

Muhammad Al Farizi
OCSWOM

ocswom@inf.elte.hu

Practice Group 5

3rd assignment/4. Task

6th December 2024

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.

Analysis

The program is a game called "Tron Battle" inspired by the Tron game.

Database

PlayerEntity:

Represents a player with their name, hashed password, and registration date. It includes methods to validate passwords and retrieve player details.

GameEntity:

Represents a game with its name and release date. It provides basic access to game properties.

HighScoreEntity:

Represents a player's score, linking a PlayerEntity with a score value.

HighScoreDB:

Manages database operations (CRUD) for high scores, players, and games, ensuring interactions like adding players, updating leaderboards, and retrieving scores.

Model

Tron:

Represents a player's "Tron" entity in the game. It manages the color, movement, score increment, bonus effects, and drawing logic for the Tron on the game board.

Player:

Represents a player by extending basic player data from PlayerEntity. It handles the score, a collection of Tron objects, and manages switching between active Trons during gameplay.

Movement:

Manages the movement logic for a player or object. It keeps track of position, direction, speed, and responds to key events to update movement accordingly.

Bonus:

Represents a bonus item on the game board. It ensures the bonus appears at random positions without overlapping with occupied areas and provides drawing logic for visualization.

GraphicsImages:

Loads and manages images used in the game, such as the scoreboard and logo, to be displayed on the UI.

Music:

Handles playing sound files in the game. It loads audio from a file and streams it for playback during gameplay.

Board:

The main game engine for the Tron Battle game, managing the game logic, rendering the game state, and handling player movements and collisions. It coordinates player actions, checks for deaths, handles bonuses, and manages the overall game loop using a timer and regular updates. It also interacts with other components like the Player, Bonus, Music, and GraphicsImages classes, and updates the user interface with BoardGUI. It supports key event handling for player movement and session saving to the high score database.

View

MenuGUI:

This class represents the main menu of the game, allowing players to start the game or exit the application. It initializes the UI components like buttons and handles their actions.

AuthenticationGUI:

It manages the player login and game setup, providing fields for username, password, and color selection. It ensures both players are authenticated and ready to play.

BoardGUI:

Displays the Tron game board, integrating the game logic with the visual interface and handling player key inputs during gameplay.

MessageGUI:

Displays informative or error messages with a back button that redirects players to the main menu.

HighScoreGUI:

Displays the list of high scores, presenting player names and their scores in a scrollable panel with a back button to return to the main menu.

Controller

TronBattle

The controller that initializes the game, sets up the database connection, and displays the main menu GUI. It manages the overall game flow, connecting to the high score database and providing access to it, while also launching the game with a default date for the game entity.

Method Description

A short description on the methods used in each class. Getters and Setters are excluded.

Database

PlayerEntity	+ PlayerEntity(String name, String password, Date registerDate, PasswordType passwordType)	Constructs a PlayerEntity object with the specified name, password, registration date, and password type.
---------------------	--	---

	+ getName(): String	Returns the player's name.
	+ getPasswordHash(): String	Returns the hashed password of the player.
	+ getRegisterDate(): Date	Returns the player's registration date.
	+ checkPassword(String password): boolean	Checks if the given plain password matches the stored hash password.
	+ toString(): String	Returns a string representation of the player.

GameEntity	+ GameEntity(String name, Date releaseDate)	Creates a GameEntity object with the specified name and release date.
	+ getName(): String	Returns the game's name.
	+ getReleaseDate(): Date	Returns the game's release date.
	+ toString(): String	Returns a string representation of the game.

HighScoreEntity	+ HighScoreEntity(PlayerEntity player, int score)	Creates a HighScoreEntity object with the specified player and score.
	+ getPlayer(): PlayerEntity	Returns the player associated with the score.
	+ getScore(): int	Returns the score of the player.
	+ toString(): String	Returns a string representation of the high score entry.

HighScoreDB	+ HighScoreDB(GameEntity game) throws SQLException	Initializes the database connection and game-related information.
	+ putHighScore(HighScoreEntity highScore): void	Adds a high score for a player.
	+ getHighScore(HighScoreEntity highScore): int	Retrieves the current high score of a player.
	+ getHighScores(): List<HighScoreEntity>	Retrieves the list of all high scores for the game.
	+ getPlayer(String name, String password): PlayerEntity	Retrieves a player by name and password.
	+ getPlayer(String name): PlayerEntity	Retrieves a player by name.
	- insertGame(GameEntity game): int	Inserts a new game into the database if it does not already exist.

