Department of Electrical & Computer Engineering
Second Semester, 2023/2024
ENCS3130 Linux Laboratory
Shell Scripting Project – Text Message Encryption and Decryption

Student Name: Asmaa Abed Al-Rahman Fares

Student ID:1210084

Student Name: Adam Samer Al-Adham

Student ID:1210396

Section:2

Instructor Name: Aziz Qaroush

TA Name: Ahed Mafarjeh

Table of content

1-Abstract

The "XO Game Shell Script" project aims to create a terminal-based game where two users can play the XO (Tic-Tac-Toe) game with added features like loading from a file and various move options. The script prompts users to input their names and the number of moves to play. Players can choose between starting with an empty grid or loading from a file . Players can make moves to place or remove their marks, exchange rows or columns, or swap marks with their opponent. Scoring is based on alignments and moves made, with points earned or deducted accordingly. The script tracks and displays the score of each player throughout the game. The project involves shell scripting to handle user inputs, manage game state, and calculate scores, enhancing the classic game of XO with strategic elements and customization options.

2-Theory

The XO game is played on a grid , where two players take turns marking empty cells. The objective is to form a horizontal, vertical, or diagonal line of three of one's own marks (X or O).

Here's a breakdown of the script's functionality:

1. Grid Initialization and Display: The grid is represented as a multi-dimensional associative array in Bash. The (display_empty_grid) function initializes and displays an empty grid of the specified size. The (display_grid_from_file) function reads the grid from a file and displays it.

2. Scoring System : The script keeps track of player scores throughout the game. The (update_scores) function updates the scores based on the alignments of marks on the grid. Players gain or lose points depending on their moves and their effects on the grid.

3. Move Functions :

1. Move 1 (Placing a Mark):
   - When a player places their mark on the grid, the script checks for alignments.
   - If the placement creates alignments of the player's mark (either horizontally, vertically, or diagonally), the player earns 2 points.
   - If the placement creates alignments of the opponent's mark, the player loses 3 points (as it might be seen as aiding the opponent).

2. Move 2 (Removing a Mark):
   - When a player removes their mark from the grid, 1 point is directly awarded .

3. Move 3 (Exchanging Rows):
   - When a player exchanges rows on the grid it decrements by 1 point and , the script checks for alignments after the exchange.
   - If the exchange creates alignments of the player's mark, the player earns 2 points.
   - If the exchange creates alignments of the opponent's mark, the player loses 3 points.

4. Move 4 (Exchanging Columns):
   - When a player exchanges columns on the grid it decrements by 1 point and , the script checks for alignments after the exchange.
   - Similar to Move 3, the player earns 2 points for creating alignments of their mark and loses 3 points for creating alignments of the opponent's mark.

5. Move 5 (Exchanging Marks' Positions):

- When a player exchanges the positions of their mark and the opponent's mark it decrements by 2 points and , the script checks for alignments after the exchange.
- Again, the player earns 2 points for creating alignments of their mark and loses 3 points for creating alignments of the opponent's mark.

4. Main Function and Game Loop: The `main` function orchestrates the game. It initializes the game settings, including the grid size, player names, and maximum number of moves. It then enters a game loop where players take turns making moves until the maximum moves are reached.

5. Winner Determination: After the game loop ends, the script determines the winner based on the scores. If there's a tie, it declares it. This adds an additional layer of strategy to the game, as players must balance making moves to increase their scores while also preventing their opponent from gaining an advantage.

6. Restart or Quit: After the game ends, players are given the option to restart the game or quit. This allows for multiple rounds of gameplay without needing to rerun the script.

Overall, the script provides a comprehensive and interactive XO game experience with additional strategic elements beyond the basic rules of Tic-Tac-Toe. Players must carefully consider their moves to outscore their opponent and emerge victorious .

