Object Oriented Programming

Final Project Report

Student Information:

Name: Gabriel Anderson **Student Id:** 2702256315

Course Code: COMP6699001

Course Name: Object Oriented Programming

Class: L2AC **Lecturer:** Jude Joseph Lamug Martinez,MCS

Type of Assignment: Final Project report

Table of Contents

A. Introduction

Background

Students are expected to create a program using what was taught in the Object Oriented Programming course and Students are also expected to use concept that was beyond what was taught in the Object Oriented Programming course

After doing some research and thinking, I decided to create a game called The Gam Game. It is a board game consisting of multiple mini games and I created the user interface using Java swing which was not taught in the course.

B. Project Specification

1. Game Overview

Title: The Gam Game
Genre: Board Game
Platform: Java Swing
Objective: The Gam Game combines multiple mini-games, where each mini-game's winner places a block in a central Tic-Tac-Toe board. The ultimate goal is to win the Tic-Tac-Toe game

2. Game Components

Main Board:

Tic-Tac-Toe board for placing 'X' and 'O' symbols.

Mini-Game Boards and Components:

Nim: Three piles of objects.

Connect 4: 7x6 grid and 'X' and 'O' symbols for each player.

Number Game: Number cards and addition ranges.

Player Pieces:

Symbols for Tic-Tac-Toe ('X' and 'O').

Symbols for Connect 4 ('X' and 'O').

Objects for Nim.

Instructions Manual:

Comprehensive guide detailing the rules for each mini-game and the overall game flow.

3. **Game Mechanics**

A. Starting the Game:
Players choose the game mode:
Player vs Player or Player vs AI.
The game randomly selects one of the three mini-games to play.

B. Playing Mini-Games:

Nim:
Players take turns removing 1-3 objects from a single pile.
The player forced to take the last object loses.

Connect 4:
Players alternate turns placing their symbols in a 7x6 grid.
The first to connect four symbols in a row wins.

Number Game:
Each player starts with a random number ranging from 0-21.
Players have 2 chances to add numbers from predefined ranges to their total.
The player with the highest number ≤ 21 wins.

Placing Symbols:
The winner of a mini-game places their symbol on the Tic-Tac-Toe board.

C. Winning the Game:

Players continue to play mini-games and place symbols on the Tic-Tac-Toe board.
The first player to align three symbols wins the Tic-Tac-Toe game.
If the board is full and no one wins, it results in a draw, and the board is reset.

4. Game Libraries/Modules

A. Swing:

Purpose: To create the graphical user interface (GUI) for the game.

Key Components:

- JFrame: Main window of the application.
- JPanel: Container for organizing UI components.
- JLabel: Display text and images.
- JButton: Create interactive buttons.
- JOptionPane: Create dialog boxes for user interactions.
- GridLayout: Manage layout for placing components in a grid.

B. AWT (Abstract Window Toolkit):

Purpose: To handle low-level windowing, events, and graphics.

Key Components:

- Font: Customize font styles and sizes.
- Dimension: Define the size of components.
- Component: Base class for all AWT components.

C. Event Handling:

Purpose: To manage user interactions with the game.

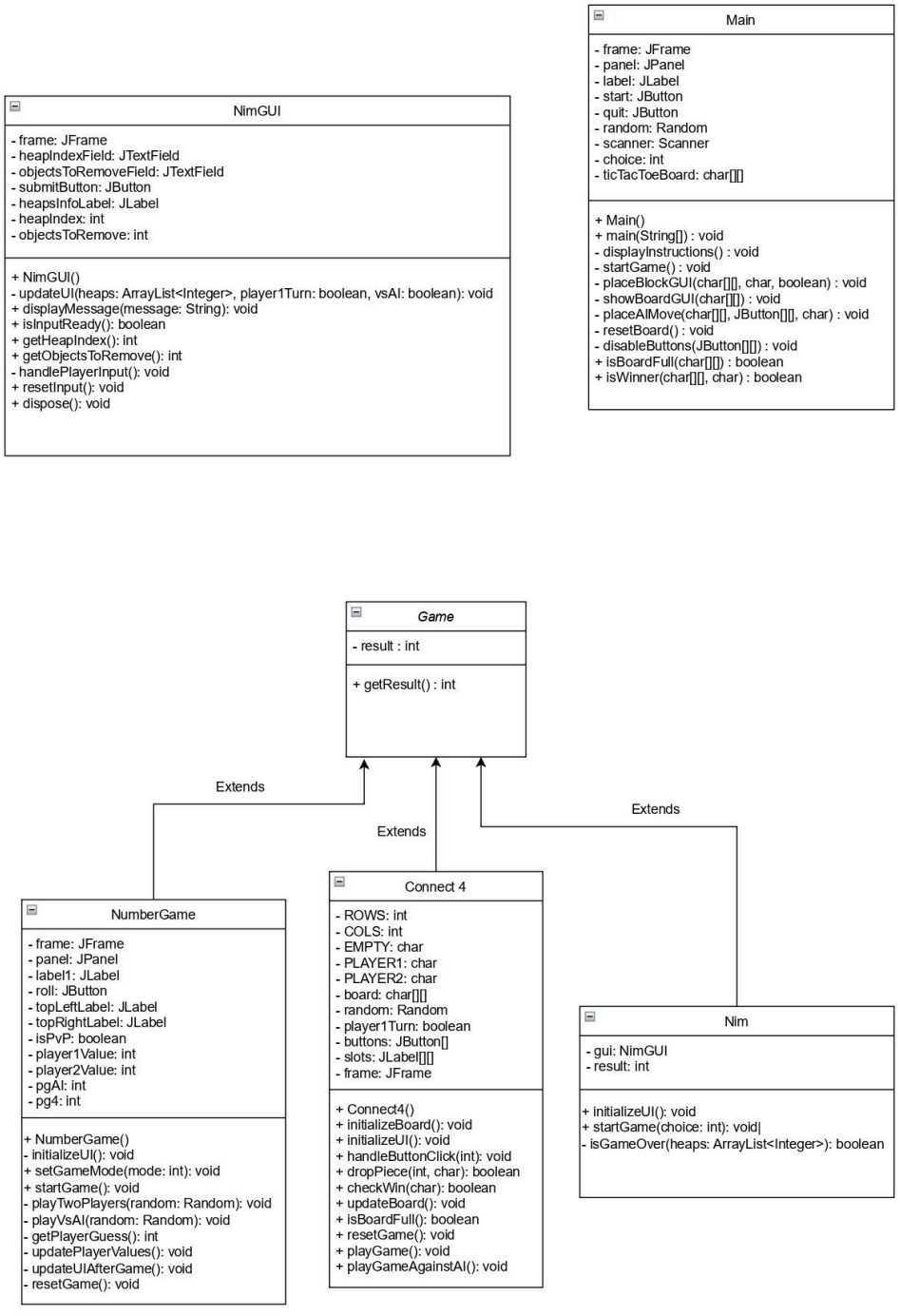
Key Components:

- ActionListener: Interface for receiving action events (e.g., button clicks).
- ActionEvent: Encapsulates information about an action event.

C. Solution Design

1. Class Diagram

The Class Diagram shows the structure of the classes that exists in the program



2. Algorithms

Main class

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Random;
import java.util.Scanner;
```

Import swing, libraries and other modules

```
final public class Main {

    // GUI components
    8 usages
    private JFrame frame;
    9 usages
    private JPanel panel;
    3 usages
    private JLabel label;
    5 usages
    private JButton start;
    5 usages
    private JButton quit;
    1 usage
    private static Random random = new Random();
    1 usage
    private static Scanner scanner = new Scanner(System.in);
    7 usages
    private int choice; // Game mode choice
    5 usages
    private char[][] ticTacToeBoard = new char[3][3]; // Tic-Tac-Toe board
```

Initialize GUI component, game variables and tictactoe board

```java
    // Constructor to initialize GUI
    1 usage
    public Main() {

        // Create main frame
        frame = new JFrame( title: "The Gam Game");
        panel = new JPanel();
        label = new JLabel( text: "Welcome to the Gam game!");
        start = new JButton( text: "Start");
        quit = new JButton( text: "Quit");

        // Set panel border and layout
        panel.setBorder(BorderFactory.createEmptyBorder( top: 200,  left: 250,  bottom: 200,  right: 250));
        panel.setLayout(new GridLayout( rows: 0,  cols: 1));

        // Set font, size, and style
        label.setFont(new Font( name: "Serif", Font.BOLD,  size: 24));

        // Customize buttons and alignment
        start.setFont(new Font( name: "Serif", Font.PLAIN,  size: 18));
        start.setAlignmentX(Component.CENTER_ALIGNMENT);
        quit.setFont(new Font( name: "Serif", Font.PLAIN,  size: 18));
        quit.setAlignmentX(Component.CENTER_ALIGNMENT);

        // Add action listeners to buttons
        start.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) { displayInstructions(); // Display instructions in a dialog }
        });
```

```java
    quit.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) { System.exit( status: 0); // Exit the application }
    });

    // Add components to the panel
    panel.add(label);
    panel.add(Box.createRigidArea(new Dimension( width: 0,  height: 20))); // Add space between label and start button
    panel.add(start);
    panel.add(Box.createRigidArea(new Dimension( width: 0,  height: 10))); // Add space between start button and quit button
    panel.add(quit);

    // Add panel to the frame
    frame.add(panel);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize( width: 800,  height: 600); // Set fixed size for the initial frame
    frame.setVisible(true);
}
```

Constructor for GUI that initializes the title, text, start and quit buttons. It also set the panel and layout of the screen as well as the font, size and style. ActionListener for buttons to work, add components to the panel and add panel to the frame.

```java
// Main method to run the program
public static void main(String[] args) {
    SwingUtilities.invokeLater(new Runnable() {
        @Override
        public void run() { new Main(); }
    });
}
```

Method to run the program by calling the Main() method

```java
// Method to display instructions
1 usage
private void displayInstructions() {

    // Display instructions in a dialog box
    JOptionPane.showMessageDialog( parentComponent: null,  message: "Welcome to the Gam game!\n\n" +
            "----------------------------------------\n"+
            "Play a randomized game and winner puts a block in tictactoe, game goes on until tictactoe is over\n\n" +
            "Game rules:\n" +
            "----------------------------------------\n"+
            "Nim:\n" + "The game is played with 3 piles of objects.\n" +
            "Two players take turns.\n" +
            "On each turn, a player must remove at least 1 and at most 3 object from a single pile.\n" +
            "The player forced to take the last object loses.\n" +
            "----------------------------------------\n"+
            "Connect 4:\n" + "Player puts a symbol within a grid with 7 columns and 6 rows.\n" +
            "Two players take turns.\nThe symbols occupies the lowest available space within the column.\n" +
            "A player wins by connecting four of their symbols in a row horizontally, vertically, or diagonally.\n" +
            "----------------------------------------\n"+
            "Number Game:\n" + "Each player gets a random number between 0 - 21\n" +
            "Players can add their numbers up to 2 times, ranging from 1-4, 5-7, 8-10\n" +
            "Players with the highest number <= 21 wins\n" +
            "----------------------------------------\n"+
            "Choose the game mode:\n" +
            "1. Player vs Player\n" +
            "2. Player vs AI",  title: "Instructions", JOptionPane.INFORMATION_MESSAGE);
```

```java
boolean validInput = false;
while (!validInput) {

    // Ask the user to enter the game mode
    String mode = JOptionPane.showInputDialog( parentComponent: null,  message: "Please enter the game mode (1 (Player vs Player) o
    try { // Check input is valid or not
        choice = Integer.parseInt(mode);
        if (choice != 1 && choice != 2) { // Input out of range
            JOptionPane.showMessageDialog( parentComponent: null,  message: "Invalid choice. Please enter 1 or 2.",  title: "Error",
        } else { // Valid input
            validInput = true;
            // Start the game in a new thread to keep the GUI responsive
            new Thread(new Runnable() {
                @Override
                public void run() { startGame(); }
            }).start();
        }
    } catch (NumberFormatException ex) { // Invalid input
        JOptionPane.showMessageDialog( parentComponent: null,  message: "Invalid input. Please enter a valid number (1 or 2).",  tit
    }
}
```

Method to display instructions, asks user to input the game mode with error handling

