Third Assignment, Task 4 (Tron)

Name: Komron Fayziboev
Neptun code: B9227I
Email: b9227i@inf.elte.hu
Group: 8

Description

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.

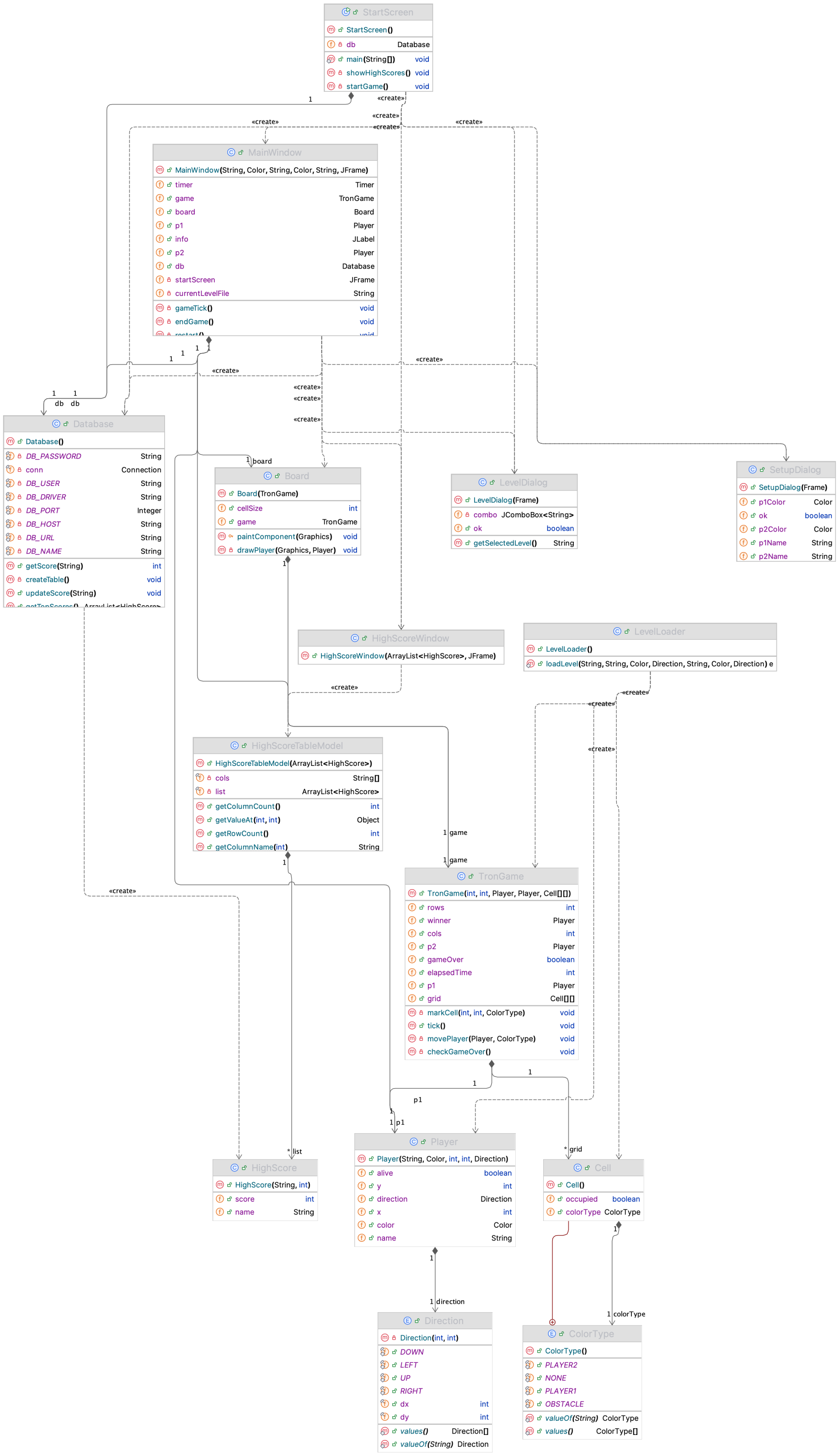
Analysis and Solution Plan

The solution involves a modular design with clearly defined responsibilities for each class and subsystem. Key components include:

- *Model*: Defines the core game logic, including the TronGame, Player, Direction, and Cell classes.
- *Persistence*: Manages high scores and database interactions using the Database class.
- *View*: Provides the UI components, including StartScreen, MainWindow, and LevelDialog.

