Chyong Artur
XOG5D9
xog5d9@inf.elte.hu
Group 5

2. assignment/7. Task

10th Novemeber 2023

Task

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.

Objectives

Game Implementation:

- Develop a Break-through game for two players on an n x n board with 2n dolls per player.
- Enable doll movements one step forward or diagonally forward, mimicking chess, with the ability to beat opponent dolls.

Board Size Selection:

- Allow users to choose board sizes (6x6, 8x8, 10x10) for each game.

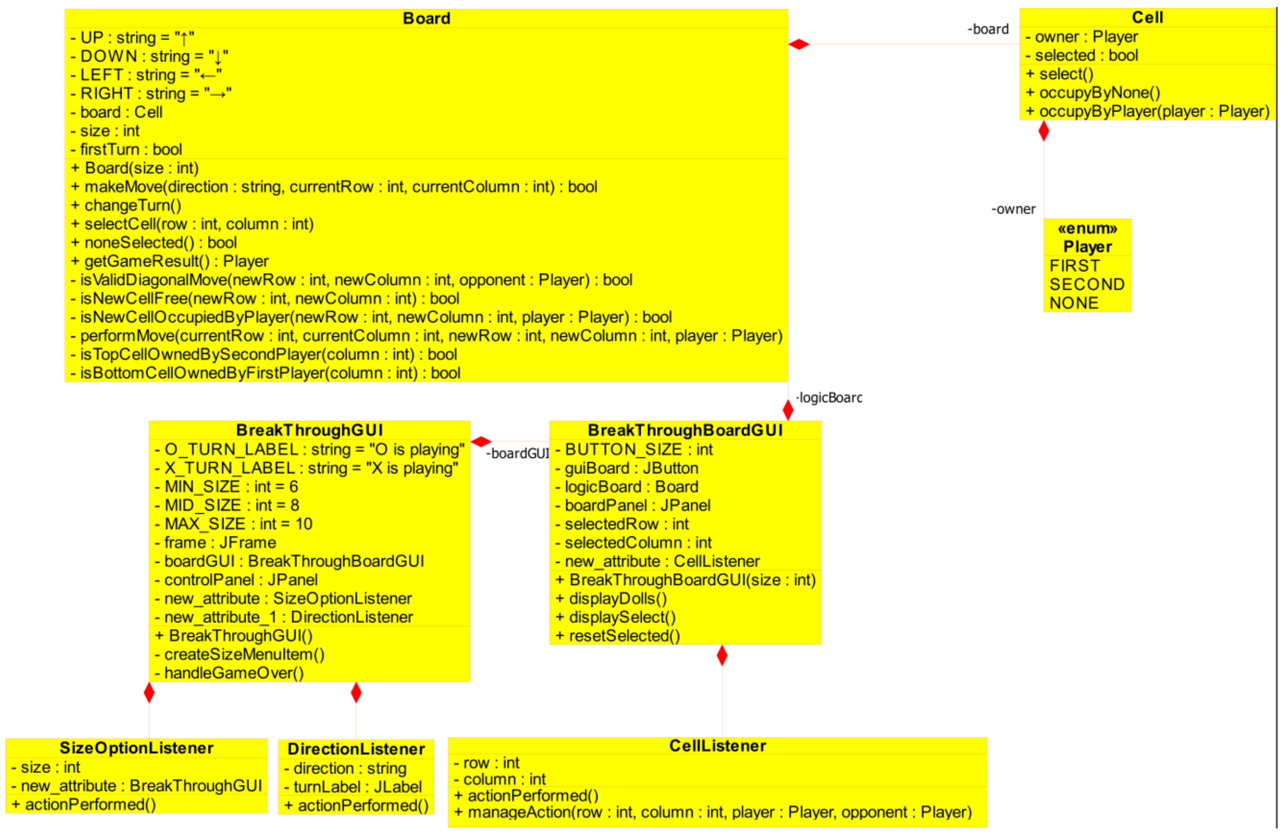
Game End Recognition:

- Implement a mechanism to detect game conclusion when a player's doll reaches the opposite board edge, displaying the winner.

