Game Documentation for Tron Game

Introduction

The **Tron Game** is a dynamic, competitive two-player strategy game where players control bikes that leave behind trails on an expanding grid. The objective is to avoid collisions while navigating the grid, with each round ending when one player crashes. The player with the highest score after 10 rounds wins.

Game Rules and Objectives

Objective

- Outlast the opponent by avoiding collisions with trails, blocks, or boundaries.
- Score points by surviving longer than the opponent in each round.
- The player with the highest score after 10 rounds is declared the winner.

Gameplay Mechanics

Grid Setup

- Initial size: 30x30 cells.
- Expands by 2 rows and 2 columns after each round.
- Random blocks are placed on the grid at the start of each round.

Player Bikes

- **Player 1**:
  - Color: Red (#FF0000).
  - Starting position: Top-left corner.
- **Player 2**:
  - Color: Blue (#0000FF).
  - Starting position: Bottom-right corner.

Trails

- Bikes leave trails as they move:
  - **Player 1**: Light Red (#FF8777).
  - **Player 2**: Light Blue (#ADD8E6).
- Trails act as barriers and cause collisions.

Round Flow

1. Both players move simultaneously using their controls.
2. The round ends when one player crashes.
3. The surviving player scores a point.

Collision Detection

- **Trail Collision**: Crashing into a trail ends the round.
- **Boundary Collision**: Moving beyond the grid's edges results in a crash.
- **Block Collision**: Moving into a random block results in a crash.

Key Handling and Event Handlers

Key Assignments

- **Player 1 Movement (Arrow Keys)**:
  - Up: ↑
  - Down: ↓
  - Left: ←
  - Right: →
- **Player 2 Movement (WASD Keys)**:
  - Up: W
  - Down: S
  - Left: A
  - Right: D

Event Handling

- **Key Listener Setup**:
  - The game grid listens for key events.
  - A `KeyAdapter` captures and processes key presses for both players.
- **Movement Workflow**:
  - Player direction is updated based on the last key press.
  - Bikes move one cell per game update.
  - Trails are marked on previously occupied cells.
- **Example**:
  - Player 1 presses ↑:
    - The bike moves up by one cell.
    - The previous cell is marked as a trail (#FF8777).

- Player 2 presses D:
    - The bike moves right by one cell.
    - The previous cell is marked as a trail (#ADD8E6).

Timer Implementation Overview

The timer in the `BattleField` game is implemented using the `Timer` class from Java's `javax.swing` package, which provides a simple way to schedule tasks for repeated execution with fixed-rate or fixed-delay intervals.

Key Components

- **Timer Object**: This is the instance that controls the timing and is responsible for triggering the timer event every second.
- **ActionListener**: This listens for timer events (ticks) and performs actions like updating the display or checking game conditions at each tick.
- **Time Variables**: Variables like `minutes`, `seconds`, or `timeLeft` are used to track the remaining time or the current time of the round.

Game Flow

Player Movement

1. The game listens for key events on the grid.
2. Bikes move in their respective directions, leaving trails behind.

Collision Detection

- After each move, the game checks for collisions:
    - Trail: Crashing into a trail ends the round.
    - Boundary: Moving outside the grid ends the round.
    - Block: Crashing into a block ends the round.

Round Transition

1. The grid size expands.
2. Player positions reset to their starting cells.
3. Blocks are placed randomly on the grid.

Winning Condition

- After 10 rounds, the player with the highest score wins.

Class Descriptions

Main Class

- Handles:
    - Game initialization.
    - Player name input.
    - Round progression and scoring.
    - Restart and exit options.

BattleField Class

- Manages:
    - Grid dimensions.
    - Player positions.
    - Collision detection.
    - Game state transitions.

BattleFieldGUI Class

- Manages the graphical interface:
    - Renders grid, trails, bikes, and blocks.
    - Handles key events for player movement.

DatabaseHandler Class

- Tracks game statistics:
    - Stores player names, scores, and match history.
    - Maintains a leaderboard of top players.

