

**Department of Electrical & Computer Engineering**

**ENCS3130, LINUX LABORATORY**

**Second Semester-2023-2024**

**Project #1 Report**

**Shell Scripting**

**Students’ Names:**

***Rama Abu Hnaish 1211751***

***Lama Rimawi 1213515***

**Instructor: *Dr.* Mohammad Jubran**

**Section: 1 M 14:00 – 16:50**

**Introduction:**

The goal of this project was to develop a shell script allowing two users to play XO game within a terminal environment. Our objective was to follow the specifications outlined, ensuring that the script enables players to take turns positioning and replacing their marks on an NxN grid until the game concludes after a certain number of moves have been made by the players.

**Idea: Describe the concept and functionality of the XO game.**

**CONCEPT:**

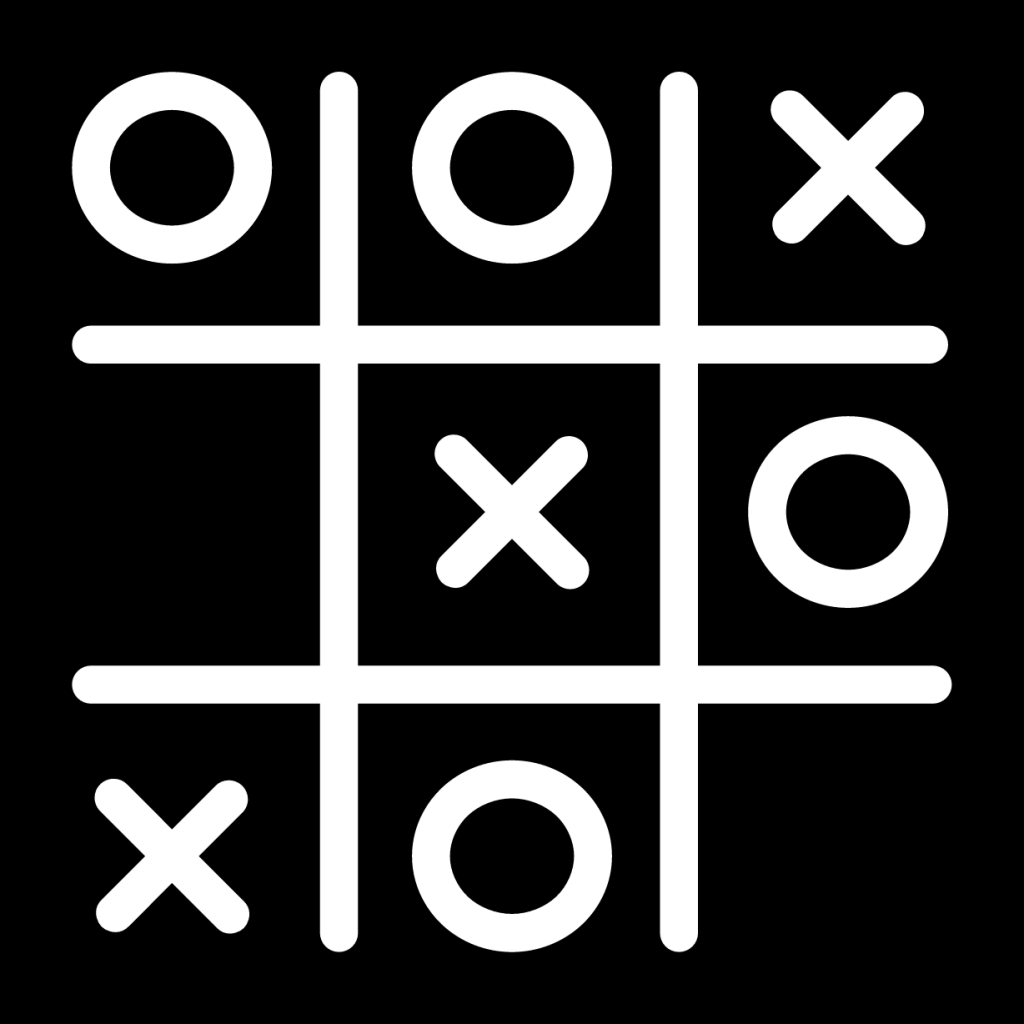
The XO game, is a classic two-player game played on a grid where each player takes turns marking the grid squares with their designated symbols, typically "X" and "O".

The main goal is to strategically place your mark (either "X" or "O") on the grid to form a line of at least 3 consecutive

Marks either horizontally, vertically,

or diagonally. The first player to achieve this

goal wins the game.

****

**Implementation:**

1. We started by carefully looking at the project specifications, figuring out what the script needed to do. From there, we organized our work to cover these main points:
   * User Input and Interaction:

Implemented prompts to capture users' names and the number of moves after which the game concludes.

* + Provided options for users to initialize an empty (new) game or load a grid configuration from a file.
  + Enabled users to input their moves, including placing marks, removing marks, and executing moves like row/column exchanges and mark exchanges.

**Game Logic and Grid Management:**

We wrote functions to manage the game grid, initialize it with specified dimensions (NxN), where N can be either 3, 4, or 5 and display it to users.

Implemented logic to check for a winner after each move by examining row, column, and diagonal alignments of marks.

Incorporated scoring rules outlined in the specifications, assigning points based on move types and alignment outcomes.

The scoring rules for the XO game are as follows:

* Earn 2 points for aligning three marks horizontally, vertically, or diagonally.
* Lose 3 points if the opponent aligns three marks.
* Gain 1 point for removing a mark (Move 2).
* a 1-point penalty for row or column exchanges (Move 3 or Move 4) 2-point for mark exchanges with the opponent (Move 5).