Automated Game Restart:

- Ensure automatic initiation of a new game with the chosen board size after the current game concludes.

Class diagram


Board
- UP : string = "↑"
- DOWN : string = "↓"
- LEFT : string = "←"
- RIGHT : string = "→"
- board : Cell
- size : int
- firstTurn : bool
+ Board(size : int)
+ makeMove(direction : string, currentRow : int, currentColumn : int) : bool
+ changeTurn()
+ selectCell(row : int, column : int)
+ noneSelected() : bool
+ getGameResult() : Player
- isValidDiagonalMove(newRow : int, newColumn : int, opponent : Player) : bool
- isNewCellFree(newRow : int, newColumn : int) : bool
- isNewCellOccupiedByPlayer(newRow : int, newColumn : int, player : Player) : bool
- performMove(currentRow : int, currentColumn : int, newRow : int, newColumn : int, player : Player)
- isTopCellOwnedBySecondPlayer(column : int) : bool
- isBottomCellOwnedByFirstPlayer(column : int) : bool
-board
Cell
- owner : Player
- selected : bool
+ select()
+ occupyByNone()
+ occupyByPlayer(player : Player)
-owner
«enum»
Player
FIRST
SECOND
NONE
-logicBoarc
BreakThroughGUI
- O_TURN_LABEL : string = "O is playing"
- X_TURN_LABEL : string = "X is playing"
- MIN_SIZE : int = 6
- MID_SIZE : int = 8
- MAX_SIZE : int = 10
- frame : JFrame
- boardGUI : BreakThroughBoardGUI
- controlPanel : JPanel
- new_attribute : SizeOptionListener
- new_attribute_1 : DirectionListener
+ BreakThroughGUI()
- createSizeMenuItem()
- handleGameOver()
-boardGUI
BreakThroughBoardGUI
- BUTTON_SIZE : int
- guiBoard : JButton
- logicBoard : Board
- boardPanel : JPanel
- selectedRow : int
- selectedColumn : int
- new_attribute : CellListener
+ BreakThroughBoardGUI(size : int)
+ displayDolls()
+ displaySelect()
+ resetSelected()
SizeOptionListener
- size : int
- new_attribute : BreakThroughGUI
+ actionPerformed()
DirectionListener
- direction : string
- turnLabel : JLabel
+ actionPerformed()
CellListener
- row : int
- column : int
+ actionPerformed()
+ manageAction(row : int, column : int, player : Player, opponent : Player)

Methods

BreakThroughGUI class:

The class possesses the *constructor* which initializes the game.

createSizeMenuItem: accepts string label and int size and returns he menu item with the given label and size.

handleGameOver: accepts string with the information who is the winner and turnlabel to change the label.

SizeOptionListener class:

The class implements ActionListener.

Its constructor accepts int size.

Since it implements ActionListener, it overrides actionPerformed method which changes the size of the current board to the chosen size.

DirectionListener class:

The class implements ActionListener.

Its constructor accepts string direction and JLabel turnLabel.

Since it implements ActionListener, it overrides actionPerformed method which listens to the button with the direction and calls for the method to perform the move and check the state of the game.

BreakThroughBoardGUI class:

It possesses the standard set of getters.

Its constructor accepts int size and initializes the logic and GUI boards.

displayDolls: it goes through the logic board and sets up the dolls on the GUI board where it is needed.

displaySelect: accepts row and column and selects/deselects the cell at this row and column by changing the background.

resetSelected: resets the selected row and column to -1.

CellListener class:

The class implements ActionListener.

Its constructor accepts int row and int column.

Since it implements ActionListener, it overrides actionPerformed method which checks if the cell at the specified row and column is selectable and in case it is it selects it.

Board class:

It possesses the standard set of getters.

