

### **Task 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 second 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 exist in the database yet, then insert a record for him. Create a menu item, which displays a high score table of the players for the 10 best scores. Also, create a menu item which restarts the game.

### **Task analysis**

#### **Goal of the Task:**

Create a Tron-style light-cycle game in a top-down view, where two players control their respective cycles. Each cycle leaves a trail behind as it moves. Both players should be able to steer their cycles, and a player loses if they crash into a wall, the game boundary, or a light trail. After the game, the winner's score is recorded, and players can view a high score table or restart the game.

#### **Key Requirements from the Task Description:**

##### **1. Two-Player Gameplay:**

- Two players (e.g., Player 1 using W/A/S/D, Player 2 using arrow keys).
- Cycles leave a continuous trail behind them as they move.
- Movement and direction control is crucial.

##### **2. Collision and Losing Conditions:**

- A player loses if they run into a boundary, a wall, or any trail (including the opponent's).
- If both players crash simultaneously, the game ends in a draw.

##### **3. Levels:**

- The game should load from predefined level files or generate them.
- At least 10 predefined levels were to be created.

- Levels are represented by a 2D grid with walls (#), empty spaces, and markers (1 and 2) for player start positions.

#### 4. **Scoring and Database:**

- After determining a winner, update the winner's score in a database.
- If the player is not in the database, insert a new record.
- Provide a high score table (top 10 players).

#### 5. **Menu and Restart Options:**

- A menu or dialog to restart the game.
- A menu or dialog to display high scores.
- A timer or display for elapsed game time.

#### 6. **User Friendliness:**

- Easy-to-use interface.
- Simple graphics with a top view of the playfield.
- Keyboard input handling for player movement.
- A start dialog for entering player names, choosing colours, and selecting a level.

### **Solution Plan**

#### 1. **Initialization:**

- Start the game by displaying StartDialog.
- Player1 and Player2 enter their names, pick their colors, and select a level file from the levels directory.

#### 2. **Loading the Level:**

- Use LevelLoader to parse the chosen file, create a Level object, identify player start positions, and gather wall data.
- Level provides methods to check collisions and retrieve wall positions.

#### 3. **Setting Up the Game Board:**

- GamePanel is initialized with the chosen level, player names, and colors.
- GamePanel sets the players at their start positions, initializes trails, and starts a timer for continuous movement.

#### 4. **Game Loop and Control:**

- On every timer tick, GamePanel moves the players according to their directions.
- Keyboard input is handled to change directions mid-game.
- Trails are recorded, and collisions are checked every tick.
- Time elapsed is tracked for display and final stats.

#### **5. Collision Detection and Game Over:**

- If a player hits a wall, boundary, or a trail, the game ends.
- Determine the winner based on who collided or if both collided simultaneously (draw).

#### **6. End Game Actions:**

- GamePanel stops the timer and shows GameOverDialog.
- The winner's score is updated in the database via DatabaseManager.
- Players can view high scores (HighScoreDialog) or choose "Play Again," which triggers StartDialog again.