**The code: Provide the complete shell script code.**

#!/bin/bash

player1=""

player2=""

player1\_score=0

player2\_score=0

move\_count=0

echo "Welcome to the XO game!"

echo "-----------------------"

#Players' Names

# Prompt players for their names

echo "Player 1, please enter your name:"

read player1

echo "Player 2, please enter your name:"

read player2

# Display players' names

echo "Player 1: $player1 (X)"

echo "Player 2: $player2 (O)"

#set game configration

echo "How many moves should the game last?"

read max\_moves

echo "Specify the dimentions of the grid (3,4,or 5):"

read N

if [[ $N -eq 3 || $N -eq 4 || $N -eq 5 ]]

then

echo "valid size"

else

echo "Invalid grid size.Please choose 3,4,or 5."

exit 1

fi

#Initialize a grid with 2D array

declare -a grid

for ((i=0;i<N\*N;i++))

do

grid[i]=' '

done

#Option to load from file

while true

do

echo "Do you want to start a new game or load from a file? (new/load)"

read choice

if [ "$choice" == "new" ]

then

break

elif [ "$choice" == "load" ]

then

echo "Enter the file name:"

read filename

if [ -f "$filename" ]

then

# Load game from file

echo "Loading game from file..."

i=0

while IFS='|' read -ra cells

do

for cell in "${cells[@]}"

do

grid[i]="$cell"

i=$((i+1))

done

done < "$filename"

break

else

echo "File not found. Please enter a valid file name."

fi

else

echo "Invalid option. Please choose 'new' or 'load'."

fi

done

#--------------------------------

#display the initial grid

display\_grid() {

for (( i=0;i<N\*N;i++))

do

if [ $((i%N)) -eq $((N-1)) ]

then

echo -n "${grid[i]} "

echo

else

echo -n "${grid[i]} |"

fi

done

}

#-----------------------------------

print\_menu() {

local player=$1

local mark=$2

local opponent\_mark

if [ "$mark" == "x" ]

then

opponent\_mark="o"

else

opponent\_mark="x"

fi

local choice

echo "It's a $player's turn ($mark)."

echo "choose your move:"

echo "1. place a mark "

echo "2.remove a mark ($mark)"

echo "3.Exchange rows"

echo "4.Exchange columns"

echo "5.Exchange marks"

echo "6.Quit the game"

echo "Enter your choice (1-6): "

read choice

case $choice in

1) place\_mark "$player" "$mark" ;;

2) remove\_mark "$player" "$mark" ;;

3) exchange\_rows "$player" ;;

4) exchange\_columns "$player" ;;

5) exchange\_marks "$player" "$mark" "$opponent\_mark" ;;

6) echo "Exiting the game."

return 1 ;;

\*) echo "Invalid option.Please try again." ;;

esac

update\_scores "$mark" "$opponent\_mark"

#Display the updated scores

echo "current scores:"

echo "$player1: $player1\_score"

echo "$player2: $player2\_score"

return 0

}

#--------------------------------------------

place\_mark() {

local player=$1

local mark=$2

local row col index

#prompt the player for coordinate

echo "$player, enter the row and column numbers to place your $mark:"

read row col

#asjust for zero based indexing

row=$((row-1))

col=$((col-1))

#claculate the index for grid

index=$((row \* N + col))

#validate the coordinates and check if the cell is empty

if [ "$row" -ge 0 ] && [ "$row" -lt "$N" ] && [ "$col" -ge 0 ] && [ "$col" -lt "$N" ] && [ "${grid[index]}" == " " ]

then

grid[index]=$mark

display\_grid

else

echo "Invalid move or cell is already occupied. please try again."

place\_mark "$player" "$mark"

fi

}

#-----------------------------------

remove\_mark() {

local player=$1

local mark=$2

local row col index

#prompt the player for coordinate

echo "$player, enter the row and column numbers of the mark you want to remove:"

read row col

#adjust for zero based indexing

row=$((row-1))

col=$((col-1))

#calc for index in grid

index=$((row \* N +col))

#validate the coordinates and check if cell contains content the players mark

if [ "$row" -ge 0 ] && [ "$row" -lt $N ] && [ "$col" -ge 0 ] && [ "$col" -lt $N ] && [ "${grid[index]}" == "$mark" ]

then

grid[index]=' '

display\_grid

#update the player's score

if [ $mark = "x" ]

then

player1\_score=$(( player1\_score + 1 ))

echo "Score updated: $player has $player1\_score points."

else

player2\_score=$(( player2\_score + 1 ))

echo "Score updated: $player has $player2\_score points."

fi

else

echo "Invalid move or cell does not contain your mark. please try again."

remove\_mark "$player" "$mark"

fi

}

#----------------------------------------

exchange\_rows() {

local player=$1

local row1 row2 temp i

#prompt the player for the row numbers

echo "$player,enter the numbers of the two rows you want to exchange:"

read row1 row2

#adjust for zero based indexing

row1=$((row1 -1))

row2=$((row2 -1))

#validate the row numbers

if [ "$row1" -ge 0 ] && [ "$row1" -lt "$N" ] && [ "$row2" -ge 0 ] && [ "$row2" -lt "$N" ] && [ "$row1" -ne "$row2" ]

then

#exchange 2 rows

for ((i=0;i<N;i++))

do

temp="${grid[$((row1 \*N + i))]}"

grid[$((row1 \* N + i ))]="${grid[$((row2 \*N + i))]}"

grid[$((row2 \* N + i ))]=$temp

done

display\_grid

if [ $mark = "x" ]

then

player1\_score=$(( player1\_score - 1 ))

echo "Score updated: $player has $player1\_score points."

else

player2\_score=$(( player2\_score - 1 ))

echo "Score updated: $player has $player2\_score points."

fi

else

echo "Invalid row numbers or rows are the same.Please try again."

exchange\_rows "$player"

fi

}

#-------------------------------------

exchange\_columns() {

local player=$1

local col1 col2 temp i

#prompt the player for the col numbers

echo "$player,enter the numbers of the two columns you want to exchange:"

read col1 col2

#adjust for zero based indexing

col1=$((col1 -1))

col2=$((col2 -1))

#validate the row numbers

if [ "$col1" -ge 0 ] && [ "$col1" -lt "$N" ] && [ "$col2" -ge 0 ] && [ "$col2" -lt "$N" ] && [ "$col1" -ne "$col2" ]

then

#exchange 2 columns

for ((i=0;i<N;i++))

do

temp="${grid[$((col1 \*N + i))]}"

grid[$((col1 \* N + i ))]="${grid[$((col2 \*N + i))]}"

grid[$((col2 \* N + i ))]=$temp

done

display\_grid

if [ $mark = "x" ]

then

player1\_score=$(( player1\_score - 1 ))

echo "Score updated: $player has $player1\_score points."

else

player2\_score=$(( player2\_score - 1 ))

echo "Score updated: $player has $player2\_score points."

fi

else

echo "Invalid column numbers or columns are the same.Please try again."

exchange\_columns "$player"

fi

}

#---------------------------------

exchange\_marks() {

local player=$1

local player\_mark=$2

local opponent\_mark=$3

local player\_row player\_col opponent\_row opponent\_col player\_index opponent\_index

