ID: EE8LMT

Assignment 2 P07

Name: Abdul Basit

Documentation for BreakThrough Game

Exercise Description:

Break-through is a two-player game, played on a board consists of n x n fields. Each player has 2n dolls in two rows, placed on at the player's side initially (similarly to the chess game, but now every dolls of a player look like the same). A player can move his doll one step forward or one step diagonally forward (can't step backward). A player can beat a doll of his opponent by stepping diagonally forward onto it. A player wins when his doll reaches the opposite edge of the board. Implement this game, and let the board size be selectable (6x6, 8x8, 10x10). 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:

Break-through is a strategic two-player game played on an n x n grid, mirroring chess in the initial setup but simplified by having each player's pieces look the same. The game's objective is for a player to navigate their dolls across the board to the opposite side, capturing their opponent's dolls by moving diagonally. A player wins when one of their dolls reaches the opposite edge of the board. The challenge in implementing Break-through lies in accurately simulating the board dynamics, doll movements, and interaction rules, ensuring the game works according to the rules set.

Key Components of the Game

1. Player

- **Description**: Represents a participant in the game, controlling dolls on the board. Each player starts with their dolls arranged in two rows at their end of the board.

2. Doll

- **Description**: Acts as the game piece for each player, capable of moving forward or diagonally to capture opposing dolls.
- **Responsibilities**:
- **Moving**: Can move to an adjacent forward or diagonal position unless blocked or at the edge of the board.
- **Capturing:** Can capture an opponent's doll by moving diagonally to the position occupied by an opposing doll.

3. GameBoard

- **Description**: A grid-based playing field where all game actions take place, sized based on player preference (6x6, 8x8, 10x10).
- **Responsibilities**:
- **Hosting Dolls**: Keeps track of all positions on the board and which dolls occupy them.
- **Checking Moves:** Ensures all moves and captures are valid according to the game rules.

- **Win Condition Monitoring:** Checks if any doll has reached the opposite edge of the board to declare a winner.

4. BreakThroughController

- **Description**: Manages the interaction between the model (Game) and the view (BreakThroughView), enforcing the rules and progression of the game.
- **Responsibilities**:
- **Game Flow Management**: Changes owner when purchased by a player.
- **Event Handling:** Processes player interactions such as doll selections and movement commands.
- **Win Condition Check:** Determines if the game has ended and triggers the winning sequence.

5. BreakThroughView

- **Description**: Provides the graphical user interface where players interact with the game, displaying the board and accepting user inputs.
- **Responsibilities**:
    - **Displaying the Board**: Shows the game board and the positions of all dolls. ○ **User Interaction:** Captures user inputs like doll selections and movements. ○ **Game Updates:** Refreshes the display based on changes in the game state.

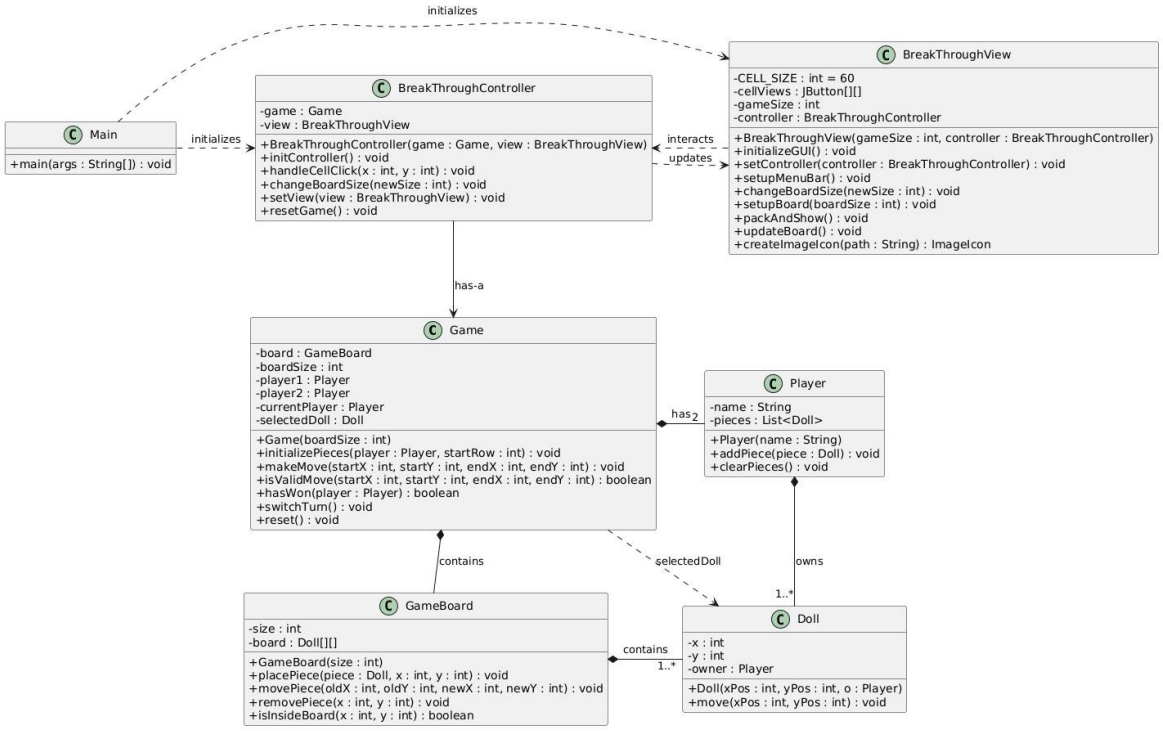
Solution plan: We will implement an abstract class for both Player and Field, setting a foundation for specific behaviors in their subclasses. The Player class will feature an abstract method decidePay, which will be customized in each subclass based on the player's strategy. Similarly, the Field class will have an abstract method enter, which will dictate the interaction each type of field has with a player. Using runtime polymorphism, java will dynamically determine the appropriate method (decidePay or enter) to execute based on the object's class (Player or Field) at runtime. We will employ the Visitor design pattern to facilitate interactions between players and fields. This pattern will enable players to "visit" fields and execute the correct action depending on both the player's characteristics and the type of field they land on.