	- insertPlayer(PlayerEntity player): int	Inserts a new player into the database if it does not already exist.
	- insertSessions(HighScoreEntity highScore): int	Inserts a new session into the database for a player.
	- insertLeaderboard(HighScoreEntity highScore): void	Inserts or updates the leaderboard with the player's high score for the game.
	- retrievePlayerNo(PlayerEntity player): int	Retrieves the player number (id) based on the player's game and password hash.
	- retrieveGameId(GameEntity game): int	Retrieves the game id based on the game's name.
	- retrieveHighScoreFromSessions(HighScoreEntity highScore): int	Retrieves the high score from the sessions for a specific player and game.
	- retrieveHighScoreFromLeaderboard(HighScoreEntity highScore): int	Retrieves the high score from the leaderboard for a specific player and game.

Model

Tron	+Tron(Color color, Movement movement)	Constructor that initializes the Tron player with a specific color and movement behavior.
	+ getColor(): Color	Returns the color of the Tron player.
	+ getMovement(): Movement	Returns the Movement object associated with the Tron player.
	+ isDead(): boolean	Checks if the player has gone out of bounds. Stops movement if out of bounds.
	+ incrementScore(Player player)	Increments the score of the specified player.
	+ draw(Graphics g, Player player)	Draws the Tron player on the screen based on its movement and position.
	+ set(int x, int y, int dx, int dy, int speed)	Sets the position, direction, and speed of the Tron player.
	+ activeBonusPass(): boolean	Decrements the active bonus counter and increases player speed if the bonus expires.

Player	+ Player(PlayerEntity playerEntity, Color color, int[][] initialPositions, int up, int down, int left, int right, int dx, int dy, int speed)	Constructor that initializes a player with a list of Trons, score, and movement keys.
---------------	--	---

	+ getScore(): int	Returns the player's current score.
	+ getTrons(): List<Tron>	Returns the list of Trons associated with the player.
	+ getCurrentTron(): Tron	Returns the currently active Tron for the player.
	+ nextCurrentTron()	Switches to the next Tron in the list, looping back to the first if needed.
	+ incrementScore()	Increments the player's score by one.

Movement	+ Movement(int x, int y, int up, int down, int left, int right, int dx, int dy, int speed)	Constructor that initializes the position, direction, speed, and movement keys.
	+ move()	Moves the object based on the current direction and speed.
	+ setX(int x)	Sets the x-coordinate position.
	+ setY(int y)	Sets the y-coordinate position.
	+ setDX(int dx)	Sets the horizontal direction (dx) of movement.
	+ setDY(int dy)	Sets the vertical direction (dy) of movement.
	+ getX(): int	Returns the x-coordinate position.
	+ getY(): int	Returns the y-coordinate position.
	+ getDX(): int	Returns the horizontal direction (dx) of movement.
	+ getDY(): int	Returns the vertical direction (dy) of movement.
	+ getSpeed(): int	Returns the movement speed.
	+ keyPressed(KeyEvent e)	Handles key press events to change movement direction.
	- isValidMovement(Movement movement, KeyEvent e): Boolean	Checks if the current key press results in a valid movement based on the player's current direction.
	- moveLeft()	Updates movement to go left.
	- moveRight()	Updates movement to go right.
	- moveUp()	Updates movement to go up.
	- moveDown()	Updates movement to go down.
	+ changeSpeed(int speed)	Changes the speed and adjusts position accordingly.

Bonus	+ Bonus(Color color, Board board)	Constructor that initializes a player with a list of Trons, score, and movement keys.
	+ getX(): int	Returns the player's current score.
	+ getY(): int	Returns the list of Trons associated with the player.
	+ setX(int x)	Returns the currently active Tron for the player.
	+ setY(int y)	Switches to the next Tron in the list, looping back to the first if needed.
	+ getColor(): Color	Returns the color of the bonus.
	+ draw(Graphics g)	Draws the bonus on the board at a random valid position.
	+ randomPosition()	Randomly assigns a position on the board where no area is already taken.
GraphicsImages	+ GraphicsImages(String scoreBoard, String logo)	Constructor that loads the scoreboard and logo images from the provided file paths.
	+ getScoreBoard(): Image	Returns the scoreboard image.
	+ getLogo(): Image	Returns the logo image.
	- loadScoreBoard(String scoreBoard)	Loads the scoreboard image from the specified file path.
	- loadLogo(String logo)	Loads the logo image from the specified file path.
Music	+ playSound(String filename)	Plays an audio file specified by the file path using a buffered audio stream.
Board	+Board(BoardGUI gui, PlayerEntity player1Entity, Color player1Color, PlayerEntity player2Entity, Color player2Color)	Initializes the game board, players, bonus system, and other components needed for the Tron Battle game.
	+paintComponent(Graphics g): void	Renders the game board, including players, score, bonus items, and highlights.
	+actionPerformed(ActionEvent e): void	Updates the game state by moving players, checking collisions, and handling