# Prompt the player for the coordinate of their mark and opponent mark

echo "$player, enter the row and column numbers of your mark to exchange:"

read player\_row player\_col

echo "$player, enter the row and column numbers of opponent mark to exchange:"

read opponent\_row opponent\_col

# Adjust for zero-based indexing

player\_row=$((player\_row - 1))

player\_col=$((player\_col - 1))

opponent\_row=$((opponent\_row - 1))

opponent\_col=$((opponent\_col - 1))

# Calculate the indexes for the grid array

player\_index=$((player\_row \* N + player\_col))

opponent\_index=$((opponent\_row \* N + opponent\_col))

# Validate the coordinates and check if the cells contain the correct marks

if [ "$player\_row" -ge 0 ] && [ "$player\_row" -lt "$N" ] && [ "$player\_col" -ge 0 ] && [ "$player\_col" -lt "$N" ] && [ "$opponent\_row" -ge 0 ] && [ "$opponent\_row" -lt "$N" ] && [ "$opponent\_col" -ge 0 ] && [ "$opponent\_col" -lt "$N" ] && [ "${grid[player\_index]}" = "$player\_mark" ] && [ "${grid[opponent\_index]}" = "$opponent\_mark" ]

then

# Perform the exchange

temp=${grid[player\_index]}

grid[player\_index]=${grid[opponent\_index]}

grid[opponent\_index]="$temp"

# Display the updated grid

display\_grid

# Update the scores (assuming $mark should be $player\_mark)

if [ "$player\_mark" = "x" ]

then

player1\_score=$((player1\_score - 2))

echo "Score updated: $player has $player1\_score points."

else

player2\_score=$((player2\_score - 2))

echo "Score updated: $player has $player2\_score points."

fi

else

echo "Invalid coordinates or incorrect marks. Please try again."

exchange\_marks "$player" "$player\_mark" "$opponent\_mark"

fi

}

#---------------------------------

check\_alignments() {

local mark=$1

local alignments=0

#check horizontal alignments

for ((row=0;row<N;row++))

do

local count=0

for ((col=0;col<N;col++))

do

if [ "${grid[$((row \* N + col))]}" == "$mark" ]

then

count=$((count +1 ))

fi

done

if [ "$count" -eq "$N" ]

then

alignments=$((alignments +1))

fi

done

#check vertical alignments

for ((col=0;col<N;col++))

do

local count=0

for ((row=0;row<N;row++))

do

if [ "${grid[$((row \* N + col))]}" == "$mark" ]

then

count=$((count +1 ))

fi

done

if [ "$count" -eq "$N" ]

then

alignments=$((alignments +1))

fi

done

#check diagonal alignment

local count\_diag1=0

local count\_diag2=0

for ((i=0;i<N;i++))

do

if [ "${grid[$((i \* N +i))]}" == "$mark" ]

then

count\_diag1=$((count\_diag1 +1 ))

fi

if [ "${grid[$(((N -i -1)\*N +i))]}" == "$mark" ]

then

count\_diag2=$((count\_diag2 +1 ))

fi

done

if [ "$count\_diag1" -eq "$N" ]

then

alignments=$((alignments +1))

fi

if [ "$count\_diag2" -eq "$N" ]

then

alignments=$((alignments +1))

fi

return $alignments

}

#-------------------------------------

update\_scores() {

local player\_mark=$1

local opponent\_mark=$2

#check alignment for player's mark

check\_alignments "$player\_mark"

local player\_alignments=$?

if [ $player\_mark = "x" ]

then

player1\_score=$(( player1\_score + player\_alignments \* 2 ))

else

player2\_score=$((player2\_score + player\_alignments \* 2 ))

fi

#check alignment for opponent's mark

check\_alignments "$opponent\_mark"

local opponent\_alignments=$?

if [ $opponent\_mark = "x" ]

then

player1\_score=$(( player1\_score - opponent\_alignments \* 3 ))

else

player2\_score=$((player2\_score - opponent\_alignments \* 3 ))

fi

}

#---------------------------------------------------------

check\_for\_win() {

local mark=$1

if ! check\_alignments "$mark"

then

echo "Win detected for $mark"

return 0 # win detected

else

echo "No win for $mark"

return 1 # no win

fi

}

#---------------------------------------------------

display\_grid

# Main game loop with options

while true

do

# Player 1 turn

echo "It's $player1 's turn x."

if ! print\_menu "$player1" "x"

then

break # Exit the loop if player chooses to quit

fi

move\_count=$((move\_count+1))

if check\_for\_win "x"

then

echo "$player1 wins!"

break

elif [ "$move\_count" -ge "$max\_moves" ]

then

echo "The game is a draw."

break

fi

# Player 2 turn

echo "It's $player2 's turn o."

if ! print\_menu "$player2" "o"

then

break

fi

move\_count=$((move\_count+1))

if check\_for\_win "o"

then

echo "$player2 wins!"

break

elif [ "$move\_count" -ge "$max\_moves" ]

then

ech #!/bin/bash

player1=""

player2=""

player1\_score=0

player2\_score=0

move\_count=0

echo "Welcome to the XO game!"

echo "-----------------------"

#Players' Names

# Prompt players for their names

echo "Player 1, please enter your name:"

read player1

echo "Player 2, please enter your name:"

read player2

# Display players' names

echo "Player 1: $player1 (X)"

echo "Player 2: $player2 (O)"

#set game configration

echo "How many moves should the game last?"

read max\_moves

echo "Specify the dimentions of the grid (3,4,or 5):"

read N

if [[ $N -eq 3 || $N -eq 4 || $N -eq 5 ]]

then

echo "valid size"

else

echo "Invalid grid size.Please choose 3,4,or 5."

exit 1

fi

#Initialize a grid with 2D array

declare -a grid

for ((i=0;i<N\*N;i++))

do

grid[i]=' '

done

#Option to load from file

while true

do

echo "Do you want to start a new game or load from a file? (new/load)"

read choice

if [ "$choice" == "new" ]

then

break

elif [ "$choice" == "load" ]

then

echo "Enter the file name:"

read filename

if [ -f "$filename" ]

then

# Load game from file

echo "Loading game from file..."

i=0

while IFS='|' read -ra cells

do

for cell in "${cells[@]}"

do

grid[i]="$cell"

i=$((i+1))

done

done < "$filename"

break

else

echo "File not found. Please enter a valid file name."

