IKHQOK
Heng Freddy Zi Ping
ikhqok@inf.elte.hu

Task

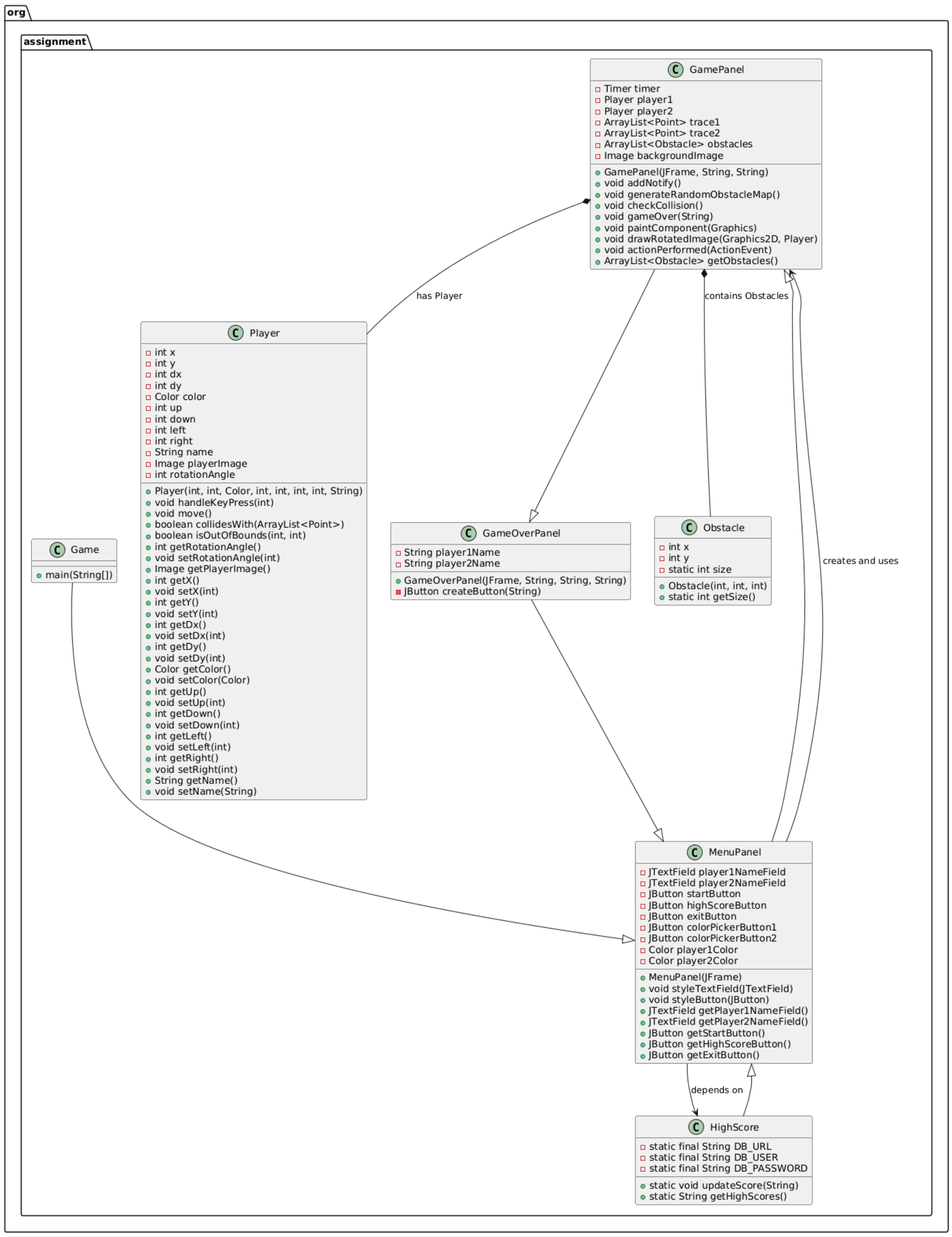
The Tron game is a two-player game where players control motorcycles that leave trails behind them. The goal of the game is to avoid crashing into walls or the trails of the opponent while trying to make the opponent crash. The game features a high score system, and players can see their names and scores in a leaderboard after each game. The game includes several panels (game, menu, game over) to manage the game's flow, as well as logic to handle player movements, collisions, and score tracking.