Its constructor accepts int size and initializes the logic board, so that the first two rows belong to the first player, the last two rows belong to the second player, and the rows between them belong to NONE player.

makeMove: checks first if the move in the given direction is valid, and if it is, it performs the move. Returns true if a move was successful and false, otherwise.

changeTurn: changes the field firstTurn to the opposite Boolean value.

selectCell: calls for the select method of the given cell.

noneSelected: checks if there is already a selected cell on the board. If there is, it returns true, otherwise, false.

getGameResult: checks the state of the game. If the second player reaches the first row, the second player wins, if the first player reaches the last row, the first player wins, otherwise, none wins.

isValidDiagonalMove: checks if a move in the given direction is valid. Returns true if it is, otherwise, false.

isNewCellFree: checks if the given cell belongs to NONE. If it does, it returns true, otherwise, false.

isNewCellOccupiedByPlayer: checks if the given cell belongs to the specified player. If it does, it returns true, otherwise, false.

performMove: releases the previous cell and occupies the new cell.

isTopCellOwnedBySecondPlayer: checks if the second player has reached the first row. If they do, it returns true, otherwise, false.

isBottomCellOwnedByFirstPlayer: checks if the first player has reached the last row. If they do, it returns true, otherwise, false.

Cell class:

It possesses the standard set of getters.

The constructor accepts the Player owner, sets it and sets selected field to false by default.

select: selects/deselects the cell which is changes the Boolean value to the opposite value.

occupyByNone: sets the owner of the cell to NONE.

occupyByPlayer: sets the owner of the cell to the given player.