```
// Method to start game
1 usage
private void startGame() {
    int play = 0;

    while (play == 0) { // Loop until the user exits the game
        // Play a random game (Nim, Connect 4, or Number Game)
        int dice = random.nextInt( bound: 3); // Randomly choose between Nim, Connect 4, or Number Game
        int result = 0;

        // Initialize games and get result
        try {
            switch (dice) {
                case 0:
                    Nim nim = new Nim();
                    nim.startGame(choice); // Start Nim game with chosen mode (1 or 2)
                    result = nim.getResult(); // Get result
                    break;
                case 1:

                    Connect4 game = new Connect4();
                    if (choice == 1) { // Player vs Player mode
                        game.playGame();
                    } else { // Player vs AI mode
                        game.playGameAgainstAI();
                    }
                    result = game.getResult(); // Get result
                    break;
```

```
    case 2:

        while (result == 0) {
            NumberGame numberGame = new NumberGame();
            if (choice == 1) { // Player vs Player mode
                numberGame.setGameMode(1);
                numberGame.startGame();
            } else { // Player vs AI mode
                numberGame.setGameMode(2);
                numberGame.startGame();
            }
            result = numberGame.getResult(); // Get result
        }
        break;
}
```

Method to start game, play a random game randomized by the random.nextint(3), playing Nim, connect 4 or Number game based on the value and get result of the game

```
// Check if there is a winner or a draw
if (result == 1 || result == 2) {
    char symbol = (result == 1) ? 'X' : 'O'; // Determine symbol based on result
    boolean isAI = (choice == 2 && result == 2); // Check if AI won in Player vs AI mode

    // Show the Tic-Tac-Toe GUI
    placeBlockGUI(ticTacToeBoard, symbol, isAI);

    // Check if the current player has won
    if (isWinner(ticTacToeBoard, symbol)) {
        if (isAI) { // AI win
            JOptionPane.showMessageDialog(frame, message: "AI Wins!", title: "Game Result", JOptionPane.INFORMATION_MESSA
        } else { // Player win
            JOptionPane.showMessageDialog(frame, message: "Player " + ((result == 1) ? '1' : '2') + " Wins!", title: "Game
        }

        // Reset tictactoe board and exit loop
        resetBoard();
        break;
    } else if (isBoardFull(ticTacToeBoard)) { // Check if the board is full and it's a draw
        JOptionPane.showMessageDialog(frame, message: "Draw! The board is full ", title: "Result", JOptionPane.INFORMATION

        // Reset tictactoe board and exit loop
        resetBoard();
        break; // Exit the loop if the board is full
    }
}
```

Check if there is a winner or a draw. Shows the tictactoe board and if game is over (there is a winner or a draw) reset board and exit loop

```
// Method to show the Tic-Tac-Toe GUI and place a block
1 usage
private void placeBlockGUI(char[][] board, char symbol, boolean isAI) {
    JFrame ticTacToeFrame = new JFrame( title: "Tic-Tac-Toe");
    JPanel ticTacToePanel = new JPanel();
    ticTacToePanel.setLayout(new BorderLayout());

    // Set text to show whose turn it is
    JLabel statusLabel = new JLabel( text: "Player " + (symbol == 'X' ? '1' : '2') + " (" + symbol + ") turn", SwingConstants.CENTER);
    statusLabel.setFont(new Font( name: "Serif", Font.BOLD,  size: 24));
    ticTacToePanel.add(statusLabel, BorderLayout.NORTH);

    // Creates panel and initialize 3x3 array to hold buttons
    JPanel boardPanel = new JPanel();
    boardPanel.setLayout(new GridLayout( rows: 3,  cols: 3));
    JButton[][] buttons = new JButton[3][3];

    // Create buttons for tictactoe board
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            buttons[i][j] = new JButton(board[i][j] == '\u0000' ? "" : String.valueOf(board[i][j])); // if not blank, show the strir
            buttons[i][j].setFont(new Font( name: "Serif", Font.PLAIN,  size: 60));
            final int row = i;
            final int col = j;
            buttons[i][j].setEnabled(board[i][j] == '\u0000'); // Disable button if already occupied
```

```
        buttons[i][j].addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                if (board[row][col] == '\u0000') {
                    board[row][col] = symbol;
                    buttons[row][col].setText(String.valueOf(symbol));
                    disableButtons(buttons); // Disable all buttons after one move
                }
            }
        });
        boardPanel.add(buttons[i][j]); // Show button placed in the GUI
    }
}

if (isAI) { // AI places move and show tictactoe board
    placeAIMove(board, buttons, symbol);
    showBoardGUI(ticTacToeBoard);
    return;
}

// Continue button to close tictactoe frame
JButton continueButton = new JButton( text: "Continue");
continueButton.setFont(new Font( name: "Serif", Font.PLAIN,  size: 24));
continueButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) { ticTacToeFrame.dispose(); // Close tictactoe frame }
});
```

```
    // Wait for the Tic-Tac-Toe window to close before continuing
    while (ticTacToeFrame.isDisplayable()) {
        try {
            Thread.sleep( millis: 100);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

Method to show the tictactoe board, button to place a symbol and a continue button to exit the tictactoe board

```java
// Method to show the current state of the Tic-Tac-Toe board in the GUI (for when AI turn)
1 usage
private void showBoardGUI(char[][] board) {
    JFrame ticTacToeFrame = new JFrame( title: "Tic-Tac-Toe Board");
    JPanel ticTacToePanel = new JPanel();
    ticTacToePanel.setLayout(new BorderLayout());

    // Show text to show it is AI's turn
    JLabel statusLabel = new JLabel( text: "AI (O) turn", SwingConstants.CENTER);
    statusLabel.setFont(new Font( name: "Serif", Font.BOLD,  size: 24));
    ticTacToePanel.add(statusLabel, BorderLayout.NORTH);

    // Creates panel and initialize 3x3 array to hold buttons
    JPanel boardPanel = new JPanel();
    boardPanel.setLayout(new GridLayout( rows: 3,  cols: 3));
    JButton[][] buttons = new JButton[3][3];

    // Show buttons placed in the GUI
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            buttons[i][j] = new JButton(board[i][j] == '\u0000' ? "" : String.valueOf(board[i][j]));
            buttons[i][j].setFont(new Font( name: "Serif", Font.PLAIN,  size: 60));
            buttons[i][j].setEnabled(false); // Disable button for viewing only
            boardPanel.add(buttons[i][j]);
        }
    }
```

```java
    // Add components to Tic-Tac-Toe panel
    ticTacToePanel.add(boardPanel, BorderLayout.CENTER);

    // Continue button to close the Tic-Tac-Toe frame
    JButton continueButton = new JButton( text: "Continue");
    continueButton.setFont(new Font( name: "Serif", Font.PLAIN,  size: 24));
    continueButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) { ticTacToeFrame.dispose(); // Close tictactoe frame }
    });

    // Add continue button to the panel
    ticTacToePanel.add(continueButton, BorderLayout.SOUTH);

    // Setup Tic-Tac-Toe frame properties
    ticTacToeFrame.add(ticTacToePanel);
    ticTacToeFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    ticTacToeFrame.setSize( width: 400,  height: 400); // Set fixed size for the Tic-Tac-Toe frame
    ticTacToeFrame.setVisible(true);

    // Wait for the Tic-Tac-Toe window to close before continuing
    while (ticTacToeFrame.isDisplayable()) {
        try {
            Thread.sleep( millis: 100);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

Method to show the tictactoe board GUI and wait for the tictactoe frame to be closed to continue

```java
// Method for AI to place its move
1 usage
private void placeAIMove(char[][] board, JButton[][] buttons, char symbol) {
    // AI places the first available move
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            if (board[i][j] == '\u0000') {
                board[i][j] = symbol;
                buttons[i][j].setText(String.valueOf(symbol));
                buttons[i][j].setEnabled(false);
                return;
            }
        }
    }
}
```

Method for AI to find the first empty slot and place AI's symbol

```java
// Method to reset the Tic-Tac-Toe board
2 usages
private void resetBoard() {
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            ticTacToeBoard[i][j] = '\u0000';
        }
    }
}

// Method to disable all buttons on the Tic-Tac-Toe board
1 usage
private void disableButtons(JButton[][] buttons) {
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            buttons[i][j].setEnabled(false);
        }
    }
}
```

Method to reset the tictactoe board by setting all the slot into blank and Methods that disable all buttons.

```java
// Method to check if the board is full
1 usage
public static boolean isBoardFull(char[][] board) {
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            if (board[i][j] == '\u0000') {
                return false;
            }
        }
    }
    return true;
}
```

Method to check if the board is full by checking if there is an empty slot

```java
    // Method to check if the current player has won
    1 usage
    public static boolean isWinner(char[][] board, char symbol) {
        // Check rows
        for (int i = 0; i < 3; i++) {
            if (board[i][0] == symbol && board[i][1] == symbol && board[i][2] == symbol) {
                return true;
            }
        }
        // Check columns
        for (int i = 0; i < 3; i++) {
            if (board[0][i] == symbol && board[1][i] == symbol && board[2][i] == symbol) {
                return true;
            }
        }
        // Check diagonal for both directions
        if (board[0][0] == symbol && board[1][1] == symbol && board[2][2] == symbol) {
            return true;
        }
        if (board[0][2] == symbol && board[1][1] == symbol && board[2][0] == symbol) {
            return true;
        }
        return false;
    }
}
```

Method to check if a player has won the tictactoe by checking the columns, rows, and diagonal

Nim

```java
import javax.swing.*;
import java.awt.*;
import java.util.ArrayList;
import java.util.Random;

2 usages
public class Nim extends Game {

    // GUI class for the game
    2 usages
    private static class NimGUI {

        // GUI components
        9 usages
        private JFrame frame;
        4 usages
        private JTextField heapIndexField;
        4 usages
        private JTextField objectsToRemoveField;
        3 usages
        private JButton submitButton;
        3 usages
        private JLabel heapsInfoLabel;

        // Variable to store the heap index and number of objects to remove input
        4 usages
        private int heapIndex = -1;
        4 usages
        private int objectsToRemove = -1;
```

Import, initializes and declare components for the game

```java
// Constructor to set up the GUI
1 usage
public NimGUI() {

    // Create main frame
    frame = new JFrame( title: "Nim Game");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize( width: 400,  height: 300);
    frame.setLocationRelativeTo(null); // Center the window

    // Display welcome message
    JOptionPane.showMessageDialog(frame,  message: "You got Nim!");
    JPanel inputPanel = new JPanel(new GridBagLayout());
    GridBagConstraints gbc = new GridBagConstraints();
    gbc.insets = new Insets( top: 5,  left: 5,  bottom: 5,  right: 5); // Add padding

    // Label for heap information
    heapsInfoLabel = new JLabel( text: "Heaps Information:");
    gbc.gridx = 0;
    gbc.gridy = 0;
    gbc.gridwidth = 2;
    inputPanel.add(heapsInfoLabel, gbc);
    gbc.gridwidth = 1; // Reset grid width
```

```java
    // Input field for heap index
    heapIndexField = new JTextField( columns: 5);
    gbc.gridx = 0;
    gbc.gridy = 1;
    inputPanel.add(new JLabel( text: "Heap Index (1-3):"), gbc);
    gbc.gridx = 1;
    inputPanel.add(heapIndexField, gbc);

