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

Task

Pebble is a two-player game, played on a board consists of n x n fields. Initially, n white and n black pebbles are placed on the board randomly. Each color belongs to only one player. The players take turns choosing one of its pebble, and then move it horizontally or vertically. The movement also affects the neighbouring pebbles in the direction (the pebble on the edge falls off). The objective of the game is to push out as much pebbles of the opponent from the board as we can, within a given number of turns (5n). A player wins, if he has more pebbles on the board at the end than his opponent. The game is draw, if they have the same number of pebbles on the board. Implement this game, and let the board size be selectable (3x3, 4x4, 6x6 → turns are 15, 20, 30). The game should recognize if it is ended, and it has to show the name of the winner in a message box (if the game is not ended with draw), and automatically begin a new game.

Analysis

1. Game Board

- **Board Representation**: The game is played on an n x n board (where n can be 3, 4, or 6). The board will hold pebbles of two colors (white and black).
- **Pebble Placement**: Initially, n white and n black pebbles are randomly placed on the board. Each player controls one color (white for Player 1, black for Player 2).
- **Board Operations**:
    - **Set Pebble**: Ability to place pebbles on the board at specified positions.
    - **Move Pebble**: A player selects a pebble and moves it horizontally or vertically. The movement can affect neighboring pebbles.
    - **Pebble Removal**: When a pebble moves to the edge, it "falls off" the board.

2. Game Logic

- **Turn-Based System**: The game alternates turns between the two players. Each player moves a single pebble per turn.
- **Movement Constraints**:
    - Pebbles are moved horizontally or vertically.
    - Each move can impact adjacent pebbles, and if a pebble is pushed off the edge, it is removed from the board.
- **Game Objective**:
    - Players aim to push as many of the opponent's pebbles off the board as possible.
    - A player wins if they have more pebbles on the board than the opponent after 5n turns.

- **Game End Condition**:
    - The game ends when all moves are completed (after 5n turns).
    - The game ends in a **draw** if both players have the same number of pebbles left on the board.
    - The game ends with a **win** if one player has more pebbles than the other.

3. Player Interactions

- **Turn Sequence**:
    - Player 1 starts first with white pebbles, and Player 2 controls black pebbles.
    - Each player selects a pebble to move, chooses the direction of movement, and the game updates the board accordingly.
- **Turn Limit**: The game has a maximum number of turns, which varies based on the size of the board:
    - For a 3x3 board: 15 turns.
    - For a 4x4 board: 20 turns.
    - For a 6x6 board: 30 turns.

4. UI and Game Flow

- **Graphical Interface**:
    - The game will have a graphical interface to display the board, pebbles, and a button for each cell.
    - Players will interact with the board by clicking on the pebbles.
- **Move Choices**: Players will choose the direction (up, down, left, right) for the pebble they selected to move.
- **Game Outcome**:
    - A message box will display the winner's name or announce a draw when the game ends.
    - After the game ends, a new game will automatically begin.

5. Game Rules and Validations

- **Player's Move Validation**:
    - A player can only move their own pebbles. This needs to be validated before processing the move.
    - The game should reject invalid moves (e.g., trying to move an opponent's pebble).
- **Move Direction**: The player chooses a direction for the pebble movement, which must be within the bounds of the board.
- **Out-of-Bounds Move**: If a move causes a pebble to go off the edge of the board, it is removed.
- **Turn Limitation**: The game must track the number of turns and enforce the turn limit.
- **Game End Logic**:
    - After each turn, the game should check if the end condition is met (either after the maximum number of turns or when all pebbles of one player are removed).
    - Display the result and reset the game..

6. Classes and Key Components

1. GameBoard Class:

- Stores the state of the board (a 2D array).
- Handles pebble placement, movement, and removal.
- Manages the count of white and black pebbles.

2. GameLogic Class:

- Controls the flow of the game, tracking turns and managing the rules (who's turn it is, checking if the game is over, etc.).
- Decides when the game ends and determines the winner.

3. Player Class:

- Manages the current player, switching between players after each turn.
- Tracks the color of the current player’s pebbles.