3-Procedure

1. Game Initialization:

```
asmaa@asmaa-virtual-machine: ~/Desktop
asmaa@asmaa-virtual-machine:~/Desktop$ bash new1.sh
Welcome to the XO Game!
Do you want to start an empty game or load from a file? (empty/load)
```

If starting with an empty grid:

```
asmaa@asmaa-virtual-machine: ~/Desktop
asmaa@asmaa-virtual-machine:~/Desktop$ bash new1.sh
Welcome to the XO Game!
Do you want to start an empty game or load from a file? (empty/load)
empty
Enter the size of the grid (3, 4, or 5):
3
Empty grid:
| | | |
| | | |
| | | |
```

If loading from file:

```
asmaa@asmaa-virtual-machine: ~/Desktop
asmaa@asmaa-virtual-machine:~/Desktop$ bash new1.sh
Welcome to the XO Game!
Do you want to start an empty game or load from a file? (empty/load)
load
Enter the path to the file:
grid.txt
Grid loaded from file:
|X|X|O|
|O|X|X|
|X|O|O|
```

2. Player Setup:

```
asmaa@asmaa-virtual-machine: ~/Desktop
asmaa@asmaa-virtual-machine:~/Desktop$ bash new1.sh
Welcome to the XO Game!
Do you want to start an empty game or load from a file? (empty/load)
load
Enter the path to the file:
grid.txt
Grid loaded from file:
|X|X|O|
|O|X|X|
|X|O|O|

Enter player 1's name:
A
Enter player 2's name:
B
```

3. Set Maximum Number of Moves:

```
|X|X|O|
|O|X|X|
|X|O|O|

Enter player 1's name:
A
Enter player 2's name:
B
Enter the maximum number of moves:
15
A's turn (X)
Choose your move:
1. Place mark
2. Remove mark
3. Exchange rows
4. Exchange columns
5. Exchange marks
```

3. Game Loop:

```
Enter the maximum number of moves:
15
A's turn (X)
Choose your move:
1. Place mark
2. Remove mark
3. Exchange rows
4. Exchange columns
5. Exchange marks
```

5. Move Execution and Scoring :

1. Move 1 : Here we can see player "A" chose move 1 to place the mark , and this mark cuased an alignment for him so his score is increased by 2.

```
Updated grid:
|X|O|O|
| |X| |
| | | |
 ____________________________________________________
|                        Scores:                     |
|____________________________________________________|
|                            |                       |
|                        A: 0| 0: B                  |
|____________________________|_______________________|

A's turn (X)
Choose your move:
1. Place mark
2. Remove mark
3. Exchange rows
4. Exchange columns
5. Exchange marks

1
Enter the row number (1-3):
3
Enter the column number (1-3):
3

Updated grid:
|X|O|O|
| |X| |
| | |X|
 ____________________________________________________
|                        Scores:                     |
|____________________________________________________|
|                            |                       |
|                        A: 2| 0: B                  |
|____________________________|_______________________|
```

2. Move 2 : Here we can see how move 2 removes a mark , also when player "B" chose it his score will increase by one.

```
Updated grid:
|X| | |
| | | |
| | | |

 _____________________________________________________
|                      Scores:                        |
|_____________________________________________________|
|                           |                         |
|                      A: 0| 0: B                    |
|___________________________|_________________________|

B's turn (O)
Choose your move:
1. Place mark
2. Remove mark
3. Exchange rows
4. Exchange columns
5. Exchange marks

2
Enter the row number (1-3):
1
Enter the column number (1-3):
1

Updated grid:
| | | |
| | | |
| | | |

 _____________________________________________________
|                      Scores:                        |
|_____________________________________________________|
|                           |                         |
|                      A: 0| 1: B                    |
|___________________________|_________________________|
```