Class Diagram:

ID: EE8LMT

Assignment 2 P07

Name: Abdul Basit



Description of Important Methods:

1. **handleCellClick(x, y) Method in BreakThroughController** • **Purpose:** Processes the action when a player clicks on a cell, determining if it is a move or selection based on the game state.

- **Functionality:** Checks if a selected doll can legally move to the clicked position or if a new doll is being selected.

Functions Called:

game.getBoard().getPiece(x, y)

- **Purpose:** Retrieves the Doll at the specified position (x, y) on the board.
- **Functionality:** This function helps determine if the clicked cell contains a doll. If it does, the function returns the Doll object at that position; otherwise, it returns null.

game.getSelectedDoll()

- **Purpose:** Checks if a doll is currently selected by the player.
- **Functionality:** Returns the currently selected Doll if one is selected, or null if no doll is selected. This is used to determine if the click should be interpreted as a selection or a move attempt.

game.setSelectedDoll(Doll doll)

- **Purpose:** Sets the currently selected doll for the player.

- **Functionality:** Assigns the provided Doll to be the selected doll for the current player. This is called to select a new doll or to clear the selection by passing null.

game.makeMove(int startX, int startY, int endX, int endY)

- **Purpose:** Executes a move for the selected doll from its current position to the target position (x, y).
- **Functionality:** This function validates the move, updates the board, and checks if the move results in a capture. It throws an IllegalStateException if the move is invalid. If the move is successful, the doll is moved to the new position.

game.hasWon(Player player)

- **Purpose:** Checks if the specified player has won the game.
- **Functionality:** This function inspects the board to see if any of the player's dolls have reached the opponent's starting row, which is the winning condition. If a win condition is met, it returns true; otherwise, it returns false.

view.updateBoard()

- **Purpose:** Refreshes the board view to reflect the current state of the game.
- **Functionality:** Updates the visual representation of the board to show the latest positions of the dolls, highlighting the selected doll or indicating valid moves if applicable.

2. makeMove(startX, startY, endX, endY) Method of Game Class

- **Purpose:** Executes a move or capture, updating the board state.
- **Functionality:** Moves a doll from one position to another, capturing an opponent's doll if applicable, and checks for a win condition after each move.

3. updateBoard() Method of BreakThroughView

- **Purpose:** Refreshes the game board display to reflect the current state of the game.
- **Functionality:** Updates the positions of all dolls on the board, highlighting possible moves and indicating the current player's turn.

.

4. resetGame() Method of BreakThroughController

- **Purpose:** Resets the game to its initial state for a new game after a player wins.
- **Functionality:** Reinitializes the board, repositions all dolls, and prepares the game for a new round.

ID: EE8LMT

Assignment 2 P07

Name: Abdul Basit

5. isValidMove(startX, startY, endX, endY) Method of Game Class

- **Purpose:** Determines if the move a player wants to make is legal according to the game rules. • **Functionality:** Checks if the move is forward or diagonally forward without stepping back or out of bounds, also ensuring no friendly doll is at the destination. If moving diagonally, it verifies that the target cell is occupied by an opponent's doll for a capture.