fi

else

echo "Invalid option. Please choose 'new' or 'load'."

fi

done

#--------------------------------

#display the initial grid

display\_grid() {

for (( i=0;i<N\*N;i++))

do

if [ $((i%N)) -eq $((N-1)) ]

then

echo -n "${grid[i]} "

echo

else

echo -n "${grid[i]} |"

fi

done

}

#-----------------------------------

print\_menu() {

local player=$1

local mark=$2

local opponent\_mark

if [ "$mark" == "x" ]

then

opponent\_mark="o"

else

opponent\_mark="x"

fi

local choice

echo "It's a $player's turn ($mark)."

echo "choose your move:"

echo "1. place a mark "

echo "2.remove a mark ($mark)"

echo "3.Exchange rows"

echo "4.Exchange columns"

echo "5.Exchange marks"

echo "6.Quit the game"

echo "Enter your choice (1-6): "

read choice

case $choice in

1) place\_mark "$player" "$mark" ;;

2) remove\_mark "$player" "$mark" ;;

3) exchange\_rows "$player" ;;

4) exchange\_columns "$player" ;;

5) exchange\_marks "$player" "$mark" "$opponent\_mark" ;;

6) echo "Exiting the game."

return 1 ;;

\*) echo "Invalid option.Please try again." ;;

esac

update\_scores "$mark" "$opponent\_mark"

#Display the updated scores

echo "current scores:"

echo "$player1: $player1\_score"

echo "$player2: $player2\_score"

return 0

}

#--------------------------------------------

place\_mark() {

local player=$1

local mark=$2

local row col index

#prompt the player for coordinate

echo "$player, enter the row and column numbers to place your $mark:"

read row col

#asjust for zero based indexing

row=$((row-1))

col=$((col-1))

#claculate the index for grid

index=$((row \* N + col))

#validate the coordinates and check if the cell is empty

if [ "$row" -ge 0 ] && [ "$row" -lt "$N" ] && [ "$col" -ge 0 ] && [ "$col" -lt "$N" ] && [ "${grid[index]}" == " " ]

then

grid[index]=$mark

display\_grid

else

echo "Invalid move or cell is already occupied. please try again."

place\_mark "$player" "$mark"

fi

}

#-----------------------------------

remove\_mark() {

local player=$1

local mark=$2

local row col index

#prompt the player for coordinate

echo "$player, enter the row and column numbers of the mark you want to remove:"

read row col

#adjust for zero based indexing

row=$((row-1))

col=$((col-1))

#calc for index in grid

index=$((row \* N +col))

#validate the coordinates and check if cell contains content the players mark

if [ "$row" -ge 0 ] && [ "$row" -lt $N ] && [ "$col" -ge 0 ] && [ "$col" -lt $N ] && [ "${grid[index]}" == "$mark" ]

then

grid[index]=' '

display\_grid

#update the player's score

if [ $mark = "x" ]

then

player1\_score=$(( player1\_score + 1 ))

echo "Score updated: $player has $player1\_score points."

else

player2\_score=$(( player2\_score + 1 ))

echo "Score updated: $player has $player2\_score points."

fi

else

echo "Invalid move or cell does not contain your mark. please try again."

remove\_mark "$player" "$mark"

fi

}

#----------------------------------------

exchange\_rows() {

local player=$1

local row1 row2 temp i

#prompt the player for the row numbers

echo "$player,enter the numbers of the two rows you want to exchange:"

read row1 row2

#adjust for zero based indexing

row1=$((row1 -1))

row2=$((row2 -1))

#validate the row numbers

if [ "$row1" -ge 0 ] && [ "$row1" -lt "$N" ] && [ "$row2" -ge 0 ] && [ "$row2" -lt "$N" ] && [ "$row1" -ne "$row2" ]

then

#exchange 2 rows

for ((i=0;i<N;i++))

do

temp="${grid[$((row1 \*N + i))]}"

grid[$((row1 \* N + i ))]="${grid[$((row2 \*N + i))]}"

grid[$((row2 \* N + i ))]=$temp

done

display\_grid

if [ $mark = "x" ]

then

player1\_score=$(( player1\_score - 1 ))

echo "Score updated: $player has $player1\_score points."

else

player2\_score=$(( player2\_score - 1 ))

echo "Score updated: $player has $player2\_score points."

fi

else

echo "Invalid row numbers or rows are the same.Please try again."

exchange\_rows "$player"

fi

}

#-------------------------------------

exchange\_columns() {

local player=$1

local col1 col2 temp i

#prompt the player for the col numbers

echo "$player,enter the numbers of the two columns you want to exchange:"

read col1 col2

#adjust for zero based indexing

col1=$((col1 -1))

col2=$((col2 -1))

#validate the row numbers

if [ "$col1" -ge 0 ] && [ "$col1" -lt "$N" ] && [ "$col2" -ge 0 ] && [ "$col2" -lt "$N" ] && [ "$col1" -ne "$col2" ]

then

#exchange 2 columns

for ((i=0;i<N;i++))

do

temp="${grid[$((col1 \*N + i))]}"

grid[$((col1 \* N + i ))]="${grid[$((col2 \*N + i))]}"

grid[$((col2 \* N + i ))]=$temp

done

display\_grid

if [ $mark = "x" ]

then

player1\_score=$(( player1\_score - 1 ))

echo "Score updated: $player has $player1\_score points."

else

player2\_score=$(( player2\_score - 1 ))

echo "Score updated: $player has $player2\_score points."

fi

else

echo "Invalid column numbers or columns are the same.Please try again."

exchange\_columns "$player"

fi

}

#---------------------------------

exchange\_marks() {

local player=$1

local player\_mark=$2

local opponent\_mark=$3

local player\_row player\_col opponent\_row opponent\_col player\_index opponent\_index

# Prompt the player for the coordinate of their mark and opponent mark

echo "$player, enter the row and column numbers of your mark to exchange:"

read player\_row player\_col

echo "$player, enter the row and column numbers of opponent mark to exchange:"

read opponent\_row opponent\_col

# Adjust for zero-based indexing

player\_row=$((player\_row - 1))

player\_col=$((player\_col - 1))

opponent\_row=$((opponent\_row - 1))

opponent\_col=$((opponent\_col - 1))

# Calculate the indexes for the grid array

player\_index=$((player\_row \* N + player\_col))

opponent\_index=$((opponent\_row \* N + opponent\_col))