3. Move 3 : Here we exchange rows in move 3 , the player "A" wanted to exchange row one with row two so at first he gets a minus one point for using move 3 then he gets an additional two points for getting an alignment to his favor.

```
Grid loaded from file:
|X|X|O|
|O|X|X|
|X|O|O|

Enter player 1's name:
A
Enter player 2's name:
B
Enter the maximum number of moves:
100
A's turn (X)
Choose your move:
1. Place mark
2. Remove mark
3. Exchange rows
4. Exchange columns
5. Exchange marks

3
Enter the rows to exchange (e.g., 'rxy' where 'x' and 'y' are row numbers):
r12

Updated grid:
|O|X|X|
|X|X|O|
|X|O|O|

 _____________________________________________________
|                      Scores:                        |
|_____________________________________________________|
|                            |                        |
|                      A: 1  | 0: B                   |
|____________________________|________________________|
```

4. Move 4 : Here we can how we can exchanges columns in move 4 , the player "B" wanted to exchange column one with column two so at first he gets a minus one point for using move 4 then he gets a minus three points for getting an alignment for his opponent.

```
Updated grid:
|X|O|O|
| |X| |
| | |X|
 ____________________________________________________
|                        Scores:                     |
|____________________________________________________|
|                           |                        |
|                       A: 2| 0: B                   |
|___________________________|________________________|

B's turn (O)
Choose your move:
1. Place mark
2. Remove mark
3. Exchange rows
4. Exchange columns
5. Exchange marks

4
Enter the columns to exchange (e.g., 'cxy' where 'x' and 'y' are column numbers):
c13

Updated grid:
|O|O|X|
| |X| |
|X| | |
 ____________________________________________________
|                        Scores:                     |
|____________________________________________________|
|                           |                        |
|                       A: 2| -4: B                  |
|___________________________|________________________|
```

5. Move 5 : Here we can how we can exchanges marks in move 5, the player "A" wanted to exchange the mark in row one & column one with the mark in row 3 & column 3 , so at first he gets a minus two points for using move 5 then he loses three points for getting an alignment for his opponent.

```
3

Updated grid:
|O|O|X|
| |X| |
|X| |O|
 _____________________________________________________
|                       Scores:                       |
|_____________________________________________________|
|                           |                         |
|                       A: 2| -7: B                   |
|___________________________|_________________________|

A's turn (X)
Choose your move:
1. Place mark
2. Remove mark
3. Exchange rows
4. Exchange columns
5. Exchange marks

5
Enter the rows and columns numbers of your mark & opponent mark to exchange
(e.g., 'exyuv' where 'e' and 'x' are your row & column numbers and 'y' and 'u' your opponent row & column number):
e1333

Updated grid:
|O|O|O|
| |X| |
|X| |X|
 _____________________________________________________
|                       Scores:                       |
|_____________________________________________________|
|                           |                         |
|                      A: -3| -7: B                   |
|___________________________|_________________________|
```

6. Check for Winner & End game :

```
1
Enter the row number (1-3):
2
Enter the column number (1-3):
1

Updated grid:
| |X|X|
|O|O| |
| |X| |
 ____________________________________________________
|                        Scores:                     |
|____________________________________________________|
|                            |                       |
|                       A: 4| 2: B                  |
|____________________________|_______________________|


 ____________________________________________________
|                        Scores:                     |
|____________________________________________________|
|                            |                       |
|                       A: 4| 2: B                  |
|____________________________|_______________________|


Player A wins !
```

Here we can see our game displays the scores and the winner name in the end of the game , also with an option if the players wants to play again or quit .

```
 ____________________________________________________
|                        Scores:                     |
|____________________________________________________|
|                            |                       |
|                       A: 1| 2: B                  |
|____________________________|_______________________|


Player B wins based on scores!
main.bash: line 526: break: only meaningful in a `for', `while', or `until' loop
Do you want to Restart or Quit? (Restart/Quit)
```

4- Conclusion

In conclusion, our experiment with the "XO Game Shell Script" project has been a fun and educational journey. We've learned a lot about shell scripting and its capabilities in creating interactive games. By implementing features like customizable grid sizes, loading from files, and various move options, we've made the classic XO game more dynamic and exciting.

