|  |  |  |  |
| --- | --- | --- | --- |
|  |  | **Tic Tac Toe** |  |

|  |
| --- |
| **Your Tasks (Mark these off as you go)** |
| * Define key vocabulary * Review the TicTacToe and TicTacToePane classes * Review and complete the TicTacToeLegend class * Review and the TicTacToeGrid class and get the game started * Receive credit for this lab guide |

* **Define key vocabulary**

**2D Array**

|  |
| --- |
|  |

* **Review the TicTacToe and TicTacToePane**

To better understand the mechanics of the Tic Tac Toe game you will be creating, lets explore the TicTacToe and TicTacToePane classes that you will be provided.

|  |
| --- |
| **TicTacToe.java** |
| public class TicTacToe {      public static void main(String[] args) {      JFrame frame = new JFrame("Tic Tac Toe");      frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);      //Creates a new TicTacToePane and packs it on the window      frame.getContentPane().add(new TicTacToePane());      frame.pack();      frame.setVisible(true);      }  } |

|  |
| --- |
| **TicTacToePane.java** |
| public class TicTacToePane extends JPanel {      private int a;      private int startPlayer = (int)(Math.random()\*2);      private TicTacToeGrid grid = new TicTacToeGrid(startPlayer);      private TicTacToeLegend legend = new TicTacToeLegend(new buttonListener(), startPlayer);      public TicTacToePane(){              setLayout(new BorderLayout());              setPreferredSize(new Dimension(600, 680));              add(grid,BorderLayout.NORTH);              add(legend, BorderLayout.CENTER);      }      //Creates an action listener on the reset button to reset the game      private class buttonListener implements ActionListener{                  @Override                  public void actionPerformed(ActionEvent e)                  {                          a = legend.getButtonAction((JButton) e.getSource());                          grid.resetAction(a);                  }      }  } |

|  |
| --- |
| Review the classes above. In the space below, briefly explain what is going on. Then draw a picture that illustrates the GUI that is created. |
|  |
|  |

* **Review and complete the TicTacToeLegend class**

The TicTacToeLegend class keeps track of whose turn it is and colors the legend the appropriate color. It also provides a button that allows for the game to be reset. Below is a working screenshot. In this section you will implement the changePlayer method.

A screenshot of a computer

Description automatically generated

When the game starts a random player is highlighted. In the example left, Player 1 was red. When player 1 clicked on the grid, the button changed to red and Player 2 was highlighted blue in the legend.

|  |
| --- |
| **TicTacToeLegend.java** |
| public class TicTacToeLegend extends JPanel {        static JLabel playerOne, playerTwo;      public TicTacToeLegend(ActionListener a, int startPlayer){          JPanel panel = new JPanel();          panel.setLayout(new FlowLayout());          setPreferredSize(new Dimension(600, 80));          Font f = new Font("Calibri",Font.BOLD,14);          playerOne = new JLabel("Player 1");          playerOne.setVerticalAlignment(JLabel.CENTER);          playerOne.setHorizontalAlignment(JLabel.CENTER);          playerOne.setForeground(Color.WHITE);          playerOne.setBorder(BorderFactory.createLineBorder(Color.black));          playerOne.setBackground(Color.BLACK);          playerOne.setOpaque(true);          playerOne.setFont(f);          playerOne.setPreferredSize(new Dimension(100,50));          add(playerOne);          playerTwo = new JLabel("Player 2");          playerTwo.setVerticalAlignment(JLabel.CENTER);          playerTwo.setHorizontalAlignment(JLabel.CENTER);          playerTwo.setForeground(Color.WHITE);          playerTwo.setBorder(BorderFactory.createLineBorder(Color.black));          playerTwo.setBackground(Color.BLACK);          playerTwo.setOpaque(true);          playerTwo.setFont(f);          playerTwo.setPreferredSize(new Dimension(100,50));          add(playerTwo);          JButton resetButton = new JButton();          resetButton.setText("Reset");          resetButton.setPreferredSize(new Dimension(100,50));          resetButton.addActionListener(a);          add(resetButton);          changePlayer(startPlayer);      }      /\*\*       \* TODO: Implement the changePlayer method which       \* changes the color of the playerOne and playerTwo labels       \* depending on the currentPlayer       \* @param currentPlayer       \*/      public static void changePlayer(int currentPlayer){        }      /\*\*       \* DO NOT EDIT       \* Returns 0 if the reset button is clicked       \*/      public int getButtonAction(JButton s){              return 0;      }  } |

|  |
| --- |
| In the space below, complete the changePlayer method. changePlayer accepts a parameter, currentPlayer, that has a value of 0 or 1.   * If the currentPlayer is 0, change the background of the playerOne button to red and background of the playerTwo button to black. * If the currentPlayer is 1, change the background of the playerTwo button to blue and the background of the playerOne button to black. |
|  |

* **Review the TicTacToeGrid class and get the game started**

The TicTacToeGrid class is where you will implement the functionality of your game. Take a minute to review the class below,