# Validate the coordinates and check if the cells contain the correct marks

if [ "$player\_row" -ge 0 ] && [ "$player\_row" -lt "$N" ] && [ "$player\_col" -ge 0 ] && [ "$player\_col" -lt "$N" ] && [ "$opponent\_row" -ge 0 ] && [ "$opponent\_row" -lt "$N" ] && [ "$opponent\_col" -ge 0 ] && [ "$opponent\_col" -lt "$N" ] && [ "${grid[player\_index]}" = "$player\_mark" ] && [ "${grid[opponent\_index]}" = "$opponent\_mark" ]

then

# Perform the exchange

temp=${grid[player\_index]}

grid[player\_index]=${grid[opponent\_index]}

grid[opponent\_index]="$temp"

# Display the updated grid

display\_grid

# Update the scores (assuming $mark should be $player\_mark)

if [ "$player\_mark" = "x" ]

then

player1\_score=$((player1\_score - 2))

echo "Score updated: $player has $player1\_score points."

else

player2\_score=$((player2\_score - 2))

echo "Score updated: $player has $player2\_score points."

fi

else

echo "Invalid coordinates or incorrect marks. Please try again."

exchange\_marks "$player" "$player\_mark" "$opponent\_mark"

fi

}

#---------------------------------

check\_alignments() {

local mark=$1

local alignments=0

#check horizontal alignments

for ((row=0;row<N;row++))

do

local count=0

for ((col=0;col<N;col++))

do

if [ "${grid[$((row \* N + col))]}" == "$mark" ]

then

count=$((count +1 ))

fi

done

if [ "$count" -eq "$N" ]

then

alignments=$((alignments +1))

fi

done

#check vertical alignments

for ((col=0;col<N;col++))

do

local count=0

for ((row=0;row<N;row++))

do

if [ "${grid[$((row \* N + col))]}" == "$mark" ]

then

count=$((count +1 ))

fi

done

if [ "$count" -eq "$N" ]

then

alignments=$((alignments +1))

fi

done

#check diagonal alignment

local count\_diag1=0

local count\_diag2=0

for ((i=0;i<N;i++))

do

if [ "${grid[$((i \* N +i))]}" == "$mark" ]

then

count\_diag1=$((count\_diag1 +1 ))

fi

if [ "${grid[$(((N -i -1)\*N +i))]}" == "$mark" ]

then

count\_diag2=$((count\_diag2 +1 ))

fi

done

if [ "$count\_diag1" -eq "$N" ]

then

alignments=$((alignments +1))

fi

if [ "$count\_diag2" -eq "$N" ]

then

alignments=$((alignments +1))

fi

return $alignments

}

#-------------------------------------

update\_scores() {

local player\_mark=$1

local opponent\_mark=$2

#check alignment for player's mark

check\_alignments "$player\_mark"

local player\_alignments=$?

if [ $player\_mark = "x" ]

then

player1\_score=$(( player1\_score + player\_alignments \* 2 ))

else

player2\_score=$((player2\_score + player\_alignments \* 2 ))

fi

#check alignment for opponent's mark

check\_alignments "$opponent\_mark"

local opponent\_alignments=$?

if [ $opponent\_mark = "x" ]

then

player1\_score=$(( player1\_score - opponent\_alignments \* 3 ))

else

player2\_score=$((player2\_score - opponent\_alignments \* 3 ))

fi

}

#---------------------------------------------------------

check\_for\_win() {

local mark=$1

if ! check\_alignments "$mark"

then

echo "Win detected for $mark"

return 0 # win detected

else

echo "No win for $mark"

return 1 # no win

fi

}

#---------------------------------------------------

display\_grid

# Main game loop with options

while true

do

# Player 1 turn

echo "It's $player1 's turn x."

if ! print\_menu "$player1" "x"

then

break # Exit the loop if player chooses to quit

fi

move\_count=$((move\_count+1))

if check\_for\_win "x"

then

echo "$player1 wins!"

break

elif [ "$move\_count" -ge "$max\_moves" ]

then

echo "The game is a draw."

break

fi

# Player 2 turn

echo "It's $player2 's turn o."

if ! print\_menu "$player2" "o"

then

break

fi

move\_count=$((move\_count+1))

if check\_for\_win "o"

then

echo "$player2 wins!"

break

elif [ "$move\_count" -ge "$max\_moves" ]

then

echo "The game is a draw."

break

fi

done

#---------------------------------------

#Display final message

echo "Game over.Final scores:"

echo "$player1: $player1\_score"

echo "$player2: $player2\_score"

o "The game is a draw."

break

fi

done

#---------------------------------------

#Display final message

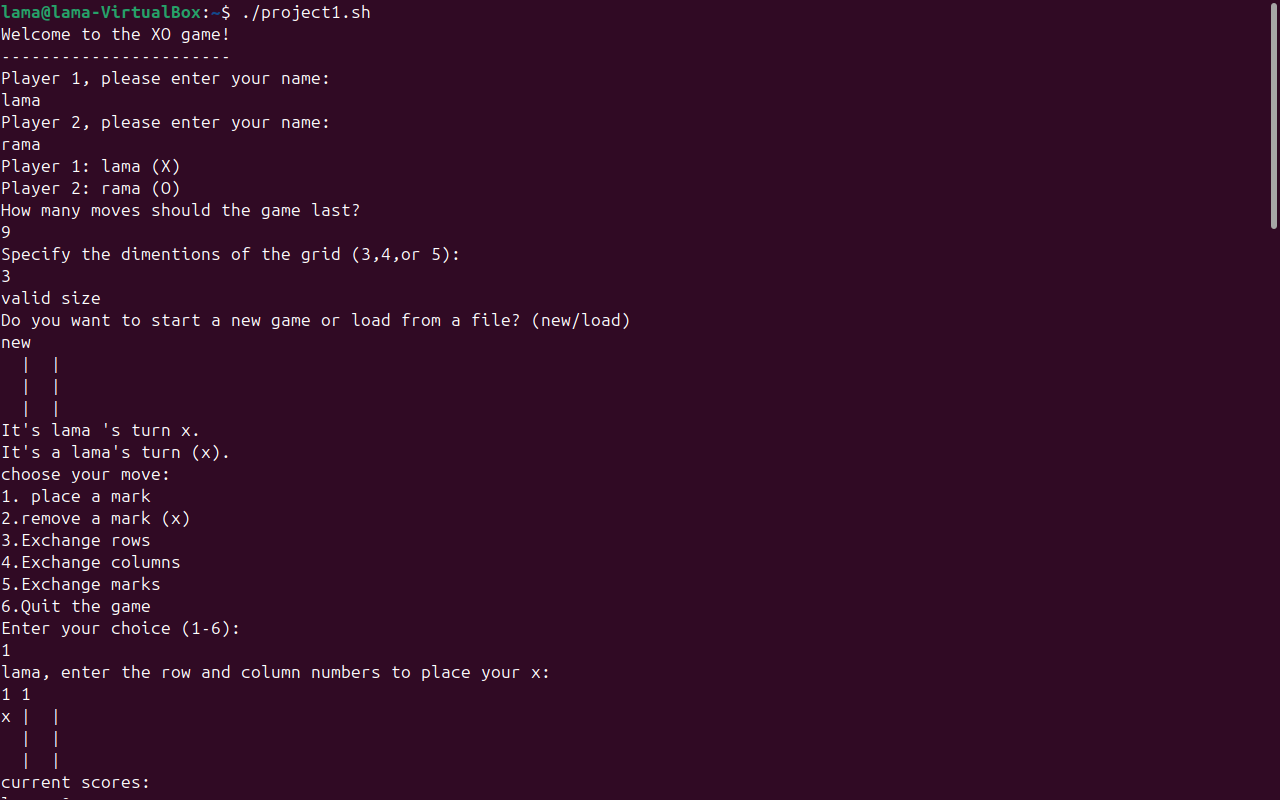
echo "Game over.Final scores:"