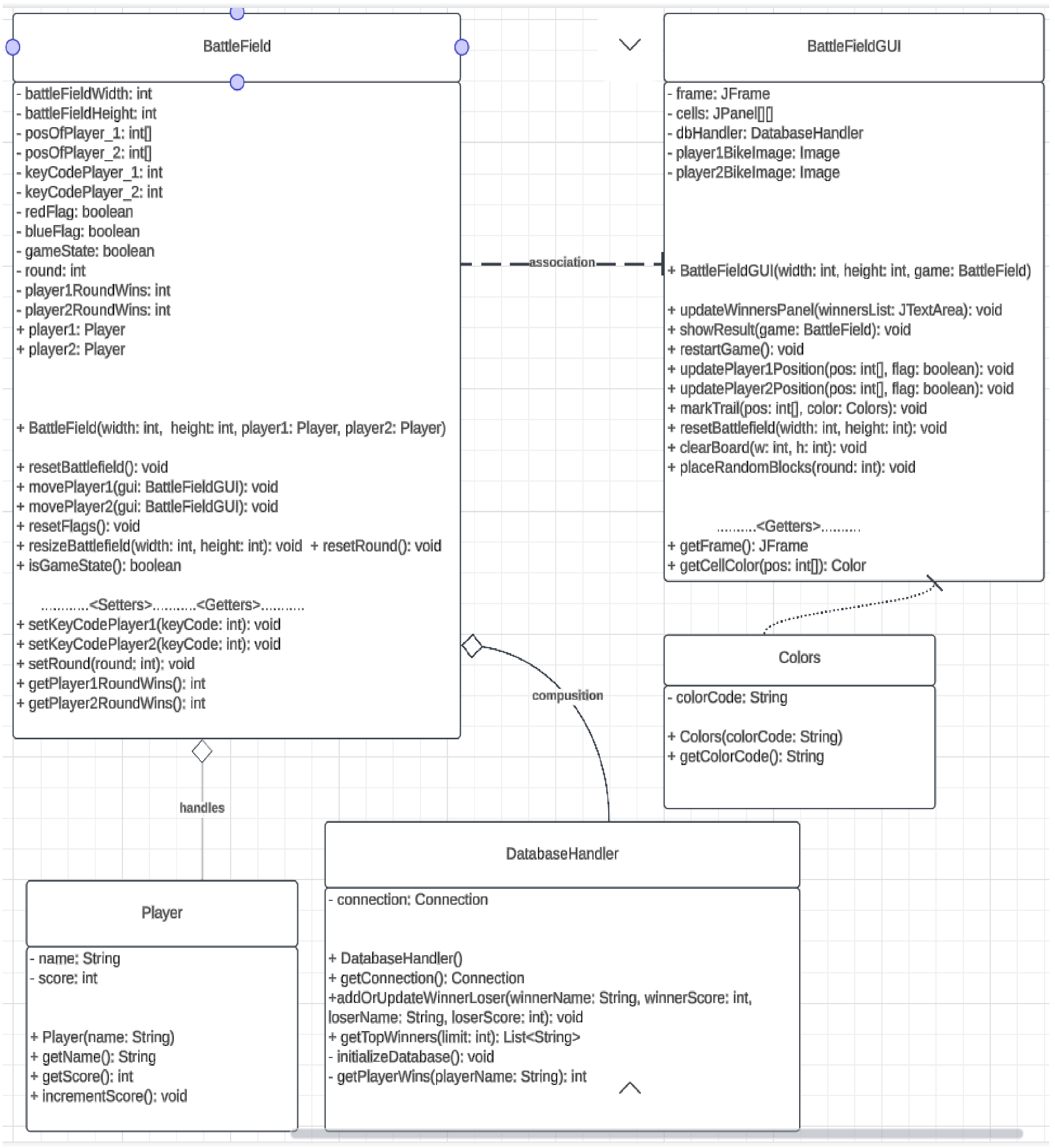
Colors Enum

- Defines color codes for visual elements:
    - Player bikes and trails.
    - Background and obstacles.

Game Features

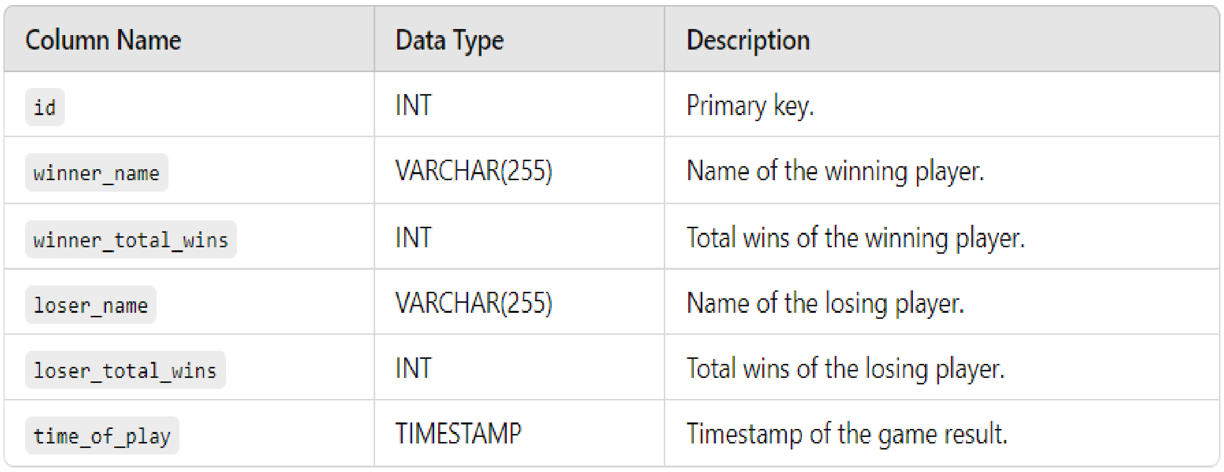
- **Dynamic Grid**: Expands with each round for added complexity.
- **Player Controls**: Supports intuitive key-based navigation.
- **Obstacles**: Randomly placed blocks add unpredictability.
- **Trail System**: Player trails act as barriers.
- **Leaderboard**: Tracks player statistics across matches.
- **Custom Visuals**: Unique bike icons and trail colors for each player.