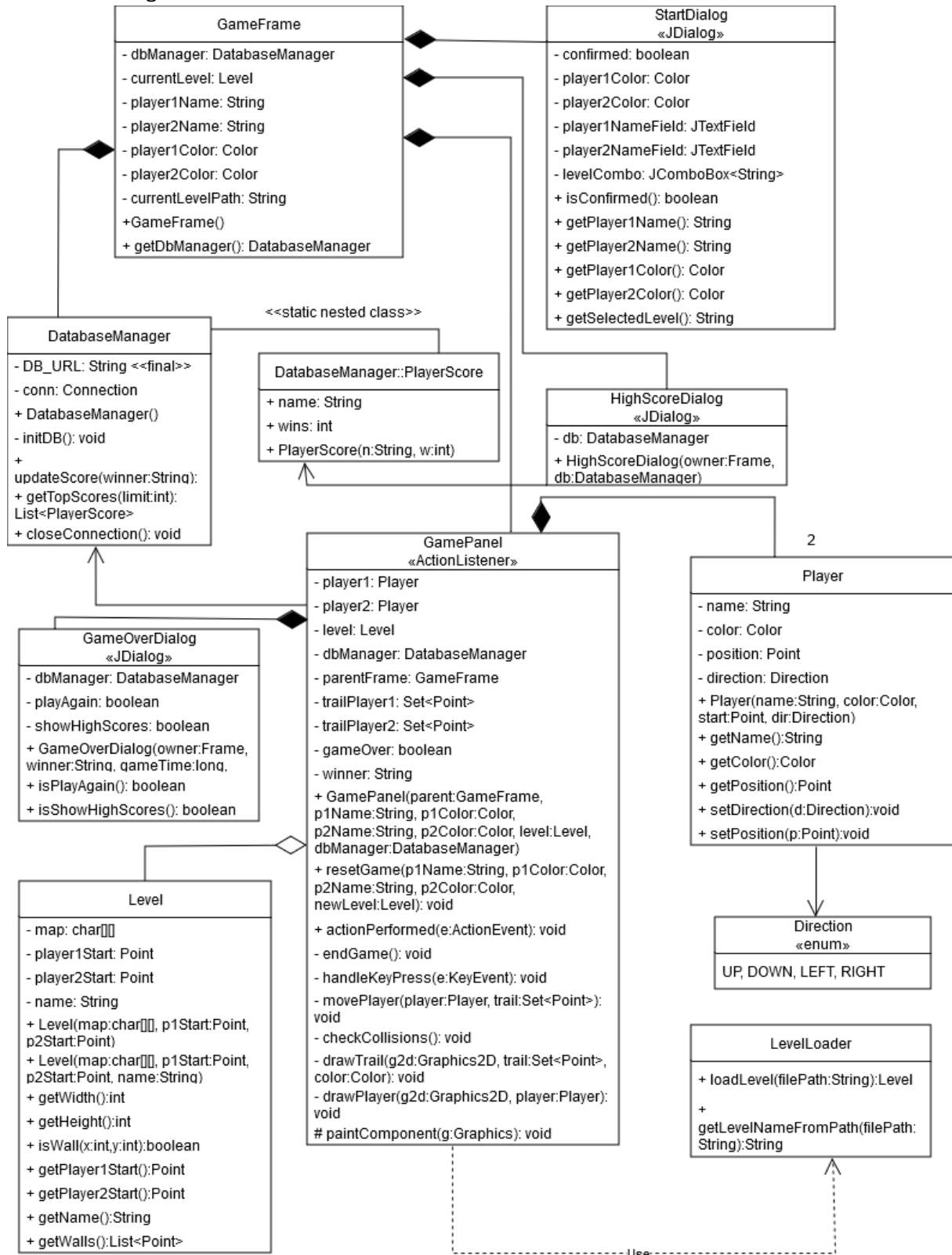
#### **7. High Scores and Database:**

- Scores are stored locally in game.db using SQLite.
- DatabaseManager ensures the table is created, updates/inserts player scores, and retrieves top 10 scores for display in HighScoreDialog.

#### **8. Restarting the Game:**

- If "Play Again" is chosen, StartDialog appears once more to select potentially different parameters (new players, colors, or level).
- GamePanel's resetGame method is called to start fresh without exiting the application.

UML class diagram:



1. GameFrame → StartDialog, HighScoreDialog:

Composition:

- The GameFrame creates and manages these dialog objects. They do not have meaning without the GameFrame, so it "owns" their lifecycles.

2. GameFrame → DatabaseManager:

Composition:

- GameFrame creates a DatabaseManager instance to handle scores. When GameFrame closes, the DatabaseManager also goes away.

3. DatabaseManager → PlayerScore:

Nested Class:

- PlayerScore is defined inside DatabaseManager. It represents a data structure returned by getTopScores. HighScoreDialog uses these results.

4. GameFrame → GamePanel:

Composition:

- GameFrame creates and contains the GamePanel. The GamePanel runs the main game logic and rendering.

5. GamePanel → Player:

Composition:

- GamePanel creates and fully controls Player objects. Players exist only during the current game session managed by GamePanel.

6. GamePanel → Level:

Aggregation:

- GamePanel uses a Level object (provided by LevelLoader) to check walls and player movement. The Level is not strictly owned by GamePanel; it can be replaced if needed.

7. GamePanel → DatabaseManager:

Association:

- GamePanel uses DatabaseManager to update scores at the end of a game, but does not create or own it (it's supplied by GameFrame).

8. Player → Direction:

Association:

- Each Player has a Direction to indicate movement. Direction is an enum that Player simply references.

9. GamePanel → LevelLoader:

Dependency:

- This class calls LevelLoader.loadLevel(...) when needed. It doesn't hold a permanent reference to LevelLoader.

## Method descriptions:

### DatabaseManager

**Purpose:**

Manages all database operations for storing and retrieving player scores.

**Methods:**

- **DatabaseManager() [Constructor]:**  
Initializes a connection to the SQLite database (game.db) and calls initDB() to ensure the scores table exists.
- **initDB():**  
Creates the scores table if it doesn't already exist. Used internally by the constructor.
- **updateScore(String winner):**  
Updates or inserts the record for the winning player in the database. If the player already exists, increments their win count by one; otherwise, inserts a new player record with one win.
- **getTopScores(int limit):**  
Retrieves a sorted list (by descending wins) of up to limit player scores from the database. Returns a List<PlayerScore>.
- **closeConnection():**  
Closes the database connection. Should be called when the application is terminating.

**Inner Class:**

- **PlayerScore:**  
A simple data class holding a player's name and wins.

**Direction (Enum)****Purpose:**

Represents possible movement directions for players.

**Enum Constants:**

- **UP, DOWN, LEFT, RIGHT:** Used to update and track player movement direction.