		bonuses.
	-resetPlayer(Player player, int[][] positions, int dx, int dy, int speed): void	Resets a player's state (position, speed) after a death or game restart.
	-markTaken(Player player): void	Marks the areas occupied by the player's trail on the game grid.
	-checkTaken(Player player): boolean	Checks if the player has collided with a wall or another player's trail.
	-checkBonus(Player player): boolean	Checks if the player has collected a bonus item.
	+getTaken(int x, int y): Boolean	Checks if the given coordinates (x, y) are occupied by either player's trail.
	+keyPressed(KeyEvent e): void	Handles key press events for player movement and saving the session.
	-saveSession(): void	Saves the current game session to the high scores database.
	-isPlayerMovementKey(int key, Player player): Boolean	Determines if a key press corresponds to a valid movement for the specified player.
	-processPlayerMovement(Player player, KeyEvent e): void	Processes the movement of a player based on the key press event.
	-isValidMovement(Movement movement, KeyEvent e): Boolean	Checks if the current key press results in a valid movement based on the player's current direction.

View

MenuGUI	+ MenuGUI()	Constructs and initializes the main menu with Start and Exit buttons.
----------------	-------------	---

	+ AuthenticationGUI()	Constructs and initializes the authentication window for player setup.
	- addEscapeKeyListenerToComponents(Component... components): void	Adds an Escape key listener to components to return to the main menu.
	- createPlayerPanel(String title, AtomicBoolean readyFlag): JPanel	Creates and configures an authentication panel for a player, including fields and buttons.

AuthenticationGUI	- createTextField(String placeholder): JTextField	Creates a text field with a placeholder for user input.
	- createPasswordField(String placeholder): JPasswordField	Creates a password field with a placeholder for user input.
	- setPlaceholder(JTextField textField, String placeholder): void	Adds placeholder text functionality to a text field.
	- setPlaceholder(JPasswordField passwordField, String placeholder): void	Adds placeholder text functionality to a password field.
	- getColorFromSelection(String colorName): Color	Converts a color name string into a corresponding Color object.
	- checkReady(): void	Checks if both players are ready, and starts the game if true.

BoardGUI	+ BoardGUI(PlayerEntity player1, Color player1Color, PlayerEntity player2, Color player2Color)	Initializes the game board GUI and assigns player colors.
	+ <u>reset(): void</u>	Disposes of the current game board frame and resets it.

