Faculty of Engineering and Technology

Electrical and Computer Engineering Department.

Linux Lab ENCS313

Shell Script Project

Prepared by :

Yousef Qawwas:1220137 & Ahmad Abu Saleem:1201315

T.A: Ahed Mafarjeh

Date: 1/may/2024

Idea & Code Snippets:

This shell script code is for creating a XO game, in which the game acquires two players to compete against each other where the player must put his mark once in the grid in each turn, here in this script, the game is able to be launched in 3x3 or 4x4 or 5x5 grids also the game runs until a certain number of moves is reached and then it calculates the score of each player according to this plan:

After a player makes any move, they earn two points for any alignment (horizontal, vertical, or diagonal) of their marks and lose three points if the move results in any alignment for the opponent's marks.

Players earn one point for playing Move 2

Players are penalized with one point for playing Move 3 or Move 4 and two points for playing Move 5.

For a better understanding let us break into what these moves really mean:

Move 1 allows a player to put his mark on an empty cell

```
# Function to place marks in an empty cell (Move 1)
move1() {

    echo "Player $current_player's turn"
    echo "Enter the row and column numbers separated by a space to place your mark ('row column'): "
    read row col

    # making sure the values are valid.
    if [ $row -lt 1 ] || [ $row -gt $N ] || [ $col -lt 1 ] || [ $col -gt $N ]; then
        echo "Please enter valid row and column numbers."
        move_1  # recall the function
        return
    fi

    # Check if the cell is empty
    if [ "${grid[$((row - 1)),$((col - 1))]}" != " " ]; then
        echo "Cell is already occupied. Please choose an empty cell."
        move_1
        return
    fi

    # Place the mark in the empty cell
    grid[$((row - 1)),$((col - 1))]=$current_mark
}
```

Move 2 allows a player to remove his mark only from a specified cell

```
move2() {

    echo "Player $current_player's turn"
    echo "Enter the row and column numbers separated by a space to remove the mark ('row column'): "
    read row col

    # check that the values are valid.
    if [ $row -lt 1 ] || [ $row -gt $N ] || [ $col -lt 1 ] || [ $col -gt $N ]; then
        echo "Invalid coordinates. Please enter valid row and column numbers."
        move2
        return
    fi

    # Check if the cell is occupied
    if [ "${grid[$((row - 1)),$((col - 1))]}" == " " ]; then
        echo "Cell is already empty. Please choose an occupied cell."
        move2
        return
    fi

    # Check if the mark in the cell belongs to the current player
    if [ $current_player -eq 1 ]; then
        if [ "${grid[$((row - 1)),$((col - 1))]}" != "X" ]; then # player1 has access to X mark only and so for player2 with O mark.

            echo "The mark is not yours to remove!"
            move2
            return
        fi
    elif [ $current_player -eq 2 ]; then
        if [ "${grid[$((row - 1)),$((col - 1))]}" != "O" ]; then
            echo "The mark is not yours to remove!"
            move2
            return
        fi
    fi

    # Remove the mark from the occupied cell
    grid[$((row - 1)),$((col - 1))]=" "
}
```

Move 3 allows a player to exchange rows in the grid

```
move3() {
    echo "Player $current_player's turn"
    echo "Enter the row numbers to exchange (format as: rxy, where x and y are row numbers): "
    read rowNums

        temp=$(echo $rowNums | cut -c1) # to check if entered the data correctly.
        if [ $temp != 'r' ]; then
        echo "Please enter as specified!"
        move3
        return
        fi

    # get row numbers
    row1=$(echo $rowNums | cut -c2)
    row2=$(echo $rowNums | cut -c3)

    # check validity of data.
    if [ $row1 -lt 1 ] || [ $row1 -gt $N ] || [ $row2 -lt 1 ] || [ $row2 -gt $N ]; then
        echo "Invalid row numbers. Please enter valid row numbers."
        return
    fi

    row1=$((row1 - 1))  # to make sure that the index start from zero as the row numbers entered starts from 1 at least.
    row2=$((row2 - 1))

    # exchanging rows
    for ((j=0; j<N; j++)); do
        temp_cell="${grid[$row1,$j]}"
        grid[$row1,$j]=${grid[$row2,$j]}
        grid[$row2,$j]=$temp_cell
    done
}
```