UML Diagram


BattleField
- battleFieldWidth: int
- battleFieldHeight: int
- posOfPlayer_1: int[]
- posOfPlayer_2: int[]
- keyCodePlayer_1: int
- keyCodePlayer_2: int
- redFlag: boolean
- blueFlag: boolean
- gameState: boolean
- round: int
- player1RoundWins: int
- player2RoundWins: int
+ player1: Player
+ player2: Player
+ BattleField(width: int, height: int, player1: Player, player2: Player)
+ resetBattlefield(): void
+ movePlayer1(gui: BattleFieldGUI): void
+ movePlayer2(gui: BattleFieldGUI): void
+ resetFlags(): void
+ resizeBattlefield(width: int, height: int): void + resetRound(): void
+ isGameState(): boolean
...........<Setters>..........<Getters>..........
+ setKeyCodePlayer1(keyCode: int): void
+ setKeyCodePlayer2(keyCode: int): void
+ setRound(round: int): void
+ getPlayer1RoundWins(): int
+ getPlayer2RoundWins(): int
association
BattleFieldGUI
- frame: JFrame
- cells: JPanel[][]
- dbHandler: DatabaseHandler
- player1BikeImage: Image
- player2BikeImage: Image
+ BattleFieldGUI(width: int, height: int, game: BattleField)
+ updateWinnersPanel(winnersList: JTextArea): void
+ showResult(game: BattleField): void
+ restartGame(): void
+ updatePlayer1Position(pos: int[], flag: boolean): void
+ updatePlayer2Position(pos: int[], flag: boolean): void
+ markTrail(pos: int[], color: Colors): void
+ resetBattlefield(width: int, height: int): void
+ clearBoard(w: int, h: int): void
+ placeRandomBlocks(round: int): void
.........<Getters>.........
+ getFrame(): JFrame
+ getCellColor(pos: int[]): Color
Colors
- colorCode: String
+ Colors(colorCode: String)
+ getColorCode(): String
composition
handles
Player
- name: String
- score: int
+ Player(name: String)
+ getName(): String
+ getScore(): int
+ incrementScore(): void
DatabaseHandler
- connection: Connection
+ DatabaseHandler()
+ getConnection(): Connection
+addOrUpdateWinnerLoser(winnerName: String, winnerScore: int, loserName: String, loserScore: int): void
+ getTopWinners(limit: int): List<String>
- initializeDatabase(): void
- getPlayerWins(playerName: String): int

Database Design

Table: game_winners

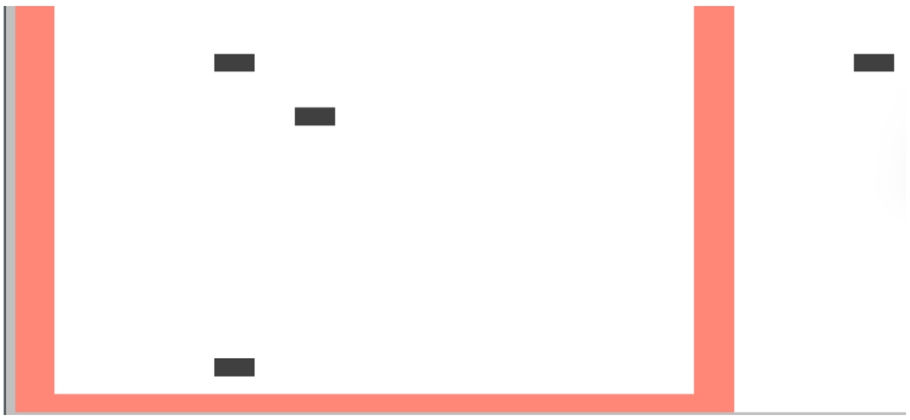


Column Name	Data Type	Description
`id`	INT	Primary key.
`winner_name`	VARCHAR(255)	Name of the winning player.
`winner_total_wins`	INT	Total wins of the winning player.
`loser_name`	VARCHAR(255)	Name of the losing player.
`loser_total_wins`	INT	Total wins of the losing player.
`time_of_play`	TIMESTAMP	Timestamp of the game result.