Analysis

1. **Game Structure**
    - **Game Panels**:
        - MenuPanel: Displays the main menu, where the player can start a new game or view the high scores.
        - GamePanel: Displays the game board, handles user input for movement, and handles the logic for player movement, collision detection, and score tracking.
        - GameOverPanel: Displays the result (win/loss) at the end of the game and allows for restarting the game.
    - **Player Class**:
      Represents a player in the game. Each player has a position, movement direction, color, and key mappings for controlling movement. Players can move their motorcycle and leave a trail behind.
2. **Game Logic**
    - **Movement**: Players can move their motorcycles in four directions (up, down, left, right) based on key presses. The motorcycle leaves a trail that cannot be crossed.
    - **Obstacles:** There are 10 predefined maps with obstacles scattered throughout each level.
    - **Collision Detection**: The game checks if a player crashes into the walls, obstacles or another player's trail (including their own).
    - **Game End**: The game ends when a player crashes, and the other player is declared the winner.
    - **High Scores**: After a game ends, the score is updated in the database and displayed in a leaderboard.
    - **Game Flow**:
      The game runs real-time for the two players, handles movement, detects collisions, and updates the score.

3. **UI Design**
    - **GameBoard**: The game is played on a grid of cells. Players' movements are displayed as colored trails behind their motorcycles.
        - Red for Player 1
        - Blue for Player 2
    - **Menus**: Players can start a new game or view high scores via the main menu. The game over panel shows the final result.

UML Diagram


org
assignment
GamePanel
- Timer timer
- Player player1
- Player player2
- ArrayList<Point> trace1
- ArrayList<Point> trace2
- ArrayList<Obstacle> obstacles
- Image backgroundImage
+ GamePanel(JFrame, String, String)
+ void addNotify()
+ void generateRandomObstacleMap()
+ void checkCollision()
+ void gameOver(String)
+ void paintComponent(Graphics)
+ void drawRotatedImage(Graphics2D, Player)
+ void actionPerformed(ActionEvent)
+ ArrayList<Obstacle> getObstacles()
has Player
contains Obstacles
creates and uses
Player
- int x
- int y
- int dx
- int dy
- Color color
- int up
- int down
- int left
- int right
- String name
- Image playerImage
- int rotationAngle
+ Player(int, int, Color, int, int, int, int, String)
+ void handleKeyPress(int)
+ void move()
+ boolean collidesWith(ArrayList<Point>)
+ boolean isOutOfBounds(int, int)
+ int getRotationAngle()
+ void setRotationAngle(int)
+ Image getPlayerImage()
+ int getX()
+ void setX(int)
+ int getY()
+ void setY(int)
+ int getDx()
+ void setDx(int)
+ int getDy()
+ void setDy(int)
+ Color getColor()
+ void setColor(Color)
+ int getUp()
+ void setUp(int)
+ int getDown()
+ void setDown(int)
+ int getLeft()
+ void setLeft(int)
+ int getRight()
+ void setRight(int)
+ String getName()
+ void setName(String)
Game
+ main(String[])
GameOverPanel
- String player1Name
- String player2Name
+ GameOverPanel(JFrame, String, String, String)
- JButton createButton(String)
Obstacle
- int x
- int y
- static int size
+ Obstacle(int, int, int)
+ static int getSize()
MenuPanel
- JTextField player1NameField
- JTextField player2NameField
- JButton startButton
- JButton highScoreButton
- JButton exitButton
- JButton colorPickerButton1
- JButton colorPickerButton2
- Color player1Color
- Color player2Color
+ MenuPanel(JFrame)
+ void styleTextField(JTextField)
+ void styleButton(JButton)
+ JTextField getPlayer1NameField()
+ JTextField getPlayer2NameField()
+ JButton getStartButton()
+ JButton getHighScoreButton()
+ JButton getExitButton()
depends on
HighScore
- static final String DB_URL
- static final String DB_USER
- static final String DB_PASSWORD
+ static void updateScore(String)
+ static String getHighScores()

Methods

Class	Methods	Description
Game	startGame()	Starts the game, initializing the game board, players, and setting the current game state.
	endGame()	Ends the game, showing the result and updating the scores.
GamePanel	paintComponent(Graphics g)	Renders the game board and player trails on the screen.
	handleKeyPress(int keyCode)	Handles player input for controlling movements.
	checkCollisions()	Checks for collisions with walls or trails.
	updatePlayerPositions()	Updates the position of players based on movement.
	resetGame()	Resets the game to start a new round.
Player	move()	Moves the player based on the current direction.
	handleKeyPress(int keyCode)	Changes direction based on key input.
	isOutOfBounds(int width, int height)	Checks if the player is out of bounds.
HighScore	updateScore(String playerName)	Updates or inserts the player's score in the database.
	getHighScores()	Retrieves and returns the top 10 high scores from the database.
MenuPanel	showMenu()	Displays the main menu with options to start a new game or view high scores.
GameOverPanel	showGameOver(String result)	Displays the end of the game result (win, loss, or draw) and allows restarting the game.

