objectOrientedGame

Quade Martin

Introduction

Using Java Objects and a few supplemental libraries, a simple turn-based game can be created in text form, then developed into a full GUI with interactive elements. By creating a game object and a GUI object, the objects can be called together to produce a fully functional game.

Game Plan

For my game, I chose to recreate the board game “Connect4” as it had the turn-based elements required for the assignment. I decided to have the game logic handled by one class, the board logic by another, and the GUI handled by a third separate class.

Back-End Board Logic

To create the Back-End of my game, I started with a text-based “grid” for the pieces to be played on. I created the game using some simple 2-D array logic.

```
private final int ROWS = 6;
private final int COLS = 7;
public int[][] board;

public Connect4Board() {
    board = new int[ROWS][COLS];
    for (int row = 0; row < ROWS; row++) {
        for (int col = 0; col < COLS; col++) {
            board[row][col] = 0;
        }
    }
}
```

Back-End Board Logic

Along with creating the board function, I created other functions that could be called to perform different actions of the board, such as adding pieces to the board, clearing the board, and checking whether a move is valid.

```
public void makeMove(int col) {
        for (int row = ROWS - 1; row
>= 0; row--) {
            if (board[row][col-1]==0)
{
                board[row][col-1] =
getCurrentPlayer();
                break;
            }
        }
        printBoard();
    }
```

```
public void clearBoard() {
        for (int row = 0; row <
board.length; row++) {
            for (int col = 0; col <
board[0].length; col++) {
                board[row][col] = 0;
            }
        }
    }
```

```
public boolean isValidMove(int col) {
        if (col < 1 || col > COLS) {
            return false;
        }
        if (board[0][col-1] != 0) {
            return false;
        }
        return true;
    }
```

Back-End Game Logic

To go along with the board functions, I created a class of game logic functions, as well as a main function to run the relevant board and game functions.

```
public Connect4Game() {
        board = new Connect4Board();
        System.out.println("Welcome to Connect4!");
    }

public int getWinner(){
        if(board.hasPlayerWon(1)){
            return(1);
        }
        if(board.hasPlayerWon(2)){
            return(2);
        }
        return(0);
    }
```

hasPlayerWon()

I also created a large function called “hasPlayerWon” with multiple inner functions that each scan the board for one of the three possible ways to win Connect4, being horitzontally, vertically, and diagonally. If a player has won, the function returns “true,” otherwise, it returns “false”

```
public boolean hasPlayerWon(int player) {
        // Check horizontal
        for (int row = 0; row < ROWS; row++) {
            for (int col = 0; col <= COLS - 4; col++) {
                if (board[row][col] == player && board[row][col+1] == player &&
                    board[row][col+2] == player && board[row][col+3] == player) {
                    return true;
        // Check vertical
        for (int row = 0; row <= ROWS - 4; row++) {
            for (int col = 0; col < COLS; col++) {
                if (board[row][col] == player && board[row+1][col] == player &&
                    board[row+2][col] == player && board[row+3][col] == player) {
                    return true;
        // Check diagonal (top left to bottom right)
        for (int row = 0; row <= ROWS - 4; row++) {
            for (int col = 0; col <= COLS - 4; col++) {
                if (board[row][col] == player && board[row+1][col+1] == player && board[row+2][col+2] == player && board[row+3][col+3] == player) {
                    return true;
        // Check diagonal (top right to bottom left)
        for (int row = 0; row <= ROWS - 4; row++) {
            for (int col = 3; col < COLS; col++) {
                if (board[row][col] == player && board[row+1][col-1] == player &&
                    board[row+2][col-2] == player && board[row+3][col-3] == player) {
                    return true;
        return false;
```

*For convenience’s sake, I have deleted some curly brackets from the code.