|  |
| --- |
| **TicTacToeGrid.java** |
| /\*\*   \* Creates a Tic Tac Toe game   \* Feel free to modify and/or add additional methods or variables as needed   \* @author Pluska   \*/  public class TicTacToeGrid extends JPanel implements ActionListener, TicTacToeInterface{        private JButton[][] tiles;      private int[][] boardValues;      private JButton clicked;      Font f;      private int[] playerStats = new int[2];      private int currentPlayer;      private int gridDimensions;      private boolean full = false;      private Point p;        /\*\*       \* constructor which creates the playing grid       \* @param d       \*/      public TicTacToeGrid(int startPlayer){          gridDimensions = 3;          setLayout(new GridLayout(gridDimensions, gridDimensions));          setPreferredSize(new Dimension(600, 600));          f = new Font("Calibri",Font.BOLD,20);          tiles = new JButton[gridDimensions][gridDimensions];          currentPlayer = startPlayer;          /\*\*           \* Populates the tiles array with buttons           \* and sets the background of each button           \* then adds the button to the grid           \*/          for(int row = 0; row < gridDimensions;row++){              for(int col = 0; col < gridDimensions; col++){                  tiles[row][col] = new JButton();                  tiles[row][col].addActionListener(this);                  add(tiles[row][col]);              }          }          //Once the grid is populated start the game.          startGame();      }        /\*\*       \* TODO: Complete startGame by initializing the boardValues array.       \* This array will be used to keep track of the values assigned to each button.       \* You need to decide how you want to keep track of each player and       \* whether or not the button has been clicked.       \* You then need to populate the boardValues array with the default values       \*/      public void startGame(){      }        /\*\*       \* TODO: Complete the setTile method which does the following.       \* Sets the tile to the appropriate color and sets the text (setText)  \* with an X or O.  \* After the tile is set you must update the boardValues array with  \* the appropriate value, you must check if there is a winner, you must       \* check if the board is full       \* @param currentTile is the button clicked       \* @param player is the currentPlayer       \*/      public void setTile(JButton currentTile, int player) {      }      /\*\*       \* TODO: Complete checkWinner which checks all the possible combinations  \* to see if the currentPlayer wins. If there is a winner the playerStats  \* must be updated and displayed by calling showStats       \* @param p is the currentPlayer       \*/      public void checkWinner(int p){      }        /\*\*       \* TODO:  Complete the switchPlayer method       \* Switches the current player       \* changes the color on the legend       \*/      public void switchPlayer(){ //this switches the player from player 1 to 2, or vice versa. This is called every time, after the program checks for a win      }          /\*\*       \* TODO: Complete the isFull method       \* Checks if the grid is full and there are no more       \* possible moves.       \* If the grid is moved alert the user       \* @return true if the grid is full       \*/      public boolean isFull() {                //Below is an example of how to create a popup that alerts the              //user that the grid is full              //JOptionPane error = new JOptionPane();              //error.showMessageDialog(null, "Sorry!  The board is full!"); //you must reset game after the error shows, otherwise it keeps saying every column is full            return full;      }        /\*\*       \* TODO: Complete the showStats method so that the       \* correct stats appear in the popup window       \* after each round       \*/      public void showStats(){          // JOptionPane stats = new JOptionPane();          // stats.showMessageDialog(null, "Stats should be displayed here");      }        /\*\*       \* TODO: Complete the resetGame method       \* resets the playing grid for new game       \* without resetting the game stats       \*/      public void resetGame() {        }      /\*\*       \* Locates which button that is clicked       \* @param c the button clicked       \* DO NOT EDIT       \*/      public void locateClicked(JButton c) {          for(int row = 0; row < gridDimensions;row++){              for(int col = 0; col < gridDimensions; col++){                  if(c == tiles[row][col]){                      clicked = c;                      //stores the current location as a point                      //you can access the row and col of the                      //button clicked as p.x and p.y, respectively                      p = new Point(row, col);                  }              }          }      }          /\*\*       \* DO NOT EDIT       \* implements the action button clicked on ConnectFourLegend       \* @param i the action of the button.  Depending on the value a different action is invoked       \*/      public void resetAction(int i) {          if(i == 0){              resetGame();          }      }      /\*\*       \* DO NOT EDIT       \*/      @Override      public void actionPerformed(ActionEvent e) {          clicked = (JButton) e.getSource();          locateClicked(clicked);          setTile(clicked, currentPlayer);      }      } |

|  |
| --- |
| Locate the startGame method. To get the game started we need to decide on some initialize board values. For example, if 0 represents player 1 and 1 represents player 2, then how do we represent a blank space? For this example, let’s use -1. Write the startGame method below. The startGame method should do the following,   * + - * Initialize the boardValues array       * Store -1 in each position of the boardValues       * Set the background of each button in the tiles array to black       * Set the text on each button in the tiles array to nothing ("") |
|  |

|  |
| --- |
| Locate the setTile method. Write the setTile method. The setTile method should do the following,   * + - * Set the button the appropriate color       * Set the text on the button to an x or o       * Update the boardValues array with the appropriate value       * Check if there is a winner by calling checkWinner       * Check if the board is full by calling isFull |
|  |

* **Write the hasAllValues method**

Now that we have a way to retrieve our columns from the Latin Square, we can check to see if each of the columns has all the values as the first row. Recall, that to be a Latin square the values that appear in the first row show also appear in every other row too.

The has AllValues method compares two arrays to see if they each have the same values. If they do, it returns true, otherwise it returns false. The hasAllValues method has the following signature,

|  |
| --- |
| public boolean hasAllValues(int arr1[], int arr2[]){} |

|  |
| --- |
| Write the hasAllValues method in the space below |
|  |

* **Write the isLatin method**

Now that we have all the necessary helper methods written we can check to see if any square is a Latin Square. The isLatin method should check for the following,

* The first row has no duplicate values
* All values in the first row of the square appear in each row of the square
* All values in the first row of the square appear n each column of the square

The signature for the isLatin method is shown below. If a square meets the above criteria, isLatin returns true, otherwise it returns false.

|  |
| --- |
| public boolean isLatin(int square[][]){} |

|  |
| --- |
| In the space below, write the isLatin method. |
|  |

* **Receive credit for this lab guide**

Submit this portion of the lab to Pluska to receive credit for the lab guide. Once received, your completed code challenges will also be graded and will count towards your final lab grade.