# Report On
# CONNECT 4

Submitted in partial fulfillment of the requirements of the Course project in
Semester III of Second Year Computer Science & Engineering
(Data Science)

by

Shardul Brid
Parth Vasave
Jash Mhatre
Megha Kapadne

Supervisor
Prof. Yogesh Pingle

**Vidyavardhini's College of Engineering & Technology**

**Department of Computer Science & Engineering (Data science)**

**(2023-24)**

# Vidyavardhini's College of Engineering & Technology

## Department of Computer Science & Engineering (Data science)

# CERTIFICATE

This is to certify that the project entitled “Connect 4” is a bonafide work of "Shardul Brid, Parth Vasave, Jash Mhatre, Megha Kapadne" submitted to the University of Mumbai in partial fulfillment of the requirement for the Course project in semester III of Second Year Computer Science & Engineering (Data Science).

**Supervisor**

Prof. Yogesh Pingle

Internal Examiner

External Examiner

Dr. Vikas Gupta

Head of Department

Dr. H.V. Vankudre
Principal

# Contents

## Abstract

Connect Four is a two players game which takes place on a 7x6 rectangular board placed vertically between them. One player has 21 yellow coins and the other 21 red coins. Each player can drop a coin at the top of the board in one of the seven columns; the coin falls down and fills the lower unoccupied square. Of course, a player cannot drop a coin in a certain column if it is already full (i.e., it already contains six coins).

Even if there's no rule about who begins first, we assume, as in chess, that the lighter side makes the first move. We also use the chess notation to represent a square on the board. That is, we number rows from 1 to 7 starting from the bottom and the columns from A to G starting from the leftmost.

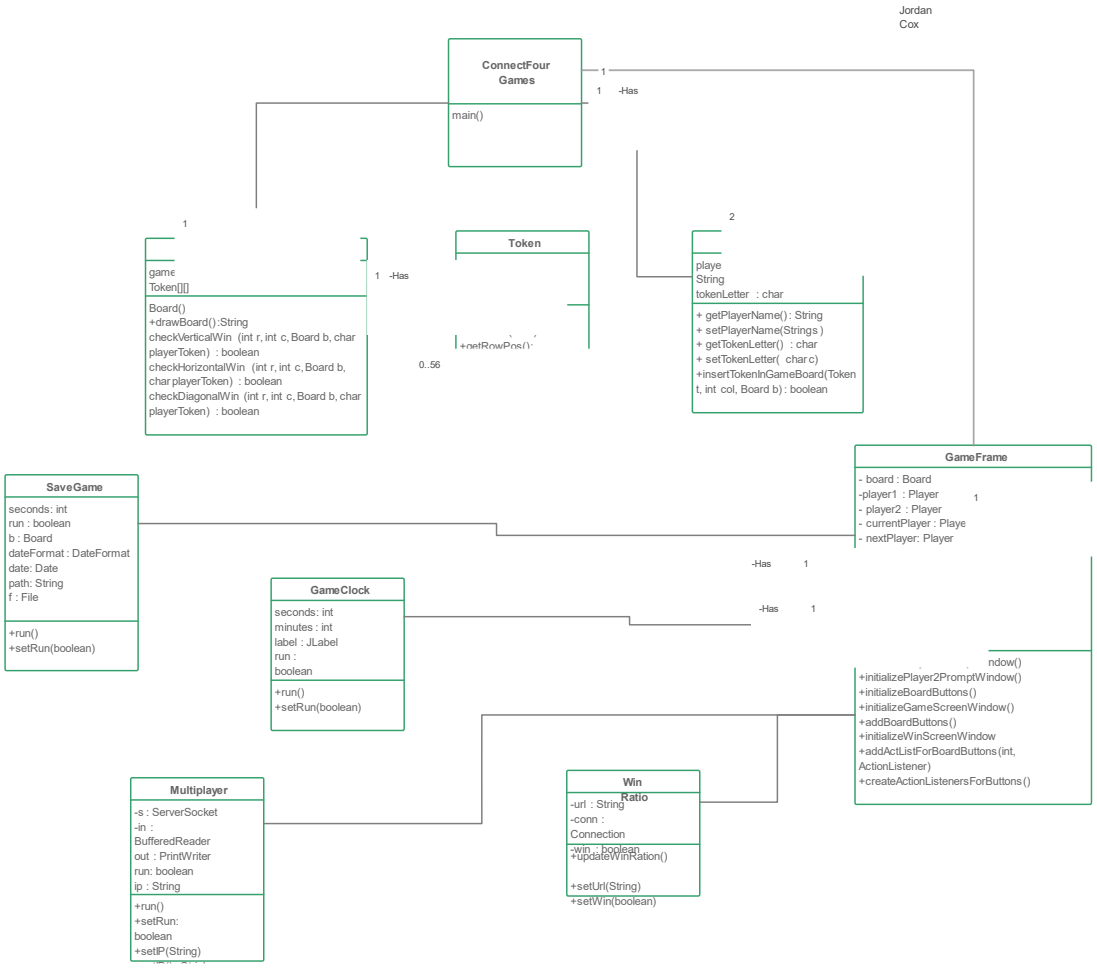
The object of the game is to connect four coins vertically, horizontally or diagonally. If the board is filled and no one has aligned four coins then the game is drawn