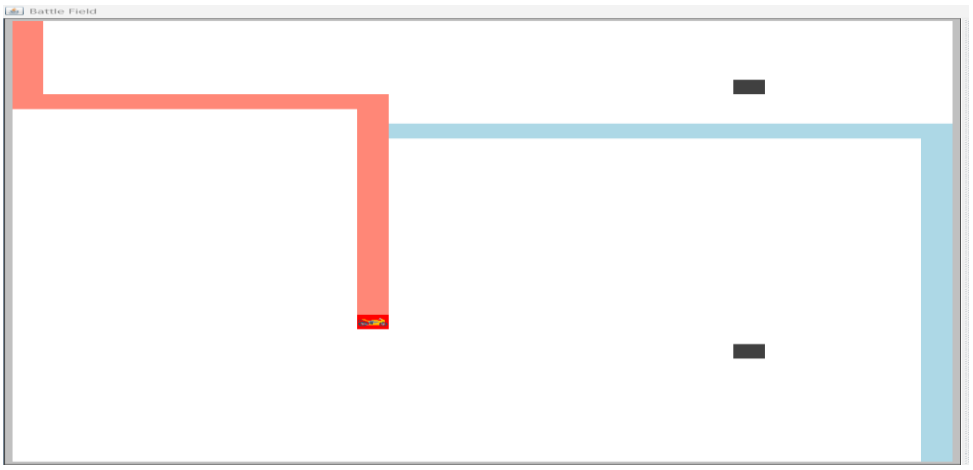
Testing

White-Box Testing

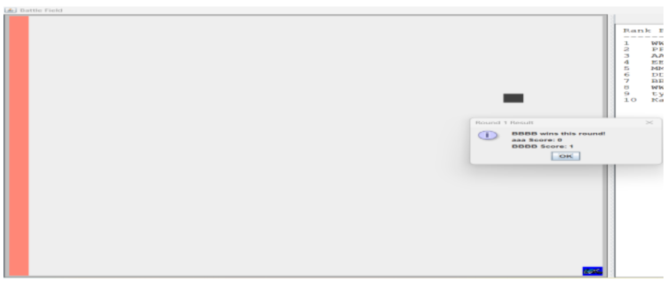
1. **Test Player 1 Movement**:
   a. **Input**: Player 1 presses the UP arrow key at any position.
   b. **Expected Output**: Player 1 moves to up and leaves a trail, and goes furher until the player changes direction or collieds.



2. **Test Player 2 Collision with Trail**:
   a. **Input**: Player 2 moves into Player 1's trail.
   b. **Expected Output**: Player 2 crashes, and Player 1 scores a point.



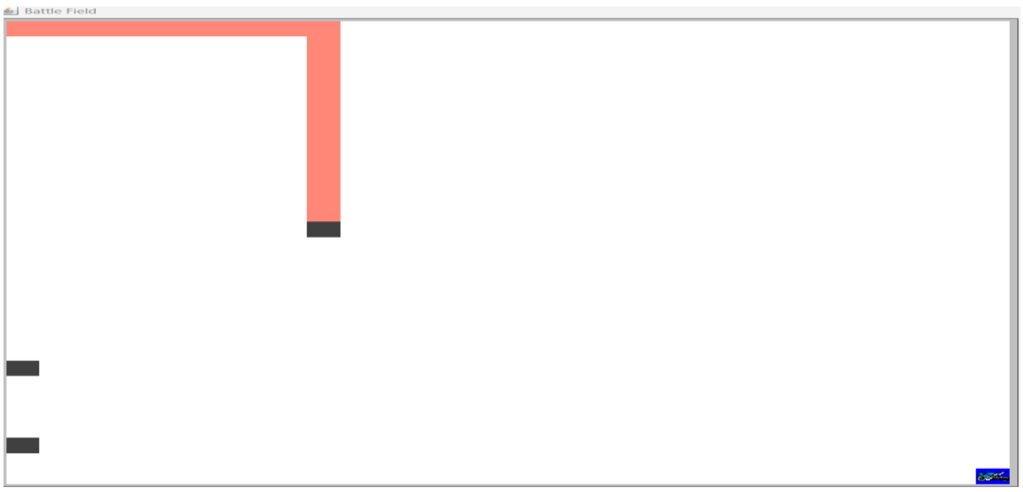
3. **Test Boundary Collision**:
   a. **Input**: Player 1 presses LEFT at `(0, 0)`.
   b. **Expected Output**: Player 1 crashes, and Player 2 wins the round.



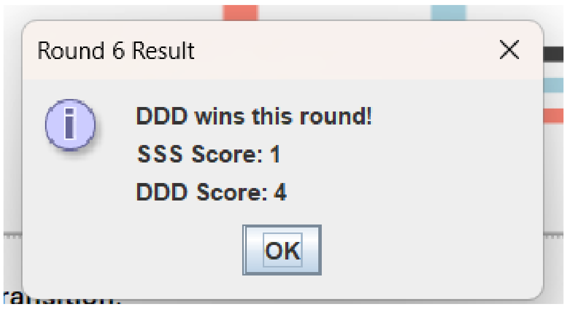
```
Rank E
------
1    WW
2    PP
3    AA
4    EE
5    MM
6    DD
7    BB
8    WW
9    ty
10   Ka
```

BBBB wins this round!
aaa Score: 0
BBBB Score: 1

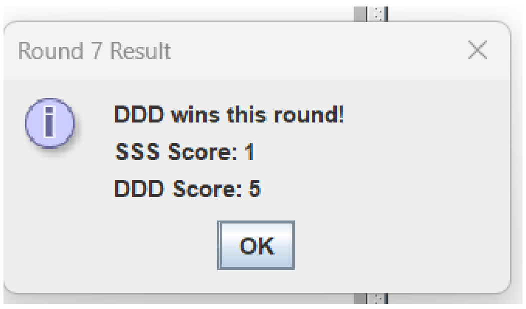
4. **Test Block Collision**:
   a. **Input**: Player moves into a block.
   b. **Expected Output**: The player crashes, and the other player scores.



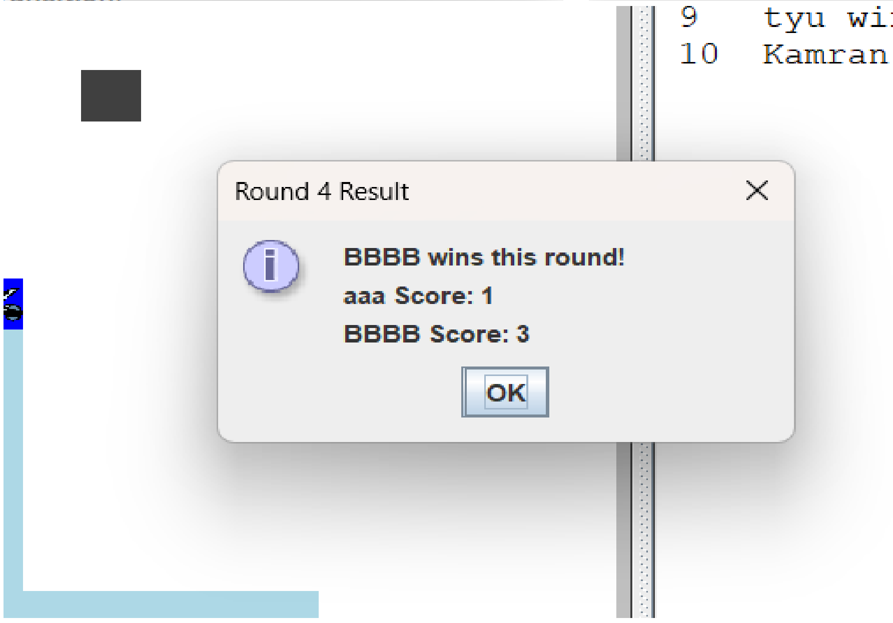
5. **Test Score Update**:
   a. **Input**: Player 1 wins a round.
   b. **Expected Output**: Player 1's score increments.


Round 6 Result

DDD wins this round!
SSS Score: 1
DDD Score: 4


Round 7 Result

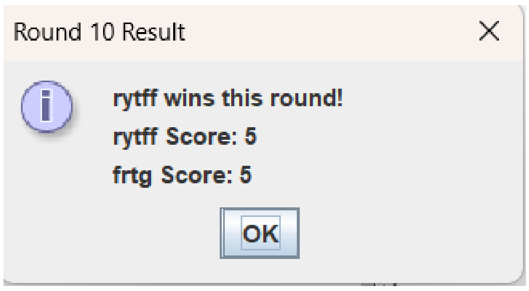
DDD wins this round!
SSS Score: 1
DDD Score: 5


```
9    tyu wins (7
10   Kamran wins
```

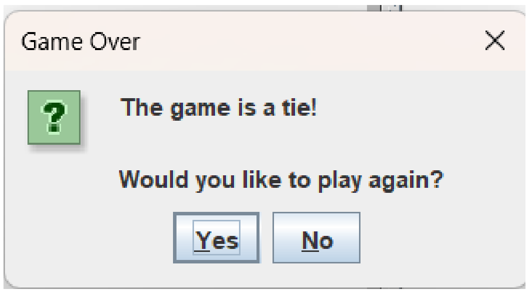
Round 4 Result

BBBB wins this round!
aaa Score: 1
BBBB Score: 3

6. **Test Game Ends as a Tie**:
    a. **Input**: both players gets same score.
    b. **Expected: Output**: The game ends in as a tie no one wins.


Round 10 Result

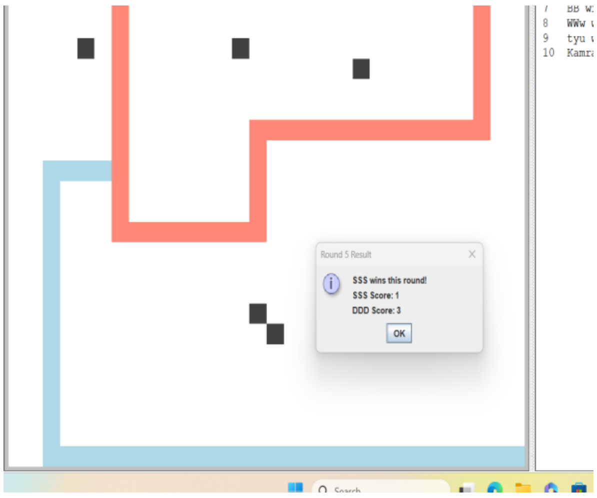
rytff wins this round!
rytff Score: 5
frtg Score: 5


Game Over

The game is a tie!

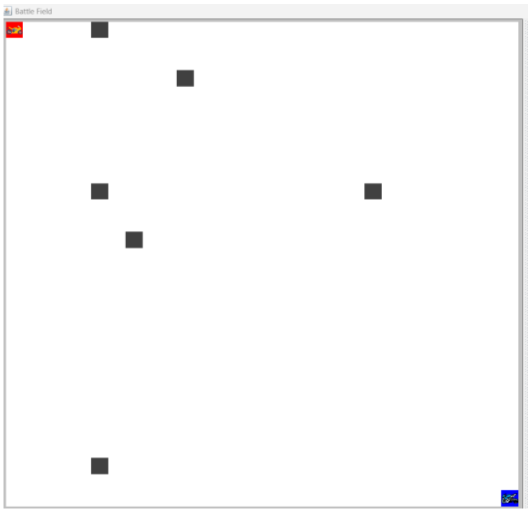
Would you like to play again?

7. **Test Game Reset After Win**:
    a. **Input**: Player achieves a win condition.
    b. **Expected Output**: The grid resets for the next round.


Round 5 Result

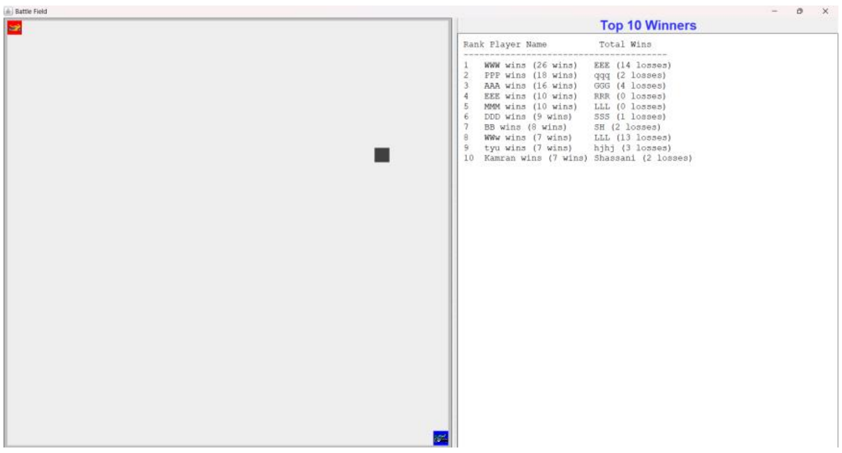
SSS wins this round!
SSS Score: 1
DDD Score: 3

```
7   BB w
8   WWw
9   tyu
10  Kamra
```