**GameFrame****Purpose:**

The main application window (JFrame) that initiates the game by showing the start dialog, loading levels, creating the game panel, and providing access to the database manager.

**Methods:**

- **GameFrame() [Constructor]:**  
Sets up the main game window, initializes the database manager, shows the StartDialog to get initial player/level info, loads the selected level, and creates the GamePanel for gameplay.
- **getDbManager():**  
Returns the DatabaseManager instance used by this game frame.
- **main(String[] args) [static]:**  
The entry point of the program. Launches the game on the Swing event thread by creating an instance of GameFrame.

## GameOverDialog

### Purpose:

Displayed when a game ends. Shows the winner, the elapsed time, and provides options to play again, view high scores, or exit.

### Methods:

- **GameOverDialog(Frame owner, String winner, long gameTime, DatabaseManager dbManager) [Constructor]:**  
Creates the dialog, displaying a congratulatory message if there's a winner or a "Draw" message. Shows the elapsed game time. Offers buttons: "Play Again", "High Scores", and "Exit".
- **isPlayAgain():**  
Returns true if the user clicked "Play Again" after the game ended.
- **isShowHighScores():**  
Returns true if the user chose to view high scores.

### Inner Class:

- **TrophyPanel:**  
A custom panel that draws a trophy icon to visually decorate the dialog.

## GamePanel

### Purpose:

The main gameplay area where players move, leave trails, and interact with the level. Handles the game loop, input, rendering, and collision detection.

### Methods:

- **GamePanel(GameFrame parent, String p1Name, Color p1Color, String p2Name, Color p2Color, Level level, DatabaseManager dbManager) [Constructor]:**  
Initializes the game with given players, colors, and level. Sets up the key listener, timer, and panel dimensions.
- **resetGame(String p1Name, Color p1Color, String p2Name, Color p2Color, Level newLevel):**  
Resets the state of the game to start a new round with possibly new players, colors, and a new level. Clears trails, repositions players, and restarts the timer.
- **actionPerformed(ActionEvent e):**  
Called by the game timer. Moves players, checks for collisions, and repaints the panel each tick if the game isn't over.
- **handleKeyPress(KeyEvent e):**  
Updates player directions based on key inputs (WASD for Player 1, arrow keys for Player 2).
- **movePlayer(Player player, Set<Point> trail):**  
Advances the player's position one cell in the current direction and records the previous position in their trail.

- **checkCollisions():**  
Checks if either player has collided with a wall, boundary, or trail. If so, determines the game outcome (win, lose, draw) and ends the game.
- **endGame():**  
Stops the timer, updates the database with the winner's score, and shows GameOverDialog. If "Play Again" is chosen, restarts or closes the game as appropriate.
- **paintComponent(Graphics g):**  
Draws the current game state: status bar, player info, level walls, trails, and player positions, along with elapsed time and level name.

## HighScoreDialog

### Purpose:

Shows a table of the top 10 player scores (sorted by wins) retrieved from the database.

### Methods:

- **HighScoreDialog(Frame owner, DatabaseManager db) [Constructor]:**  
Queries the database for top scores, then creates and displays a scrollable table showing player names and their win counts. Provides a "Close" button.

## Level

### Purpose:

Represents a game level's layout, including walls, empty spaces, player start positions, and a name.

### Methods:

- **Level(char[][] map, Point p1Start, Point p2Start) [Constructor]:**  
Creates a level with given map data and player start positions, using a default name.
- **Level(char[][] map, Point p1Start, Point p2Start, String name) [Constructor]:**  
Same as above but allows specifying a custom level name.
- **getWidth(), getHeight():**  
Return the width and height of the level in cells.
- **isWall(int x, int y):**  
Checks if a given coordinate is outside the level or a # wall.
- **getPlayer1Start(), getPlayer2Start():**  
Return the starting positions for Player 1 and Player 2, respectively.
- **getName():**  
Returns the name of the level.
- **getWalls():**  
Returns a list of all wall coordinates in the level.

## LevelLoader

### Purpose:

Loads a level layout from a .txt file, parsing walls, empty spaces, and special markers for player start positions.



### Methods:

- **loadLevel(String filePath):**  
Reads the file line by line, builds a 2D char array for the map, finds 1 and 2 for player starts, and returns a Level object. If no start positions are found, uses defaults. The level name is inferred from the file name.
- **getLevelNameFromPath(String filePath):**  
Extracts a human-readable level name from the file name by removing extensions and formatting words.

### Player

#### Purpose:

Represents a single player's state: name, color, position, direction, and movement history.

#### Methods:

- **Player(String name, Color color, Point start, Direction dir) [Constructor]:**  
Creates a player with given attributes. Sets initial position and direction.
- **getName(), getColor(), getPosition(), getPreviousPosition(), getDirection():**  
Getter methods for player attributes.
- **setDirection(Direction d):**  
Updates the player's current movement direction.
- **setPosition(Point p):**  
Updates the player's position and stores the old position as previousPosition.