echo "$player1: $player1\_score"

echo "$player2: $player2\_score"

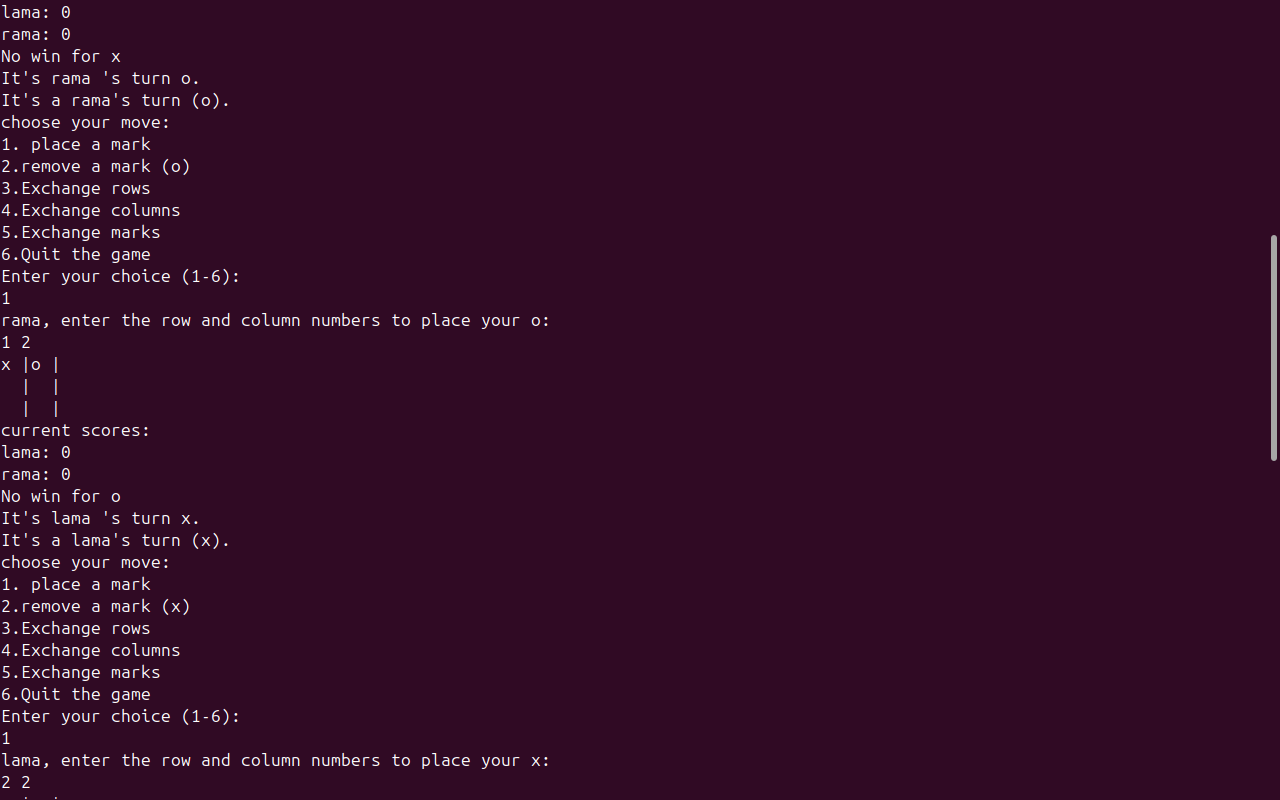
**Screenshots: Include screenshots of the game screen and game grid demonstrating various gaming scenarios.**