In the realm of classic board games, Connect4 stands as an enduring emblem of strategic engagement and intellectual rivalry. Developed and implemented in Java, this digital adaptation of Connect4 encapsulates the essence of the traditional game, where two players engage in a contest of wit and foresight to connect four of their coloured discs in a row, either horizontally, vertically, or diagonally, on a grid. The Java-based interface seamlessly combines user-friendly interactivity with the underlying complexities of the game's logic, fostering a dynamic environment for players to anticipate and counter each other's moves. As they take turns, the game unfolds as a captivating test of strategy and tactical acumen, providing an engaging and intellectually stimulating experience that upholds the timeless appeal of Connect4 in the digital age.

## Problem Statement

Design and develop a computer-based implementation of the classic game Connect4, with the objective of creating an engaging and interactive platform for players to enjoy the game. The software should provide a user-friendly graphical interface, allowing two players to take turns and strategically place their colored discs on a 6x7 grid until one player successfully connects four of their discs in a row, either horizontally, vertically, or diagonally, or until the grid is full, resulting in a draw. The program should incorporate efficient algorithms to manage game logic, including checking for win conditions, preventing illegal moves, and offering players clear feedback on game progress. Additionally, it should include options for game customization, such as adjustable grid size, player vs. computer mode with varying difficulty levels, and the ability to replay or save game sessions. The project aims to offer a robust and enjoyable digital rendition of Connect4, fostering critical thinking, strategic decision-making, and an immersive gaming experience.

# Block Diagram, Its Description and Working




Working and Description

1. Game Board: Connect4 is played on a vertical grid with 6 rows and 7 columns, initially empty. In Java, you can represent the board using a 2D array.

2. Players: Two players take turns, one using red discs, and the other using yellow discs.

3. Objective: The goal is to connect four of your own colored discs in a row either horizontally, vertically, or diagonally.

4. Player Input: In a Java implementation, players can input their moves by selecting a column to drop their disc into. You can use a graphical interface or a text-based interface to collect these inputs.

5. Game Logic: The game logic involves checking the validity of each move, ensuring discs stack in the lowest available position within the selected column, and checking for win conditions after every move.

6. Win Conditions: After each move, the Java program should check if the current player has formed a winning combination of four discs. This involves scanning the board horizontally, vertically, and diagonally for a winning pattern.

7. Game Loop The game continues until a player wins, resulting in a game over, or until the board is full, leading to a draw. In Java, you can implement a game loop to handle the flow of the game, which includes input validation, updating the board, and checking for win or draw conditions.

In a Java implementation of Connect4, we would use classes and methods to organize the code effectively, including a class for the game board, methods for validating moves, and functions for checking win conditions. Additionally, you'd likely create a user interface to provide a visual representation of the game and allow players to interact with it.