    // Input field for number of objects to remove
    objectsToRemoveField = new JTextField( columns: 5);
    gbc.gridx = 0;
    gbc.gridy = 2;
    inputPanel.add(new JLabel( text: "Objects to Remove (1-3):"), gbc);
    gbc.gridx = 1;
    inputPanel.add(objectsToRemoveField, gbc);

    // Submit button for player input
    submitButton = new JButton( text: "Enter");
    submitButton.addActionListener(e -> handlePlayerInput());
    gbc.gridx = 0;
    gbc.gridy = 3;
    gbc.gridwidth = 2;
    gbc.anchor = GridBagConstraints.CENTER;
    inputPanel.add(submitButton, gbc);

    frame.add(inputPanel, BorderLayout.CENTER);
    frame.setVisible(true);
}
```

Constructor for the GUI by setting up the mainframe, displaying welcome message, information texts and their placement for the game and submit button for player input

```
// Method to update the GUI
2 usages
public void updateUI(ArrayList<Integer> heaps, boolean player1Turn, boolean vsAI) {

    // Displays the current state
    StringBuilder message = new StringBuilder("<html>Current State:<br>");
    for (int i = 0; i < heaps.size(); i++) {
        message.append("Heap ").append(i + 1).append(": ").append(heaps.get(i)).append(" objects<br>");
    }
    message.append("<br>");

    // Displays whose turn it is
    if (vsAI) {
        message.append(player1Turn ? "Player 1's turn." : "AI's turn.");
    } else {
        message.append(player1Turn ? "Player 1's turn." : "Player 2's turn.");
    }
    message.append("</html>");
    heapsInfoLabel.setText(message.toString());
}
```

Method to display the updated heap, whose turn it is and a bit of indentation using html

```
// Method to display message
9 usages
public void displayMessage(String message) { JOptionPane.showMessageDialog(frame, message); }

// Method to check if player input is ready
1 usage
public boolean isInputReady() { return heapIndex >= 0 && objectsToRemove >= 0; }

// Getter for player input
1 usage
public int getHeapIndex() { return heapIndex; }

// Getter for objects to remove
1 usage
public int getObjectsToRemove() { return objectsToRemove; }
```

Method to display message in the GUI, check if input is ready and getter for both heap index and objects to remove from user input

```java
// Error handling method to handle player input from text fields
1 usage
private void handlePlayerInput() {
    try {
        heapIndex = Integer.parseInt(heapIndexField.getText()) - 1;
        objectsToRemove = Integer.parseInt(objectsToRemoveField.getText());
        heapIndexField.setText("");
        objectsToRemoveField.setText("");
    } catch (NumberFormatException e) {
        displayMessage("Invalid input. Please enter numbers only.");
    }
}
```

Error handling method to ensure user enters integers

```java
    // Reset player input
    4 usages
    public void resetInput() {
        heapIndex = -1;
        objectsToRemove = -1;
    }

    // Close frame
    1 usage
    public void dispose() { frame.dispose(); }
}
```

Method to reset player input and close the frame of currently open frame

```java
private NimGUI gui;

// Method to initialize UI
1 usage
public void initializeUI() { gui = new NimGUI(); }
```

Declare and call NimGUI in gui

```java
// Method to start the game
1 usage
public void startGame(int choice) {
    initializeUI();

    // Set up initial state for each heap
    ArrayList<Integer> heaps = new ArrayList<>();
    Random random = new Random();

    // Each heap with random numbers
    heaps.add(random.nextInt( bound: 2) + 3);
    heaps.add(random.nextInt( bound: 2) + 4);
    heaps.add(random.nextInt( bound: 2) + 5);

    // Main game loop
    boolean player1Turn = true; // Variable to track whose turn it is
    boolean vsAI = choice == 2; // Determine if the game is against AI based on the user's choice
    boolean gameOver = false; // Flag to track if the game is over
```

Method to start and set the game, setting up the initial state of heaps and fill each heap with random numbers. Declare booleans value to track whose turn it is, check if game is over and determine which game mode it is.

```java
while (!gameOver) {
    // Update GUI with current state of heaps
    gui.updateUI(heaps, player1Turn, vsAI);

    // Check if the game is over
    if (isGameOver(heaps)) {
        if (player1Turn) { // If it's player 1's turn when game ends
            if (vsAI) {
                gui.displayMessage("You win!"); // Player 1 wins against AI
                result = 1;
            } else {
                gui.displayMessage("Player 1 wins!"); // Player 1 wins against Player 2
                result = 1;
            }
        } else { // If it's not player 1's turn when game ends
            if (vsAI) {
                gui.displayMessage("AI wins!"); // AI wins
                result = 2;
            } else {
                gui.displayMessage("Player 2 wins!"); // Player 2 wins
                result = 2;
            }
        }

        // Exit game (Close frame and exit loop)
        gui.dispose();
        gameOver = true;
        break;
    }
```

If game is not over, update the GUI with the current state of heaps. Check if game is over, if true, exit game (closes the frame and exit loop)

```
// Player's turn
if (player1Turn || !vsAI) {
    while (!gui.isInputReady()) {
        // Wait for player input
        try {
            Thread.sleep( millis: 100); // Sleep briefly to reduce CPU usage
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    // Get player's input for heap index and objects to remove
    int heapIndex = gui.getHeapIndex();
    int objectsToRemove = gui.getObjectsToRemove();

    // Validate player input
    if (heapIndex < 0 || heapIndex >= heaps.size()) { // Invalid heap input
        gui.displayMessage("Invalid heap index. Please enter a number between 1 and " + heaps.size() + ".");
        gui.resetInput();
        continue;
    }

    // Invalid number of objects
    if (heaps.get(heapIndex) == 0) { // Heap is empty
        gui.displayMessage("Heap " + (heapIndex + 1) + " is empty. Please choose another heap.");
        gui.resetInput();
        continue;
    }
```

```
    if (objectsToRemove < 1 || objectsToRemove > Math.min(heaps.get(heapIndex), 3)) { // Input out of available range
        gui.displayMessage("Invalid number of objects. Please enter a number between 1 and " + Math.min(heaps.get(heapIndex), 3) + ".
        gui.resetInput();
        continue;
    }

    // Update heap with player input
    heaps.set(heapIndex, heaps.get(heapIndex) - objectsToRemove);
    gui.resetInput();
} else {
    // AI's turn
    int heapIndex, objectsToRemove;
    do {
        heapIndex = random.nextInt(heaps.size()); // Random heap to remove from
    } while (heaps.get(heapIndex) == 0); // Ensure selected heap is not empty

    // Random objects from 1-3 to remove and update heap
    objectsToRemove = random.nextInt(Math.min(3, heaps.get(heapIndex))) + 1;
    heaps.set(heapIndex, heaps.get(heapIndex) - objectsToRemove);

    // Update GUI and display AI removes how many objects from which heap
    gui.updateUI(heaps, player1Turn, vsAI);
    gui.displayMessage("AI removes " + objectsToRemove + " objects from heap " + (heapIndex + 1));
}

player1Turn = !player1Turn; // Switch turns
```

If it is the player's turn, wait for input. Store player's input for heap index and number of objects to remove in a variable. Error handling for invalid heap index, heap is empty, and invalid number of objects to remove, update heap with the input. If it is AI's turn, choose random heap to remove and ensure that heap is not empty, and then the AI chooses a random number of objects to remove 1-3 and update heap and switch turns.

```
    // Check if the game is over (all heaps are empty)
    1 usage
    public static boolean isGameOver(ArrayList<Integer> heaps) {
        for (int heap : heaps) {
            if (heap > 0) {
                return false; // Game is not over if any heap still has objects
            }
        }
        return true; // Game is over if all heaps are empty
    }
}
```

Method to check if there is still heap in heaps, checking if game is over or not

Connect 4

```java
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Random;

2 usages
public class Connect4 extends Game {

    // Constants for game parameters
    10 usages
    private static final int ROWS = 6;
    17 usages
    private static final int COLS = 7;
    4 usages
    private static final char EMPTY = '-';
    5 usages
    private static final char PLAYER1 = 'X';
    4 usages
    private static final char PLAYER2 = 'O';

    // Game state variables
    22 usages
    private char[][] board;
    3 usages
    private Random random;
    18 usages
    private boolean player1Turn = true;
    4 usages
    private JButton[] buttons;
    6 usages
    private JLabel[][] slots;
```

Import, initializes and declare components for the game

```
private JFrame frame;

// Constructor to initialize game
1 usage
public Connect4() {
    board = new char[ROWS][COLS];
    random = new Random();
    // Initialize Board and GUI
    initializeBoard();
    initializeUI();
}
```

constructor to initialize game by calling initializeBoard and initializeUI function

```
// Initialize the game board with empty slots
2 usages
private void initializeBoard() {
    for (int i = 0; i < ROWS; i++) {
        for (int j = 0; j < COLS; j++) {
            board[i][j] = EMPTY;
        }
    }
}
```

Initialize the board with empty slots

```java
// Initialize the GUI
1 usage
private void initializeUI() {

    // Create main frame
    frame = new JFrame( title: "Connect 4");
    frame.setLayout(new BorderLayout());
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    JPanel boardPanel = new JPanel();

    // Display welcome message
    JOptionPane.showMessageDialog(frame,  message: "You got Connect 4!");

    // Shows the buttons per column and labels per slot
    boardPanel.setLayout(new GridLayout( rows: ROWS + 1, COLS));
    buttons = new JButton[COLS];
    slots = new JLabel[ROWS][COLS];

    // Create buttons for each column
    for (int i = 0; i < COLS; i++) {
        buttons[i] = new JButton( text: "Drop");
        int col = i;
        buttons[i].addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) { handleButtonClick(col); // Button click event handler }
        });
        boardPanel.add(buttons[i]); // Add button to panel
    }
```

Method to create the mainframe, displaying welcome message and show buttons and labels per slot

```java
    // Create labels for each board slot
    for (int i = 0; i < ROWS; i++) {
        for (int j = 0; j < COLS; j++) {
            slots[i][j] = new JLabel(String.valueOf(EMPTY), SwingConstants.CENTER);
            slots[i][j].setBorder(BorderFactory.createLineBorder(Color.BLACK));
            slots[i][j].setFont(new Font( name: "Serif", Font.PLAIN,  size: 24));
            boardPanel.add(slots[i][j]); // Add label to board panel
        }
    }

    // Add board panel to main frame and setting the frame
    frame.add(boardPanel, BorderLayout.CENTER);
    frame.setSize( width: 700,  height: 600);
    frame.setLocationRelativeTo(null);
    frame.setVisible(true);
}
```

Creates the label for each slot and add the board panel to the mainframe and sets up the frame.

```java
// Event handler for column drop buttons
1 usage
private void handleButtonClick(int col) {
    if (dropPiece(col, player1Turn ? PLAYER1 : PLAYER2)) { // Try to drop piece in column
        updateBoard();
        if (checkWin(player1Turn ? PLAYER1 : PLAYER2)) {

            // Win message and reset game state
            JOptionPane.showMessageDialog(frame, message: "Player " + (player1Turn ? "1" : "2") + " wins!");
            result = player1Turn ? 1 : 2;
            resetGame();
        } else if (isBoardFull()) {

            // Draw message and reset game state
            JOptionPane.showMessageDialog(frame, message: "It's a draw!");
            result = 0;
            resetGame();
        } else {
            player1Turn = !player1Turn; // Switch turns
        }
    }
}
```

Event handler for column drops, checks if any player has win or the board is full, resetting the game.