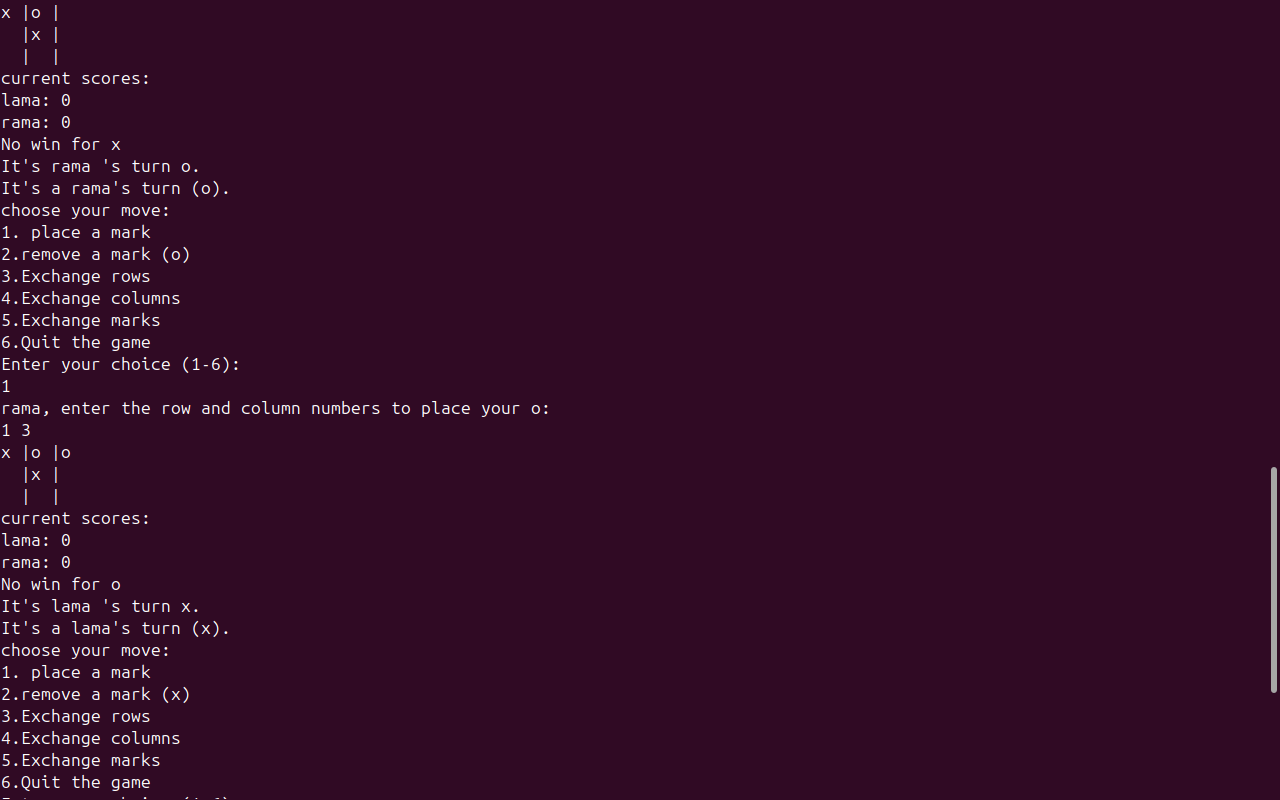
Initial Game Setup and First Move in the XO Game:

****

initial setup and first move in a terminal-based XO game. Players Lama (X) and Rama (O) enter their names and select a 3x3 grid. Lama makes the first move by placing 'X' in row 1, column 1. The updated grid and current scores are displayed.

Gameplay Progression in the XO Game.



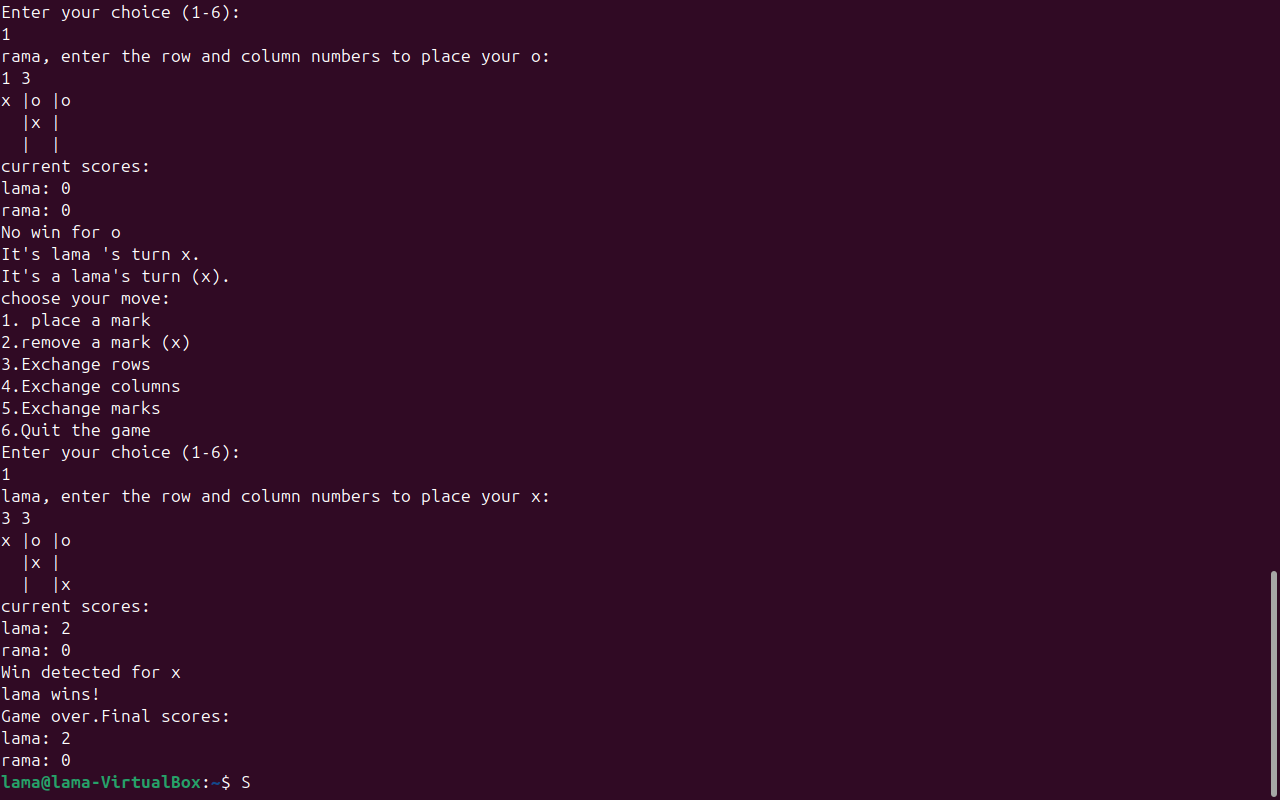


The game has ended with Lama winning by aligning three 'X' marks vertically in the middle column of the 3x3 game board.