# Software And Hardware Used

## Hardware:

- Intel core i3 or above
- 256gb storage
- 4gb of RAM

## Software:

- Windows 7,8 or above
- Platform: java language

# Code

```
package APCS2019.Connect4Stuff;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;

class Connect4 extends JPanel implements ActionListener, MouseListener, MouseMotionListener {

   private static final int WIDTH, HEIGHT, widthUnit, heightUnit, boardLength, boardHeight;
   private static JFrame frame;
   private static Connect4 instance;
   private static Point p1, p2;

   public static void main(String[] args) {
      instance = new Connect4();
   }

   public Connect4() {
      setBackground(Color.WHITE);

      frame = new JFrame("Connect 4");
      frame.setBounds(50, 50, WIDTH, HEIGHT);
      frame.add(this);
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setVisible(true);

      javax.swing.Timer timer = new javax.swing.Timer(10, this);
      timer.start();

      frame.addMouseListener(this);
      frame.addMouseMotionListener(this);
   }

   static {
      int initialWidth = 1800;
      int initialHeight = 1000;
      boardLength = 7;
      boardHeight = 6;
      widthUnit = initialWidth / (boardLength + 2);
      WIDTH = widthUnit * (boardLength + 2);
      heightUnit = initialHeight / (boardHeight + 2);
      HEIGHT = heightUnit * (boardHeight + 2);
   }

   public void actionPerformed(ActionEvent e) {
```

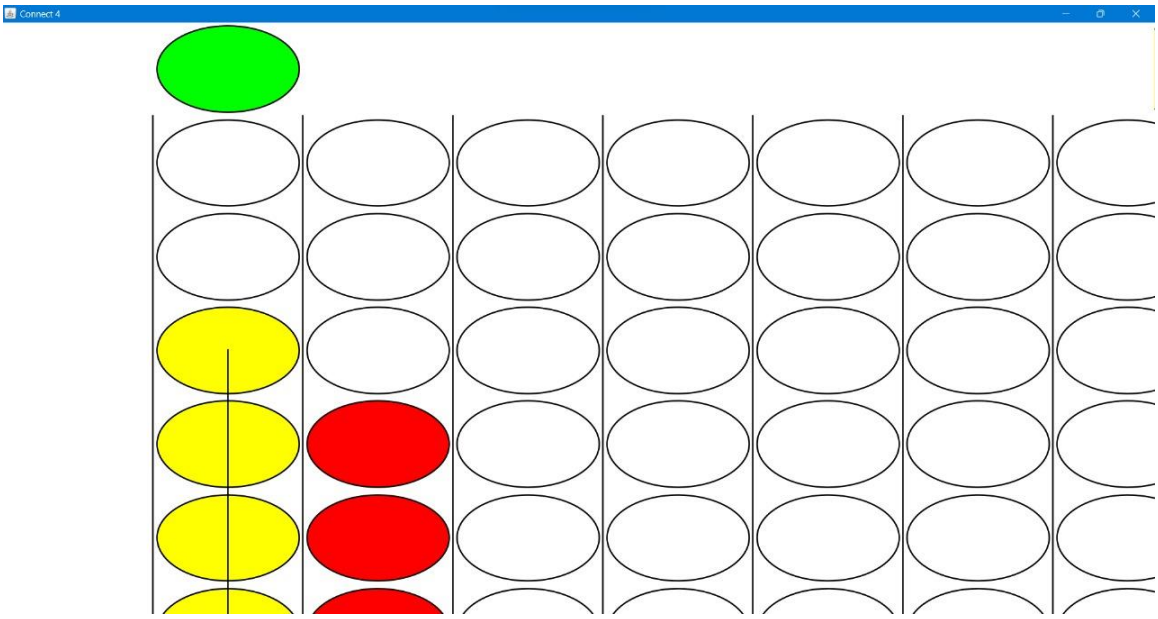
```
        repaint();
    }

    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        Board.draw(g);
    }

    public void mouseMoved(MouseEvent e) {
        Board.hover(e.getX());
    }

    public void mousePressed(MouseEvent e) {
        Board.drop();
    }

    public void mouseReleased(MouseEvent e) {}
    public void mouseClicked(MouseEvent e) {}
    public void mouseEntered(MouseEvent e) {}
    public void mouseExited(MouseEvent e) {}
    public void mouseDragged(MouseEvent e) {}

    static class PointPair {
        public Point p1, p2;

        PointPair(int x1, int y1, int x2, int y2) {
            p1 = new Point(x1, y1);
            p2 = new Point(x2, y2);
        }
    }

    static class Board {
        static Color[][] board;
        static Color[] players;
        static int turn;
        static int hoverX, hoverY;
        static boolean gameDone;

        static {
            board = new Color[boardLength][boardHeight];
            for (Color[] colors : board) {
                Arrays.fill(colors, Color.WHITE);
            }
            players = new Color[]{Color.YELLOW, Color.RED};
            turn = 0;
        }

        public static void draw(Graphics g) {
            ((Graphics2D)g).setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);
            ((Graphics2D)(g)).setStroke(new BasicStroke(2.0f));
```