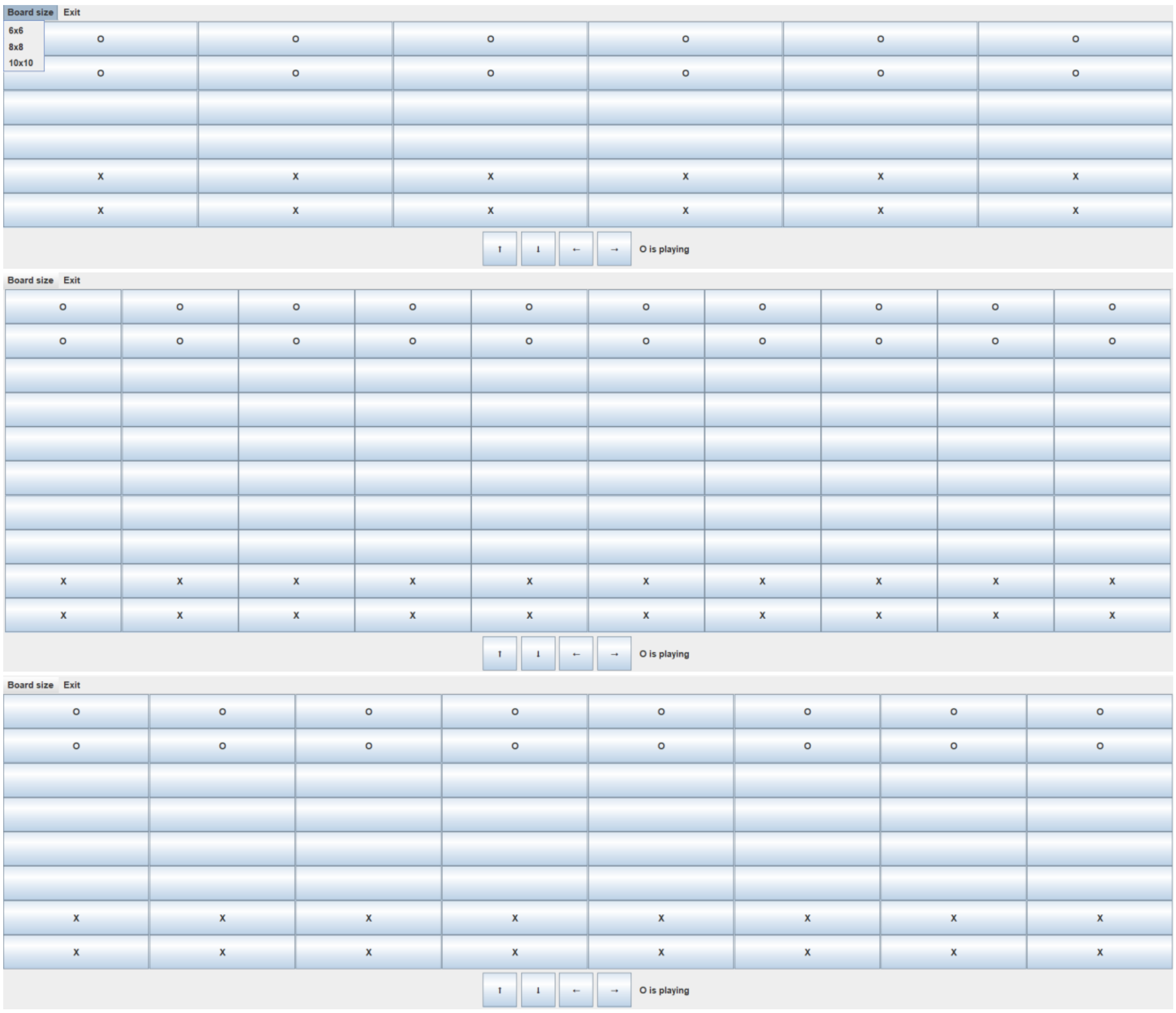
Manual testing

Test 1

Choosing the size.

When the size is chosen, the size of the board changes to the chosen size.

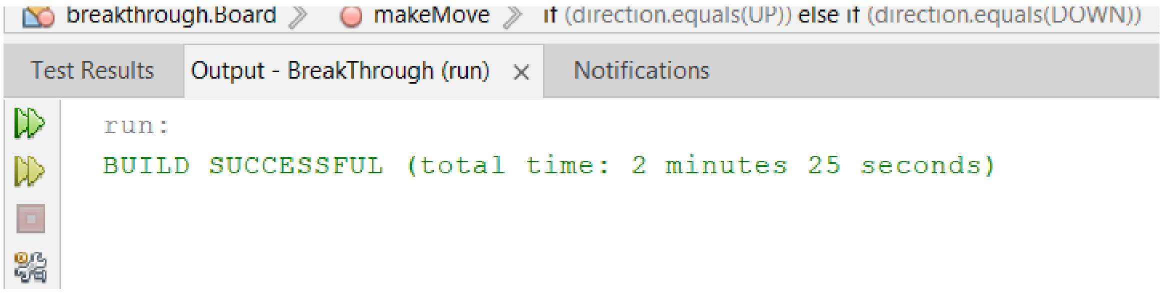
It works with every size given.



Test 2

Exiting on the exit button.

When pressing the exit button, the process stops and the application exits.

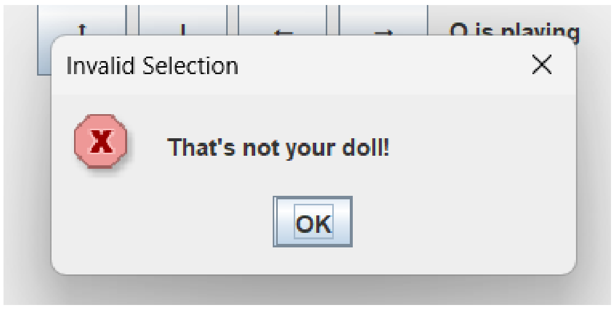

breakthrough.Board
makeMove
if (direction.equals(UP)) else if (direction.equals(DOWN))
Test Results
Output - BreakThrough (run)
Notifications

```
run:
BUILD SUCCESSFUL (total time: 2 minutes 25 seconds)
```

Test 3

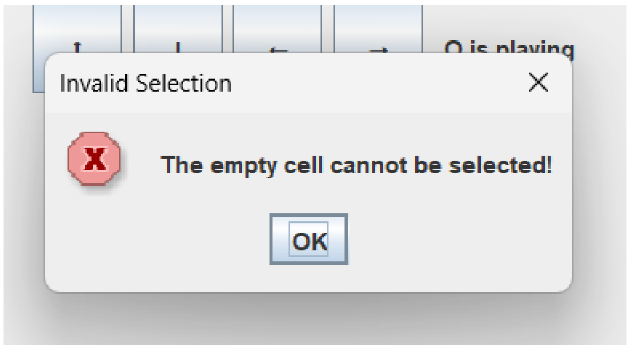
Selecting the cell.