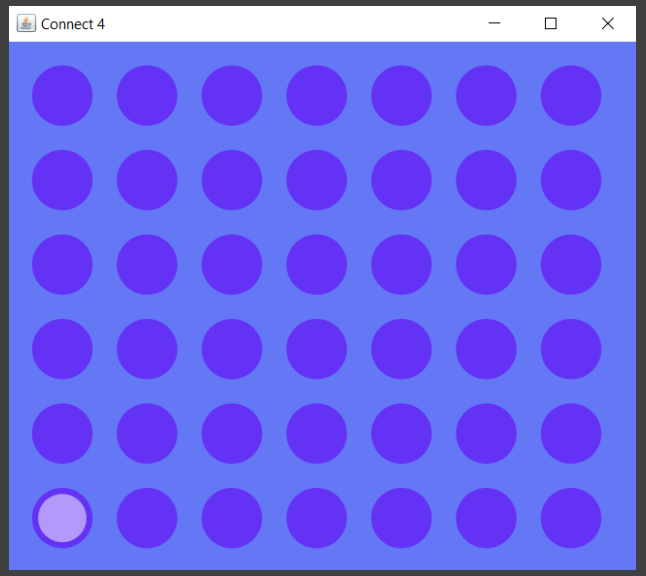
Game GUI

After completing the text-based game logic, I replaced the Scanner-based input system with a click map that placed a piece based on the location clicked.

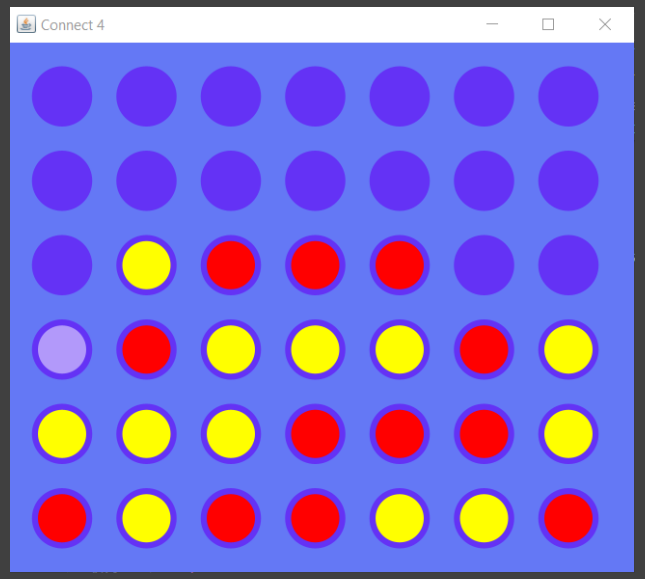
```
public void mouseMoved(MouseEvent e) {
        Point mousePos = e.getPoint();
        int x = (int) mousePos.getX();
        int y = (int) mousePos.getY();
        col = (x - 20) / 70;
        transform = new Point2D.Double(20 + col * 70, -20);
        repaint();
    }
    public void mousePressed(MouseEvent e) {
        int col = (e.getX() - 20) / 70;
        board.makeMove(col+1);
        repaint();
        }
```

Final Product

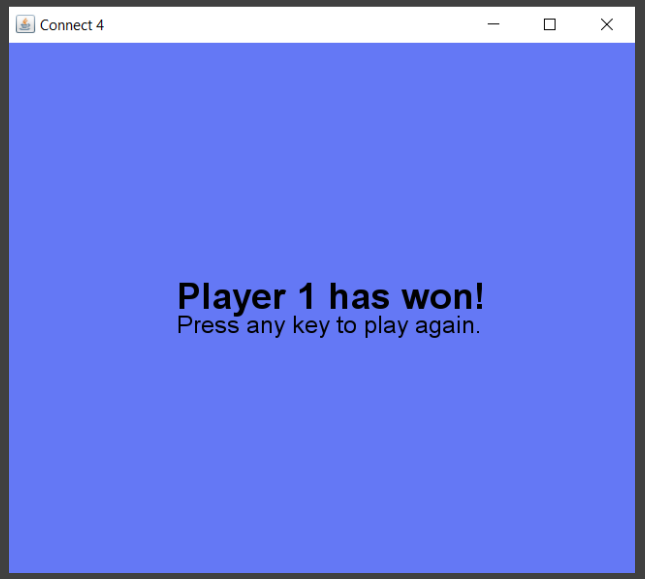
When the GUI code file is run, the following is produced:



Board with no pieces placed and mouse over first column



Board while a game is being played



Board after a game has finished with Player 1 winning