6. setupMenuBar() Method in BreakThroughView

- **Purpose**:
  - The setupMenuBar method initializes and sets up the menu bar for the game's user interface. It provides options for the user to select the board size, reset the game, and exit the application.
- **Functionality**:
  - The method creates a menu bar with three main elements:
    1. Board Size Menu: Allows the user to choose the size of the board (6x6, 8x8, or 10x10).
    2. Reset Item: Lets the user reset the game to its initial state.
    3. Exit Item: Exits the application.
- **Functions Called:**

1. controller.changeBoardSize(int newSize)

- **Purpose**: Changes the size of the game board and reinitializes the game with the specified board size.
- **Functionality**:
  - This function resets the game model (Game) to a new board size. It recreates the board and redistributes the pieces for both players on the new board. The view is then updated to reflect the change.
  - Called when a user selects a board size option (e.g., "6x6", "8x8", "10x10") from the Board Size menu.

2. controller.resetGame()

- **Purpose**: Resets the game to its initial state with the current board size.
- **Functionality**:
  - This function resets the game model, reinitializes the board, and redistributes pieces for both players. It effectively restarts the game without changing the board size.
  - Called when the Reset menu item is selected.

3. System.exit()

- **Purpose**: Exits the application.
- **Functionality**:
  - This function terminates the Java application. System.exit(0) indicates a normal exit without any errors.
  - Called when the Exit menu item is selected.

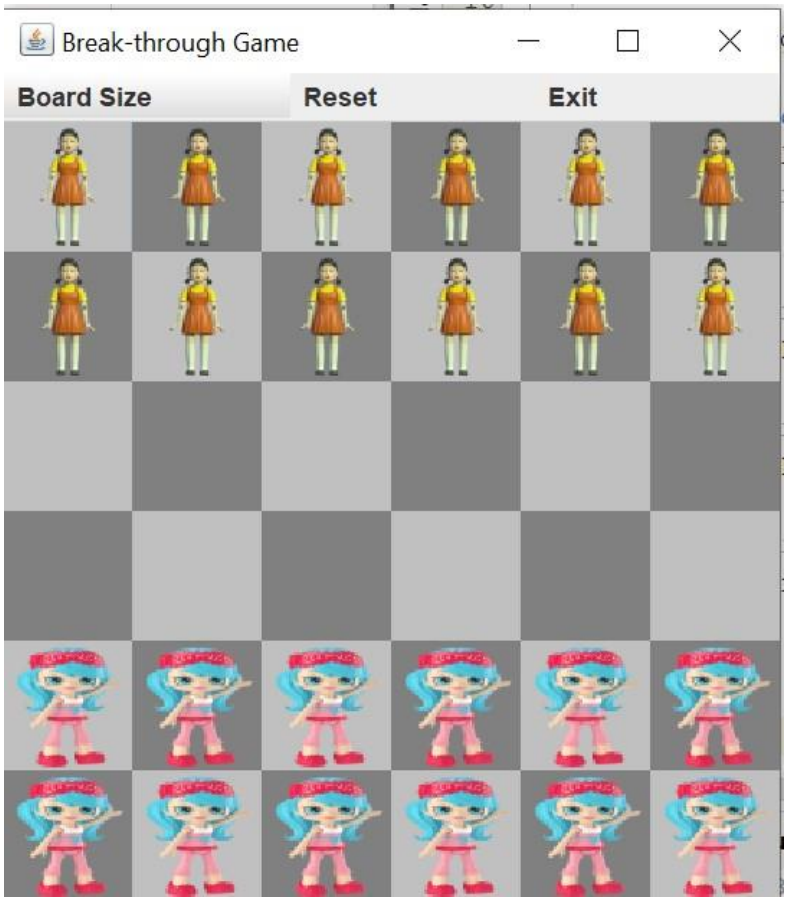
Input (6X6 Board):

Output:

ID: EE8LMT

Assignment 2 P07

Name: Abdul Basit



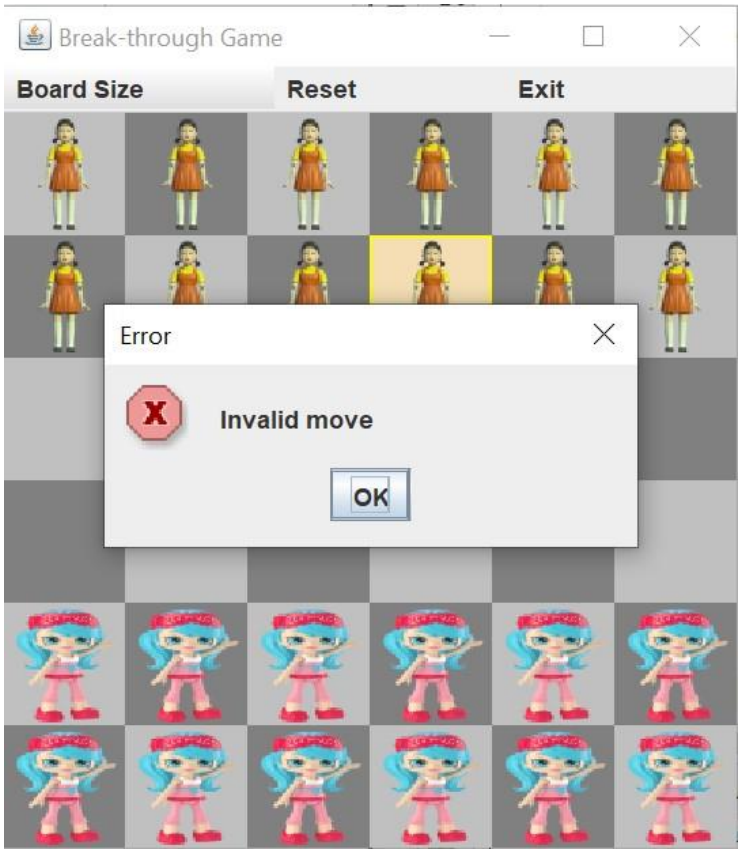
Input (illegal Move Exception):

Output:

ID: EE8LMT

Assignment 2 P07

Name: Abdul Basit



Input (Winning Condition):

Output:

ID: EE8LMT

Assignment 2 P07

Name: Abdul Basit

- **Black Box Testing Description:**

In the black box testing approach, we test the **external behavior** of the program without testing the inner workings we will focus on the input and output.

1. **TestWinCondition**:

   AS A game developer,
   I WANT TO ensure that the game correctly identifies and announces a win.
   GIVEN a player's doll is one move away from reaching the opposite side of the board.
   WHEN the player moves their doll into the winning position.
   THEN the game should declare that player as the winner and trigger the end-game sequence.
   Input:
   Player moves doll from (N-2, X) to (N-1, X) on an N x N board.

2. **TestCaptureMovement**:

   AS A game developer,
   I WANT TO verify that dolls can capture opponent dolls correctly.
   GIVEN one player's doll is diagonally adjacent to an opponent's doll.
   WHEN the player moves their doll to the position occupied by the opponent's doll.
   THEN the opponent's doll should be removed from the board, and the capturing doll moves to that position.
   Input:
   Player 1 moves the doll from (X, Y) to (X+1, Y+1) capturing Player 2's doll at (X+1, Y+1).

3. **TestInvalidMove**:

   AS A game developer,
   I WANT TO check that invalid moves are correctly prevented.
   GIVEN a player attempts a move that is backward or onto a space occupied by another doll of the same player.
   WHEN the move is attempted.
   THEN the game should not allow the move and should show an error message.
   Input:
   Player 1 tries to move a doll from (X, Y) to (X-1, Y) or to another doll's position at (X+1, Y)..

4. **TestClickOutsideBoard:**
   AS A game developer,
   I WANT TO ensure that clicking outside the playable board area does not affect the game.
   GIVEN clicks are made outside the game grid boundaries.
   WHEN these clicks occur.

THEN there should be no change in the game state and an exception or error message indicating an illegal move should be displayed.
Input:
Click at position (N+1, N+1) on an N x N board.

- **White Box Testing Description:**

In white box testing, the **internal structure** of the program is tested. Test cases are described below.

Test Cases in White Box Testing:

1. **IsValidMoveFunctionalityTest**:

   AS A game developer,
   I WANT TO ensure the isValidMove method accurately validates all move rules.
   GIVEN various board scenarios that test all branches of the method including valid moves, invalid moves, captures, and boundary conditions.
   WHEN a move is attempted under these conditions.
   THEN the method should return true for valid moves and false for invalid ones.
   Input:
   Test moves like (X, Y) to (X+1, Y), (X, Y) to (X, Y+1), etc., including boundary and capture scenarios.

2. **GameInitializationTest**:

   AS A game developer,
   I WANT TO verify that the game initializes correctly.
   GIVEN the game is started with two players and a specified board size.
   WHEN the game setup is executed.
   THEN all dolls should be correctly positioned on the board according to the game rules, and all internal states are properly initialized.
   Input:
   Initialize game with an 8x8 board.

3. **PlayerTurnSwitchTest:**
   AS A game developer,
   I WANT TO confirm that turns are correctly switched between players.
   GIVEN each player makes a valid move.
   WHEN these moves are made.
   THEN control should switch between players appropriately, reflecting the updated game state after each move.
   Input:
   Player 1 moves a doll from (0, 0) to (1, 0), followed by Player 2 moving a doll from (7, 0) to (6, 0).