When trying to select the opponent's cell, the error message shows up.


Invalid Selection

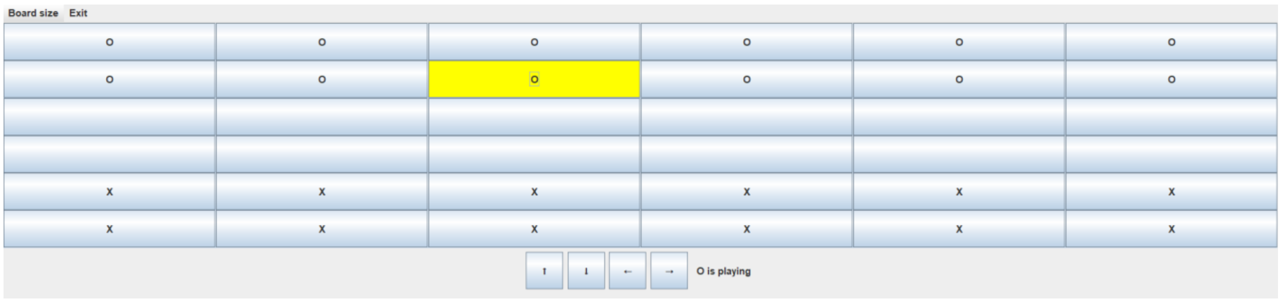
That's not your doll!

When trying to select the NONE's cell, the error message shows up.


Invalid Selection

The empty cell cannot be selected!

When trying to select the current player's cell, it gets selected (the Boolean value) and the background changes to the yellow color.

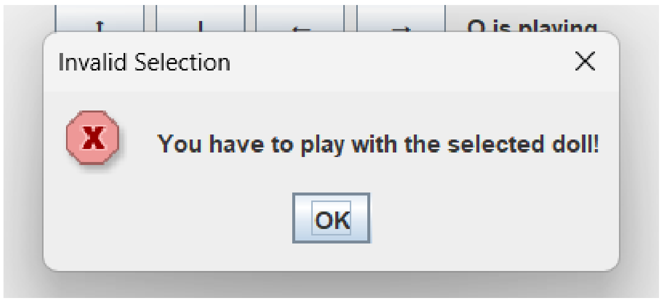


o	o	o	o	o	o
o	o	o	o	o	o
x	x	x	x	x	x
x	x	x	x	x	x

Test 4

Deselecting the cell.

When the cell is selected, I try to deselect it by clicking on it, the error message shows up, because I have to play with the doll I've already selected.


Invalid Selection

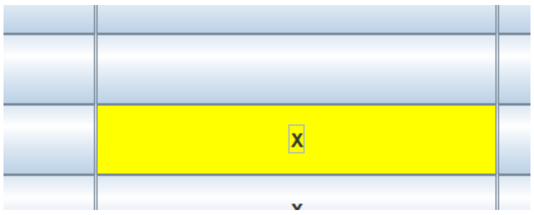
You have to play with the selected doll!

