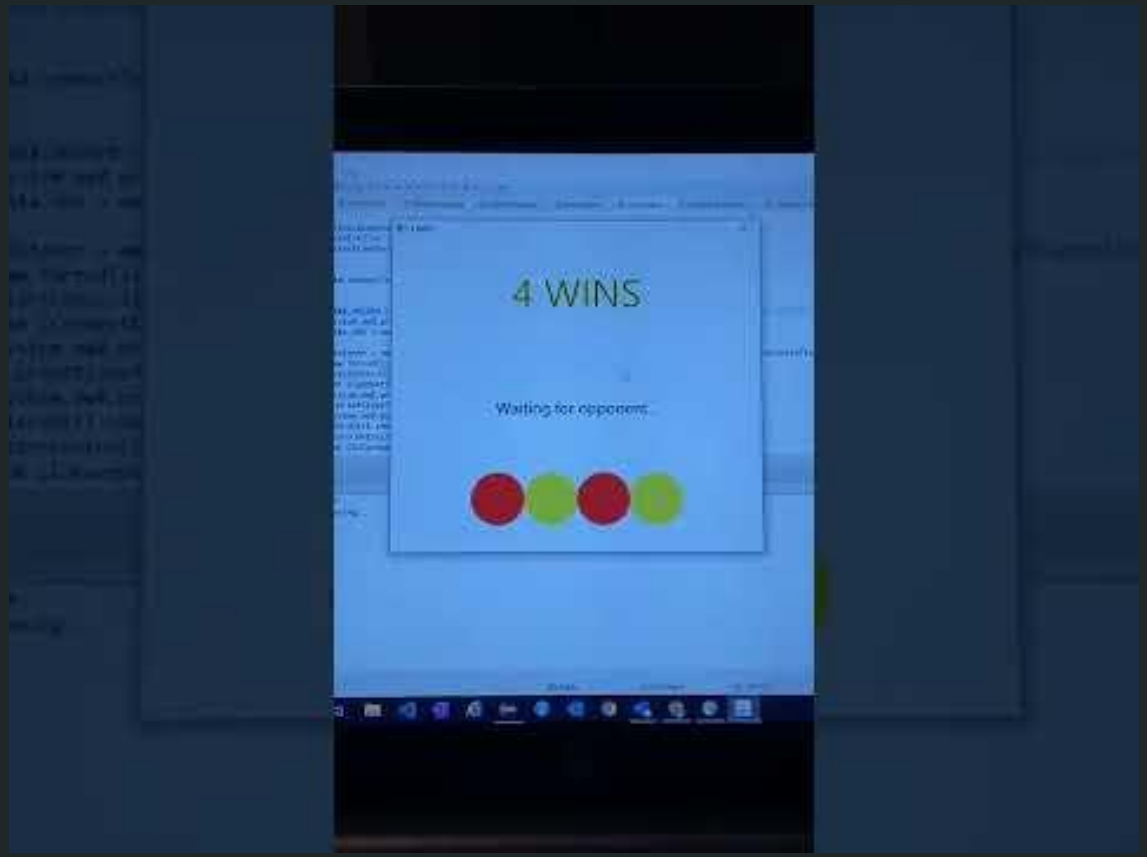
4 Gewinnt

Projekt von Furkan Akbas, Noel Thorwesten & Mohammad Azizi

Video



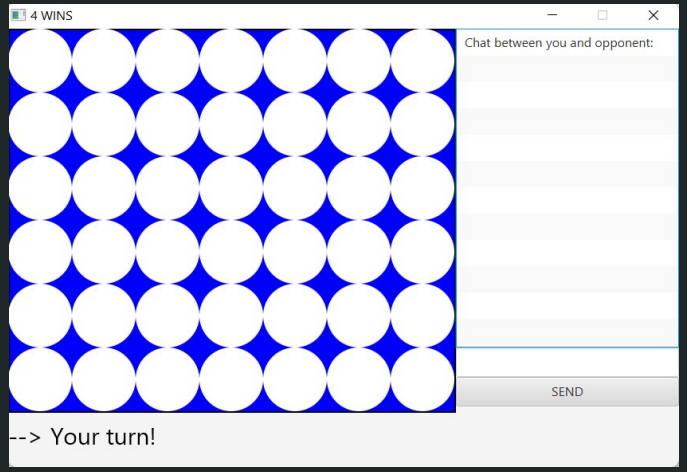
Game

StartView


```
public void showStartInterface() {
    // create play button
    button = new Button( s: "START GAME");
    button.setPrefHeight(40);
    button.setPrefWidth(160);

    hBox = new HBox();
    createCircles();
    hBox.setAlignment(Pos.CENTER);

    Text title = new Text( s: "4 WINS");
    title.setFont(Font.font( v: 60));
    title.setStrokeWidth(2);
    title.setStroke(Color.YELLOW);

    waitingText = new Text( s: "Waiting for opponent...");
    waitingText.setFont(Font.font( v: 24));
    waitingText.setVisible(false);

    VBox vBox = new VBox();
    vBox.getChildren().add(button);
    vBox.getChildren().add(waitingText);
    vBox.setAlignment(Pos.CENTER);

    // create borderpane
    BorderPane borderPane = new BorderPane();
    borderPane.setPrefWidth(560);
    borderPane.setPrefHeight(480);
    borderPane.setPadding(new Insets( v: 40));
    borderPane.setBackground(new Background(new BackgroundFill(Color.WHITE, CornerRadii.EMPTY, Insets.EMPTY)));
    borderPane.setCenter(vBox);
    borderPane.setBottom(hBox);
    borderPane.setTop(title);
    BorderPane.setAlignment(title, Pos.CENTER);

    // create scene
    Scene scene = new Scene(borderPane);
    stage.setScene(scene);
```

Game

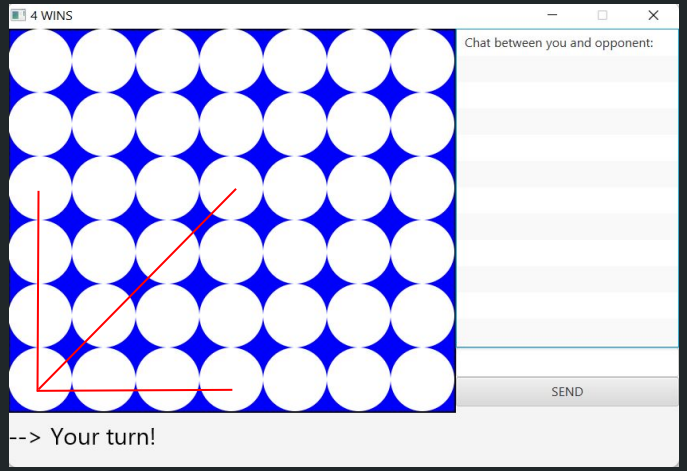
GameView




```
public void drawBoard() {