White Box Testing

Test Name	Class	Method	Description	Expected Outcome
testInitialPosition	Player	getX, getY	Verifies the player's initial X and Y position.	Player's X and Y positions should match the initialized values (e.g., 100, 100).
testMoveUp	Player	handleKeyPress, move	Tests the player's upward movement.	Player's Y position should decrease by 10, while X remains unchanged.
testMoveDown	Player	handleKeyPress, move	Tests the player's downward movement.	Player's Y position should increase by 10, while X remains unchanged.
testMoveLeft	Player	handleKeyPress, move	Tests the player's leftward movement.	Player's X position should decrease by 10, while Y remains unchanged.
testMoveRight	Player	handleKeyPress, move	Tests the player's rightward movement.	Player's X position should increase by 10, while Y remains unchanged.
testOutOfBoundsLeft	Player	move, isOutOfBounds	Verifies detection of the player moving out of the left boundary.	isOutOfBounds() should return true when the player moves past the left boundary.

testOutOfBoundsRight	Player	move, isOutOfBounds	Verifies detection of the player moving out of the right boundary.	isOutOfBounds() should return true when the player moves past the right boundary.
testOutOfBoundsTop	Player	move, isOutOfBounds	Verifies detection of the player moving out of the top boundary.	isOutOfBounds() should return true when the player moves past the top boundary.
testOutOfBoundsBottom	Player	move, isOutOfBounds	Verifies detection of the player moving out of the bottom boundary.	isOutOfBounds() should return true when the player moves past the bottom boundary.
testChangeDirectionWhileMoving	Player	handleKeyPress, move	Verifies direction change while moving is correctly handled.	The player should continue in the initial direction if a conflicting input is pressed.
testCollisionWithTrace	Player	collidesWith	Tests collision detection with a trace.	collidesWith() should return true if the player's position matches any point in the trace.

Black Box Testing

Test Name	Class	Method	Description	Expected Outcome
testPlayer1NameField	MenuPanel	getPlayer1NameField	Verifies if the Player 1 name field correctly retrieves entered text.	The retrieved text should match the entered value.
testPlayer2NameField	MenuPanel	getPlayer2NameField	Verifies if the Player 2 name field correctly retrieves entered text.	The retrieved text should match the entered value.
testStartButtonAction	MenuPanel	getStartButton	Verifies the action triggered when the Start Game button is clicked.	The GamePanel should be set as the content pane of the frame.
testHighScoreButtonAction	MenuPanel	getHighScoreButton	Verifies the action triggered when the High Score button is clicked.	The expected action should execute without errors.
testLayout	MenuPanel	Component Methods	Verifies the layout of the components in the MenuPanel.	All components (buttons, text fields, etc.) should be non-null and have reasonable font sizes.

testUpdateScore_NewPlayer	HighScore	updateScore	Tests updating the score for a new player in the database.	The new player should have a score of 1 in the database.
testUpdateScore_ExistingPlayer	HighScore	updateScore	Tests updating the score for an existing player in the database.	The existing player's score should increment correctly.
testGetHighScores	HighScore	getHighScores	Verifies retrieval of high scores from the database.	Retrieved scores should include all players and be ordered correctly.
testGetHighScores_EmptyDatabase	HighScore	getHighScores	Verifies retrieval of high scores when the database is empty.	The retrieved string should be empty.
testHighScoresAreLimitedToTop10	HighScore	getHighScores	Ensures the high scores are limited to the top 10 entries.	The result should only contain the top 10 players.
testGameInitialization	GamePanel	Constructor	Tests the initial state of the game including player positions.	Players should start at their defined initial positions (e.g., 100, 300 and 700, 300).

testPlayerMovement	GamePanel	addKeyListener, move	Tests player movement based on simulated key presses.	Players' positions should update correctly according to their movement keys.
testPaintComponent	GamePanel	paintComponent	Verifies that the rendering method executes without errors.	The method should render the game without exceptions or rendering issues.

Event Handlers

Event	Trigger	Handler	Description
Game Initialization	Main method in Game	Game.startGame()	Initializes the game, displays the main menu, and starts a new game on button click.
Player Move	Arrow key press on GamePanel	Player.handleKeyPress()	Changes the player's direction based on the key pressed.
Game Over	Player crash or turn limit reached	Game.endGame()	Ends the game and displays the result (win/loss/draw).

High Score Update	After game ends	HighScore.updateScore()	Updates the player’s score in the database.
View High Scores	Button click on MenuPanel	HighScore.getHighScores()	Displays the top 10 high scores from the database.
Game Reset	After game over	Game.resetGame()	Resets the game to start a new round.
Exit Game	Button click in MenuPanel or GameOverPanel	Game.exit()	Exits the game and closes the application.