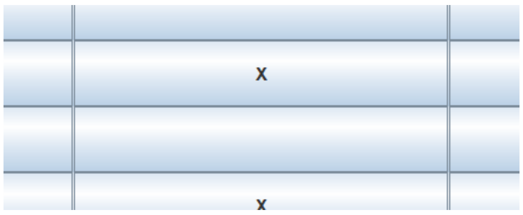
Test 5

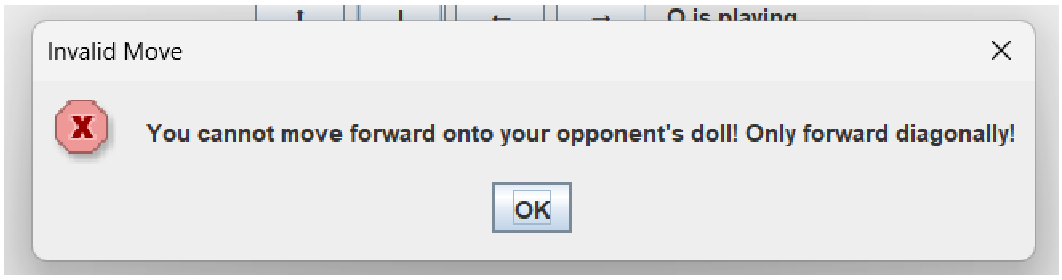
Making a move.

UP:

- When the selected cell belongs to the first player, it cannot move up. It has to move to the opposite side of the board.
- When the selected cell belongs to the second player and there is NONE'S cell in front of it, it moves up. Otherwise, the error message shows up.



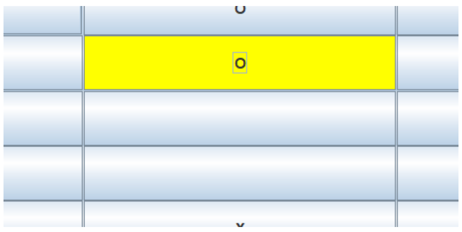


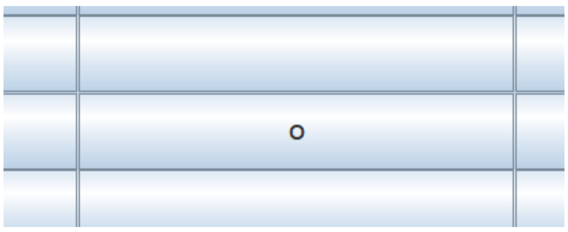


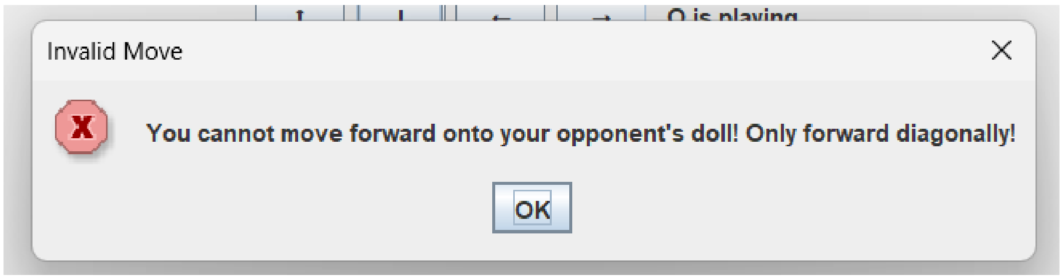
You cannot move forward onto your opponent's doll! Only forward diagonally!

DOWN:

- When the selected cell belongs to the first player and there is NONE's cell in front of it, it moves down. Otherwise, the error message shows up.
- When the selected cell belongs to the second player, it cannot move down. It has to move to the opposite side of the board.





