|  |  |  |
| --- | --- | --- |
| **Muhammad Al Farizi**  OCSWOM | **2nd assignment/8. Task** | 12th November 2024 |
| [ocswom@inf.elte.hu](mailto:ocswom@inf.elte.hu) |  |  |
| Practice Group 5 |  |  |

# Task

This two-player game is played on a board consists of n x n fields. Initially, two white and two black knights are placed at the corners of the board (knights of the same color are placed at the opposite corners).

Players step alternately, and knights can move only in an L shape, like in chess. The board is initially grey, but after each step, the visited places are colored with the color of the visitor knight (the previous color of the field doesn’t matter). A player wins, if there are 4 adjacent fields (horizontally, vertically, or diagonally) which colored to player’s color. The game ends, when there are no more grey fields.

Implement this game, and let the board size be selectable (4x4, 6x6, 8x8). The game should recognize if it is ended, and it has to show in a message box which player won. After this, a new game should be started automatically.

# Analysis

The program is a game called "Knight Tournament" that simulates knights moving around a board.

The **`Knight`** class is all about the knights—keeping track of where they are, their color, and making sure they move the right way (in an L-shape, like in chess).

The **`Player`** class manages each player’s knights, keeps track of which one is currently active, and remembers which squares they have captured.

The **`Field`** class is simple—it just represents a square on the board and keeps track of its color.

The **`Board`** class is the game's main brain, handling the layout, checking if moves are allowed, switching turns between players, and figuring out when someone wins.

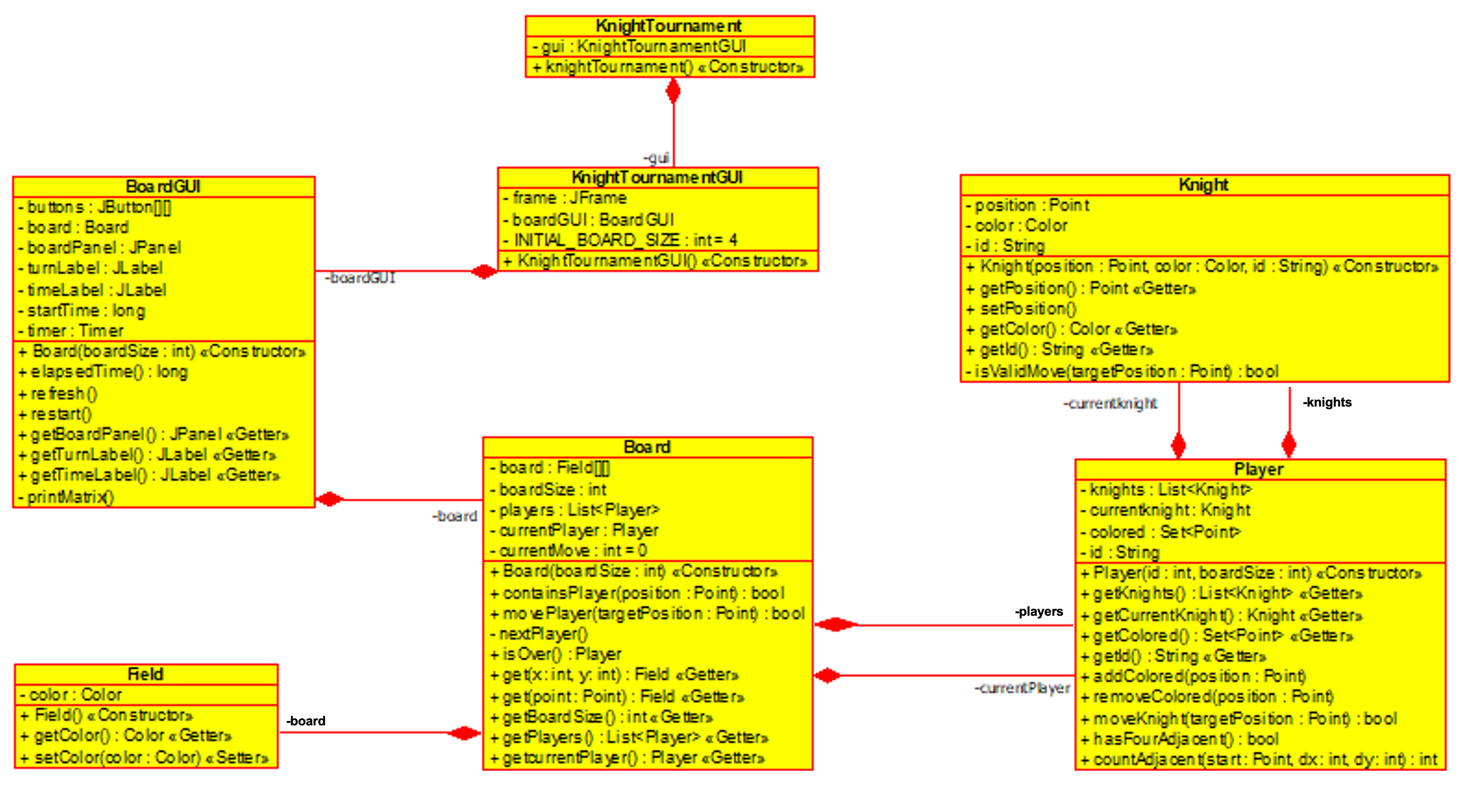
The **`BoardGUI`** class is the visual part of the game, letting players click on buttons to move their knights, and updating what’s shown on the screen.