    for (int i = 0; i < 7; i++) {
        for (int j = 0; j < 6; j++) {
            // create new circles holes for empty spaces in board
            // game board has 7 columns and six rows each has height and width of 80
            Circle circle = new Circle( v: (i * 80) + 40,  v1: (j * 80) + 40,  v2: 40, Color.WHITE);

            // if current index of array has a value, fill with corresponding color
            if (board[i][j] == 1)
                circle.setFill(Color.RED);
            if (board[i][j] == 2)
                circle.setFill(Color.YELLOW);
            pane.getChildren().add(circle);
        }
    }

    // reset value of whose turn it is
    text.setText(player == 1 ? "Your turn!" : "Opponents turn!");
```

Game

Gewinnabfragen




```
public void hasPlayerWon() {
    // check if any player has won
    hasFourEqualCirclesInColumn();
    hasFourEqualCirclesInRow();
    hasFourEqualCirclesDiagonal();
    checkIfTiedGame();
}
```

Game

X,Y Tabelle der Kreise für MouseClick

(40, 40)	(120, 40)	(200, 40)	(280, 40)	(360, 40)	(440, 40)	(520, 40)
(40, 120)	(120, 120)	(200, 120)	(280, 120)	(360, 120)	(440, 120)	(520, 120)
(40, 200)	(120, 200)	(200, 200)	(280, 200)	(360, 200)	(440, 200)	(520, 200)
(40, 280)	(120, 280)	(200, 280)	(280, 280)	(360, 280)	(440, 280)	(520, 280)
(40, 360)	(120, 360)	(200, 360)	(280, 360)	(360, 360)	(440, 360)	(520, 360)
(40, 440)	(120, 440)	(200, 440)	(280, 440)	(360, 440)	(440, 440)	(520, 440)