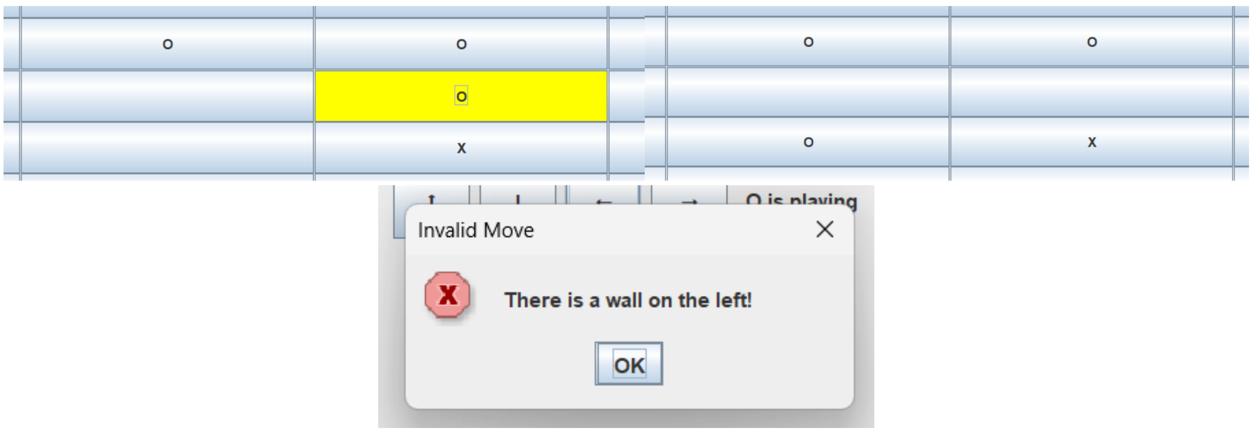


You cannot move forward onto your opponent's doll! Only forward diagonally!

LEFT:

- When there is a wall on the left from the selected cell, it cannot move left. The error message shows up.
- When there is the current's player doll on the left upper cell from the selected cell, it cannot move left. The error message shows up.

- When there is the opponent’s doll on the left upper cell from the selected cell, it moves left, and the previous cells becomes NONE’s.

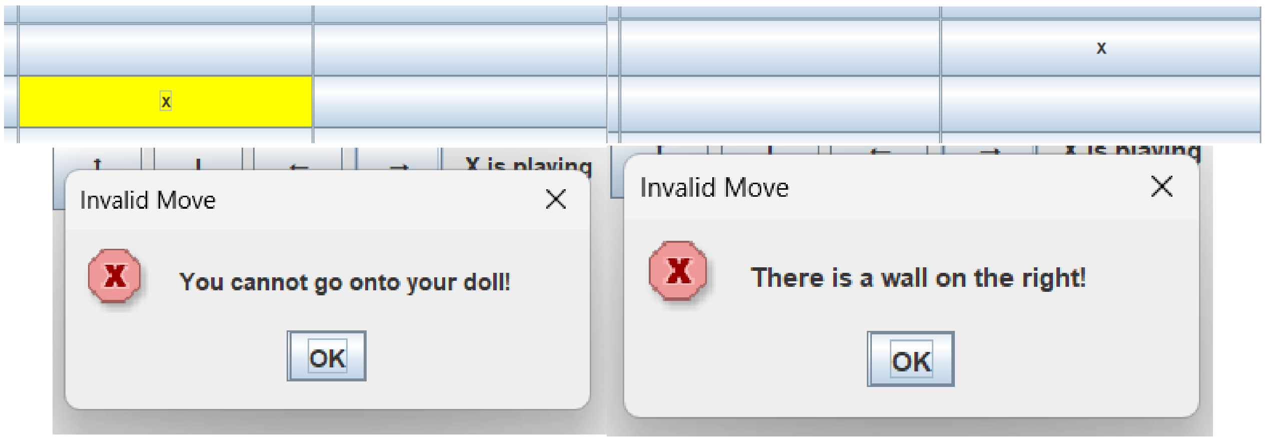


Invalid Move

There is a wall on the left!

RIGHT:

- When there is a wall on the right from the selected cell, it cannot move right. The error message shows up.
- When there is the current’s player doll on the right upper cell from the selected cell, it cannot move left. The error message shows up.
- When there is the opponent’s doll on the right upper cell from the selected cell, it moves right, and the previous cells becomes NONE’s.



Invalid Move

You cannot go onto your doll!

Invalid Move

There is a wall on the right!

Connection between the events and event handlers

SizeOptionListener:

This action listener is added to every menu item under the board size tab.

It listens to the action (click on the menu item) and changes the size of the board as per the size specified.

DirectionListener:

This action listener is added to every direction button.

It listens to the action (click on the button) and performs the move of the doll.

CellListener:

This action listener is added to every cell on the GUI board.

It listens to the action (click on the button) and selects the button on which the click was made by changing the Boolean value of the cell on the logic board and the background color of the cell on the GUI board.