The plan involves developing the classes iteratively, starting with the core game mechanics and gradually integrating UI and database functionality. Edge cases and performance considerations are addressed during testing.

Class Diagram



Method Descriptions

1. **`tick()`**:
   - **Purpose**: Advances the game by one tick, moving both players' motorcycles based on their current directions. Updates the game state, including checking for collisions or game over.
   - **Key Logic**:
     - Moves each player using `movePlayer()` and checks if the player collides with the boundary, obstacles, or any existing light trail.
     - Updates the elapsed time counter.
     - Calls `checkGameOver()` to determine if the game should end.
2. **`movePlayer(Player p, Cell.ColorType ct)`**:
   - **Purpose**: Handles the movement logic for a player's motorcycle, ensuring it follows the rules of movement and collision detection.
   - **Key Logic**:
     - Calculates the next position based on the current direction.
     - Checks for boundary collision or collision with light trails (including self-trails).
     - Updates the player's position and marks the new cell with the player's trail.
3. **`checkGameOver()`**:
   - **Purpose**: Determines if the game has ended due to a collision or both players losing simultaneously.
   - **Key Logic**:
     - Evaluates if both players are no longer alive.
     - Identifies the winner if one player remains alive, or sets the result to a draw.
4. **`markCell(int x, int y, Cell.ColorType ct)`**:
   - **Purpose**: Updates the game board by marking a cell as occupied with a specific player's trail.
   - **Key Logic**:
     - Sets the `occupied` status of the cell.
     - Assigns the cell's `colorType` to the respective player or obstacle.
5. **`updateScore(String name)`** (Database class):
   - **Purpose**: Updates the player's score in the database after a game ends.
   - **Key Logic**:
     - Increments the score by one for the winning player.
     - Uses an upsert SQL query to insert a new record if the player does not exist or update an existing record.
6. **`loadLevel(String levelFile, String p1Name, Color p1Color, Direction p1Dir, String p2Name, Color p2Color, Direction p2Dir)`** (LevelLoader class):

  - **Purpose**: Reads the game board layout from a level file and initializes the grid, obstacles, and player positions.
  - **Key Logic**:
    - Reads the level file line by line to construct the grid.
    - Identifies obstacle cells and assigns starting positions for players.
7. **`restart()`** (MainWindow class):
   - **Purpose**: Resets the game and navigates back to the setup or level selection dialog.
   - **Key Logic**:
     - Disposes of the current game window.
     - Reinitializes player details and loads the selected level for a new game.
8. **`endGame()`** (MainWindow class):
   - **Purpose**: Handles the logic for game termination, including updating scores, showing results, and returning to the start screen.
   - **Key Logic**:
     - Stops the game timer.
     - Displays a winner or draw message.
     - Updates the database with the winning player's score.

Connections Between Events and Event Handlers

The following are the connections between major events and their corresponding event handlers in the game:

- **Start Game Button Click**

  - **Event**: Clicking the "Start Game" button on the `StartScreen`.
  - **Handler**: ActionListener attached to the button in the `StartScreen` class.
  - **Functionality**: Opens the `SetupDialog` to gather player details and then the `LevelDialog` to select a level. Starts the `MainWindow` with the chosen level.

- **Level Selection Dialog**

  - **Event**: Clicking "Select" in the `LevelDialog`.
  - **Handler**: ActionListener attached to the "Select" button.
  - **Functionality**: Stores the selected level file and closes the dialog to proceed with game initialization in the `MainWindow`.

- **Player Movement**

  - **Event**: Pressing WASD or arrow keys during gameplay in the `MainWindow`.
  - **Handler**: KeyListener in the `MainWindow` class.
  - **Functionality**: Updates the direction of the corresponding player by modifying their `direction` attribute.