```java
// Attempts to drop piece into the specified column by the current player
4 usages
private boolean dropPiece(int col, char player) {
    if (col < 0 || col >= COLS) {
        JOptionPane.showMessageDialog(frame, message: "Invalid column. Please choose a column between 0 and 6.");
        return false;
    }

    // Check empty slot in the column
    for (int i = ROWS - 1; i >= 0; i--) {
        if (board[i][col] == EMPTY) {
            board[i][col] = player;
            return true;
        }
    }
    JOptionPane.showMessageDialog(frame, message: "Column is full. Please choose another column.");
    return false;
}
```

Method to drop piece into the specified column and check if input is valid or not

```java
// Method to check if current player has won the game
3 usages
private boolean checkWin(char player) {

    // Check horizontally
    for (int i = 0; i < ROWS; i++) {
        for (int j = 0; j <= COLS - 4; j++) {
            if (board[i][j] == player && board[i][j + 1] == player && board[i][j + 2] == player && board[i][j + 3] == player) {
                frame.dispose();
                return true;
            }
        }
    }

    // Check vertically
    for (int j = 0; j < COLS; j++) {
        for (int i = 0; i <= ROWS - 4; i++) {
            if (board[i][j] == player && board[i + 1][j] == player && board[i + 2][j] == player && board[i + 3][j] == player) {
                frame.dispose();
                return true;

            }
        }
    }
```

```java
// Check diagonally for both directions
for (int i = 0; i <= ROWS - 4; i++) {
    for (int j = 0; j <= COLS - 4; j++) {
        if (board[i][j] == player && board[i + 1][j + 1] == player && board[i + 2][j + 2] == player && board[i + 3][j + 3] == player)
            frame.dispose();

            return true;
        }
        if (board[i][j + 3] == player && board[i + 1][j + 2] == player && board[i + 2][j + 1] == player && board[i + 3][j] == player)
            frame.dispose();
            return true;
        }
    }
}
return false;
```

Method to check if any player has won by checking horizontally, vertically, and diagonally for both directions. If true, close frame and exit loop, if false, continue the game.

```java
private void updateBoard() {
    for (int i = 0; i < ROWS; i++) {
        for (int j = 0; j < COLS; j++) {
            slots[i][j].setText(String.valueOf(board[i][j]));
        }
    }
}

// Check if board is full
3 usages
private boolean isBoardFull() {
    for (int i = 0; i < COLS; i++) {
        if (board[0][i] == EMPTY) {
            return false; // Empty slot, board not full
        }
    }
    return true; // Board is full
}

// Reset game state
4 usages
private void resetGame() {

    // Clear game board and reset  to player 1 turn
    initializeBoard();
    updateBoard();
    player1Turn = true;
}
```

Method to show the current state of the board and check if the board is full and resets the game, resetting the board and back to player 1 turn.

```
// Method for player vs player
1 usage
public void playGame() {

    // Reset game state to start a new game
    resetGame();
    boolean gameEnd = false;

    while (!gameEnd) {
        char currentPlayer = (player1Turn) ? PLAYER1 : PLAYER2; // Determine current player

        // Display turn information and ask for column selection
        JTextField textField = new JTextField();
        Object[] message = {
                "Player " + (currentPlayer == PLAYER1 ? "1" : "2") +" ("+currentPlayer +") turn, Enter column (0-" + (COLS - 1) + '
                textField
        };
        int option = JOptionPane.showOptionDialog( parentComponent: null, message, title: "Enter Column", JOptionPane.OK_CANCEL_OPTION,
                JOptionPane.PLAIN_MESSAGE, icon: null, options: null, initialValue: null);

        if (option == JOptionPane.OK_OPTION) {
            try {
                int col = Integer.parseInt(textField.getText().trim()); // Parse column input
                if (dropPiece(col, currentPlayer)) { // Try to drop piece in chosen column
                    updateBoard(); // Update GUI
                } else {
                    continue; // Invalid move, continue loop
                }
            } catch (NumberFormatException e) {
                JOptionPane.showMessageDialog(frame, message: "Invalid input. Please enter a valid column number.");
                continue; // Input parsing error, continue loop
            }
        } else {
            System.exit( status: 0); // Exit the program if user clicked cancel or close the dialog
        }

        // Check win or draw condition after each move
        if (checkWin(currentPlayer)) {

            // Win message and reset game state
            JOptionPane.showMessageDialog(frame, message: "Player " + (player1Turn ? "1" : "2") + " wins!");
            result = player1Turn ? 1 : 2;
            gameEnd = true;
        } else if (isBoardFull()) {

            // Draw message and reset game state
            JOptionPane.showMessageDialog(frame, message: "It's a draw!");
            result = 0;
            gameEnd = true;
        } else {
            player1Turn = !player1Turn; // Switch turns
        }
    }
}
```

Method to start the game for player vs player, asking for the user input for the columns and error handling for the input. If input is valid, update board and check win or draw after the move is made, if any of those are true, game is over and get result value, else switch turns

```java
// Method for player vs AI
1 usage
public void playGameAgainstAI() {

    // Reset game state to start a new game
    resetGame();
    boolean gameEnd = false;

    while (!gameEnd) {
        char currentPlayer = (player1Turn) ? PLAYER1 : PLAYER2; // Determine current player

        if (player1Turn) {
            System.out.println("Player's turn (symbol: " + currentPlayer + ")");
            String input = JOptionPane.showInputDialog("Enter column (0-" + (COLS - 1) + "):"); // Ask for input
            try {
                int col = Integer.parseInt(input); // Parse column input
                if (dropPiece(col, currentPlayer)) { // Try to drop piece in chosen column
                    updateBoard(); // Update GUI
                } else {
                    continue; // Invalid move, continue loop
                }
            } catch (NumberFormatException e) {
                JOptionPane.showMessageDialog(frame, message: "Invalid input. Please enter a valid column number.");
                continue; // Input parsing error, continue loop
            }
        } else {
            int col = random.nextInt(COLS); // AI chooses a random column
            while (!dropPiece(col, currentPlayer)) { // AI tries to drop piece in chosen column
                col = random.nextInt(COLS); // Retry if column is full
            }
        } else {
            int col = random.nextInt(COLS); // AI chooses a random column
            while (!dropPiece(col, currentPlayer)) { // AI tries to drop piece in chosen column
                col = random.nextInt(COLS); // Retry if column is full
            }
            updateBoard(); // Update GUI after AI moves
        }

        // Check win or draw condition after each move
        if (checkWin(currentPlayer)) {

            // Win message and reset game state
            JOptionPane.showMessageDialog(frame, message: (player1Turn ? "Player" : "AI") + " wins!");
            result = player1Turn ? 1 : 2;
            gameEnd = true;
        } else if (isBoardFull()) {

            // Draw message and reset game state
            JOptionPane.showMessageDialog(frame, message: "It's a draw!");
            result = 0;
            gameEnd = true;
        } else {
            player1Turn = !player1Turn; // Switch turns
        }
    }
}
}
```

Method to start the game for player vs AI, asking for the user input for the columns and error handling for the input. If input is valid, update board and check win or draw after the move is made, if any of those are true, game is over and get result value, else switch turns. If it is AI's turn, AI chooses a

random column and drop piece in the chosen column, retries if column is full and update board after the move is successfully made.

Number Game

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Random;

2 usages
public class NumberGame extends Game {

    // GUI components
    31 usages
    private JFrame frame;
    7 usages
    private JPanel panel;
    2 usages
    private JLabel label1;
    5 usages
    private JButton roll;
    14 usages
    private JLabel topLeftLabel;
    20 usages
    private JLabel topRightLabel;

    // Mode flag and player's value
    7 usages
    private boolean isPvP;
    30 usages
    private int player1Value;
    18 usages
    private int player2Value;
    21 usages
    public int pgAI;
```

Import, initializes and declare components for the game

```java
// Constructor to initialize GUI
1 usage
public NumberGame() { initializeUI(); }

// Method to set up the GUI
1 usage
private void initializeUI() {

    // Create main frame
    frame = new JFrame( title: "Number Game");
    panel = new JPanel(new BorderLayout());

    // Display welcome message
    JOptionPane.showMessageDialog(frame,  message: "You got Number Game!");

    // Create and set up the top panel with player labels
    JPanel topPanel = new JPanel(new BorderLayout());
    topLeftLabel = new JLabel( text: "Player 1: 0");
    topRightLabel = new JLabel( text: "Player 2: 0", SwingConstants.RIGHT);
    topPanel.add(topLeftLabel, BorderLayout.WEST);
    topPanel.add(topRightLabel, BorderLayout.EAST);

    // Content label and button
    label1 = new JLabel( text: "Welcome to the Number game!", SwingConstants.CENTER);
    roll = new JButton( text: "Roll");

    // Add action listener to the roll button
    roll.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            startGame(); // Start the game logic when roll button is clicked
        }
```

```java
    // Set up main panel layout and components
    panel.setBorder(BorderFactory.createEmptyBorder( top: 20,  left: 20,  bottom: 20,  right: 20));
    panel.setLayout(new BorderLayout());
    panel.add(topPanel, BorderLayout.PAGE_START); // Add top panel to the main panel
    panel.add(label1, BorderLayout.CENTER);
    panel.add(roll, BorderLayout.SOUTH);

    // Add the main panel to the frame
    frame.add(panel);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setPreferredSize(new Dimension( width: 800,  height: 600));
    frame.pack();
    frame.setVisible(true);
}
```

Method to call initializeUI method. InitializeUI method to create the mainframe, display welcome message and set up the text and button in the frame. Action

listener to make the button work and setting up the mainframe, adding the main panel to the frame

```
// Method to set game mode
2 usages
public void setGameMode(int mode) {
    if (mode == 1) {
        // Player vs Player
        isPvP = true;
        topRightLabel.setText("Player 2: 0");
    } else {
        // Player vs AI
        isPvP = false;
        topRightLabel.setText("AI: 0"); // Update the label to "AI" for AI mode
    }
}

// Method to start game
3 usages
public void startGame() {
    Random random = new Random();

    // Decide player vs player or player vs AI
    if (isPvP) {
        playTwoPlayers(random);
    } else {
        playVsAI(random);
    }

    // Update the UI after game is finished
    updateUIAfterGame();
}
```

Method to set game mode, player vs player or player vs AI and method to start game depending on the choice.

```java
// Method to play a game between two players
1 usage
public void playTwoPlayers(Random random) {

    // Loop counter for player turn and variable to store the random number added to the player's number
    int loop = 0;
    int pg4 = 0;

    // Player 1's turn to get an initial number
    player1Value = random.nextInt( bound: 21); // Generate a random number between 0 and 20

    // Update value and display what number did player 1 get
    updatePlayerValues();
    JOptionPane.showMessageDialog(frame,  message: "Player 1 got "+player1Value+ "!");

    // Player 1's turn to make guesses and add to their number
    while (loop < 2) {
        // Ask player 1 to add guess or not and display remaining chance to add
        int response = JOptionPane.showConfirmDialog(frame,  message: "Player 1: Add a guess?\nRemaining chance to add: "+(2-loop),

        if (response == JOptionPane.YES_OPTION) { // Display and add number guess to player 1 value
            pg4 = getPlayerGuess();
            player1Value += pg4;
            updatePlayerValues();
            JOptionPane.showMessageDialog(frame,  message: "Player 1 got "+pg4+ "!\nTotal: "+player1Value);

        } else if (response == JOptionPane.NO_OPTION) { // Exit if player 1 don't want to add guesses
            break;
        } else { // Error handling
            JOptionPane.showMessageDialog(frame,  message: "Invalid input, try again");
```

```java
    loop++;
}

// Reset loop counter and start player 2's turn
loop = 0;

// Player 2's turn to get an initial number
player2Value = random.nextInt( bound: 21); // Generate a random number between 0 and 20

// Update value and display what number did player 1 get
updatePlayerValues();
JOptionPane.showMessageDialog(frame,  message: "Player 2 got "+player2Value+ "!");

// Player 2's turn to make guesses and add to their number
while (loop < 2) {
    // Ask player 2 to add guess or not and display remaining chance to add
    int response = JOptionPane.showConfirmDialog(frame,  message: "Player 2: Add a guess?\nRemaining chance to add: "+(2-loop),

    if (response == JOptionPane.YES_OPTION) { // Display and add number guess to player 2 value
        pg4 = getPlayerGuess();
        player2Value += pg4;
        updatePlayerValues();
        JOptionPane.showMessageDialog(frame,  message: "Player 2 got "+pg4+ "!\nTotal: "+player2Value);

    } else if (response == JOptionPane.NO_OPTION) { // Exit if player 2 don't want to add guesses
        break;
    } else { // Error handling
        JOptionPane.showMessageDialog(frame,  message: "Invalid input, try again");
    }
    loop++;
```

