Tron Game

Written and

Implemented By

ISROILBEK JAMOLOV (DXFV5Y)

Course: Programming

Technology

Faculty of Informatics

Eötvös Loránd University

Date: 12 DEC 2023

Contents

Task description

The task is to create a two-player game inspired by the movie “Tron”. Each player controls a motor that leaves a light trail on the dipslay of the board. The game ends when a player collides with a trail or the game boundary. The game includes a leaderboard that displays the top 10 scores. Also, the game can be restarted using menu item. In addition, there are 10 levels where they are responsible for the modifications of board size, speed and seld-collision. In other words, the game screen will shrink by levels, and the speed of the bikes will increase, as well as after 6’s level the seld-collision be turned off.

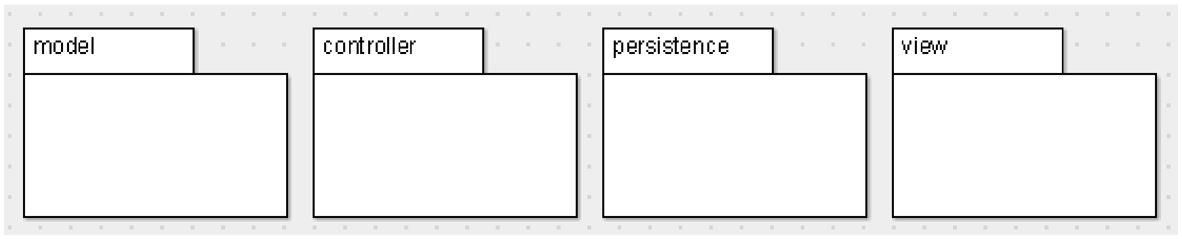
Task analysis

The game can be broken down into several components, each with its own functionality:

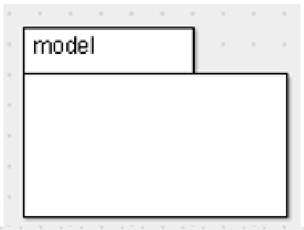
- Game: Represents the overall state of the game.
- Level: Represents a level in the game.
- Motorcycle: Represents a motorcycle in the game.
- Player: Represents a player in the game.
- Trail: Represents a trail left by a motorcycle.
- GameController: Controls the game.
- InputHandler: Handles user input.
- MainWindow: The main window of the game.
- Board: Represents the game board.
- Database: Handles database operationa.