### StartDialog

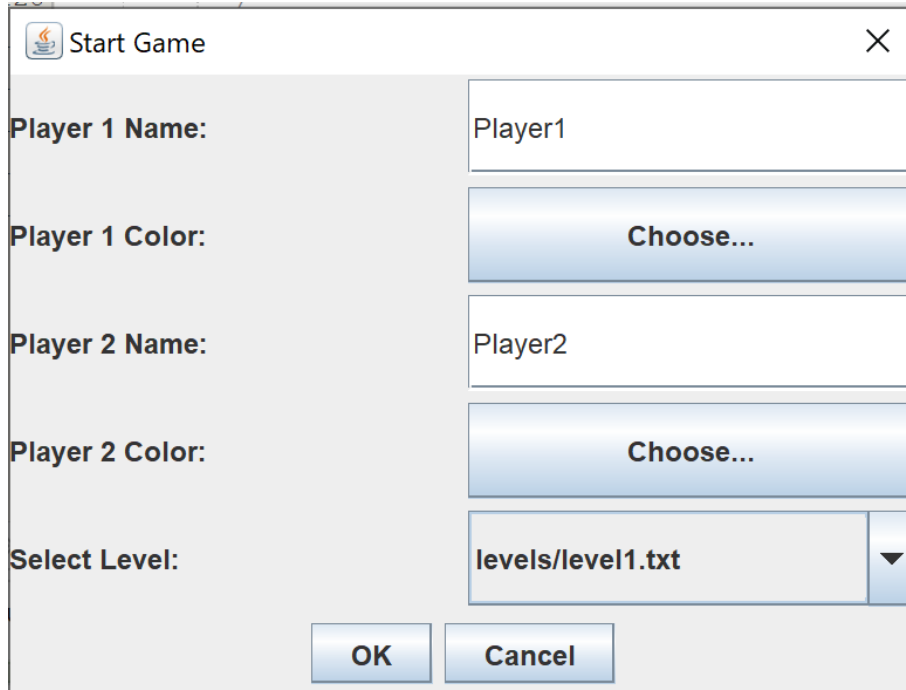
#### Purpose:

Shown at the beginning of the game or after "Play Again" is chosen, allowing users to input player names, select colors, and choose a level file.

#### Methods:

- **StartDialog(Frame owner) [Constructor]:**  
Builds a dialog with text fields for player names, buttons for choosing colors, and a dropdown for selecting a level file. "OK" confirms the settings; "Cancel" aborts.
- **isConfirmed():**  
Returns true if the player clicked "OK" to start the game, otherwise false.
- **getPlayer1Name(), getPlayer2Name():**  
Return the entered names for Player 1 and Player 2.
- **getPlayer1Color(), getPlayer2Color():**  
Return the chosen colors for each player.
- **getSelectedLevel():**  
Returns the file path of the selected level.

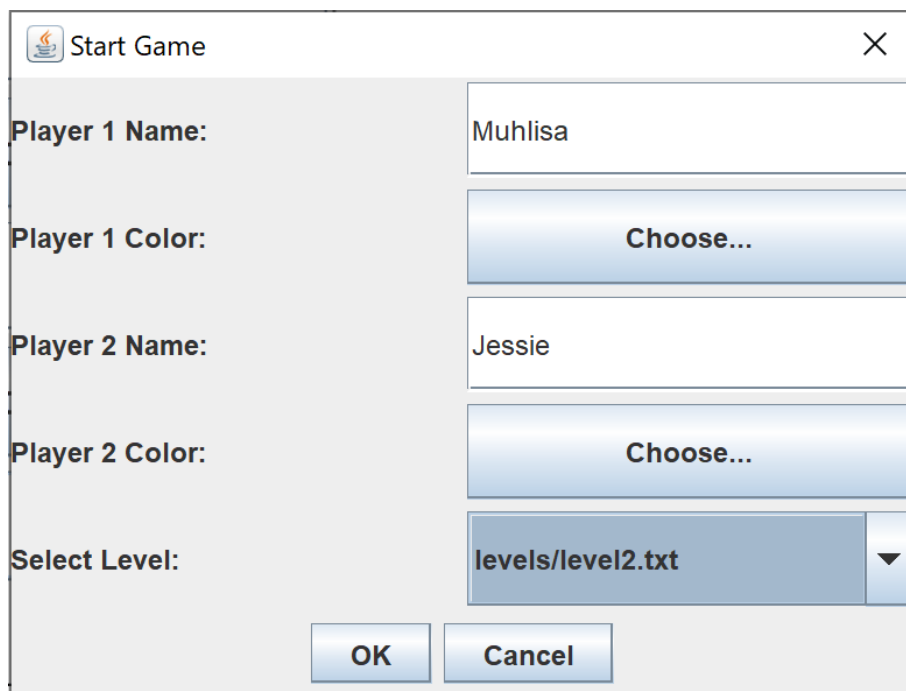
### Application Screenshots:



A screenshot of a 'Start Game' dialog box. The dialog has a title bar with a small icon and the text 'Start Game' and a close button. The main area contains five labels on the left and corresponding input fields on the right. The labels are 'Player 1 Name:', 'Player 1 Color:', 'Player 2 Name:', 'Player 2 Color:', and 'Select Level:'. The input fields are text boxes for names, buttons labeled 'Choose...' for colors, and a dropdown menu for the level. At the bottom are 'OK' and 'Cancel' buttons.