- **Game Tick**
  - **Event**: Timer firing at regular intervals in the `MainWindow`.
  - **Handler**: Timer's ActionListener in the `MainWindow` class.
  - **Functionality**: Calls the `tick()` method in the `TronGame` class to update player positions, check collisions, and determine the game's status.

- **Game Over**
  - **Event**: Triggered when a player collides with a boundary, obstacle, or light trail.
  - **Handler**: `checkGameOver()` in the `TronGame` class.
  - **Functionality**: Ends the game and calls `endGame()` in `MainWindow` to handle post-game logic, such as updating scores and returning to the start screen.

- **Restart Game**
  - **Event**: Clicking the "Restart" menu item in the `MainWindow`.
  - **Handler**: ActionListener attached to the menu item.
  - **Functionality**: Calls the `restart()` method to reinitialize the game setup.

- **High Scores**
  - **Event**: Clicking the "High Scores" menu item in the `MainWindow` or `StartScreen`.
  - **Handler**: ActionListener attached to the menu item.
  - **Functionality**: Retrieves top scores from the database and displays them in a `HighScoreWindow` dialog.

- **Exit Button**
  - **Event**: Clicking the "Exit" button on the `StartScreen`.
  - **Handler**: ActionListener attached to the button.
  - **Functionality**: Exits the application using `System.exit(0)`.

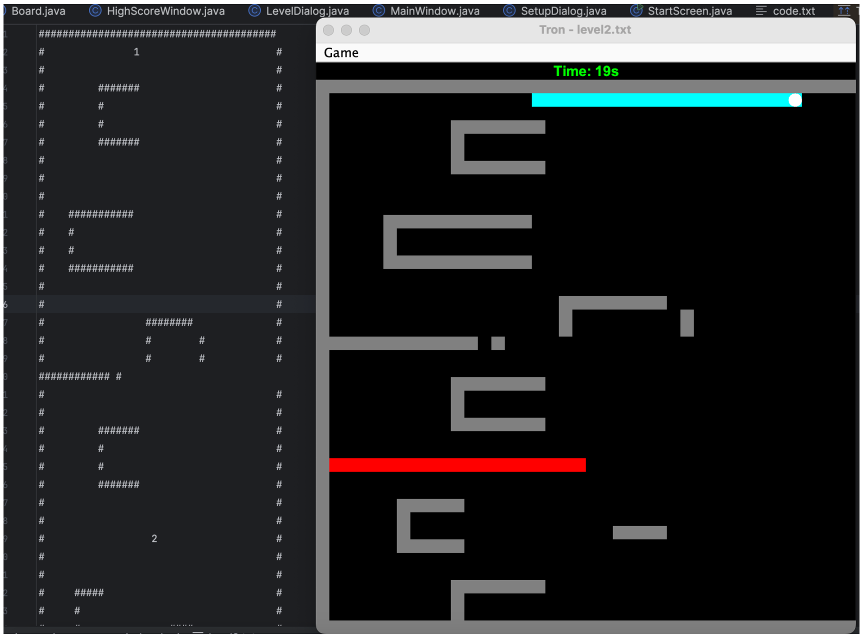
Test Report

Black Box Test Cases

1. **Test Case 1: Player Color Selection**
   - **AS A player,**
   - **I WANT TO** choose my motor color,
   - **GIVEN** the player setup dialog,

- **WHEN** I select a color,
- **THEN** the game should start with the motor trail of my chosen color.