Move 4 allows a player to exchange columns in the grid

```
move4() {

    echo "Player $current_player's turn"
    echo "Enter the column numbers to exchange (as format of :cxy, where x and y are column numbers): "
    read colNums

        tmp=$(echo $col_nums | cut -c1) # to check if entered the data correctly.
        if [ $tmp != 'c' ]; then
        echo "Please enter as specified!"
        move4
        return
        fi

    # get column numbers
    col1=$(echo $colNums | cut -c2)
    col2=$(echo $colNums | cut -c3)

    # make sure data passed correctly.
    if [ $col1 -lt 1 ] || [ $col1 -gt $N ] || [ $col2 -lt 1 ] || [ $col2 -gt $N ]; then
        echo "Please enter valid column numbers."
        move4
        return
    fi
        col1=$((col1 - 1)) # making sure index starts from 0 when exchanging.
        col2=$((col2 - 1))

    # exchange columns
    for ((i=0; i<N; i++)); do
        temp_cell="${grid[$i,$col1]}"
        grid[$i,$col1]=${grid[$i,$col2]}
        grid[$i,$col2]=$temp_cell
    done
}
```

Move 5 allows a player to exchange one of his marks with one of his opponent's

```
move5() {
    echo "Player $current_player's turn"
    echo "Enter the positions to exchange marks (as 'exyuv', where e stands for exchange, x and y are player's credentials, u and v opponent's): "
    read positions

        var=$(echo $positions | cut -c1)
        if [ $var != 'e' ]; then
                echo "please check input data!"
                move5
                return
        fi

    # get positions
    player_row=$(echo $positions | cut -c2)
    player_col=$(echo $positions | cut -c3)
    opponent_row=$(echo $positions | cut -c4)
    opponent_col=$(echo $positions | cut -c5)

    # make sure are valid
    if [ $player_row -lt 1 ] || [ $player_row -gt $N ] || [ $player_col -lt 1 ] || [ $player_col -gt $N ] || [ $opponent_row -lt 1 ] || [ $opponent_
        echo "Please enter valid positions."
        move5
        return
    fi
        player_col=$((player_col - 1)) # also to make sure indexing starts at zero.
        player_row=$((player_row - 1))
        opponent_row=$((opponent_row - 1))
        opponent_col=$((opponent_col - 1))


    # exchange marks
    tempMark="${grid[$((player_row)),$((player_col))]}"
    grid[$((player_row)),$((player_col))]=${grid[$((opponent_row)),$((opponent_col))]}
    grid[$((opponent_row)),$((opponent_col))]=$tempMark
}
```

The score function:

```
score() {

    player1_score=0
    player2_score=0

    # Function to check for alignment of marks which is called after every case of possible score change.
    check_alignment() {

        local marks="$1" # local variable that is initialized according to what it has been passed by other functions.

        if echo "$marks" | grep -q "XXX"; then
            if [ $current_player -eq 1 ]; then          # making sure that the score updates according to each player's mark and his moves.
                player1_score=$((player1_score + 2))
            else
                player2_score=$((player2_score - 3))
            fi
        elif echo "$marks" | grep -q "OOO"; then
            if [ $current_player -eq 1 ]; then
                player1_score=$((player1_score - 3))
            else
                player2_score=$((player2_score + 2))
            fi
        fi
    }

    # Check horizontal alignments
    for ((i=0; i<N; i++)); do
        row=""
        for ((j=0; j<N; j++)); do
            row+="${grid[$i,$j]}"
        done
        check_alignment "$row" # sends $row as parameter.
    done

    # Check vertical alignments
    for ((j=0; j<N; j++)); do
        col=""
        for ((i=0; i<N; i++)); do
```

```
            col+="${grid[$i,$j]}"
        done
        check_alignment "$col"
    done

    # Check diagonal alignments (from the top left to the bottom right.)
    diag1=""
    for ((i=0; i<N; i++)); do
        diag1+="${grid[$i,$i]}"
    done
    check_alignment "$diag1"

    # Check diagonal alignments (top-right to bottom-left)
    diag2=""
    for ((i=0; i<N; i++)); do
        diag2+="${grid[$i,$((N - i - 1))]}"
    done
    check_alignment "$diag2"

    # update score for different move types

    case $move_choice in
        2)  # Move 2 ( removing )
            if [ $current_player -eq 1 ]; then
                player1_score=$((player1_score + 1))
            else
                player2_score=$((player2_score + 1))
            fi
            ;;
        3|4)  # Move 3 or Move 4: exchanging either rows or columns.
            if [ $current_player -eq 1 ]; then
                player1_score=$((player1_score - 1))
            else
                player2_score=$((player2_score - 1))
            fi
            ;;
        5)  # Move 5: exchanging marks
            if [ $current_player -eq 1 ]; then
```

```
        5)  # Move 5: exchanging marks
            if [ $current_player -eq 1 ]; then
                player1_score=$((player1_score - 2))
            else
                player2_score=$((player2_score - 2))
            fi
            ;;
    esac

    # Display scores
    echo "Player 1 score: $player1_score"
    echo "Player 2 score: $player2_score"
}
```