Throughout the experiment, we faced challenges and learned how to handle user inputs, manage game state, and calculate scores effectively. We also explored the importance of strategic thinking in gameplay through the scoring mechanisms we implemented.

Overall, this experiment has been a great learning experience. It has shown us the potential of shell scripting in game development and has sparked our creativity in exploring new ways to enhance classic games. We look forward to applying what we've learned in future projects and continuing to explore the world of scripting and game development.
,

5- Appendix

The code :

```
#!/bin/bash

#-----------------------------------------------------------Initialize--------
-------------------------------------------------------
# initialize grid as a multi-dimensional array
declare -A grid
declare player1_score=0
declare player2_score=0
declare winner=""
#-----------------------------------------------------------display_empty_grid-
-------------------------------------------------------
display_empty_grid() {
    local size=$1
    for ((i = 1; i <= size; i++)); do
        for ((j = 1; j <= size; j++)); do
            grid[$i,$j]=" "  # populate the grid with empty spaces
            echo -n "| " # display delimiter for each cell
        done
        echo "|"  # end of row
    done
}
#-------------------------------------------------------
display_grid_from_file-------------------------------------------------------
display_grid_from_file() {
    local file_path=$1
    # read the size of the grid from the first line of the file
    local size=$(head -n 1 "$file_path" | awk -F '|' '{print NF-2}')
    # update global variable N
    N=$size
    # read file content line by line and populate the grid
    while IFS= read -r line; do
        # remove the first and last '|' characters from the line
        line=${line#|}
        line=${line%|}
        # split the line using '|' as the delimiter
        IFS='|' read -ra cells <<< "$line"
        # trim the array to the size of the grid
        cells=("${cells[@]:0:size}")
        for ((i = 0; i < size; i++)); do
```

```
            grid[$((line_num + 1)),$((i + 1))]=${cells[i]} # Populate the
grid with characters from the row
        done
        ((line_num++)) #increment line number
    done < "$file_path"
    display_grid # display the loaded grid
}
#-----------------------------------------------------------------display_grid-
-----------------------------------------------------
display_grid() {
    for ((i = 1; i <= N; i++)); do
        for ((j = 1; j <= N; j++)); do
            echo -n "|${grid[$i,$j]}" # display delimiter for each cell
        done
        echo "|"  # end of row
    done
    }
#-----------------------------------------------------------------display_scores--
-----------------------------------------------------
display_scores(){
    echo " ___________________________________________________________ "
    echo "|                        Scores:                            |"
    echo "|___________________________________________________________|"
    echo "|                            |                              |"
    printf "| %26s| %-26s |\n" "$player1: $player1_score" "$player2_score:
$player2"
    echo "|___________________________|______________________________|"
    echo
}
#-----------------------------------------------------------------update_scores-----
-----------------------------------------------------
update_scores() {
    local player_mark=$1
    local opponent_mark=$2
    # Initialize score changes
    local player_score_change=0
    local opponent_score_change=0
    # Check for alignments in the grid
    if check_alignment "$player_mark"; then
        ((player_score_change += 2)) # Player gets +2 points for alignment
    fi
    if check_alignment "$opponent_mark"; then
        ((player_score_change -= 3)) #player gets -3 for causing an alignment
for opponent

    fi
```

```
    # Update the scores

    if [[ "$((moves % 2))" -eq 0 ]]; then
        ((player1_score += player_score_change))
        ((player2_score += opponent_score_change))
    else
        ((player2_score += player_score_change))
        ((player1_score += opponent_score_change))
    fi