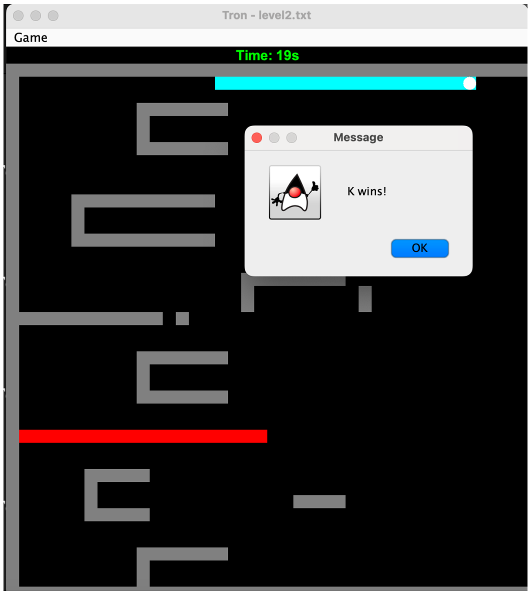
2. **Test Case 2: Level File Loading**
    - **AS A player,**
    - **I WANT TO** select a predefined level,
    - **GIVEN** the level selection dialog,
    - **WHEN** I choose a level file,
    - **THEN** the game grid should match the layout specified in the chosen file.



```
#########################################
#               1                       #
#                                       #
#         #######                       #
#         #                             #
#         #                             #
#         #######                       #
#                                       #
#                                       #
#                                       #
#    ###########                        #
#    #                                  #
#    #                                  #
#    ###########                        #
#                                       #
#                                       #
#                 ########              #
#                 #        #            #
#                 #        #            #
############ #
#                                       #
#                                       #
#         #######                       #
#         #                             #
#         #                             #
#         #######                       #
#                                       #
#                                       #
#                  2                    #
#                                       #
#                                       #
#     #####                             #
#     #                                 #
```

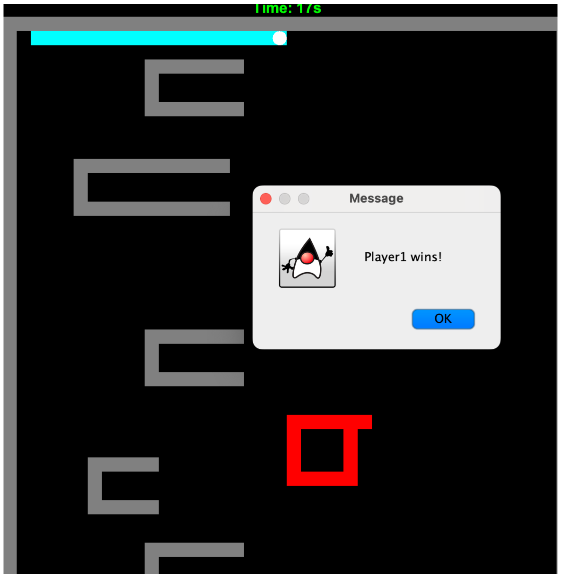
3. **Test Case 3: Player Movement**
    - **AS A player,**
    - **I WANT TO** steer my motor,
    - **GIVEN** a running game,
    - **WHEN** I press the WASD keys or arrow keys,
    - **THEN** the motor moves in the desired direction.

4. **Test Case 4: Collision Detection**
    - **AS A player,**
    - **I WANT TO** lose if I collide with a boundary, obstacle, or light trail,
    - **GIVEN** a running game,
    - **WHEN** my motor collides with these elements,
    - **THEN** the game ends with the opponent declared as the winner.



K wins!

5. **Test Case 5: Game Over on Self-Trail Collision**
    - ○ **AS A player,**
    - ○ **I WANT TO** lose if I collide with my own light trail,
    - ○ **GIVEN** a running game,
    - ○ **WHEN** my motor crosses my light trail,
    - ○ **THEN** the game ends, and the opponent wins.

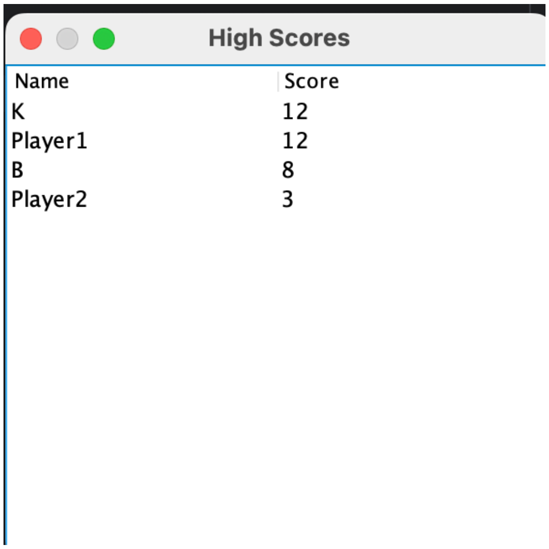


Player1 wins!

