isValidPos1();
+ isValidPos2();
+ player1step();
+ player2step();
+ printLvl();
GameID
- mode : String
- lvl : int
+ GameID();
+ hashCode();
+ equals();
Tron2D
+ main(String[] args);
DataBaseSQL
-conn: Connection
+ insert();
+ leaderboard();
+ showLeaderBoard();
+ closeConnection();
Game
- gm: HashMap
- gameLevel : GameLevel
- colorMappings : Map
+ Game();
+ load();
+printGameLvl();
+ colorToField();
+ isValidPos1();
+ isValidPos2();
+ move1();
+ move2():
- readLvl();
- addNewLvl();
- readNxt();
- readID();
- initializeColorMappings();
<enumeration> Direction
DOWN,
LEFT,
UP,
RIGHT
- Direction (int x, int y)
Board
- game: Game
- p1, p2 : Image
- tile : int
- scale : double
- new_size = int
- levelImages<Levels, Image> : Map
+ Board(Game g);
- initializeImages();
+ fixScale(double scale) ;
+ getNew_size();
+ isValidPosP1();
+ isValidPosP2();
+ refresh ();
- getPlayerIMG();
+ setP1IMG();
+ setP2IMG();
# paintComponent(Graphics g);

Test Cases:

1. Player1 loses as it goes out of the boundary:

2. Player2 loses as it runs into the light trail of the Player1:



3. Player1 loses as it crashes to a wall: