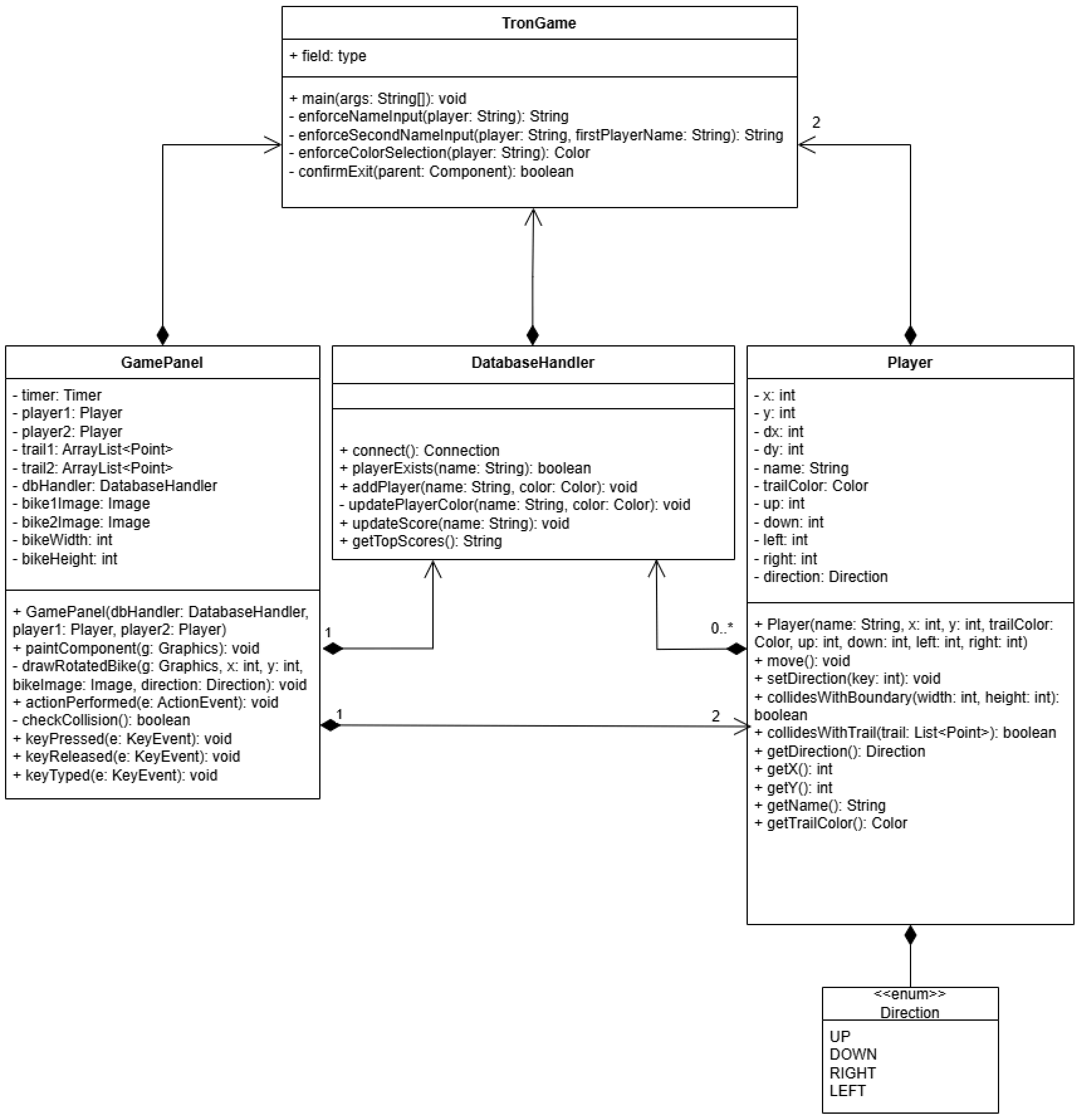
Programming technology 3rd assignment

Name: Abu Snaineh Ata Atosh
Neptun code: XP6PGI

Description of the exercise:
4. Tron 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.

Class Diagram:


TronGame
+ field: type
+ main(args: String[]): void
- enforceNameInput(player: String): String
- enforceSecondNameInput(player: String, firstPlayerName: String): String
- enforceColorSelection(player: String): Color
- confirmExit(parent: Component): boolean
2
GamePanel
- timer: Timer
- player1: Player
- player2: Player
- trail1: ArrayList<Point>
- trail2: ArrayList<Point>
- dbHandler: DatabaseHandler
- bike1Image: Image
- bike2Image: Image
- bikeWidth: int
- bikeHeight: int
+ GamePanel(dbHandler: DatabaseHandler, player1: Player, player2: Player)
+ paintComponent(g: Graphics): void
- drawRotatedBike(g: Graphics, x: int, y: int, bikeImage: Image, direction: Direction): void
+ actionPerformed(e: ActionEvent): void
- checkCollision(): boolean
+ keyPressed(e: KeyEvent): void
+ keyReleased(e: KeyEvent): void
+ keyTyped(e: KeyEvent): void
DatabaseHandler
+ connect(): Connection
+ playerExists(name: String): boolean
+ addPlayer(name: String, color: Color): void
- updatePlayerColor(name: String, color: Color): void
+ updateScore(name: String): void
+ getTopScores(): String
1
0..*
1
2
Player
- x: int
- y: int
- dx: int
- dy: int
- name: String
- trailColor: Color
- up: int
- down: int
- left: int
- right: int
- direction: Direction
+ Player(name: String, x: int, y: int, trailColor: Color, up: int, down: int, left: int, right: int)
+ move(): void
+ setDirection(key: int): void
+ collidesWithBoundary(width: int, height: int): boolean
+ collidesWithTrail(trail: List<Point>): boolean
+ getDirection(): Direction
+ getX(): int
+ getY(): int
+ getName(): String
+ getTrailColor(): Color
<<enum>>
Direction
UP
DOWN
RIGHT
LEFT

Description of the Methods:

Main Class: TronGame

`main(args: String[]): void`

The entry point of the application. Initializes the game window, players, database handler, and game panel. It also configures menu options for viewing high scores and restarting the game.

`enforceNameInput(player: String): String`
Prompts the user to input a valid name for a player. Ensures that the name is not empty and allows the user to cancel or confirm exiting the game.

`enforceSecondNameInput(player: String, firstPlayerName: String): String`
Prompts the user to input a valid name for the second player. Ensures that the name is not the same as the first player's name.

`enforceColorSelection(player: String): Color`
Opens a color chooser dialog for the user to select a trail color for a player. Ensures a valid color is selected before proceeding.

`confirmExit(parent: Component): boolean`
Displays a confirmation dialog asking the user if they want to exit the game. Returns `true` if the user confirms; otherwise, `false`.

Class: DatabaseHandler

`connect()`
Establishes a connection to the MySQL database using JDBC. Returns a `Connection` object or `null` if the connection fails.

`playerExists(name: String): boolean`
Checks whether a player with the given name exists in the database. Returns `true` if the player exists; otherwise, `false`.

`addPlayer(name: String, color: Color): void`
Adds a new player with the specified name and trail color to the database. If the player already exists, it updates their trail color.

`updatePlayerColor(name: String, color: Color): void`
Updates the trail color of an existing player in the database.

`updateScore(name: String): void`
Increments the score of the player with the specified name by 1 in the database.

`getTopScores(): String`
Retrieves the top 10 players with the highest scores from the database and returns a formatted leaderboard string.

Class: GamePanel

`GamePanel(dbHandler: DatabaseHandler, player1: Player, player2: Player)`

Constructor that initializes the game panel, including players, trails, and the timer. It also loads bike images and starts the game.

`paintComponent(g: Graphics): void`
Draws the game state, including the players' trails and bikes, onto the panel.

`drawRotatedBike(g: Graphics, x: int, y: int, bikeImage: Image, direction: Direction): void`
Draws a bike image rotated according to the player's direction.

`actionPerformed(e: ActionEvent): void`
Moves both players and checks for collisions after every timer tick. Updates the trails and repaints the game state.

`checkCollision(): boolean`
Checks if either player has collided with the boundaries or the other player's trail. Stops the game and updates the database if a collision occurs.

`keyPressed(e: KeyEvent): void`
Handles keyboard input to change the direction of the players based on their assigned controls.

`keyReleased(e: KeyEvent): void`
Empty implementation; required by the `KeyListener` interface.

`keyTyped(e: KeyEvent): void`
Empty implementation; required by the `KeyListener` interface.

Class: Player

`Player(name: String, x: int, y: int, trailColor: Color, up: int, down: int, left: int, right: int)`
Constructor that initializes the player with a name, initial position, trail color, and key bindings for movement.

`move(): void`
Updates the player's position based on their current direction.

`setDirection(key: int): void`
Changes the player's direction based on the pressed key, ensuring it aligns with the player's assigned controls.

`collidesWithBoundary(width: int, height: int): boolean`
Checks if the player has collided with the boundaries of the game panel.

`collidesWithTrail(trail: List<Point>): boolean`
Checks if the player has collided with a given trail, including their own.

`getDirection(): Direction`
Returns the current movement direction of the player.

`getX(): int`
Returns the current X-coordinate of the player.

`getY(): int`

Returns the current Y-coordinate of the player.

`getName(): String`
Returns the name of the player.

`getTrailColor(): Color`
Returns the trail color of the player.

Enum: Direction

Values:

- **UP:** Indicates upward movement.
- **DOWN:** Indicates downward movement.
- **LEFT:** Indicates movement to the left.
- **RIGHT:** Indicates movement to the right.

Test cases:

Duplicate Names

- **As a player** I want to avoid using the same name for both players.
- **Given** both players are entering their names during setup.
- **When** I enter the same name as the other player.
- **Then** I should receive an error message indicating that names must be unique.

Duplicate Colors

- **As a player** I want to avoid using the same trail color as the other player.
- **Given** both players are selecting their trail colors during setup.
- **When** I choose the same color as the other player.
- **Then** I should receive an error message indicating that colors must be unique.

Empty Name Input

- **As a player** I want to avoid leaving my name blank during setup.
- **Given** I am prompted to enter my name.
- **When** I leave the input field blank or press OK without entering a name.
- **Then** I should receive an error message indicating that the name cannot be empty.

Winning the Game

- **As a player** I want to be notified when I win the game.
- **Given** I achieve four adjacent squares of my color.
- **When** the winning condition is satisfied.
- **Then** the game declares me the winner and displays a congratulatory message.

Resetting the Game

- **As a player** I want to reset the game if needed.
- **Given** I am in the middle of a game.
- **When** I select the restart option from the menu.
- **Then** the game resets to the initial state with all scores cleared.

Top Scores Display

- **As a player** I want to view the top scores at any time.
- **Given** the game has been played by multiple players.
- **When** I select the "View High Scores" option from the menu.
- **Then** the leaderboard displays the top 10 scores in descending order.

Collision Detection

- **As a player** I want to know when I collide with a boundary or trail.
- **Given** I am controlling my knight.
- **When** my knight collides with the edge of the board or another trail.
- **Then** the game stops, the other player wins, and their score is updated.

Invalid Key Press

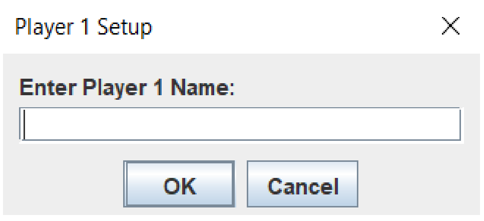
- **As a player** I want to ensure my knight doesn't move on invalid key presses.
- **Given** I am controlling my knight.
- **When** I press a key that is not assigned for movement (e.g., an unbound key).
- **Then** my knight should remain in its current position, and no movement occurs.

Restart After Win

- **As a player** I want to start a new game after a winner is declared.
- **Given** a player has won the game.
- **When** I select the "Restart Game" option from the menu.
- **Then** the game resets, and both players start with their original scores and positions.

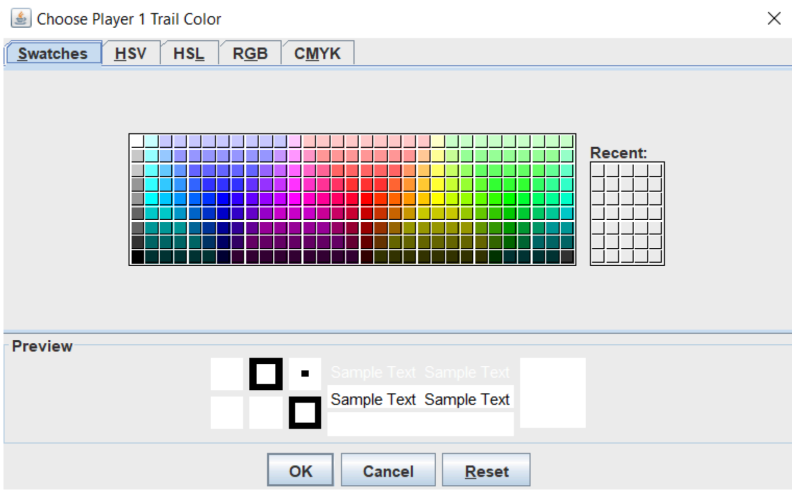
Game Screenshots:

Player 1 name input field:

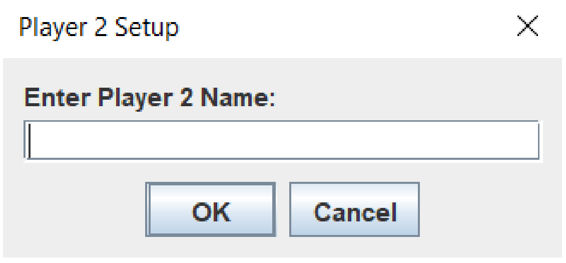

Player 1 Setup

Enter Player 1 Name:

Player 1 color choosing window:

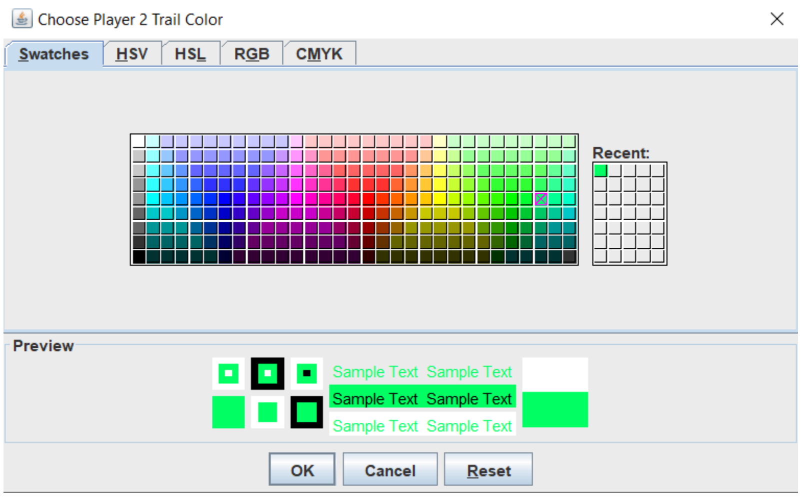

Choose Player 1 Trail Color

Player 2 name input field:

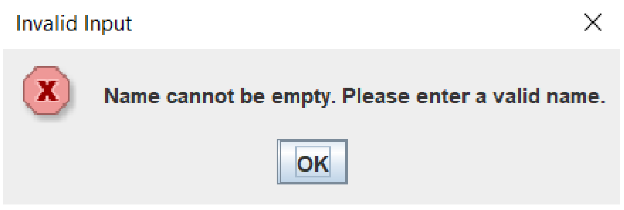

Player 2 Setup

Enter Player 2 Name:

Player 2 color choosing window:

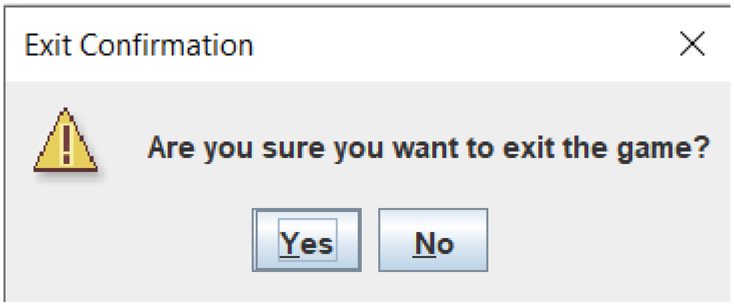

Choose Player 2 Trail Color

Empty name error message:


Invalid Input

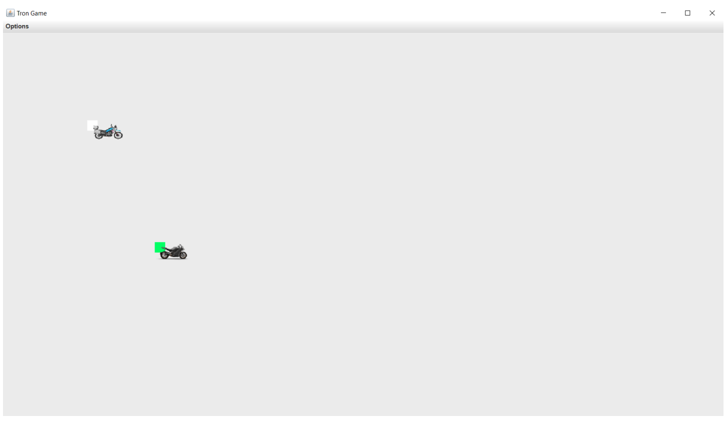
Name cannot be empty. Please enter a valid name.

Exit from application confirmation message:

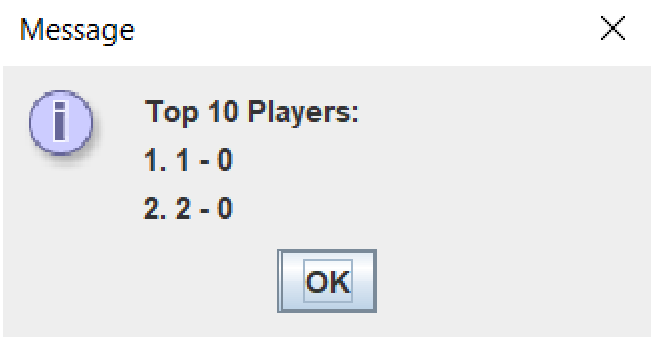

Exit Confirmation

Are you sure you want to exit the game?

Game panel:

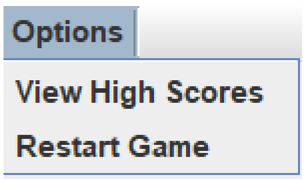


Leader board:


Message

Top 10 Players:
1. 1 - 0
2. 2 - 0

Menu items:


Options
View High Scores
Restart Game