}
#-----------------------------------------------------------------check_alignment--
---------------------------------------------
check_alignment() {
    local mark=$1
    local aligned=false
    # Check horizontal alignments
    for ((i = 1; i <= N; i++)); do
        local count=0
        for ((j = 1; j <= N; j++)); do
            if [[ "${grid[$i,$j]}" == "$mark" ]]; then
                ((count++))
            fi
        done
        if [[ $count -eq N ]]; then
            aligned=true
            break
        fi
    done
    # Check vertical alignments
    if ! $aligned; then
        for ((j = 1; j <= N; j++)); do
            local count=0
            for ((i = 1; i <= N; i++)); do
                if [[ "${grid[$i,$j]}" == "$mark" ]]; then
                    ((count++))
                fi
            done
            if [[ $count -eq N ]]; then
                aligned=true
                break
            fi
        done
    fi
    # Check diagonal "\" alignment
    if ! $aligned; then
        local count=0
        for ((i = 1; i <= N; i++)); do
```

```
            if [[ "${grid[$i,$i]}" == "$mark" ]]; then
                ((count++))
            fi
        done
        if [[ $count -eq N ]]; then
            aligned=true
        fi
    fi
    # Check diagonal "/" alignment
    if ! $aligned; then
        local count=0
        for ((i = 1; i <= N; i++)); do
            if [[ "${grid[$i,$((N - i + 1))]}" == "$mark" ]]; then
                ((count++))
            fi
        done
        if [[ $count -eq N ]]; then
            aligned=true
        fi
    fi

    # Check diagonal "\" alignment
    if ! $aligned; then
        local count=0
        for ((i = 1; i <= N; i++)); do
            if [[ "${grid[$i,$i]}" == "$mark" ]]; then
            ((count++))
            fi
        done
        if [[ $count -eq N ]]; then
        aligned=true
        fi
    fi
    $aligned
}
#-------------------------------------------------------------move_1-----------
-------------------------------------------
# Function to place player marks in an empty cell
move_1() {
    local size=$N
    echo "Enter the row number (1-$size):"
    read row
    echo "Enter the column number (1-$size):"
    read col
    while true; do
        # Validate the input coordinates
        if [[ "$row" -lt 1 || "$row" -gt "$size" || "$col" -lt 1 || "$col" -
gt "$size" ]]; then
```

```
            echo "Invalid coordinates. Your turn if over :( ."
            return
        fi
        # Check if the cell is already occupied
        if [[ "${grid[$row,$col]}" != " " ]]; then
            echo "Cell already occupied. Your turn if over :( ."
            return
        fi
        # Place the mark in the selected cell
        if [[ "$((moves % 2))" -eq 0 ]]; then
            grid[$row,$col]="X"
            update_scores "X" "O"
        else
            grid[$row,$col]="O"
            update_scores "O" "X"
        fi
        # Display the updated grid
        echo
        echo "Updated grid:"
        display_grid
        display_scores
        break
    done
}
#-----------------------------------------------------------------move_2---------
-------------------------------------------------------
# Function to remove player marks from an occupied cell
move_2() {
    local size=$N
    echo "Enter the row number (1-$size):"
    read row
    echo "Enter the column number (1-$size):"
    read col

    # Add 1 point for playing Move 2
    if [[ "$((moves % 2))" -eq 0 ]]; then
        (( player1_score += 1 ))
    else
        (( player2_score += 1 ))
    fi

    while true; do
        # Validate the input coordinates
        if [[ "$row" -lt 1 || "$row" -gt "$size" || "$col" -lt 1 || "$col" -
gt "$size" ]]; then
            echo "Invalid coordinates. Your turn if over :( ."
            return
        fi
```

```
        # Check if the cell is already empty
        if [[ "${grid[$row,$col]}" == " " ]]; then
            echo "Cell is already empty. Your turn if over :( ."
            display_grid
            return
        fi
        # Remove the mark from the selected cell
        grid[$row,$col]=" "
        # Display the updated grid
        echo
        echo "Updated grid:"
        display_grid
        display_scores