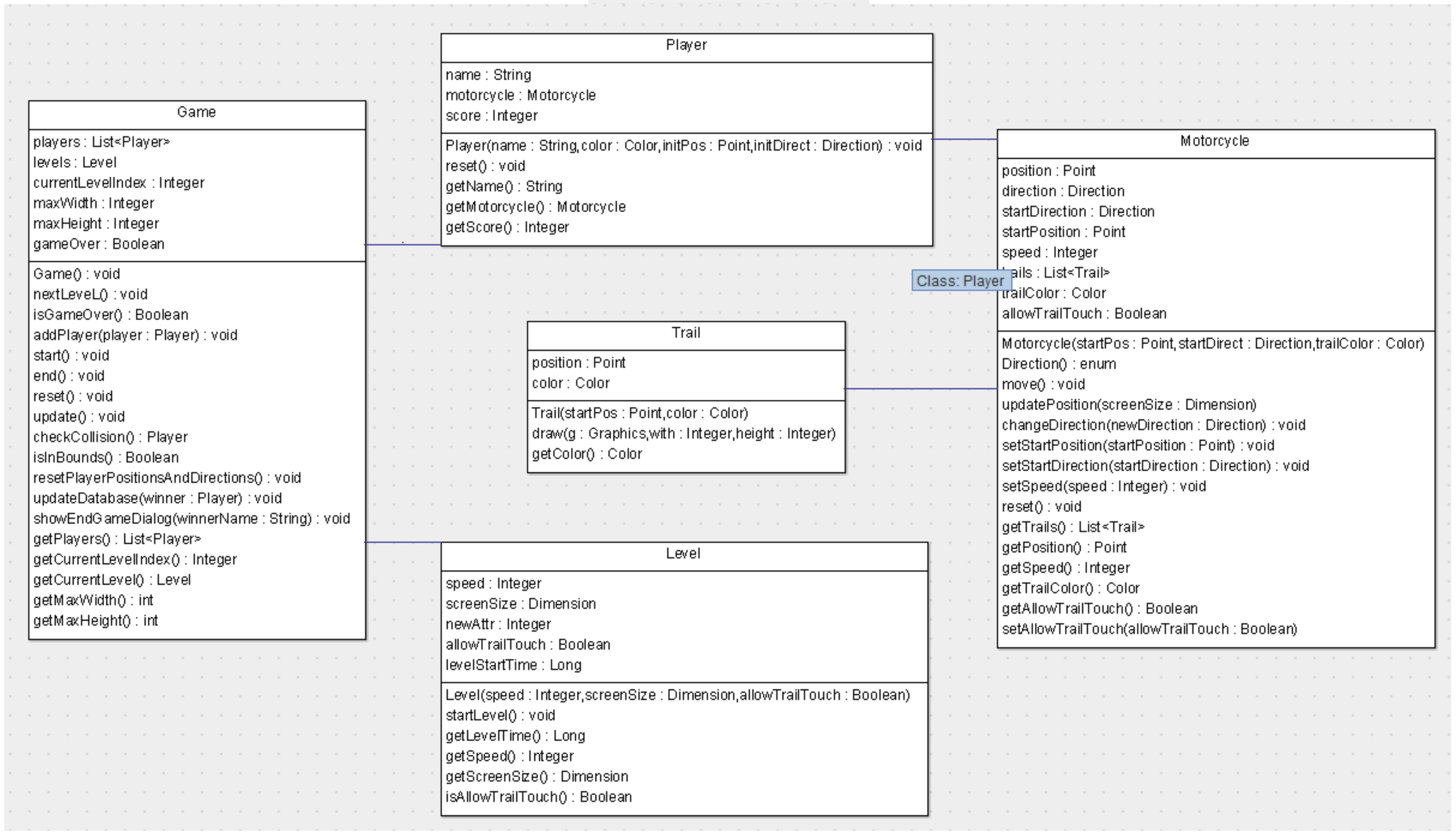
The structure of the program (UML diagram)