Player 1 Name:	Player1
Player 1 Color:	Choose...
Player 2 Name:	Player2
Player 2 Color:	Choose...
Select Level:	levels/level1.txt ▼

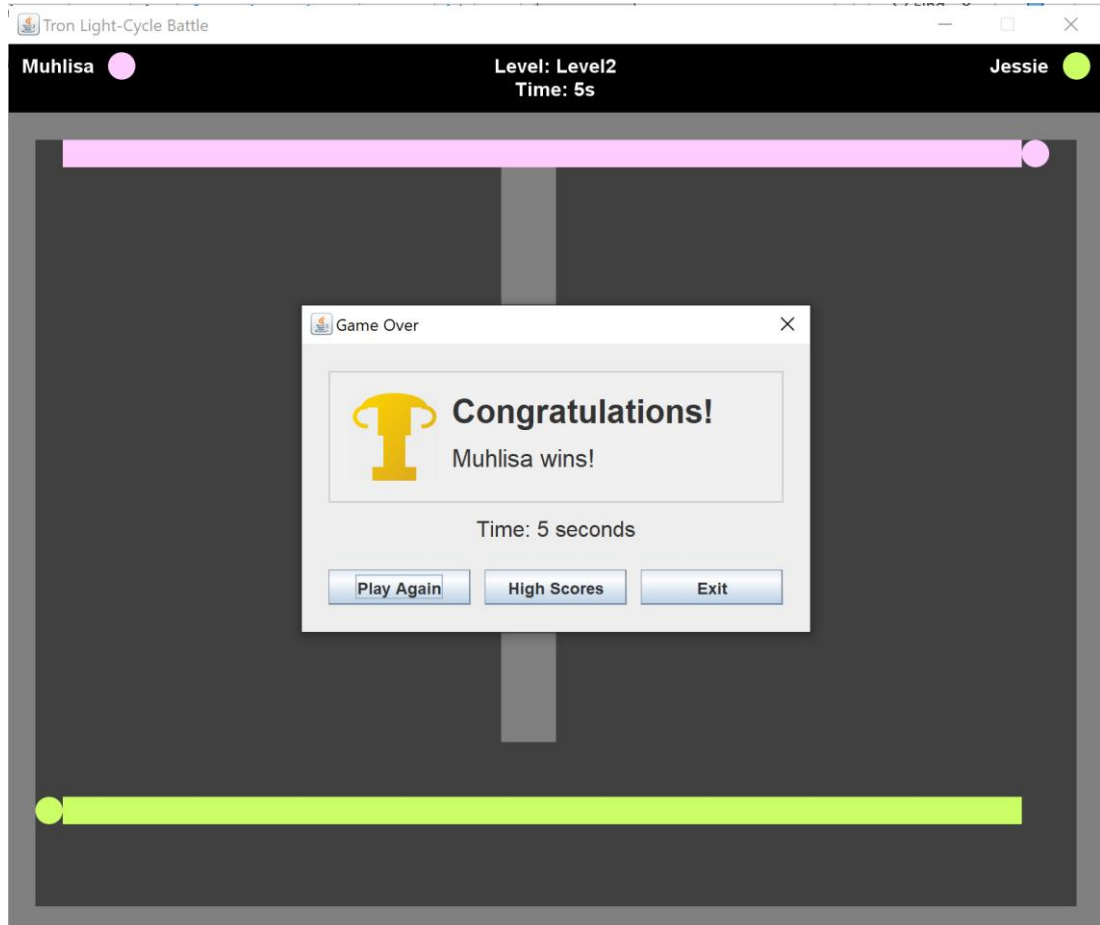
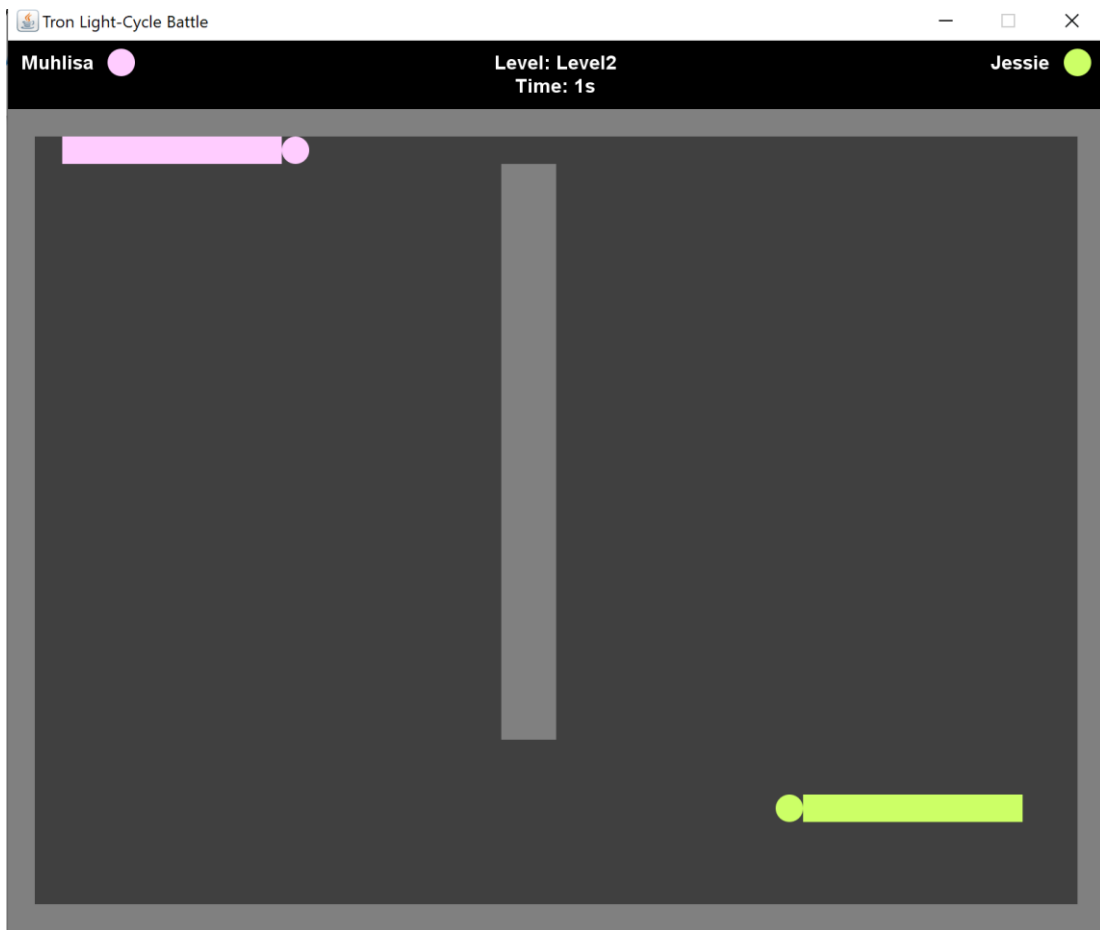
OK Cancel