Method to start the game between two players, getting a random number for the player's value and displaying what they got. A loop counter to track the number of chance the player have to add numbers to their value and pg4 to store the additional number that is to be added to the player's value and a error handling for invalid input. Reset loop counter for player 2

```
if ((player1Value <= 21) && (player2Value <= 21)) { // Both player's value is within a valid range
    if (player1Value > player2Value) {
        topLeftLabel.setText("Player 1 wins with " + player1Value);
        topRightLabel.setText("Player 2 loses with " + player2Value);

    } else if (player2Value > player1Value) {
        topLeftLabel.setText("Player 1 loses with " + player1Value);
        topRightLabel.setText("Player 2 wins with " + player2Value);

    } else {
        topLeftLabel.setText("Draw!");
        topRightLabel.setText("Draw!");
        frame.dispose();
        result = 0;
        resetGame();
    }

} else if (player1Value > 21 && player2Value <= 21) { // Player 1 exceed range and player 2 within a valid range
    topLeftLabel.setText("Player 1 loses with " + player1Value);
    topRightLabel.setText("Player 2 wins with " + player2Value);

} else if (player1Value <= 21 && player2Value > 21) { // Player 2 exceed range and player 1 within a valid range
    topLeftLabel.setText("Player 1 wins with " + player1Value);
    topRightLabel.setText("Player 2 loses with " + player2Value);

} else { // Both player exceed valid range
    topLeftLabel.setText("Both players lose (Draw)");
    topRightLabel.setText("Both players lose (Draw)");
    result = 0;
    frame.dispose();
    resetGame();
```

Check players final value after getting additional number or not, if both within valid range, the one with greater value wins or if both are the same, draw. If one of them exceed limit, the other wins but if both exceed limit, both player loses which is a draw. Close frame after game is over and restart game if the result is a draw.

```java
// Method to play a game against AI
1 usage
public void playVsAI(Random random) {

    // Loop counter for player turn and variable to store the random number added to the player's number
    int loop = 0;
    int pg4 = 0;

    // Player 1's turn to get an initial number
    player1Value = random.nextInt( bound: 21); // Generate a random number between 0 and 20

    // Update value and display what number did player 1 get
    updatePlayerValues();
    JOptionPane.showMessageDialog(frame, message: "Player 1 got "+player1Value+ "!");

    // Player 1's turn to make guesses and add to their number
    while (loop < 2) {

        // Ask player 1 to add guess or not and display remaining chance to add
        int response = JOptionPane.showConfirmDialog(frame, message: "Player 1: Add a guess?\nRemaining chance to add: "+(2-loop),

        if (response == JOptionPane.YES_OPTION) { // Display and add number guess to player 1 value
            pg4 = getPlayerGuess();
            player1Value += pg4;
            updatePlayerValues();
            JOptionPane.showMessageDialog(frame, message: "Player 1 got "+pg4+ "!\nTotal: "+player1Value);

        } else if (response == JOptionPane.NO_OPTION) { // Exit if player 1 don't want to add guesses
            break;
        } else { // Error handling
            JOptionPane.showMessageDialog(frame, message: "Invalid input, try again");
```

Method to start the game between player and AI, getting a random number for the player's value and displaying what they got. A loop counter to track the number of chance the player have to add numbers to their value and pg4 to store the additional number that is to be added to the player's value and a error handling for invalid input

```java
loop = 0;

pgAI = random.nextInt( bound: 21); // Generate a random number between 0 and 20

// Update value and display what number did AI get
topRightLabel.setText("AI: " + pgAI);
JOptionPane.showMessageDialog(frame, message: "AI got "+pgAI+ "!");

// AI's turn to make guesses and add to their number
while (loop < 2 && pgAI <= 21) { // While AI still have chance to add number and number within valid range
    int pg3 = random.nextInt( bound: 3); // Randomly chooses addition range
    switch (pg3) {
        case 0:
            pg4 = random.nextInt( bound: 4) + 1;
            pgAI += pg4;
            break;
        case 1:
            pg4 = random.nextInt( bound: 3) + 5;
            pgAI += pg4;
            break;
        case 2:
            pg4 = random.nextInt( bound: 3) + 8;
            pgAI += pg4;
            break;
    }
```

Reset loop for AI turn, generate random number 1-21 for AI and if AI still have the chance to add a number to their value and their value have not exceeded the limit, add the number to the value.

```
    topRightLabel.setText("AI: " + pgAI);
    // Handling for AI number to not exceed valid range
    if ((pgAI)<21) { // If total AI number is within valid range
        JOptionPane.showMessageDialog(frame, message: "AI got " + pg4 + "!\nTotal: " + pgAI);
    }else { // If not within valid range, AI's number - guess number and display AI got 0 as the guess number
        pgAI -= pg4;
        JOptionPane.showMessageDialog(frame, message: "AI got 0" + "!\nTotal: " + pgAI);
    }
    loop++;
}

// Determine the winner or a draw
if ((player1Value <= 21) && (pgAI <= 21)) { // Both player's value is within a valid range
    if (player1Value > pgAI) {
        topLeftLabel.setText("Player 1 wins with " + player1Value);
        topRightLabel.setText("AI loses with " + pgAI);

    } else if (pgAI > player1Value) {
        topLeftLabel.setText("Player 1 loses with " + player1Value);
        topRightLabel.setText("AI wins with " + pgAI);
    }
    else{ // Both player and AI's value are equal, reset game
        frame.dispose();
        result = 0;
        resetGame();
    }
} else if (player1Value > 21 && pgAI <= 21) { // Player's value exceed valid range
    topLeftLabel.setText("Player 1 loses with " + player1Value);
    topRightLabel.setText("AI wins with " + pgAI);
}
```

Displays what number did the AI got and if the total value of AI exceeded the limit, change the additional number to 0, preventing the AI value from exceeding the limit. Check players final value after getting an additional number or not, if the player is within valid range, the one with greater value wins or if both are the same, draw. If player value exceeds the limit, AI wins. Close frame after game is over and restart the game if the result is a draw.

```java
// Method to get player guess from input using GUI
3 usages
private int getPlayerGuess() {

    // Options for addition range
    Object[] options = {"Add 1 to 4", "Add 5 to 7", "Add 8 to 10"};
    int choice = JOptionPane.showOptionDialog(frame, message: "Choose how to add the number:", title: "Choose Addition Range",
            JOptionPane.DEFAULT_OPTION, JOptionPane.INFORMATION_MESSAGE, icon: null, options, options[0]);

    Random random = new Random();

    // Variable to store the random number that is added to the players value
    int pg4 = 0;
    switch (choice) { // Random to get number within addition specified range
        case 0:
            pg4 = random.nextInt( bound: 4) + 1;
            break;
        case 1:
            pg4 = random.nextInt( bound: 3) + 5;
            break;
        case 2:
            pg4 = random.nextInt( bound: 3) + 8;
            break;
    }

    return pg4;
}
```

Method for adding an additional number to the players value, asking the user to choose the range of the additional number, 1-4, 5-7, 8-10.

```java
// Update the labels with the current values of the players
6 usages
private void updatePlayerValues() {
    topLeftLabel.setText("Player 1: " + player1Value);
    if (isPvP) {
        topRightLabel.setText("Player 2: " + player2Value);
    } else {
        topRightLabel.setText("AI: " + player2Value);
    }
}
```

Updates the players value on top of the frame

```java
// Update UI after game ends
1 usage
private void updateUIAfterGame() {
    // Disable the roll button after game ends
    roll.setEnabled(false);

    // Show a dialog or update label for game result
    String resultMessage;

    // Get result and close frame if any player wins
    if (topLeftLabel.getText().contains("wins")) {

        // Player 1 wins
        if (isPvP) { // Player 1 wins in Player vs Player mode
            result = 1;
            resultMessage = ("Player 1 wins with " + player1Value + " over Player 2 with "+player2Value);
            frame.dispose();
        } else{ // Player 1 wins in Player vs AI mode
            result = 1;
            resultMessage = ("Player 1 wins with " + player1Value + " over AI with "+pgAI);
            frame.dispose();
        }

        // Player 1 loses
    } else if (topRightLabel.getText().contains("wins")) {
        if (isPvP) { // PLayer 1 loses in Player vs Player mode
            result = 2;
            resultMessage = ("Player 2 wins with " + player2Value + " over Player 1 with "+player1Value);
            frame.dispose();
        }
```

```java
        else { // Player 1 loses in Player vs AI mode
            result = 2;
            resultMessage = ("AI wins with " + pgAI + " over Player 1 with "+player1Value);
            frame.dispose();
        }
    } else { // Draw and reset game
        result = 0;
        resultMessage = "It's a draw!";
        resetGame();
    }

    JOptionPane.showMessageDialog(frame, resultMessage, title: "Game Result", JOptionPane.INFORMATION_MESSAGE);
}
```

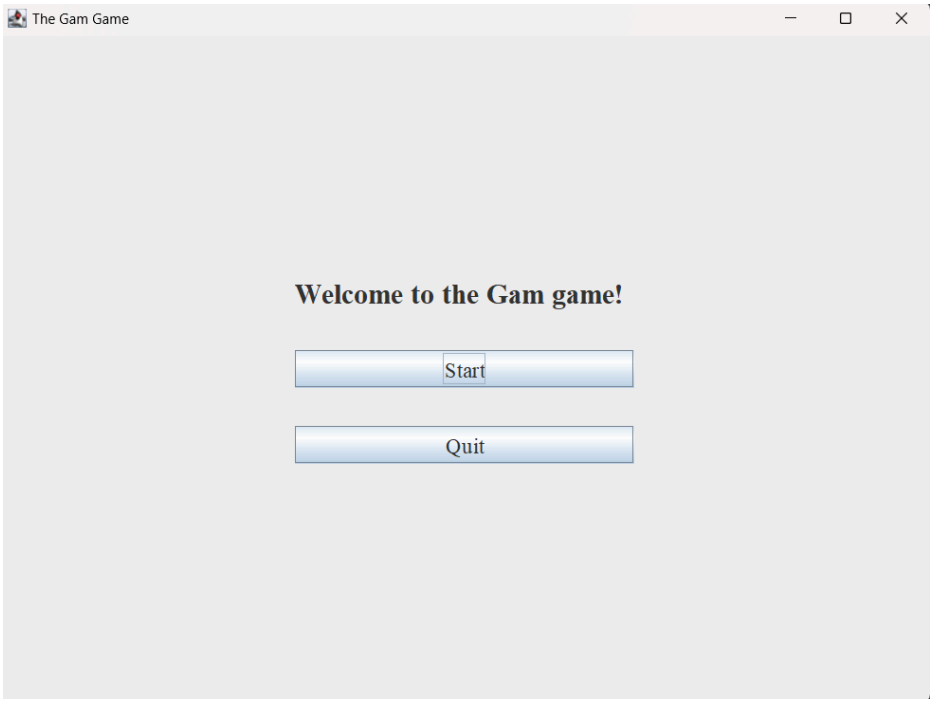
Method to show results after game is over, showing who won and close the frame or it’s a draw and restart the game

```java
// Method to reset game and player's value
4 usages
private void resetGame() {
    roll.setEnabled(true);
    player1Value = 0;
    player2Value = 0;
    topLeftLabel.setText("Player 1: 0");
    if (isPvP) {
        topRightLabel.setText("Player 2: 0");
    } else {
        topRightLabel.setText("AI: 0");
    }
}
```