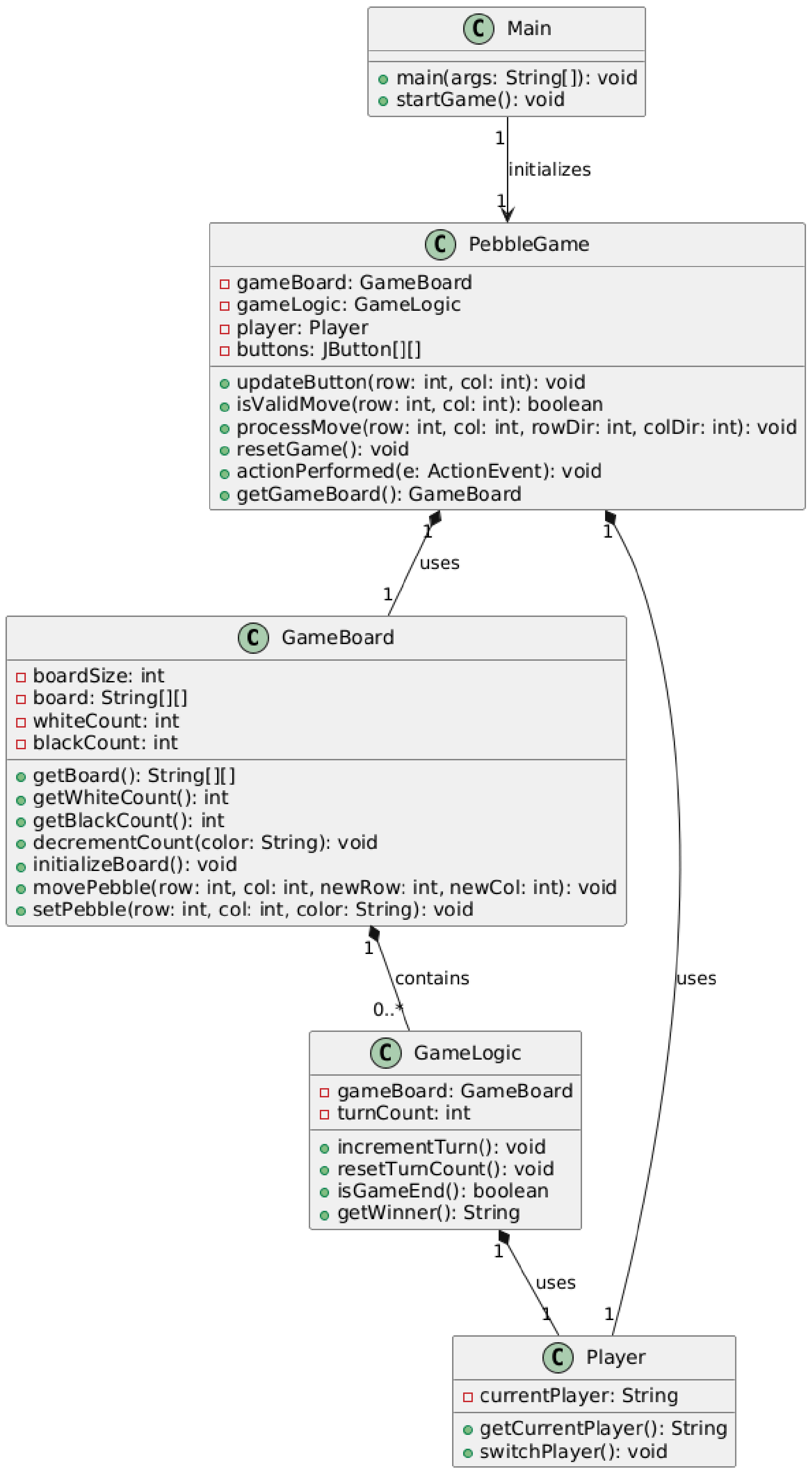
4. PebbleGame (Main Game) Class:

- Handles the UI and interactions, connecting the board, logic, and players.
- Displays the board and allows the players to make moves.
- Handles the game outcome and restarts the game after each round.

5. Game Flow and UI:

- Display the game board as buttons for players to click on.
- Implement message dialogs to show the result (win or draw) after the game ends.

UML Diagram



Main
main(args: String[]): void
startGame(): void
1
initializes
1
PebbleGame
gameBoard: GameBoard
gameLogic: GameLogic
player: Player
buttons: JButton[][]
updateButton(row: int, col: int): void
isValidMove(row: int, col: int): boolean
processMove(row: int, col: int, rowDir: int, colDir: int): void
resetGame(): void
actionPerformed(e: ActionEvent): void
getGameBoard(): GameBoard
1
uses
1
1
uses
GameBoard
boardSize: int
board: String[][]
whiteCount: int
blackCount: int
getBoard(): String[][]
getWhiteCount(): int
getBlackCount(): int
decrementCount(color: String): void
initializeBoard(): void
movePebble(row: int, col: int, newRow: int, newCol: int): void
setPebble(row: int, col: int, color: String): void
1
contains
0..*
GameLogic
gameBoard: GameBoard
turnCount: int
incrementTurn(): void
resetTurnCount(): void
isGameEnd(): boolean
getWinner(): String
1
uses
1
1
Player
currentPlayer: String
getCurrentPlayer(): String
switchPlayer(): void

Methods

Class	Methods	Description
GameBoard	`initializeBoard()`	Initializes the board with the starting configuration (randomly places white and black pebbles).
	`getBoard()`	Returns the current board (2D array of pebbles).
	`getWhiteCount()`	Returns the count of white pebbles remaining on the board.
	`getBlackCount()`	Returns the count of black pebbles remaining on the board.
	`decrementCount(String color)`	Decreases the count of a specific color pebble (white or black).
	`movePebble(int row, int col, int newRow, int newCol)`	Moves a pebble from a given position to a new position (For Testing)
	`setPebble(int row, int col, String color)`	Sets a pebble at the given position with the specified color. (For Testing)
	`removePebble(int row, int col)`	Removes a pebble from a specific position on the board. (For Testing)
GameLogic	`incrementTurn()`	Increments the turn counter.
	`resetTurnCount()`	Resets the turn counter back to 0.
	`getTurnCount()`	Returns the current turn count. (For Testing)
	`isGameEnd()`	Checks if the game has ended (either due to turn limit or no pebbles of one color remaining).