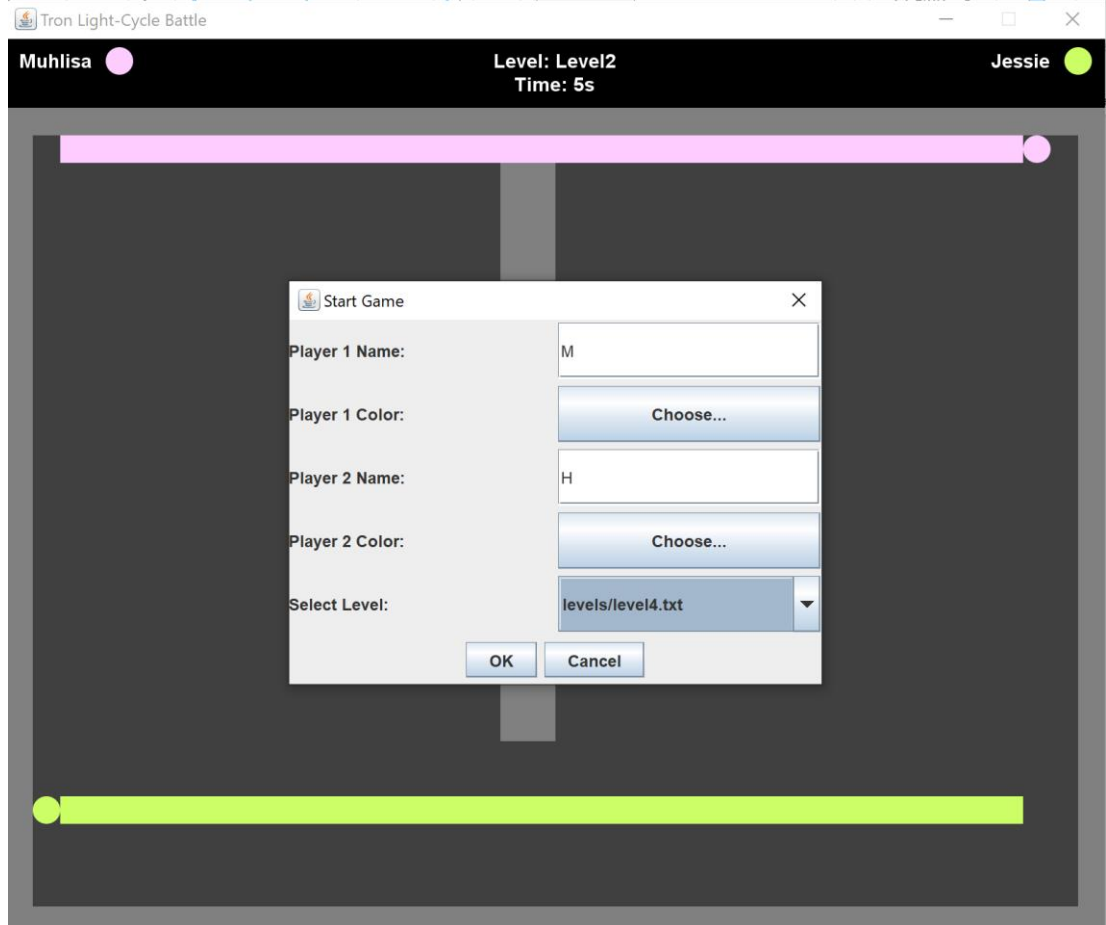
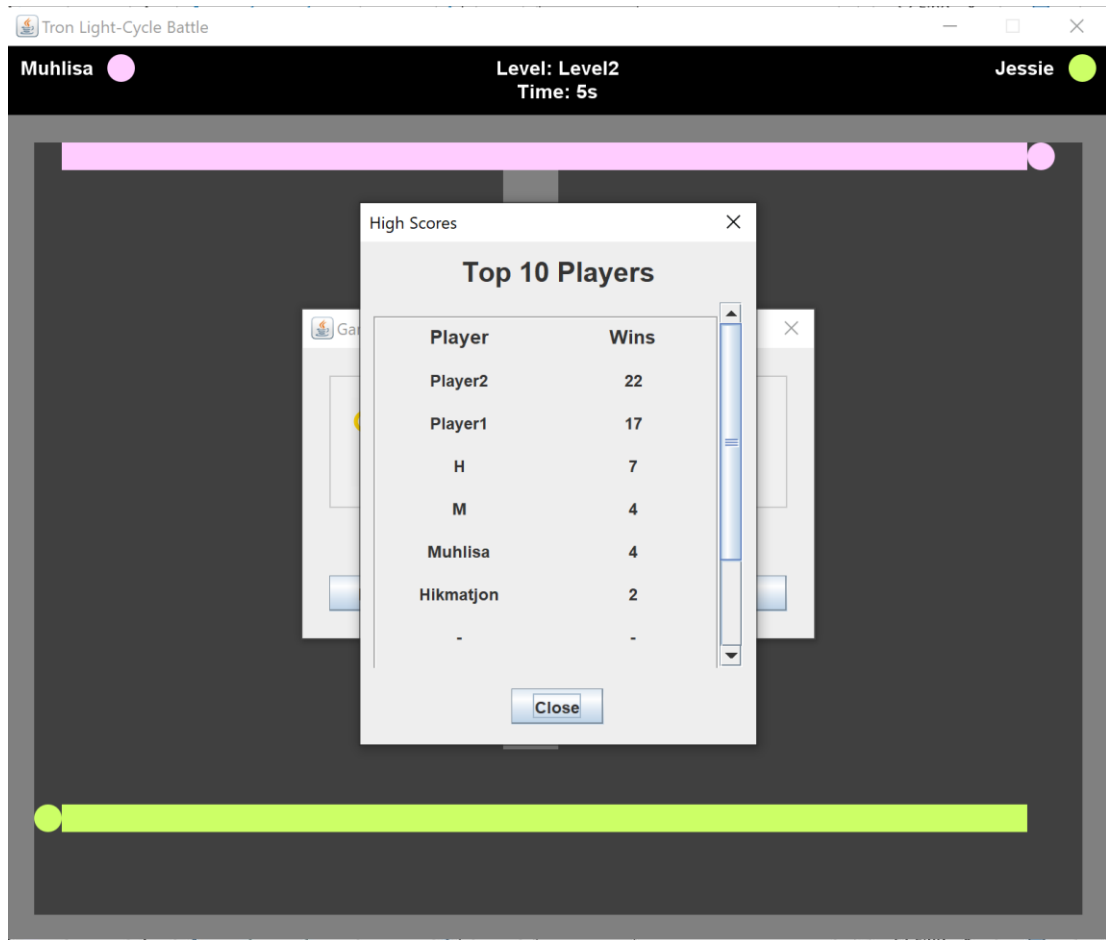