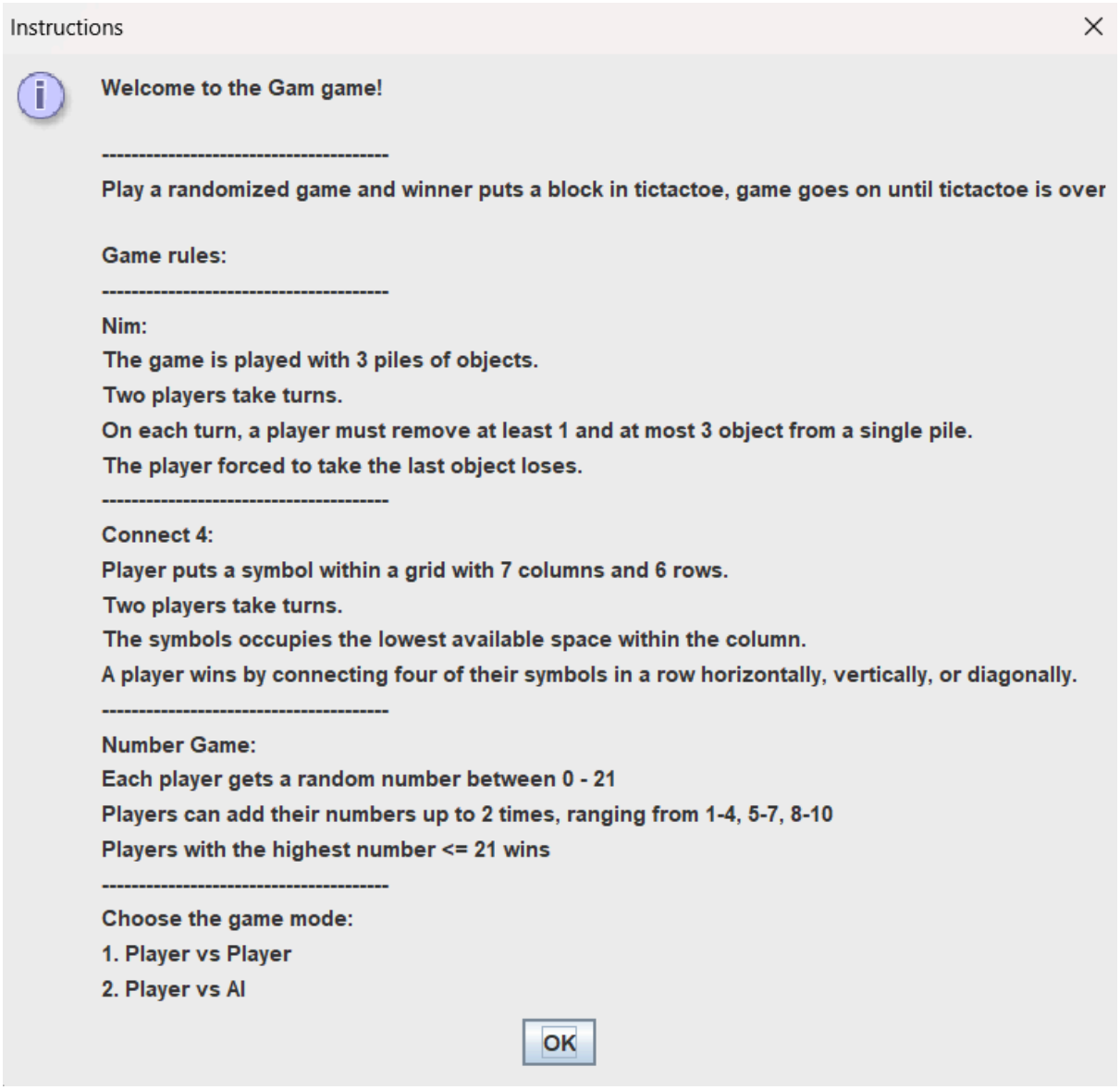
Method to restart the game, resetting the players and AI value