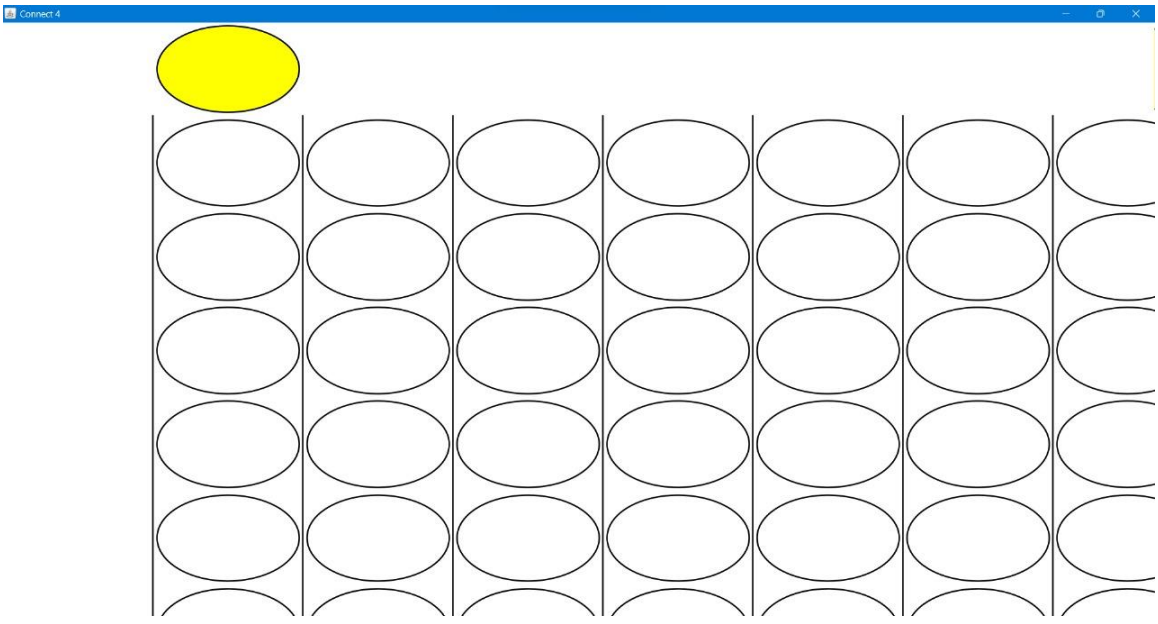
```
            for (int i = widthUnit; i <= WIDTH - widthUnit; i += widthUnit) {
                g.setColor(Color.BLACK);
                g.drawLine(i, heightUnit, i, HEIGHT - heightUnit);
                if (i == WIDTH - widthUnit) continue;
                for (int j = heightUnit; j < HEIGHT - heightUnit; j += heightUnit) {
                    g.setColor(board[i/widthUnit - 1][j/heightUnit - 1]);
                    g.fillOval(i + 5, j + 5, widthUnit - 10, heightUnit - 10);
                    g.setColor(Color.BLACK);
                    g.drawOval(i + 5, j + 5, widthUnit - 10, heightUnit - 10);
                }
            }
            g.drawLine(widthUnit, HEIGHT - heightUnit, WIDTH - widthUnit, HEIGHT -
heightUnit);

            g.setColor(gameDone ? Color.GREEN : players[turn]);
            g.fillOval(hoverX + 5, hoverY + 5, widthUnit - 10, heightUnit - 10);
            g.setColor(Color.BLACK);
            g.drawOval(hoverX + 5, hoverY + 5, widthUnit - 10, heightUnit - 10);

            g.setColor(Color.BLACK);
            if (p1 != null && p2 != null) {
                g.drawLine(p1.x, p1.y, p2.x, p2.y);
            }

        }

        public static void hover(int x) {
            x -= x%widthUnit;
            if (x < widthUnit) x = widthUnit;
            if (x >= WIDTH - widthUnit) x = WIDTH - 2*widthUnit;
            hoverX = x;
            hoverY = 0;
        }

        public static void drop() {
            if (board[hoverX/widthUnit - 1][0] != Color.WHITE) return;
            new Thread(() -> {
                Color color = players[turn];
                int x = hoverX;
                int i;
                for (i = 0; i < board[x/widthUnit - 1].length && board[x/widthUnit - 1][i] ==
Color.WHITE; i++) {
                    board[x/widthUnit - 1][i] = color;
                    try { Thread.currentThread().sleep(200); } catch(Exception ignored) {}
                    board[x/widthUnit - 1][i] = Color.WHITE;
                    if (gameDone) return;
                }
                if (gameDone) return;
                board[x/widthUnit - 1][i - 1] = color;
                checkConnect(x/widthUnit - 1, i - 1);
```

```
            }).start();
            try { Thread.currentThread().sleep(100); } catch(Exception ignored) {}
            if (gameDone) return;
            turn = (turn + 1) % players.length;
        }

        public static void checkConnect(int x, int y) {
            if (gameDone) return;

            PointPair pair = search(board, x, y);

            if (pair != null) {
                p1 = new Point((pair.p1.x + 1) * widthUnit + widthUnit / 2, (pair.p1.y + 1) *
heightUnit + heightUnit / 2);
                p2 = new Point((pair.p2.x + 1) * widthUnit + widthUnit / 2, (pair.p2.y + 1) *
heightUnit + heightUnit / 2);
                frame.removeMouseListener(instance);
                gameDone = true;
            }
        }

        public static PointPair search(Color[][] arr, int i, int j) {
            Color color = arr[i][j];
            int left, right, up, down;

            // check horizontally left to right