Black-Box Testing

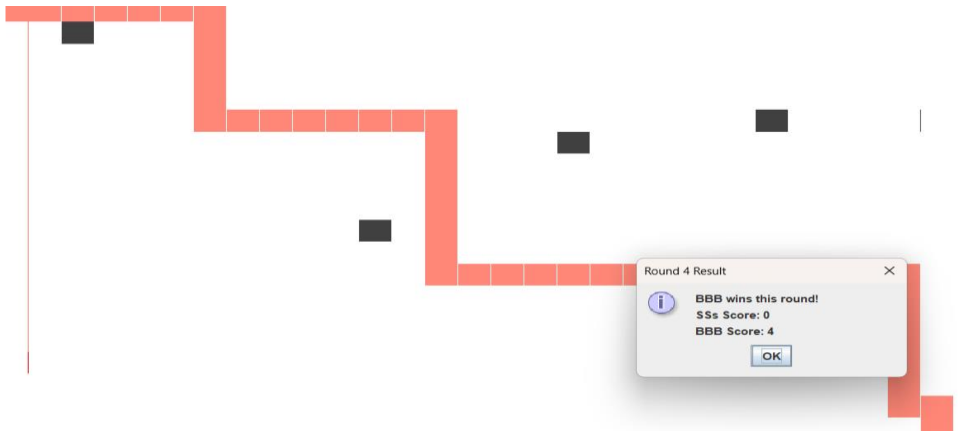
1. **Test Game Initialization**:
    a. **Input**: Start a new game.
    b. **Expected Output**: Grid initializes with bikes in starting positions.



Top 10 Winners

```
Rank Player Name             Total Wins
----------------------------------------
1    WWW wins (26 wins)    EEE (14 losses)
2    PPP wins (18 wins)    qqq (2 losses)
3    AAA wins (16 wins)    GGG (4 losses)
4    EEE wins (10 wins)    RRR (0 losses)
5    MMM wins (10 wins)    LLL (0 losses)
6    DDD wins (9 wins)     SSS (1 losses)
7    BB wins (8 wins)      SH (2 losses)
8    WWw wins (7 wins)     LLL (13 losses)
9    tyu wins (7 wins)     hjhj (3 losses)
10   Kamran wins (7 wins) Shassani (2 losses)
```

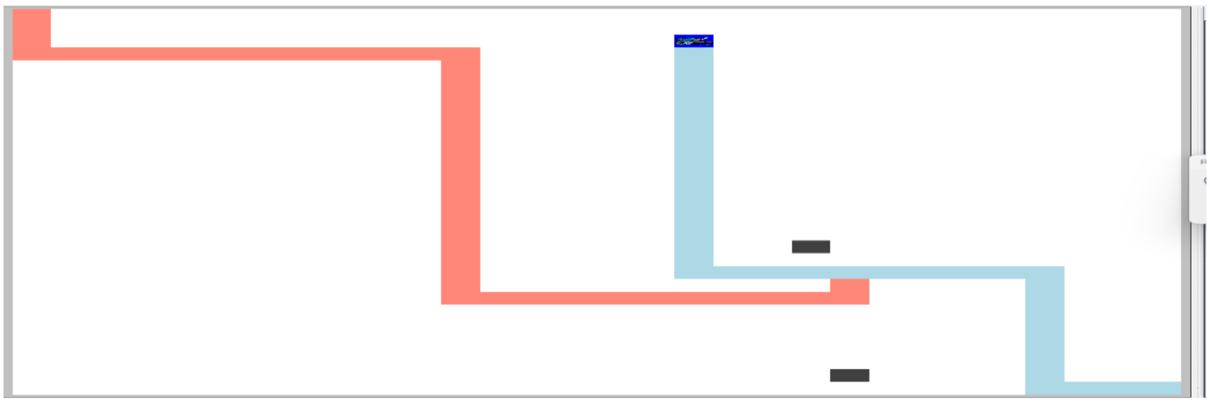
2. **Test Player Movement**:
    a. **Input**: Player 1 presses DOWN at `(0, 0)`.
    b. **Expected Output**: Player 1 moves to `(1, 0)`.

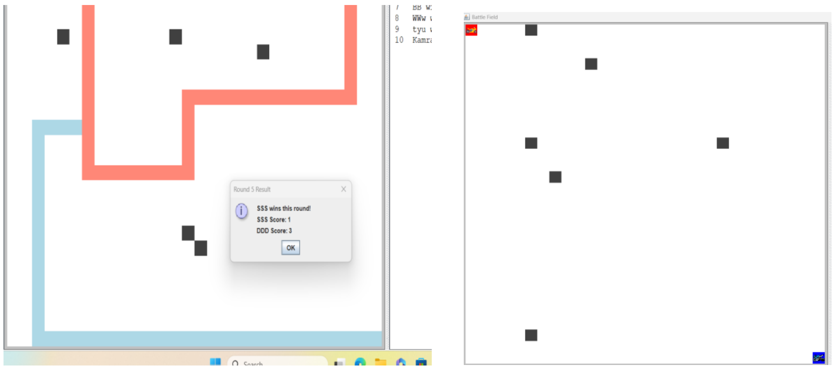

Round 4 Result

BBB wins this round!
SSs Score: 0
BBB Score: 4

3. **Test Collision Of Both Players**:
    a. Input: players move into the trails of each other.
    b. Expected Output: The match will end in a draw.

4. **Test Collision with Trail**:
    a. **Input**: Player moves into a trail.
    b. **Expected Output**: The player crashes, and the other player scores.