The **`KnightTournamentGUI`** class sets up the game window and menu, allowing you to start new games with different board sizes.

Finally, the **`KnightTournament`** class is just the starting point of the game—it kicks off the whole program and shows the main game window. Each class does its own job, making the game work and look the way it does.

**-players**

# UML Class Diagram



# Method Description

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

|  |  |  |
| --- | --- | --- |
| **Knight** | +Knight.Knight(position: Point, color: Color, id: String) | Constructor for creating a Knight with a specific position, color, and ID. |
| +Knight.getPosition(): Point | Getter for the knight's current position. |
| +Knight.setPosition(position: Point): boolean | Attempts to set a new position for the knight, returning true if the move is valid. |
| +Knight.getColor(): Color | Getter for the knight's color. |
| +Knight.getId(): String | Getter for the knight's ID. |
| -Knight.isValidMove(targetPosition: Point): boolean | Checks if a target position is a valid L-shaped move for the knight. |

|  |  |  |
| --- | --- | --- |
| **Player** | +Player.Player(id: int, boardSize: int) | Constructor for creating a Player with a specific ID and initializing knights based on the board size. |
| +Player.getKnights(): List<Knight> | Getter for the list of knights belonging to the player. |
| +Player.getCurrentKnight(): Knight | Getter for the player's current knight. |
| +Player.getColored(): Set<Point> | Getter for the set of colored fields controlled by the player. |
| +Player.getId(): String | Getter for the player's ID. |
| +Player.addColored(position: Point) | Adds a field to the set of colored fields controlled by the player. |
| +Player.removeColored(position: Point) | Removes a field from the set of colored fields controlled by the player. |
| +Player.moveKnight(targetPosition: Point): boolean | Attempts to move the current knight to the target position and updates the colored fields if successful. |
| +Player.hasFourAdjacent(): boolean | Checks if the player has four adjacent colored fields in any direction. |
| -Player.countSequence(start: Point, dx: int, dy: int): int | Counts the number of consecutive colored fields starting from a given point in a specific direction. |

|  |  |  |
| --- | --- | --- |
| **Field** | +Field.Field() | Constructor for creating a default field with a light gray color. |
| +Field.getColor(): Color | Getter for the field's current color. |
| +Field.setColor(color: Color) | Setter for the field's color. |

|  |  |  |
| --- | --- | --- |
| **Board** | +Board.Board(boardSize: int) | Constructor for creating a game board of a specified size and initializing players. |
| +Board.containsPlayer(position: Point): boolean | Checks if a specific position on the board contains a player’s knight. |
| +Board.movePlayer(targetPosition: Point): boolean | Attempts to move the current player's knight to a target position and updates game state. |
| -Board.nextPlayer() | Switches the turn to the next player after two moves. |
| +Board.isOver(): Player | Checks if the game is over and returns the winning player if there is one. |
| +Board.get(x: int, y: int): Field | Getter for the field at a specific coordinate on the board. |
| +Board.get(point: Point): Field | Getter for the field at a specific point on the board. |
| +Board.getBoardSize(): int | Getter for the board size. |
| +Board.getPlayers(): List<Player> | Getter for the list of players in the game. |
| +Board.getCurrentPlayer(): Player | Getter for the current player. |