        break
    done
}
#-----------------------------------------------------------------move_3--------
-----------------------------------------------
# Function to exchange rows on the grid
move_3() {
    local size=$N
    echo "Enter the rows to exchange (e.g., 'rxy' where 'x' and 'y' are row
numbers):"
    read rows

    # Penalize players for Move 3
    if [[ "$((moves % 2))" -eq 0 ]]; then
        (( player1_score -= 1 ))
    else
        (( player2_score -= 1 ))
    fi

    while true; do
        # Extract row numbers from user input
        rchar=${rows:0:1}
        row1=${rows:1:1}
        row2=${rows:2:1}
        # Validate row numbers
        if [[ "$rchar" != "r" || "$row1" -lt 1 || "$row1" -gt "$size" ||
"$row2" -lt 1 || "$row2" -gt "$size" ]]; then
            echo "Invalid input format or row numbers. Your turn if
over :( ."
            return
        fi
        # Exchange rows
        for ((i = 1; i <= size; i++)); do
            temp=${grid[$row1,$i]}
            grid[$row1,$i]=${grid[$row2,$i]}
```

```
            grid[$row2,$i]=$temp
        done

        # checking for alignments
        if [[ "$((moves % 2))" -eq 0 ]]; then
            update_scores "X" "O"
        else
            update_scores "O" "X"
        fi
        # Display the updated grid
        echo
        echo "Updated grid:"
        display_grid
        display_scores
        break
    done
}
#-------------------------------------------------------------move_4---------
-----------------------------------------------------
# Function to exchange columns on the grid
move_4() {
    local size=$N
    echo "Enter the columns to exchange (e.g., 'cxy' where 'x' and 'y' are
column numbers):"
    read columns

    # Penalize players for Move 4
    if [[ "$((moves % 2))" -eq 0 ]]; then
        (( player1_score -= 1 ))
    else
        (( player2_score -= 1 ))
    fi

    while true; do
        # Extract column numbers from user input
        charc=${columns:0:1}
        col1=${columns:1:1}
        col2=${columns:2:1}
        # Validate column numbers
        if [[ "$charc" != "c" || "$col1" -lt 1 || "$col1" -gt "$size" ||
"$col2" -lt 1 || "$col2" -gt "$size" ]]; then
            echo "Invalid input format or column numbers. Your turn if
over :( ."
            return
        fi
        # Exchange columns
        for ((i = 1; i <= size; i++)); do
            temp=${grid[$i,$col1]}
```

```
            grid[$i,$col1]=${grid[$i,$col2]}
            grid[$i,$col2]=$temp
        done

        # checking for alignments
        if [[ "$((moves % 2))" -eq 0 ]]; then
            update_scores "X" "O"
        else
            update_scores "O" "X"
        fi
        # Display the updated grid
        echo
        echo "Updated grid:"
        display_grid
        display_scores
        break
    done
}
#-------------------------------------------------------------move_5----------
-------------------------------------------------
# Function to exchange the positions of the player mark and the opponent mark
move_5() {
    local size=$N
    echo "Enter the rows and columns numbers of your mark & opponent mark to
exchange "
    echo "(e.g., 'exyuv' where 'e' and 'x' are your row & column numbers and
'y' and 'u' your opponent row & column number):"
    read xyuv

    # Penalize players for Move 5
    if [[ "$((moves % 2))" -eq 0 ]]; then
        (( player1_score -= 2 ))
    else
        (( player2_score -= 2 ))
    fi

    while true; do
        # Extract player and opponent coordinates
        chare=${xyuv:0:1}
        row1=${xyuv:1:1}
        col1=${xyuv:2:1}
        row2=${xyuv:3:1}
        col2=${xyuv:4:1}
        # Validate coordinates
        if [[ "$chare" != "e" || "$row1" -lt 1 || "$row1" -gt "$size" ||
"$col1" -lt 1 || "$col1" -gt "$size" || "$row2" -lt 1 || "$row2" -gt "$size"
|| "$col2" -lt 1 || "$col2" -gt "$size" ]]; then
            echo "Invalid coordinates. Your turn if over :( ."
```

```
            return
        fi
        # Exchange marks
        temp=${grid[$row1,$col1]}
        grid[$row1,$col1]=${grid[$row2,$col2]}
        grid[$row2,$col2]=$temp

        # checking for alignments
        if [[ "$((moves % 2))" -eq 0 ]]; then
            update_scores "X" "O"
        else
            update_scores "O" "X"