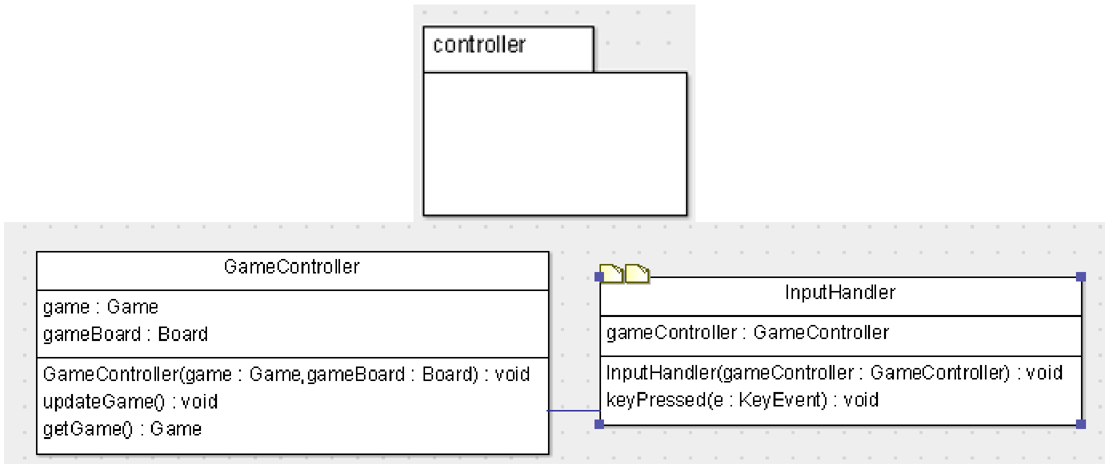
ALL PACKAGES

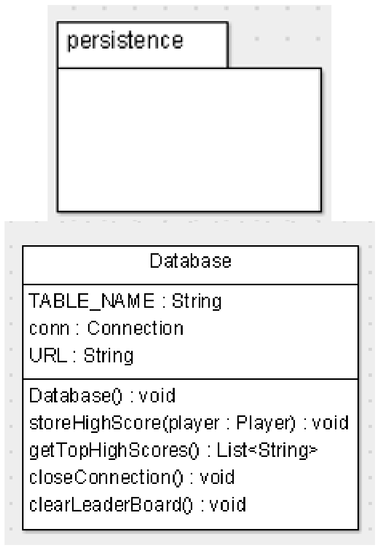


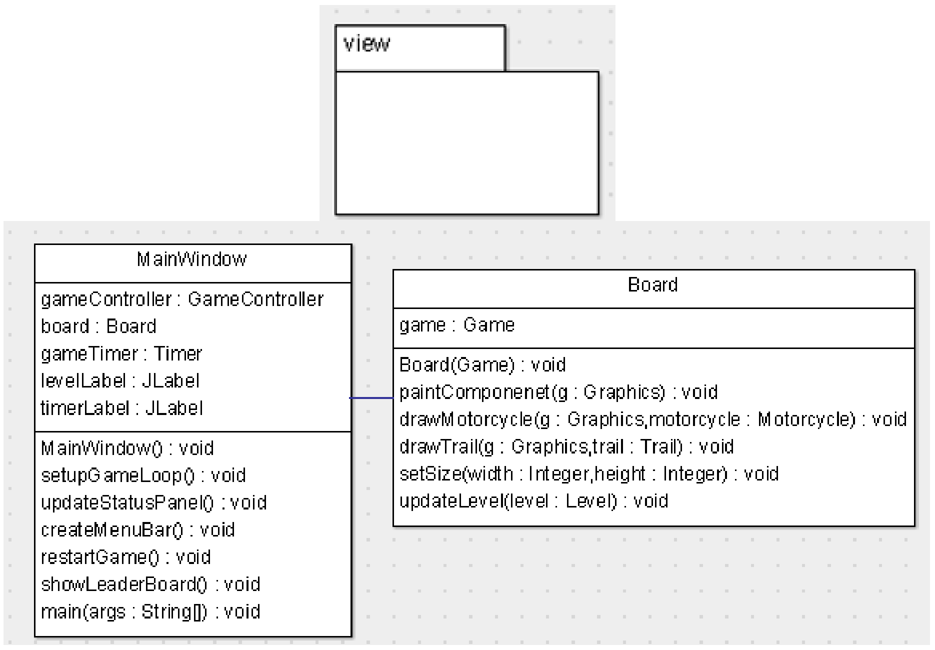
PACKAGES WITH OWN CLASSES



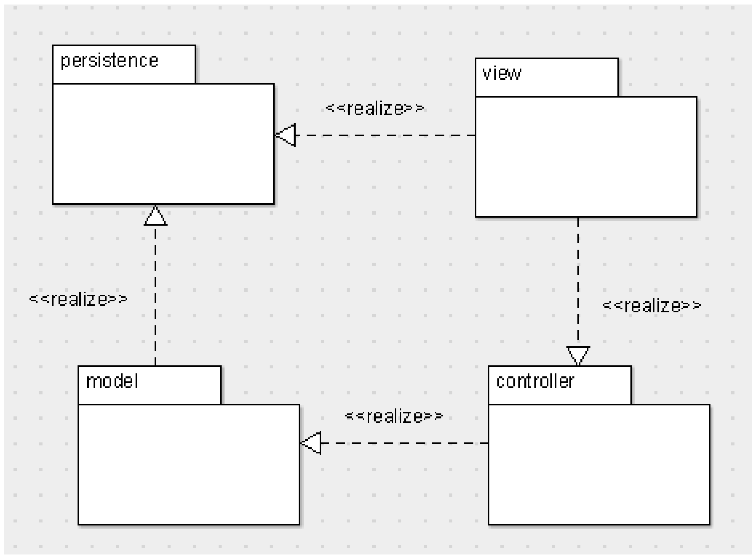

Player
name : String
motorcycle : Motorcycle
score : Integer
Player(name : String,color : Color,initPos : Point,initDirect : Direction) : void
reset() : void
getName() : String
getMotorcycle() : Motorcycle
getScore() : Integer
Game
players : List<Player>
levels : Level
currentLevelIndex : Integer
maxWidth : Integer
maxHeight : Integer
gameOver : Boolean
Game() : void
nextLeveL() : void
isGameOver() : Boolean
addPlayer(player : Player) : void
start() : void
end() : void
reset() : void
update() : void
checkCollision() : Player
isInBounds() : Boolean
resetPlayerPositionsAndDirections() : void
updateDatabase(winner : Player) : void
showEndGameDialog(winnerName : String) : void
getPlayers() : List<Player>
getCurrentLevelIndex() : Integer
getCurrentLevel() : Level
getMaxWidth() : int
getMaxHeight() : int
Motorcycle
position : Point
direction : Direction
startDirection : Direction
startPosition : Point
speed : Integer
trails : List<Trail>
trailColor : Color
allowTrailTouch : Boolean
Motorcycle(startPos : Point,startDirect : Direction,trailColor : Color)
Direction() : enum
move() : void
updatePosition(screenSize : Dimension)
changeDirection(newDirection : Direction) : void
setStartPosition(startPosition : Point) : void
setStartDirection(startDirection : Direction) : void
setSpeed(speed : Integer) : void
reset() : void
getTrails() : List<Trail>
getPosition() : Point
getSpeed() : Integer
getTrailColor() : Color
getAllowTrailTouch() : Boolean
setAllowTrailTouch(allowTrailTouch : Boolean)
Class: Player
Trail
position : Point
color : Color
Trail(startPos : Point,color : Color)