|  |  |  |
| --- | --- | --- |
| **BoardGUI** | +BoardGUI.BoardGUI(boardSize: int) | Constructor for setting up the game board's visual interface for a specified board size. |
| +BoardGUI.elapsedTime(): long | Returns the elapsed time since the game started. |
| +BoardGUI.refresh() | Attempts to move the current Refreshes the board's visual display, updating the knights' positions and colors. |
| +BoardGUI.restart() | Restarts the game, resetting the board and timer. |
| +BoardGUI.getBoardPanel(): JPanel | Getter for the game board's panel component. |
| +BoardGUI.getTurnLabel(): JLabel | Getter for the label indicating the current player's turn. |
| +BoardGUI.getTimeLabel(): JLabel | Getter for the label displaying the elapsed time. |
| -BoardGUI.printMatrix() | Prints a debug matrix representation of the game board's current state. |
| BoardGUI.ButtonListener | Class for the ButtonListener ActionEvent |
| +BoardGUI.ButtonListener.ButtonListener(x: int, y: int) | Constructor for initializing a button listener for a specific board cell. |
| +BoardGUI.ButtonListener.actionPerformed(e: ActionEvent) | Handles the button click event to move a knight if the target position is valid. |

|  |  |  |
| --- | --- | --- |
| **KnightTournament** | +KnightTournament.main(args: String[]) | Main entry point of the application. Initializes the game GUI. |

|  |  |  |
| --- | --- | --- |
| **KnightTournament**  **GUI** | +KnightTournamentGUI.KnightTournamentGUI() | Constructor for creating the main game GUI, initializing the window and game settings. |

# Screenshots

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

# Testing

Test Case 1: Knight Movement Test

AS A player in the Knight Tournament game

I WANT TO move my knight across the board

GIVEN a knight placed at an initial position on the board

WHEN I attempt to move the knight in an L-shape pattern

THEN the move should be successful if it's a valid move according to the rules.

Test Case 2: Invalid Knight Movement Test

AS A player in the Knight Tournament game

I WANT TO attempt an invalid move with my knight

GIVEN a knight at a specific position on the board

WHEN I try to move the knight in a pattern that is not an L-shape

THEN the move should be rejected and the knight's position should remain unchanged.

Test Case 3: Knight Capture Test

AS A player in the Knight Tournament game

I WANT TO move my knight to a position occupied by another player's knight

GIVEN two knights from different players on the board

WHEN I attempt to move one knight to the position of another

THEN the move should be blocked, and the game should prompt that the position is occupied.

Test Case 4: Turn Switching Test

AS A player in the Knight Tournament game

I WANT TO ensure turns are switched correctly between players

GIVEN two players with their knights on the board

WHEN one player completes their move

THEN the game should correctly switch to the other player's turn.

Test Case 5: Game Winning Condition Test

AS A player in the Knight Tournament game

I WANT TO win by creating four adjacent colored fields

GIVEN a player with colored fields on the board

WHEN four adjacent fields are aligned horizontally, vertically, or diagonally

THEN the game should recognize the win and display a winning message.

Test Case 6: Board Initialization Test

AS A user starting a new game

I WANT TO initialize the game board correctly

GIVEN a specific board size selected at the start

WHEN the game begins

THEN the board should be created with the correct size and knights should be positioned at their starting locations.

Test Case 7: Timer Functionality Test

AS A player in the Knight Tournament game

I WANT TO track the time taken for the game

GIVEN a timer displayed on the game GUI

WHEN the game starts

THEN the timer should begin counting and stop when the game ends.

Test Case 8: Restart Game Test

AS A player in the Knight Tournament game

I WANT TO restart the game if needed

GIVEN an ongoing game session

WHEN I choose to restart the game

THEN the game board should reset, and the timer should restart.

Test Case 9: Field Color Change Test

AS A player in the Knight Tournament game

I WANT TO see the fields change color when occupied

GIVEN a move to a new board position

WHEN a knight successfully moves to a target field

THEN the field's color should update to match the knight's color.

Test Case 10: GUI Button Interaction Test

AS A player using the game GUI

I WANT TO interact with the board by clicking buttons

GIVEN a button representing a board cell

WHEN I click a button to move my knight

THEN the knight should move if valid, and the game should update the display accordingly.