Program Run :

Here the user is asked to insert the initialization data

```
Enter Player 1's name:
yousef
Enter Player 2's name:
ahmad
Enter the number of moves in which the game will end after:
6
you want to load an existing file?(y/n)n
Enter the dimensions of the grid (NxN, where N can be 3, 4, or 5):
4
```

After, this grid and option show:

```
Game grid:
   |   |   |
   |   |   |
   |   |   |
   |   |   |
Player 1's turn
Choose your move:
1. Place mark in an empty cell
2. Remove mark from an occupied cell
3. Exchange rows on the grid
4. Exchange columns on the grid
5. Exchange positions of marks
1
Player 1's turn
Enter the row and column numbers separated by a space to place your mark ('row column'):
1 1
```

When player one wants to add his mark to the first cell this happens:

```
Game grid:
 X |   |   |
   |   |   |
   |   |   |
   |   |   |
Player 2's turn
Choose your move:
1. Place mark in an empty cell
2. Remove mark from an occupied cell
3. Exchange rows on the grid
4. Exchange columns on the grid
5. Exchange positions of marks
```

Note that the previous statements were cleared to maintain the clarity of the game grid

Now if player 2 wants to add his mark in the middle cell he enters 1 to add a new mark and then 2 2 to place it in the right cell

```
 X |   |   |
   | O |   |
   |   |   |
   |   |   |
Player 1's turn
Choose your move:
1. Place mark in an empty cell
2. Remove mark from an occupied cell
3. Exchange rows on the grid
4. Exchange columns on the grid
5. Exchange positions of marks
```

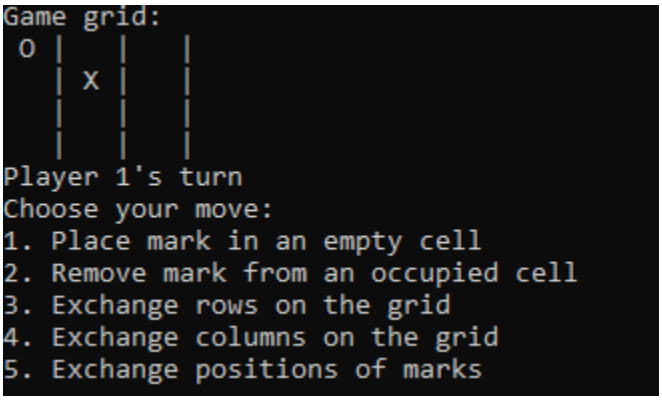
Now if player one wants to exchange the first row with the second, he must enter as follows:

```
Game grid:
 X |   |   |
   | O |   |
   |   |   |
   |   |   |
Player 1's turn
Choose your move:
1. Place mark in an empty cell
2. Remove mark from an occupied cell
3. Exchange rows on the grid
4. Exchange columns on the grid
5. Exchange positions of marks
3
Player 1's turn
Enter the row numbers to exchange (format as: rxy, where x and y are row numbers):
r12
```