            left = right = i;
            while (left >= 0 && arr[left][j] == color) left--;
            left++;
            while (right < arr.length && arr[right][j] == color) right++;
            right--;
            if (right - left >= 3) {
                return new PointPair(left, j, right, j);
            }

            // check vertically top to bottom
            down = j;
            while (down < arr[i].length && arr[i][down] == color) down++;
            down--;
            if (down - j >= 3) {
                return new PointPair(i, j, i, down);
            }

            // check diagonal top left to bottom right
            left = right = i;
            up = down = j;
            while (left >= 0 && up >= 0 && arr[left][up] == color) { left--; up--; }
            left++; up++;
            while (right < arr.length && down < arr[right].length && arr[right][down] ==
color) { right++; down++; }
            right--; down--;
```

```
            if (right - left >= 3 && down - up >= 3) {
                return new PointPair(left, up, right, down);
            }

            // check diagonal top right to bottom left
            left = right = i;
            up = down = j;
            while (left >= 0 && down < arr[left].length && arr[left][down] == color) {left--;
down++;}
            left++; down--;
            while (right < arr.length && up >= 0 && arr[right][up] == color) {right++; up--;}
            right--; up++;
            if (right - left >= 3 && down - up >= 3) {
                return new PointPair(left, down, right, up);
            }

            return null;
        }

    }
}
```

# Outputs



## Conclusion

In this mini project, in conclusion, the Connect4 mini project demonstrates an engaging and interactive implementation of the classic Connect 4 game using Java and Swing. The code showcases a well-structured and efficient approach to building the game, allowing two players to take turns dropping discs into a grid while monitoring for a winning combination. The game's graphical user interface provides an enjoyable user experience, with smooth animations and visual feedback.

This project not only serves as a great example of a Java-based game development but also highlights the use of event-driven programming, graphics rendering, and creative problem-solving to create a fun and challenging gaming experience. The code structure and the organization of classes and methods make it easy to understand and modify, making it an excellent learning resource for those looking to explore game development in Java.

Overall, the Connect4 mini project demonstrates the power of Java and its libraries in creating interactive and engaging applications and serves as an excellent starting point for further development and enhancements.

## References

https://en.wikipedia.org/wiki/Connect_Four
https://www.crio.do/projects/python-multiplayer-game-connect4/
https://medium.com/analytics-vidhya/artificial-intelligence-at-play-connect-four-minimax-algorithm-explained-3b5fc32e4a4f
https://www.researchgate.net/publication/331552609_Research_on_Different_Heuristics_for_Minimax_Algorithm_Insight_from_Connect-4_Game