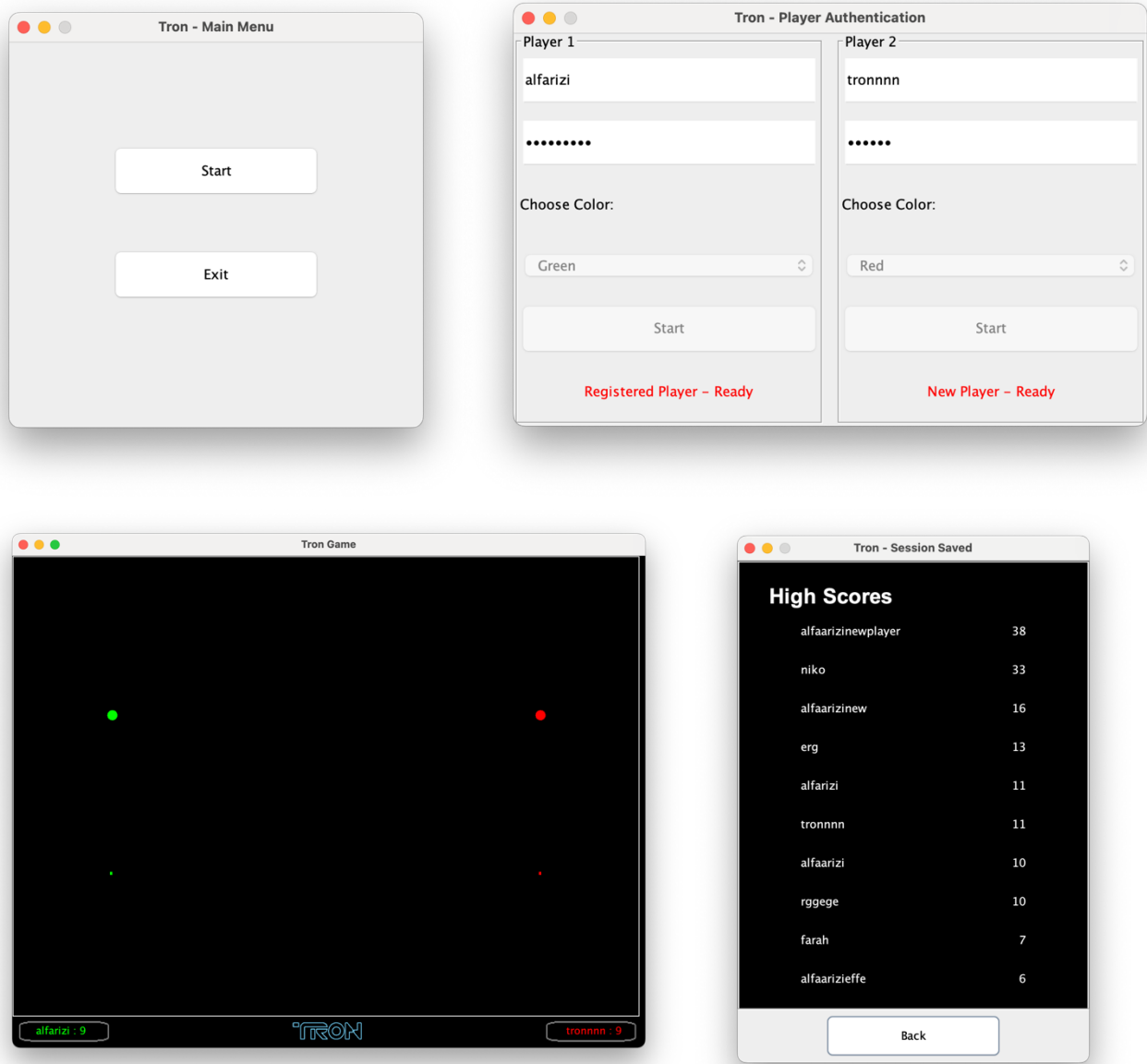
MessageGUI	+ MessageGUI(String title, String message, Color color)	Displays a message window with a back button to return to the main menu.
-------------------	---	--

HighScoreGUI	+ HighScoreGUI(String title, List<HighScoreEntity> highScores, Color color)	Constructs a window displaying a list of high scores with a back button.
---------------------	---	--

Controller

TronBattle	+ TronBattle()	Constructs the TronBattle game, initializes the database, and displays the main menu GUI
	+ <u>getDatabase(): HighScoreDB</u>	Returns the HighScoreDB instance for accessing high scores. Throws an exception if the database is not initialized.

Screenshots



Testing

Test Player Movement

As a player

I want to control my Tron using keyboard keys

Given the game is running and my Tron is visible

When I press the assigned movement keys (W/A/S/D or Arrow keys)

Then my Tron should move in the corresponding direction without any delays

Test Collision Detection

As a player

I want to lose the game if I collide with my trail, the opponent's trail, or the game boundary

Given my Tron is moving

When my Tron collides with a trail or boundary

Then the game should register my Tron as "dead" and increment the opponent's score

Test Bonus Pickup

As a player

I want to collect bonuses to gain temporary advantages

Given a bonus is visible on the board

When my Tron passes over the bonus

Then the bonus effect should activate (e.g., speed increase) and the bonus should disappear

Test Score Update

As a player

I want to see my score updated when my opponent loses

Given the opponent's Tron collides with a trail or the boundary

When the opponent is registered as "dead,"

Then my score should increment and be displayed on the scoreboard.

Test Game Reset After Death

As a player

I want to restart the game after one player dies

Given one player has lost the round

When the game resets

Then both players' Trons should return to their starting positions, and the board should reset without trails or active bonuses.

Test High Score Saving

As a player

I want to save my game session high scores

Given I press the spacebar to save the session

When the game session ends

Then my score should be stored in the high score database and displayed in the high score GUI.

Test Menu Navigation

As a user

I want to access the main menu to start the game

Given the game is launched

When I interact with the menu options

Then I should be able to start the game or view the high scores.

Test Bonus Effects Duration

As a player

I want bonuses to have temporary effects

Given I pick up a speed bonus

When the bonus duration ends

Then my Tron's speed should return to its original state

Test Multiple Bonuses

As a player

I want to pick up bonuses without conflicting effects

Given there are multiple bonuses on the board

When I collect one bonus

Then only one effect should activate, and the others should remain visible for collection

Test Database Connection Failure

As a developer,

I want to ensure the game handles database errors gracefully,

Given the database connection fails during initialization,

When the game attempts to access the high score database,

Then the game should display an error message and continue running without crashing.