Server

```java
public static final int SERVER_PORT = 7777;

FuziPavuzi
public static void main(String[] args) {
    DataOutputStream player1dos;
    DataOutputStream player2dos;

    while (true) {
        try (ServerSocket serverSocket = new ServerSocket(SERVER_PORT);) {
            System.out.println("4Wins -> Server running...");

            // accept first player
            Socket player1Socket = serverSocket.accept();
            System.out.println("1st client connected");
            player1dos = new DataOutputStream(player1Socket.getOutputStream());
            player1dos.writeUTF(Protocol.WAITING_FOR_OPPONENT);
            player1dos.writeInt( v: 1);

            // accept second player
            Socket player2Socket = serverSocket.accept();
            System.out.println("2nd client connected");
            player2dos = new DataOutputStream(player2Socket.getOutputStream());
            player2dos.writeUTF(Protocol.WAITING_FOR_OPPONENT);
            player2dos.writeInt( v: 2);

            player2dos.writeUTF(Protocol.OPPONENT_FOUND);
            player1dos.writeUTF(Protocol.OPPONENT_FOUND);

            // start listener thread for client messages
            new Thread(new ClientHandler(new DataInputStream(player1Socket.getInputStream()), player2dos)).start();
            new Thread(new ClientHandler(new DataInputStream(player2Socket.getInputStream()), player1dos)).start();
        } catch (SocketException s) {
            System.out.println("Client disconnected.");
            break;
        } catch (IOException e) {
            System.out.println("Something went wrong.");
            break;
        }
    }
}
```

Server

Protocol

```
public class Protocol {
    3 usages
    public static final String WAITING_FOR_OPPONENT = "waiting";
    3 usages
    public static final String OPPONENT_FOUND = "opponent found";
    5 usages
    public static final String GAME_OVER = "game over";
    4 usages
    public static final String MOVE = "move";
    8 usages
    public static final String QUIT = "quit";
}
```

ServerListener

```
public void run() {
    System.out.println("Server listener running...");
    command = "";

    try {
        // run while command is not equal to quit string
        while (!command.equals(Protocol.QUIT)) {
            command = this.dis.readUTF();
            System.out.println("Remote: " + command);

            // process command
            switch (command) {
            case Protocol.WAITING_FOR_OPPONENT:
                int turn = this.dis.readInt();
                System.out.println("You are player " + turn);
                gameController.setPlayer(turn);
                startController.showAlert();
                break;
            case Protocol.OPPONENT_FOUND:
                startController.startGame();
                break;
            case Protocol.MOVE:
                int row = this.dis.readInt();
                int column = this.dis.readInt();
                gameController.setOpponentCircle(row, column);
                break;
            case Protocol.GAME_OVER:
                break;
            case Protocol.QUIT:
                System.out.println("Opponent quit");
                gameController.send(Protocol.QUIT);
                this.command = Protocol.QUIT;
                Platform.runLater(() -> gameController.showDialog( msg: "Opponent left the game! Start new game."));
                break;
            default:
                ChatHandler.addMessage(command,  player: 2);
                Platform.runLater(() -> gameController.showChatMessages());
                break;
            }
        }
    } catch (IOException e) {
        System.out.println("Error occured.");
    }
    System.out.println("Listener stopped.");
}
```

Client

```java
public void sendMoveToOpponent(int row, int column) {
    try {
        this.dos.writeUTF(Protocol.MOVE);
        this.dos.writeInt(row);
        this.dos.writeInt(column);
    } catch (IOException e) {
        e.printStackTrace();
    }
}

// sends info about game over
1 usage  FuziPavuzi
public void sendGameOver() {
    try {
        this.dos.writeUTF(Protocol.GAME_OVER);
    } catch (IOException e) {
        e.printStackTrace();
    }
}

// sends any message
2 usages  FuziPavuzi
public void send(String message) {
    try {
        this.dos.writeUTF(message);
    } catch (IOException e) {
        e.printStackTrace();
    }
}

// send info about quitting
1 usage  FuziPavuzi
public void quit() {
    try {
        this.dos.writeUTF(Protocol.QUIT);
        listener.stop();
    } catch (IOException e) {
        System.out.println("Something went wrong");
    }
}
```

ClientHandler

```java
public void run() {
    try {
        String command = "";

        // read input streams as long as input is not equal to game over string
        while (!command.equals(Protocol.GAME_OVER)) {
            command = playerDis.readUTF();
            process(command);
        }
    } catch (SocketException s) {
        System.out.println("Client disconnected.");
    } catch (IOException e) {
        e.printStackTrace();
    }
}

// take commands and mediate to opponent on opponents outputstream
1 usage  FuziPavuzi
public void process(String command) throws IOException {
    switch (command) {
    case Protocol.MOVE:
        int row = this.playerDis.readInt();
        int column = this.playerDis.readInt();

        opponentDos.writeUTF(Protocol.MOVE);
        opponentDos.writeInt(row);
        opponentDos.writeInt(column);
        break;
    case Protocol.GAME_OVER:
        opponentDos.writeUTF(Protocol.GAME_OVER);
        break;
    case Protocol.QUIT:
        opponentDos.writeUTF(Protocol.QUIT);
        break;
    default:
        opponentDos.writeUTF(command);
        break;
    }
}
```