	`getWinner()`	Returns the winner of the game ("White", "Black", or "Draw").
Player	`getCurrentPlayer()`	Returns the color of the current player (either "White" or "Black").
	`switchPlayer()`	Switches the current player from one color to the other.
PebbleGame	`actionPerformed(ActionEvent e)`	Handles player actions, processing moves when a player clicks a pebble.
	`updateButton(int row, int col)`	Updates the display of a specific button on the board (reflects the state of the corresponding cell in the game).
	`resetGame()`	Resets the game state and UI (including reinitializing the board and player counts).
	`isValidMove(int row, int col)`	Checks if a pebble at the specified position belongs to the current player and is eligible to be moved.
	`processMove(int row, int col, int rowDir, int colDir)`	Processes the movement of a pebble recursively in the specified direction.
	`getGameBoard()`	Returns the `GameBoard` instance for the game (used for testing and validation).

Testing

Test Name	Description	User Story	Test Type
Test the Game Logic (Pebble Movement)	Test whether pebbles move in the correct	As a developer, I want to ensure that the pebble	**White-box (Unit)**

	direction based on input.	moves correctly.	
Test the `GameBoard` Initialization	Test if the game board is initialized with the correct number of pebbles.	As a developer, I want to ensure the board initializes correctly.	**White-box (Unit)**
Test Turn Counting	Ensure turn counter increments correctly after each move.	As a developer, I want to track the number of turns.	**White-box (Unit)**
Test Decrement of Pebble Count	Ensure the count of pebbles decreases after a move.	As a developer, I want to ensure the game tracks the pebble count.	**White-box (Unit)**
Test Movement Validation	Ensure the validity of the move based on the current player's pebble.	As a developer, I want to ensure only valid moves are allowed.	**White-box (Unit)**
Test Boundary Conditions	Test moving pebbles to boundary positions.	As a developer, I want to ensure boundary conditions are handled.	**White-box (Unit)**
Test Turn Limit	Ensure that the turn limit is correctly set for each board	As a developer, I want to be able to make sure that the number of turns per each board is correct	**White-Box (Unit)**
Test Game Initialization (UI)	Ensure the game board displays correctly with pebbles.	As a user, I want the board initialized properly with pebbles.	**Black-box (Functional)**
Test User's Move Interaction	Test user's ability to move pebbles by clicking and selecting direction.	As a user, I want to move my pebbles interactively.	**Black-box (Functional)**
Test Game End	Ensure the game ends correctly, displaying winner or draw.	As a user, I want to be informed of the game result.	**Black-box (System)**
Test Invalid Move Behavior	Ensure error handling for invalid	As a user, I want to be notified if I	**Black-box (System)**

	moves.	make an invalid move.	
Test User’s Interaction with Direction Choice	Ensure proper interaction with the direction-choice prompt.	As a user, I want to choose the direction of my move.	**Black-box (Functional)**

Event Handlers

Events	**Trigger**	**Method (Handler)**	**Description**
Game Initialization	main method in PebbleGameMain	PebbleGameMain.main	Prompts user to select board size and initializes `PebbleGame` with chosen size.
Board Cell Click	Button click on a cell in grid	PebbleGame.actionPerformed(ActionEvent e)	Checks if cell is a valid move for the current player, then prompts for move direction.
Direction Selection	Option dialog within actionPerformed	PebbleGame.actionPerformed (option handling)	Sets row/column direction values based on user selection and calls `movePebble`to execute the move.
Move Execution	movePebblemethod call	PebbleGame.movePebble(int row, int col, int rowDir, int colDir)	Recursively moves pebbles in the selected direction, updating the board and decrementing counts if pebbles fall off.

Update Button	Within movePebble	PebbleGame.updateButton(int row, int col)	Updates the visual state of buttons to reflect board changes, with background color and text for pebbles.
Switch Player	End of actionPerformed	PebbleGame.switchPlayer()	Switches `currentPlayer` to the next player, toggling between "White" and "Black".
Turn & Game End Check	End of movePebble and each turn	PebbleGame.checkGameEnd()	Checks if all pebbles of one color are removed or turn limit reached, displays winner/draw, then resets if ended.
Game Reset	When the game ends (isGameEnd)	PebbleGame.resetGame()	Resets turn count, reinitializes the board, and updates buttons for a new game round.