        fi

        # Display the updated grid
        echo
        echo "Updated grid:"
        display_grid
        display_scores
        break
    done
}
#-----------------------------------------------------------------main-----------
-------------------------------------------------
# main function to control the flow of the game
main() {
    # Reset scores
    player1_score=0
    player2_score=0
    moves=0

    # Game initialization
    echo "Welcome to the XO Game!"
    echo "Do you want to start an empty game or load from a file?
(empty/load)"
    while true; do
        read game_option
        if [[ "$game_option" == "empty" ]]; then
            break
        elif [[ "$game_option" == "load" ]]; then
            break
        else
            echo "Invalid option. Please enter 'empty' or 'load'."
        fi
    done
    if [[ "$game_option" == "empty" ]]; then
        while true; do
            echo "Enter the size of the grid (3, 4, or 5):"
```

```
            read N
            if [[ "$N" == "3" || "$N" == "4" || "$N" == "5" ]]; then
                break
            else
                echo "Invalid grid size. Please enter 3, 4, or 5."
            fi
        done
        echo "Empty grid:"
        display_empty_grid "$N" # Display an empty grid of specified size
    elif [[ "$game_option" == "load" ]]; then
        echo "Enter the path to the file:"
        while true; do
            read file_path
            if [[ -f "$file_path" ]]; then
                break
            else
                echo "File not found. Please enter a valid file path with the
extention."
            fi
        done
        echo "Grid loaded from file:"
        display_grid_from_file "$file_path"  # Display the grid loaded from
the file
    fi

    echo
    echo "Enter player 1's name:"
    read player1
    echo "Enter player 2's name:"
    read player2


    echo "Enter the maximum number of moves:"
    while true; do
    read max_moves
      if [[ "$max_moves" -gt 2 ]]; then
        break
      else
        echo "Invalid input. Please enter a number larger than 2."
      fi
    done
    # game loop
    moves=0
    while [[ "$moves" -lt "$max_moves" ]]; do

        # display current player's turn
        if [[ "$((moves % 2))" -eq 0 ]]; then
            echo "$player1's turn (X)"
```

```
        else
            echo "$player2's turn (O)"
        fi
        # prompt user for move
        while true; do
            echo "Choose your move:"
            echo "1. Place mark"
            echo "2. Remove mark"
            echo "3. Exchange rows"
            echo "4. Exchange columns"
            echo "5. Exchange marks"
            echo
            read choice
            # execute chosen move
            case $choice in
                1) move_1 && break ;;
                2) move_2 && break ;;
                3) move_3 && break ;;
                4) move_4 && break ;;
                5) move_5 && break ;;
                *) echo "Invalid choice. Please choose a number from 1 to
5." ;;
            esac
        done
        ((moves++)) # Increment moves counter
    done
        # winner based on score
        if [[ "$moves" -eq "$max_moves" ]]; then
            if [[ "$player1_score" -gt "$player2_score" ]]; then
                display_scores
                echo " "
                echo "Player $player1 wins !"
                break
            elif [[ "$player2_score" -gt "$player1_score" ]]; then
                display_scores
                echo " "
                echo "Player $player2 wins based on scores!"
                break
            else
                display_scores
                echo " "
                echo "It's a tie!"
                break
            fi
        fi

        # ask for restart or quit
        echo "Do you want to Restart or Quit? (Restart/Quit)"
```

```
        read des
        while true; do
            if [[ "$des" == "Restart" ]]; then
                    main
                    break
            elif [[ "$des" == "Quit" ]]; then
                    exit
            else
                    echo "Invalid choice. Please enter 'Restart' or 'Quit'."
                    read des
            fi
        done

}
# start the game
main
```