draw(g : Graphics,with : Integer,height : Integer)
getColor() : Color
Level
speed : Integer
screenSize : Dimension
newAttr : Integer
allowTrailTouch : Boolean
levelStartTime : Long
Level(speed : Integer,screenSize : Dimension,allowTrailTouch : Boolean)
startLevel() : void
getLevelTime() : Long
getSpeed() : Integer
getScreenSize() : Dimension
isAllowTrailTouch() : Boolean


controller
GameController
game : Game
gameBoard : Board
GameController(game : Game,gameBoard : Board) : void
updateGame() : void
getGame() : Game
InputHandler
gameController : GameController
InputHandler(gameController : GameController) : void
keyPressed(e : KeyEvent) : void


persistence
Database
TABLE_NAME : String
conn : Connection
URL : String
Database() : void
storeHighScore(player : Player) : void
getTopHighScores() : List<String>
closeConnection() : void
clearLeaderBoard() : void


view
MainWindow
gameController : GameController
board : Board
gameTimer : Timer
levelLabel : JLabel
timerLabel : JLabel
MainWindow() : void
setupGameLoop() : void
updateStatusPanel() : void
createMenuBar() : void
restartGame() : void
showLeaderBoard() : void
main(args : String[]) : void
Board
game : Game
Board(Game) : void
paintComponenet(g : Graphics) : void
drawMotorcycle(g : Graphics,motorcycle : Motorcycle) : void
drawTrail(g : Graphics,trail : Trail) : void
setSize(width : Integer,height : Integer) : void
updateLevel(level : Level) : void

RELATIONSHIP OF PACKAGES



Implementation of algorithms

1 Game Progression

The game progression is handled by the `nextLevel` method in the `Game` class. This method increases the current level index, adjusts the maximum width and height of the game area, and updates the speed of the motorcycles. If the current level index exceeds the number of levels, it resets to the first level.