D. Evidence of working program

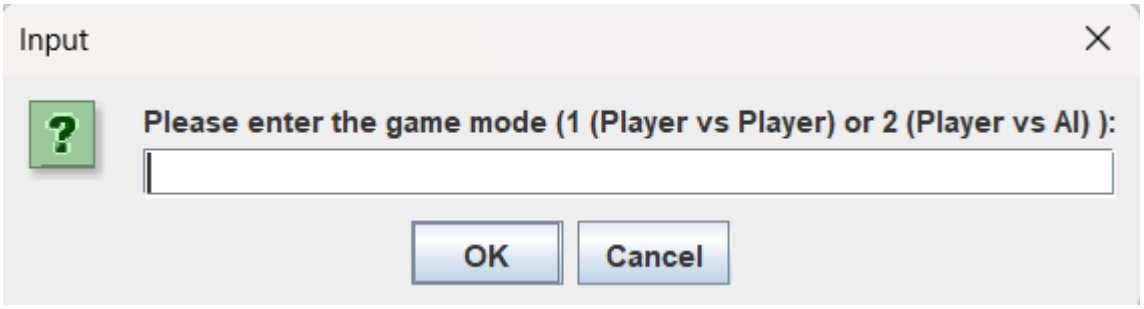
Menu



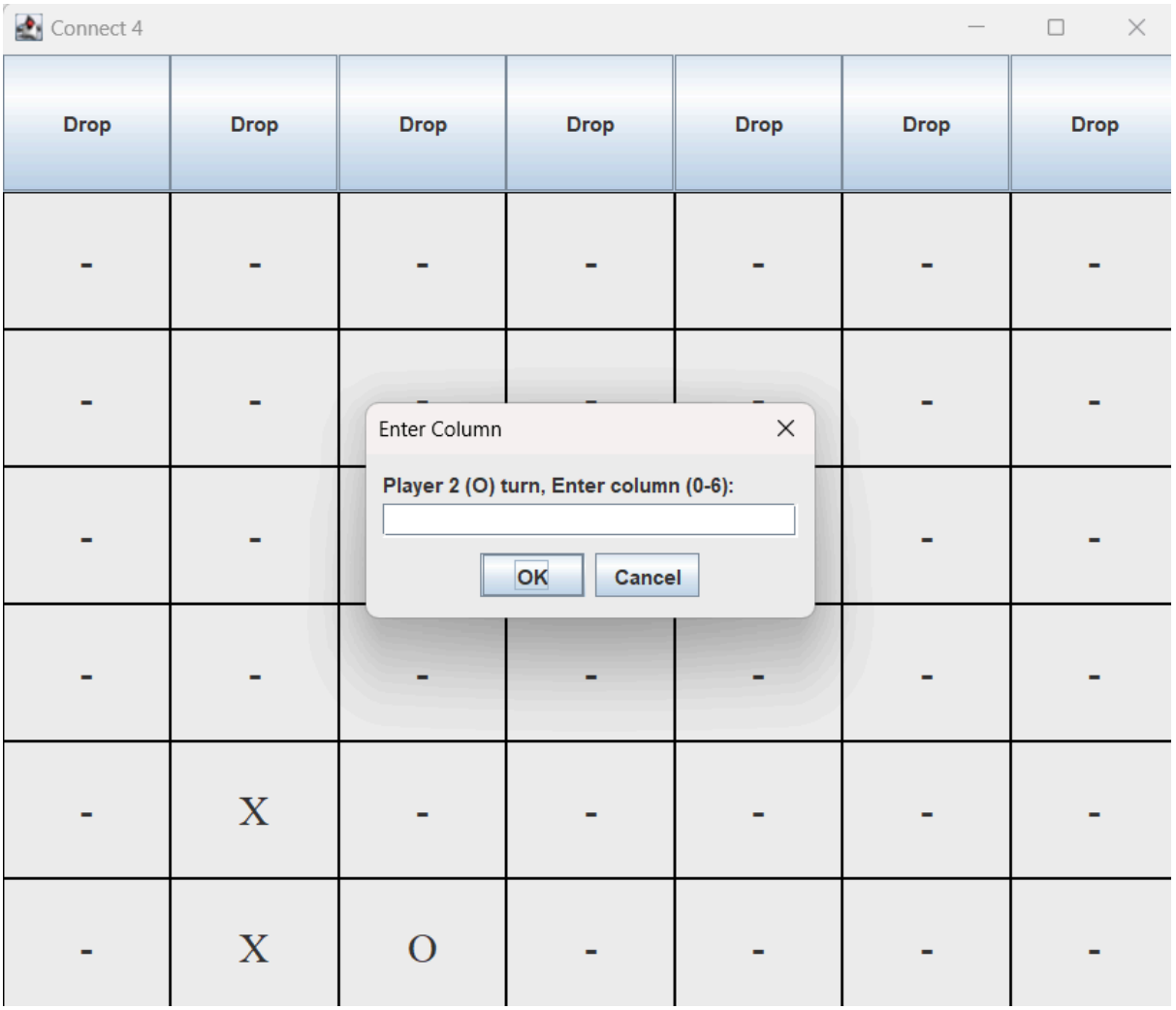
Instructions



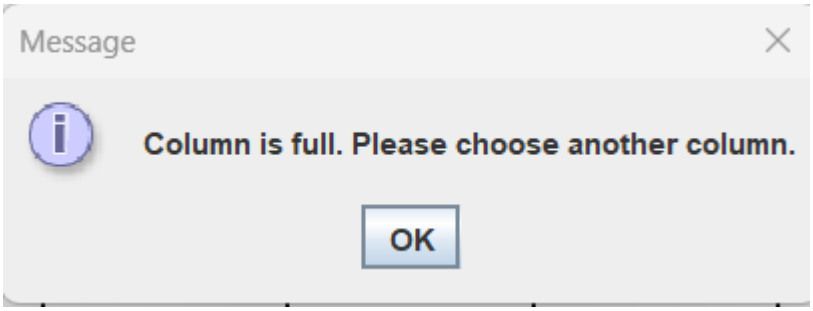
Determine game mode



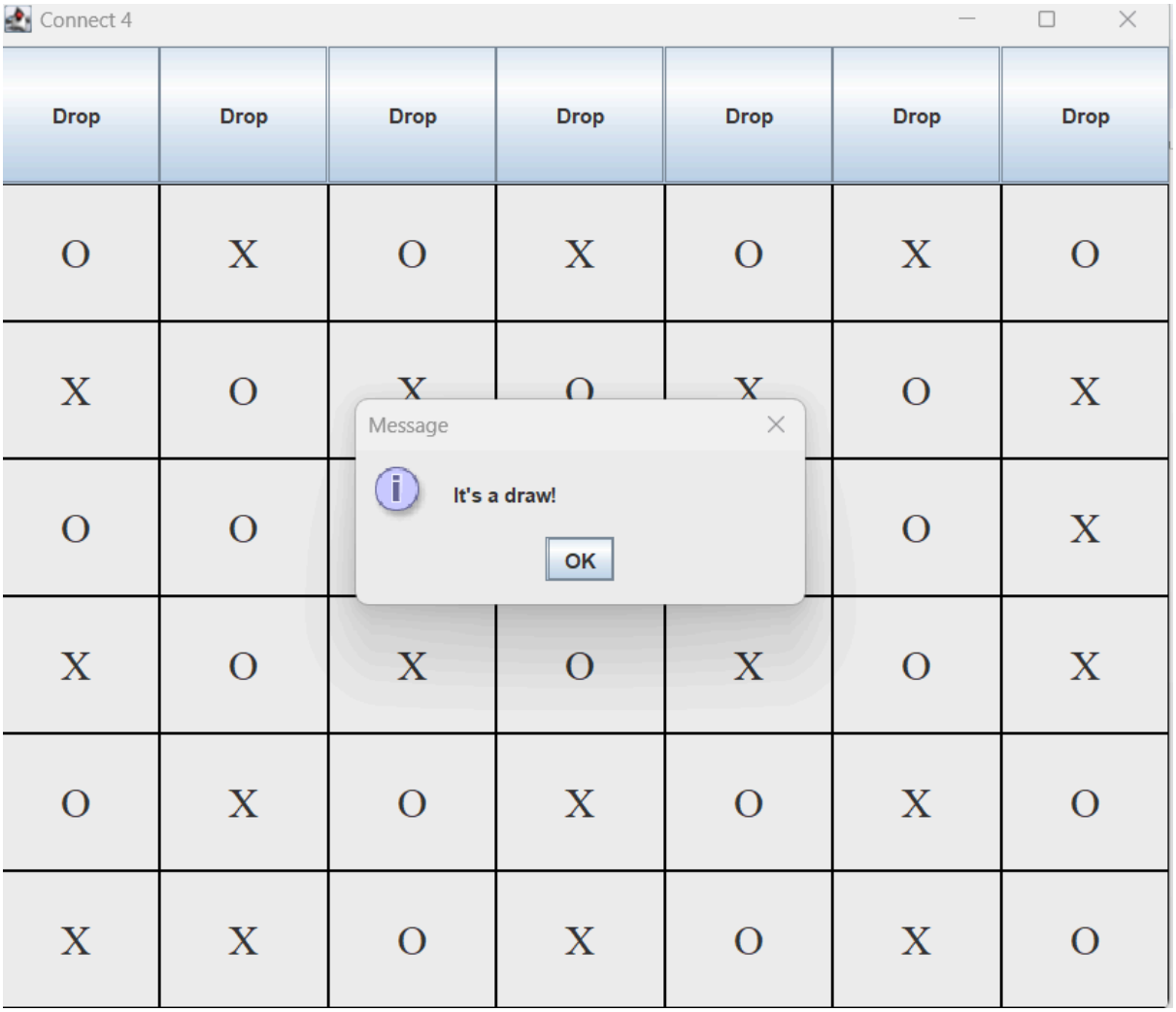
Connect 4



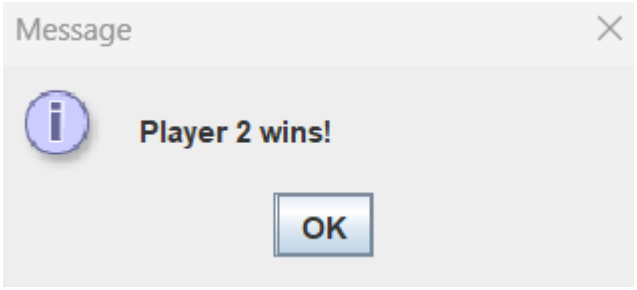
Column is full



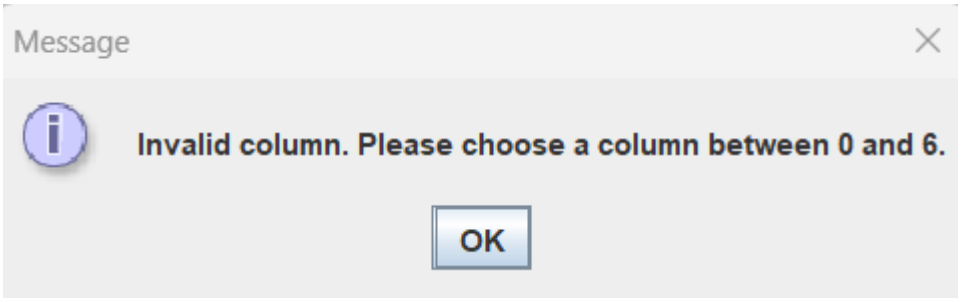
Board is full



Game Over



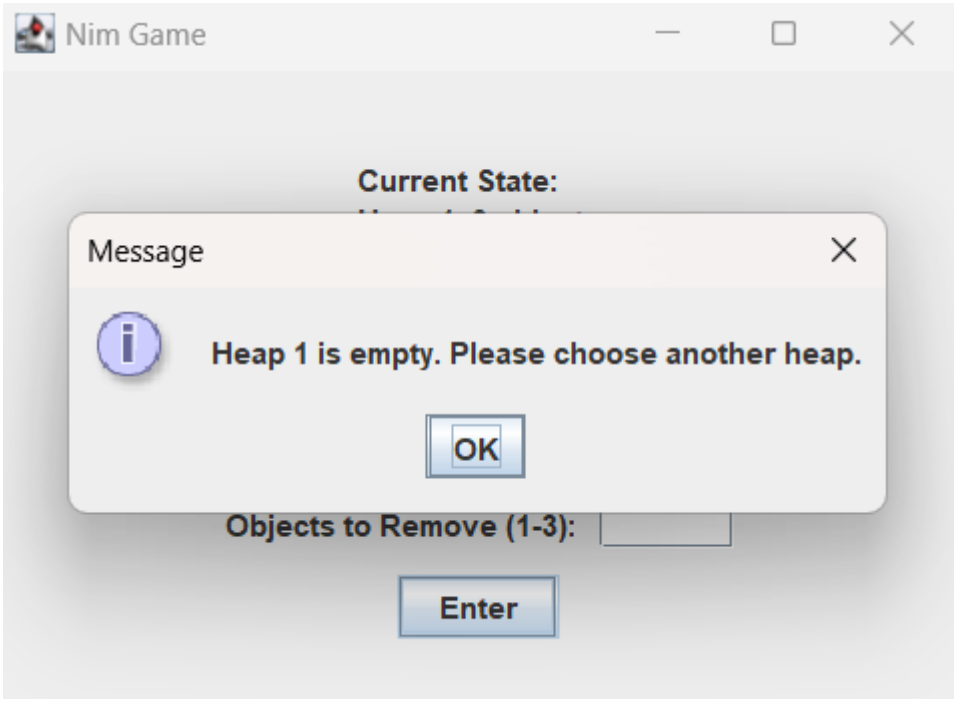
Input out of range



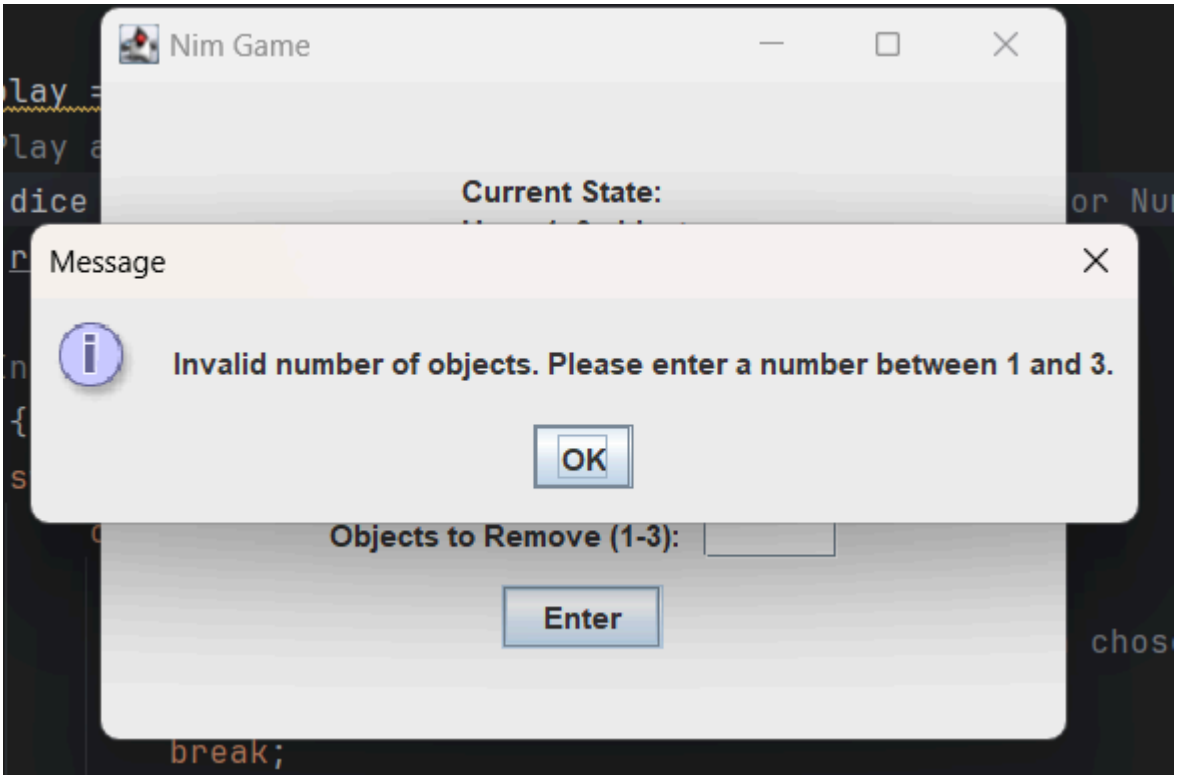
Invalid input

Nim

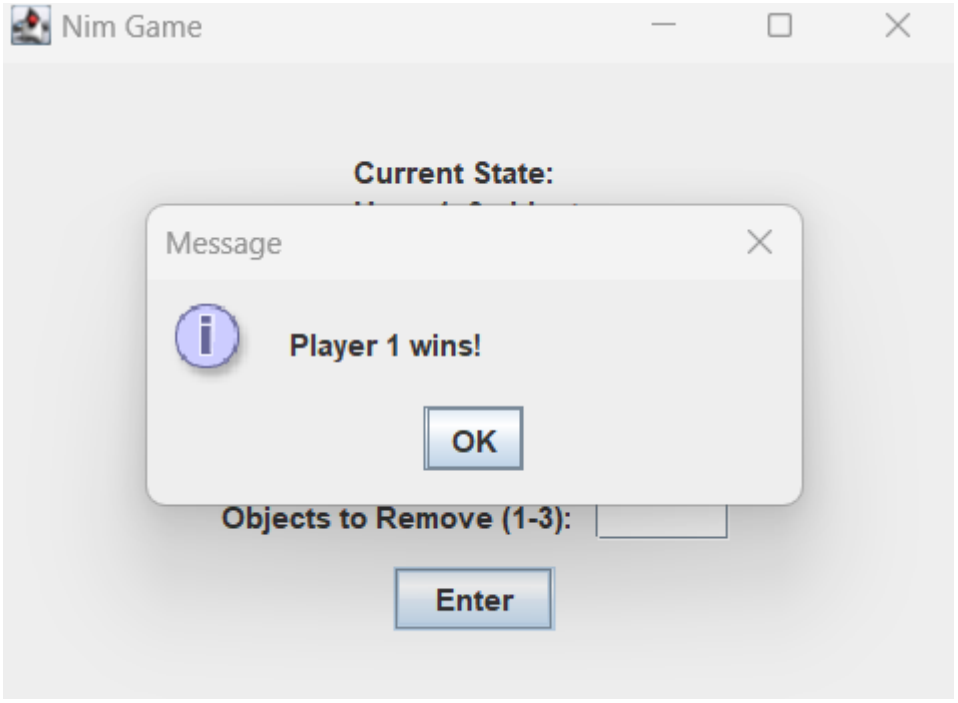
Remove object from empty heap