So, the output is:

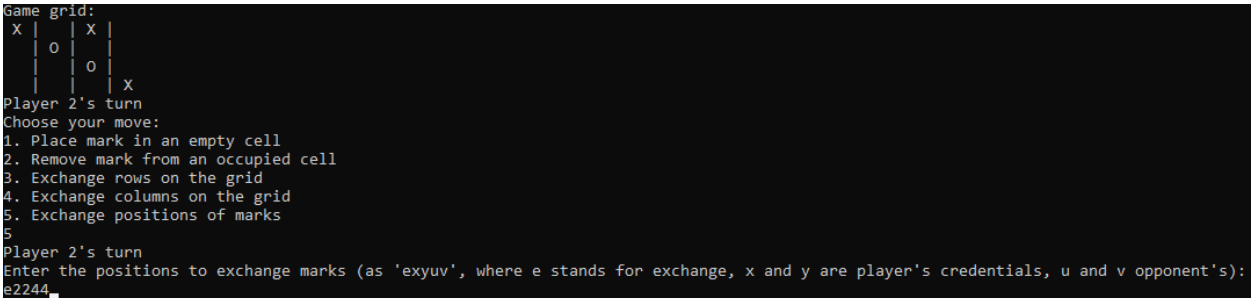
```
Game grid:
   | O |   |
 X |   |   |
   |   |   |
   |   |   |
Player 2's turn
Choose your move:
1. Place mark in an empty cell
2. Remove mark from an occupied cell
3. Exchange rows on the grid
4. Exchange columns on the grid
5. Exchange positions of marks
```

For exchanging columns player two must insert (c12) so the grid is modified as shows:

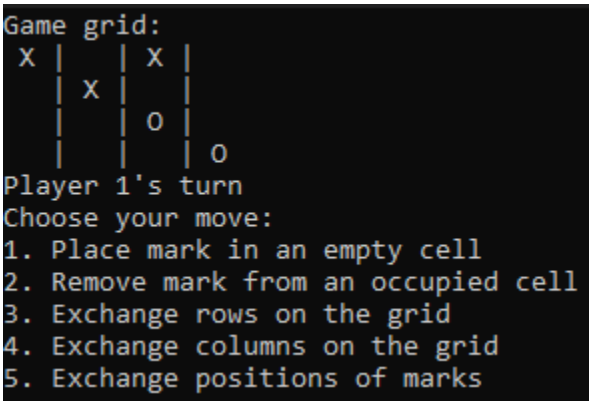


After going through with the game, let's try move 5 :

Suppose that the game has reached to this, now after inserting data as follows



This should happen:



Notice that cell 22 and cell 44 were exchanged.

If a player wants to delete his mark from a cell:

```
Game grid:
 X |   |
   | X | O
   |   | O
Player 1's turn
Choose your move:
1. Place mark in an empty cell
2. Remove mark from an occupied cell
3. Exchange rows on the grid
4. Exchange columns on the grid
5. Exchange positions of marks
2
Player 1's turn
Enter the row and column numbers separated by a space to remove the mark ('row column'):
1 1
```

```
Game grid:
   |   |
   | X | O
   |   | O
Player 2's turn
Choose your move:
1. Place mark in an empty cell
2. Remove mark from an occupied cell
3. Exchange rows on the grid
4. Exchange columns on the grid
5. Exchange positions of marks
```

If a player tries to delete a mark which is not his this would happen:

```
Game grid:
   |   |
   | X | O
   |   | O
Player 2's turn
Choose your move:
1. Place mark in an empty cell
2. Remove mark from an occupied cell
3. Exchange rows on the grid
4. Exchange columns on the grid
5. Exchange positions of marks
2
Player 2's turn
Enter the row and column numbers separated by a space to remove the mark ('row column'):
2 2
The mark is not yours to remove!
Player 2's turn
Enter the row and column numbers separated by a space to remove the mark ('row column'):
```