2 Collision Detection

Collision detection is handled by the `checkCollision` method in the `Game` class. This method checks if a motorcycle’s position is out of bounds or if it collides with another player’s trail. If a collision is detected, it returns the player who lost.

3 Game Update

The game state is updated in the `update` method in the `Game` class. This method moves each motorcycle and checks for collisions. If a collision is detected, it ends the game.

4 User Input Handling

User input is handled by the `keyPressed` method in the `InputHandler` class. This method listens for key presses and changes the direction of the motorcycles accordingly.

5 Game Rendering

The game rendering is handled by the `paintComponent` method in the `Board` class. This method draws the motorcycles and trails on the game board.

6 High Score Management

High score management is handled by the `Database` class. This class provides methods to store high scores, retrieve the top high scores, and clear the leaderboard.

These algorithms work together to create a complete, functioning game. 😊

Connections between the events and their handlers

1 User Input Events

User input events are handled by the `InputHandler` class. When a user presses a key, the `keyPressed` method in the `InputHandler` class is triggered. This method changes the direction of the motorcycles based on the key pressed by the user.

2 Game Update Events

Game update events are handled by the `GameController` class. The `GameController` class sets up a timer that triggers an event every millisecond. When this event is triggered, the `updateGame` method in the `GameController` class is called. This method updates the game state and repaints the game board.

3 Game Over Events

Game over events are handled by the `Game` class. When a collision is detected, the `end` method in the `Game` class is called. This method sets the game over state to true and stops the game timer.

4 Game Restart Events

Game restart events are handled by the `MainWindow` class. When the user selects the “Restart Game” menu item, the `restartGame` method in the `MainWindow` class is called. This method resets the game state, starts the game, and restarts the game timer.

5 Leaderboard Display Events

Leaderboard display events are handled by the `MainWindow` class. When the user selects the “Show Leaderboard” menu item, the `showLeaderboard` method in the `MainWindow` class is called. This method retrieves the top high scores from the database and displays them in a dialog.

Testing

1. User doesn’t write name

Test Case: Attempt to start the game without entering a name for a player.

Expected Result: The game should not start and the user should be prompted again to enter a name.

2. User doesn’t choose color

Test Case: Attempt to start the game without choosing a color for a player.

Expected Result: The player will be given black color by default.

3. Incrementing existing name

Test Case: Win a game with a player name that already exists in the leaderboard.

Expected Result: The score for that player name in the leaderboard should be incremented by one, if that player exists, otherwise a new player will be added with score 1.

4. Leaderboard size

Test Case: Add more than 10 players to the leaderboard.

Expected Result: Only the top 10 players (based on score), in a descending order, should be displayed on the leaderboard.

5. Self-touch before 7th level

Test Case: Make a player’s motorcycle touch its own trail before the 7th level.

Expected Result: The player should lose the game.

6. Timer after each level

Test Case: Advance to the next level.

Expected Result: The timer should reset to 0 at the start of each level.

7. Level Progression

Test Case: Complete a level and observe the changes in the next level.

Expected Result: The speed of the motorcycles should increase and the size of the game area should decrease with each level, and after level 7 the self-touching will not bring to collision.

8. Collision with Other Player’s Trail

Test Case: Make a player’s motorcycle touch the other player’s trail.

Expected Result: The player should lose the game.

9. Collision with Game Boundary

Test Case: Make a player’s motorcycle touch the game boundary.

Expected Result: The player should lose the game.

10. Restart Game

Test Case: Restart the game after a player loses.

Expected Result: The game state should reset, the game should start again, and the game timer , players’ properties, except name and the color, should be reset as well.

11. Show Leaderboard

Test Case: Select the “Show Leaderboard” menu item.

Expected Result: The top high scores from the database should be displayed in a dialog. The leaderboard can be seen in the list of menu items.

12. Motorcycle Movement

Test Case: Press a key to change the direction of a motorcycle.

Expected Result: The motorcycle should move in the direction set by the user every second. Valid keyboard are “W,A,S,D” and directional keyboards, pressing the rest of the keyboards do not effect the flow of the game.