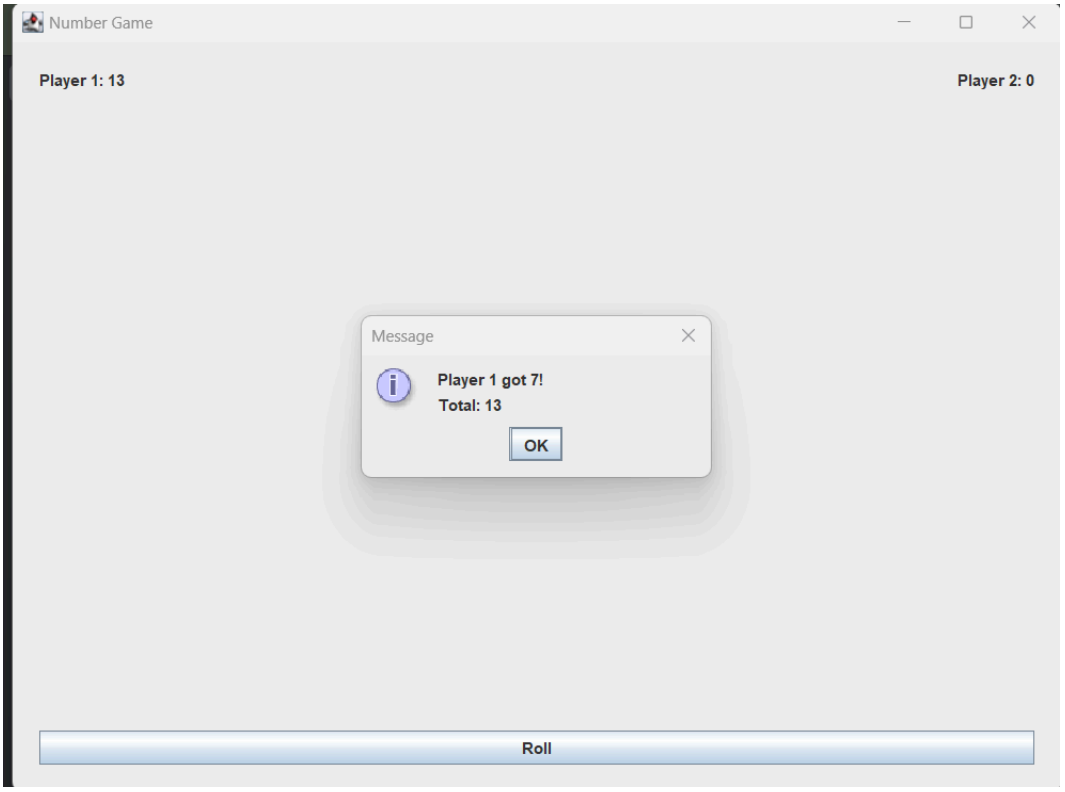
Invalid number of Object to remove



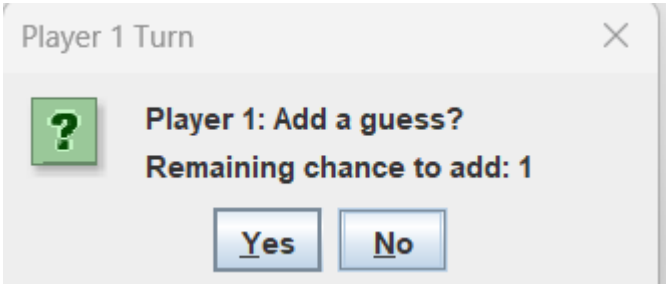
Game Over



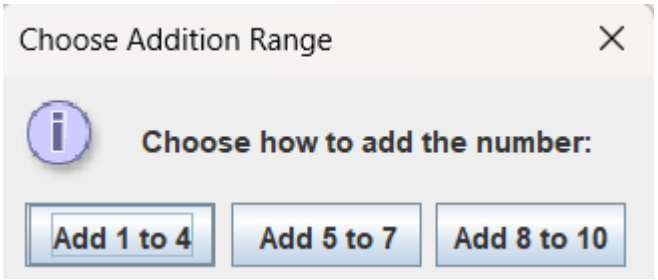
Number Game



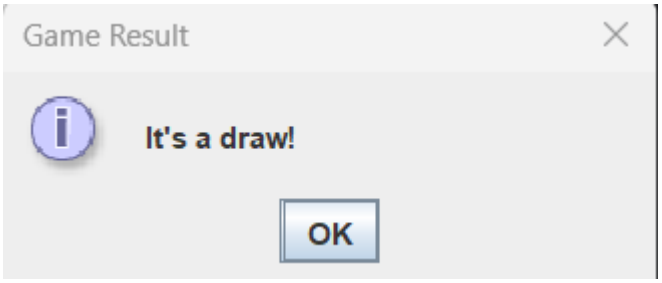
Ask the user to add guess or not



Addition range to add

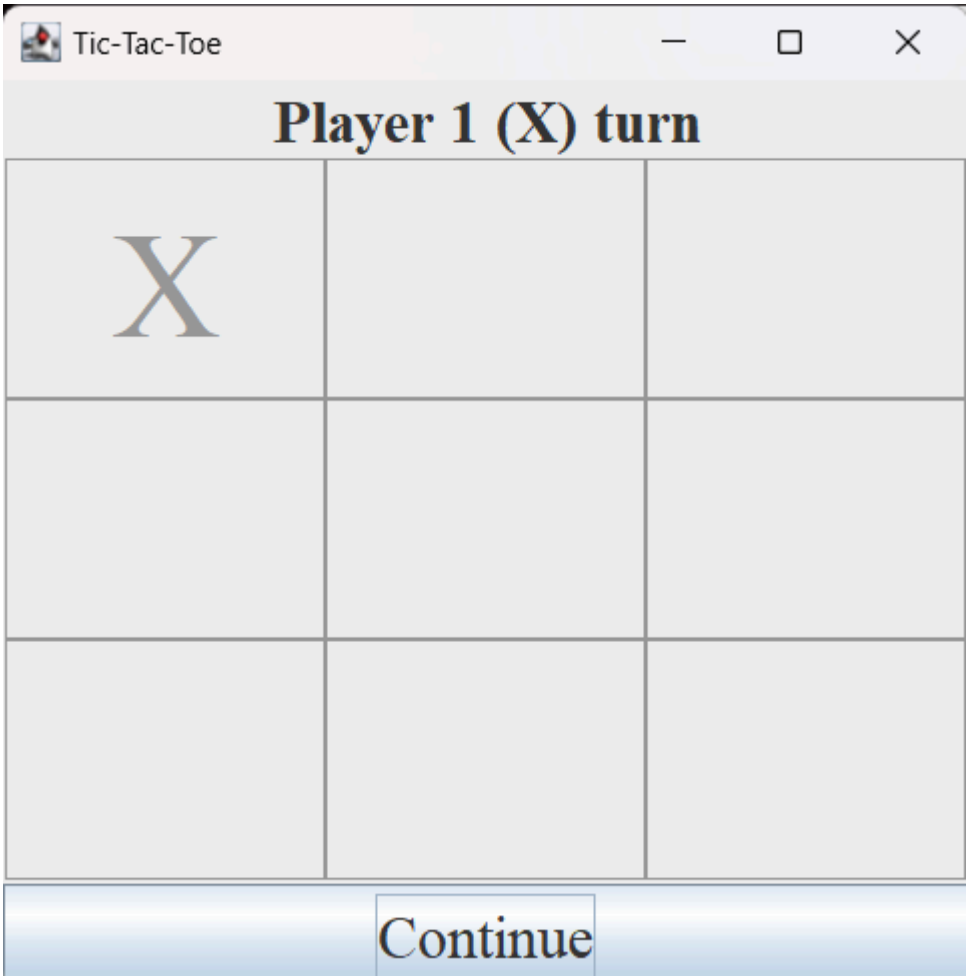


Draw

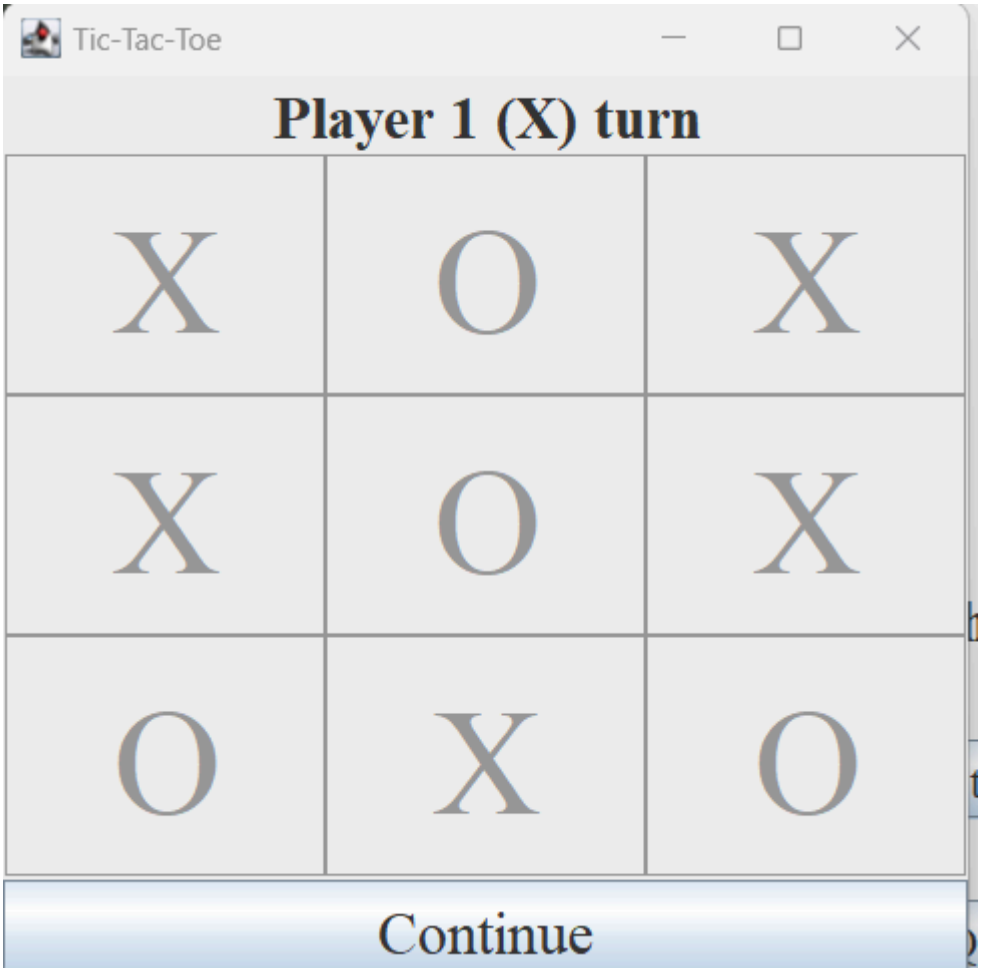


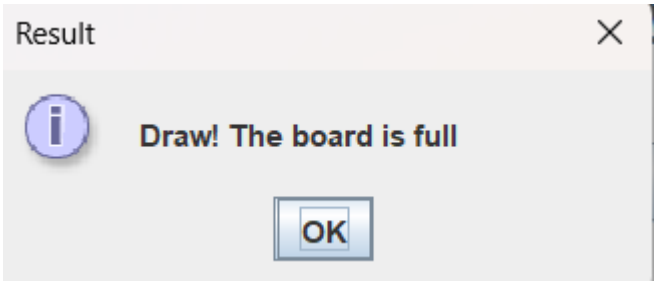
Game Over

TicTacToe

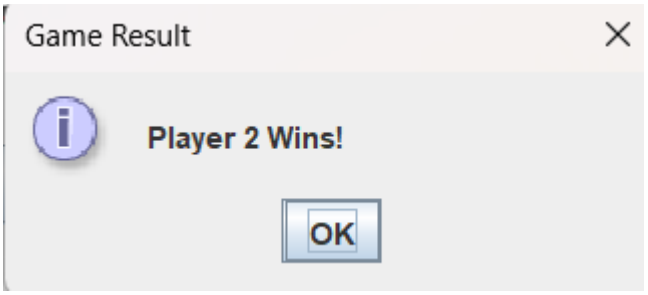


Board is full





Game Over



E. Resources

Project program files : https://github.com/gamakagami/OOP-FP