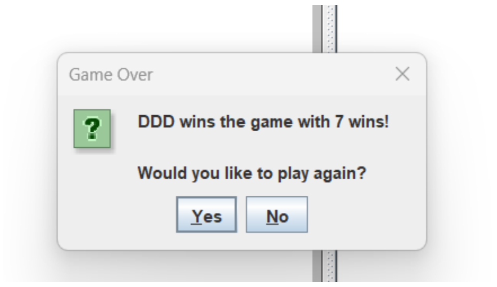


5. **Test Round Transition**:
    a. **Input**: A round ends.
    b. **Expected Output**: Grid expands, and players reset.
    c. 
Round 5 Result

SSS wins this round!
SSS Score: 1
DDD Score: 3

6. **Test Game Restart**:
    a. **Input**: Select "Yes" to restart after a round ends.

b. **Expected Output**: Game restarts with scores retained.

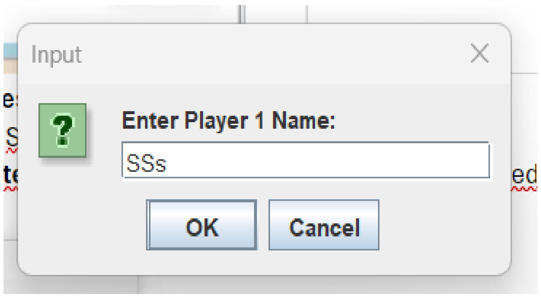


Game Over

DDD wins the game with 7 wins!

Would you like to play again?

Yes No

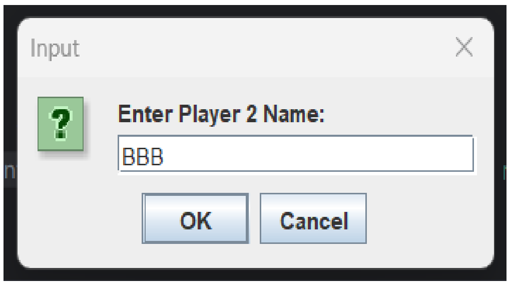


Input

Enter Player 1 Name:

SSs

OK Cancel

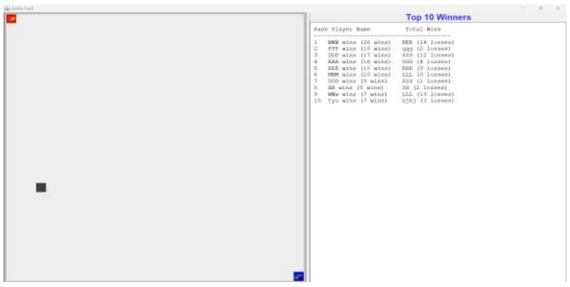


Input

Enter Player 2 Name:

BBB

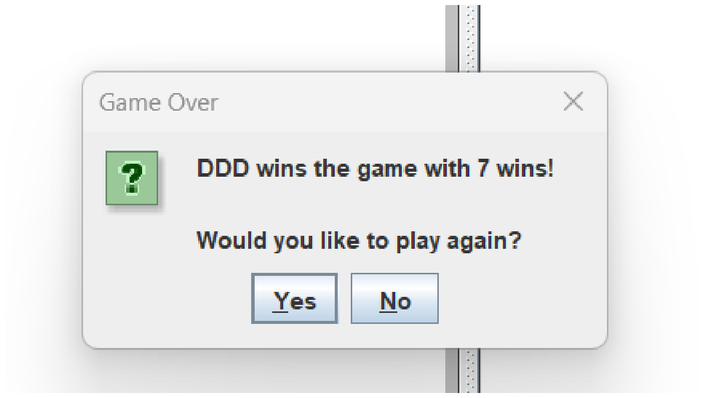
OK Cancel



Top 10 Winners

```
Rank Player Name          Total Wins
----------------------------------------
1    WWW wins (26 wins)    EEE (14 losses)
2    FFF wins (18 wins)    qqq (2 losses)
3    DDD wins (17 wins)    SSS (12 losses)
4    AAA wins (16 wins)    GGG (4 losses)
5    EEE wins (10 wins)    RRR (0 losses)
6    MMM wins (10 wins)    LLL (0 losses)
7    DDD wins (9 wins)     SSS (1 losses)
8    BB wins (8 wins)      SH (2 losses)
9    WWw wins (7 wins)     LLL (13 losses)
10   tyu wins (7 wins)     hjhj (3 losses)
```

7. **Test Winning Condition**:
   a. **Input**: Player wins 10 rounds.
   b. **Expected Output**: Game declares the winner and offers to restart or exit.



Game Over

DDD wins the game with 7 wins!

Would you like to play again?

Yes No