A second screenshot of the 'Start Game' dialog box, showing the same layout as the first but with user input. The 'Player 1 Name' field contains 'Muhlisa', the 'Player 2 Name' field contains 'Jessie', and the 'Select Level' dropdown is set to 'levels/level2.txt'. The 'Player 1 Color' and 'Player 2 Color' buttons remain 'Choose...'. The 'OK' and 'Cancel' buttons are at the bottom.

Player 1 Name:	Muhlisa
Player 1 Color:	Choose...
Player 2 Name:	Jessie
Player 2 Color:	Choose...
Select Level:	levels/level2.txt ▼

OK Cancel





## Test Report

### 1. Empty Player Name:

**AS A** player

**I WANT TO** start the game without typing my name

**GIVEN** the start dialog prompts for player names

**WHEN** I leave a player's name field empty and click "OK"

**THEN** the name should default to "Player1" or "Player2" and the game should start normally.

### 2. No Levels Available:

**AS A** player

**I WANT TO** select a level to play on

**GIVEN** there are no .txt files in the levels directory

**WHEN** I open the start dialog

**THEN** I should see no levels in the dropdown and be unable to start the game until a level is added.

### 3. Nonexistent Level File:

**AS A** player

**I WANT TO** start a game with a chosen level

**GIVEN** I select a level file that doesn't actually exist on disk

**WHEN** I click "OK" in the start dialog

**THEN** I should see an error message indicating the level can't be loaded, and the game should not start.

### 4. Player Starting Inside a Wall:

**AS A** player

**I WANT TO** have a fair starting position

**GIVEN** a level file places player 1's start position at a # wall cell

**WHEN** the game begins

**THEN** player 1 should immediately crash, resulting in either a direct loss for player 1 and a win for player 2, or a recognized invalid level scenario.

### 5. Both Players Crash Simultaneously:

**AS A** player

**I WANT TO** see what happens if both players collide at the same time

**GIVEN** a situation where both players head toward each other or a wall simultaneously

**WHEN** both players move and crash on the same tick

**THEN** the game should end in a "Draw" and no one's score should increment.

### 6. High Scores with No Prior Games:

**AS A** player

**I WANT TO** view high scores before any games have been played

**GIVEN** the database is empty

**WHEN** I open the "High Scores" dialog

**THEN** I should see either an empty list or placeholders, indicating no scores are recorded yet.

### 7. Canceling Color Selection:

**AS A** player

**I WANT TO** pick a custom color for my cycle

**GIVEN** the color chooser appears

**WHEN** I click "Cancel" without selecting a color  
**THEN** my player's color should remain the default (e.g., Blue for Player 1).

#### **8. Attempting to Move Outside Boundaries:**

**AS A** player

**I WANT TO** see what happens if I steer directly into the boundary

**GIVEN** the cycle is one step away from the wall

**WHEN** I press a direction key that leads outside the level

**THEN** my player should crash, ending the game and awarding the other player the win.

#### **9. No Movement by Any Player:**

**AS A** player

**I WANT TO** see what happens if no one presses any controls

**GIVEN** both players start facing a direction blocked by a wall

**WHEN** the timer moves them forward without input

**THEN** both might crash immediately, resulting in a "Draw" scenario.

#### **10. Immediate Restart After Game Over:**

**AS A** player

**I WANT TO** quickly start another round

**GIVEN** the game just ended and showed the Game Over dialog

**WHEN** I click "Play Again" and choose a new level and player names

**THEN** the GamePanel should reset correctly, starting the new match without errors or leftover data.

#### **11. Duplicate Player Name in Database:**

**AS A** player

**I WANT TO** keep using my favourite name even if it already exists in the database

**GIVEN** a player name that has played before

**WHEN** I enter that name again and win a game

**THEN** the database should increment the existing record's wins rather than creating a duplicate record.

#### **12. Empty Database File on Startup:**

**AS A** player

**I WANT TO** run the game with a fresh database

**GIVEN** game.db does not exist or is empty

**WHEN** I start the game

**THEN** the game should create a new database file and scores table without errors, allowing normal gameplay.

#### **13. Invalid Input Keys:**

**AS A** player

**I WANT TO** try pressing keys that do not correspond to movement controls

**GIVEN** arbitrary keyboard input (like pressing 'X' or 'Q')

**WHEN** I press non-movement keys during gameplay

**THEN** the direction should not change, and the game should continue without issues.

#### **14. No Start Positions in the Level:**

**AS A** player

**I WANT TO** ensure players spawn correctly

**GIVEN** a level file with no '1' or '2' markers

**WHEN** the game loads this level

**THEN** default fallback positions (e.g., (5,5) and (width-6, height-6)) are used, ensuring the game can still start.

**15. All Walls and No Free Space:**

**AS A** player

**I WANT TO** see what happens if the level is mostly walls with no room to move

**GIVEN** a level filled almost entirely with #, leaving the players trapped

**WHEN** the game starts

**THEN** players immediately crash or cannot move at all, resulting in an immediate end to the game (Draw or one player winning if starts differ).