6. **Test Case 6: Draw Condition**
    - ○ **AS A player,**
    - ○ **I WANT TO** see a draw when both players lose at the same tick,
    - ○ **GIVEN** a running game,
    - ○ **WHEN** both players collide simultaneously,

- **THEN** the game declares a draw.

7. **Test Case 7: High Score Update**
    - **AS A player,**
    - **I WANT TO** have my score recorded,
    - **GIVEN** that I win a game,
    - **WHEN** the game ends,
    - **THEN** my score increases by 1 in the database.
8. **Test Case 8: High Score Display**
    - **AS A player,**
    - **I WANT TO** view the top 10 scores,
    - **GIVEN** the high score menu,
    - **WHEN** I open the high score dialog,
    - **THEN** it should display the 10 highest scores sorted in descending order.

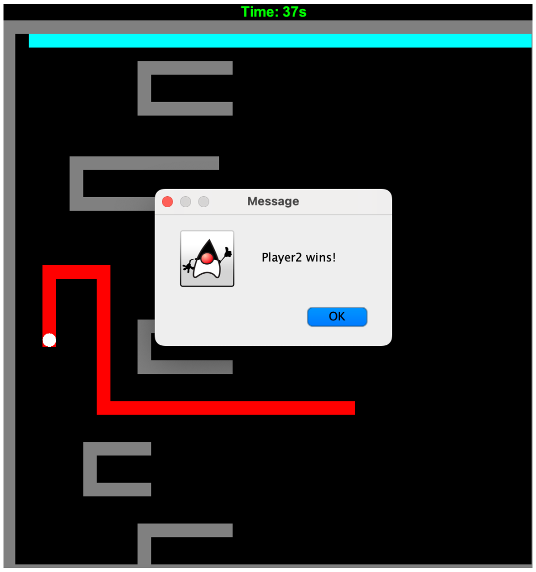


Name	Score
K	12
Player1	12
B	8
Player2	3

9. **Test Case 9: Restart Game**
    - **AS A player,**
    - **I WANT TO** restart the game,
    - **GIVEN** the game has ended,
    - **WHEN** I choose "Restart Game",
    - **THEN** the game resets and starts from the setup dialog.
10. **Test Case 10: Timer Functionality**
    - **AS A player,**
    - **I WANT TO** see the elapsed time during the game,
    - **GIVEN** a running game,
    - **WHEN** time progresses,
    - **THEN** the timer updates in real time.

White Box Test Cases

11. **Test Case 11: Obstacle Initialization**
    - **AS A developer,**
    - **I WANT TO** ensure all obstacles are correctly marked,
    - **GIVEN** a level file with obstacles,
    - **WHEN** the grid is initialized,
    - **THEN** all cells marked as '#' in the file should have `occupied=true` and `colorType=OBSTACLE`.
12. **Test Case 12: Player Initialization**
    - **AS A developer,**
    - **I WANT TO** ensure players are correctly initialized,
    - **GIVEN** a level file with starting positions,
    - **WHEN** the level is loaded,
    - **THEN** the players should start at their designated positions with the correct direction.
13. **Test Case 13: Light Trail Marking**
    - **AS A developer,**
    - **I WANT TO** ensure light trails are correctly marked,
    - **GIVEN** a running game,
    - **WHEN** a player moves to a new cell,
    - **THEN** the new cell should have `occupied=true` and `colorType` matching the player's type.
14. **Test Case 14: Boundary Collision**
    - **AS A developer,**
    - **I WANT TO** verify boundary collision detection,
    - **GIVEN** a motor at the edge of the grid,
    - **WHEN** it moves outside the grid,
    - **THEN** the player's `alive` attribute should be set to `false`.


Time: 37s

Player2 wins!

15. **Test Case 15: Draw Detection Logic**
    - **AS A developer,**
    - **I WANT TO** verify draw detection,
    - **GIVEN** both players collide simultaneously,
    - **WHEN** `checkGameOver()` is called,
    - **THEN** the `winner` attribute should be `null`, and `gameOver` should be `true`.