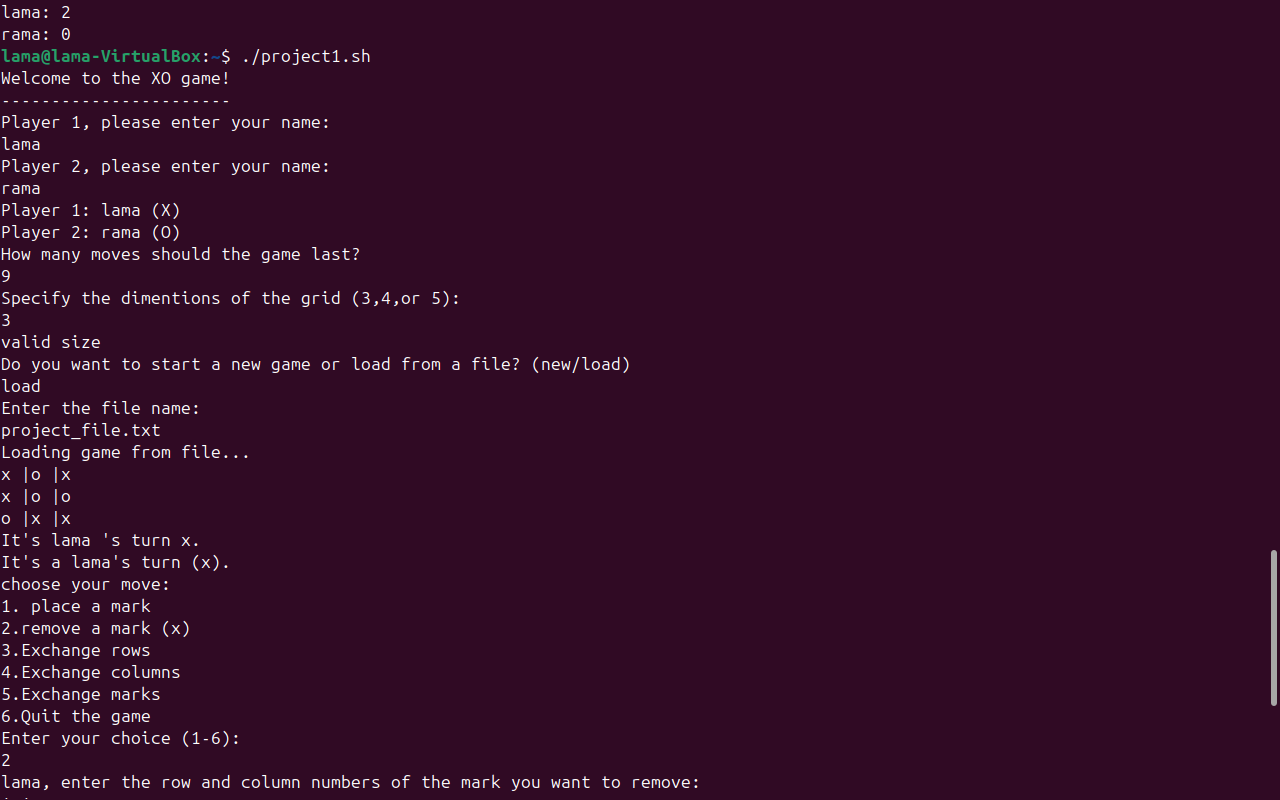
The program displayed the final score: (based on score rules)

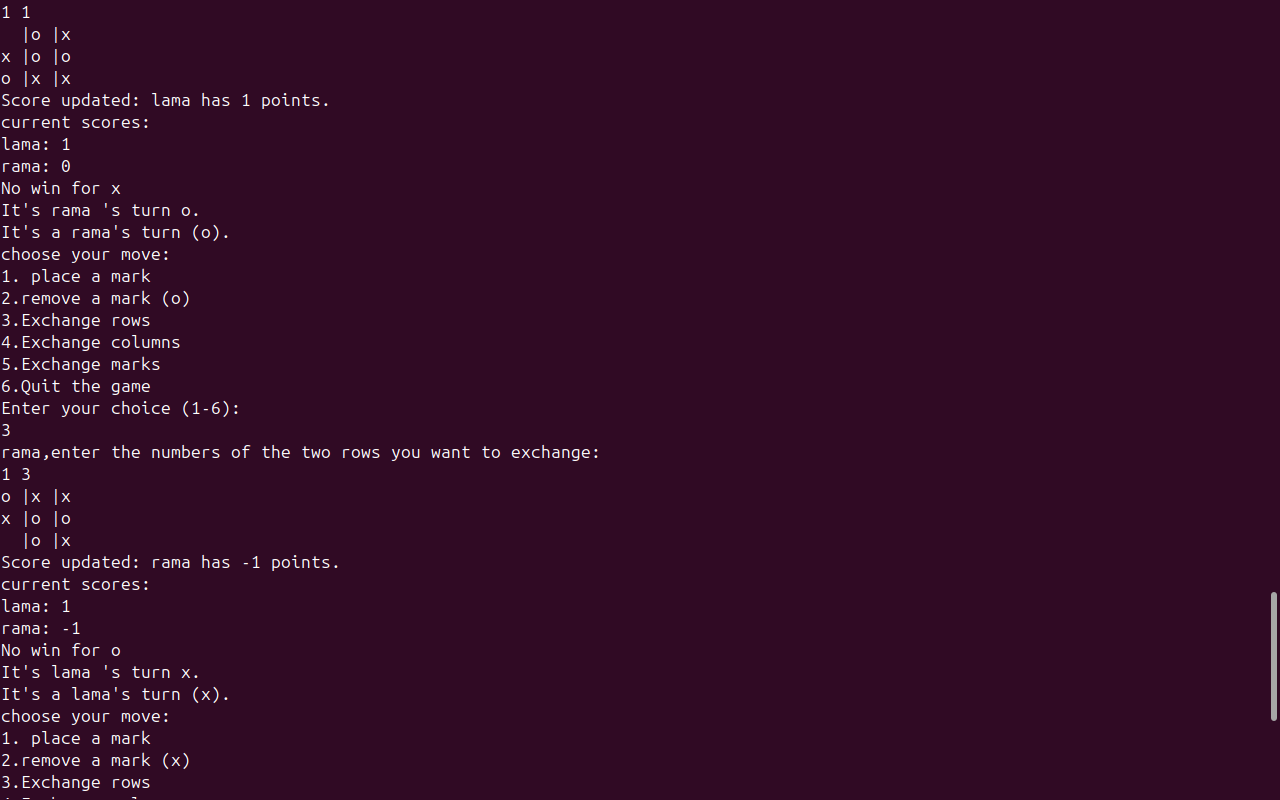
Lama: 2

Rama: 0



Players choose to start a new game or load a game from a file. Here, a grid is loaded from a file named **project\_file.txt**



****

Lama chooses the option to remove a mark (move 2), specifies the position [1,1] to remove an 'X'.

This action updates Lama's score to 1 point.

****

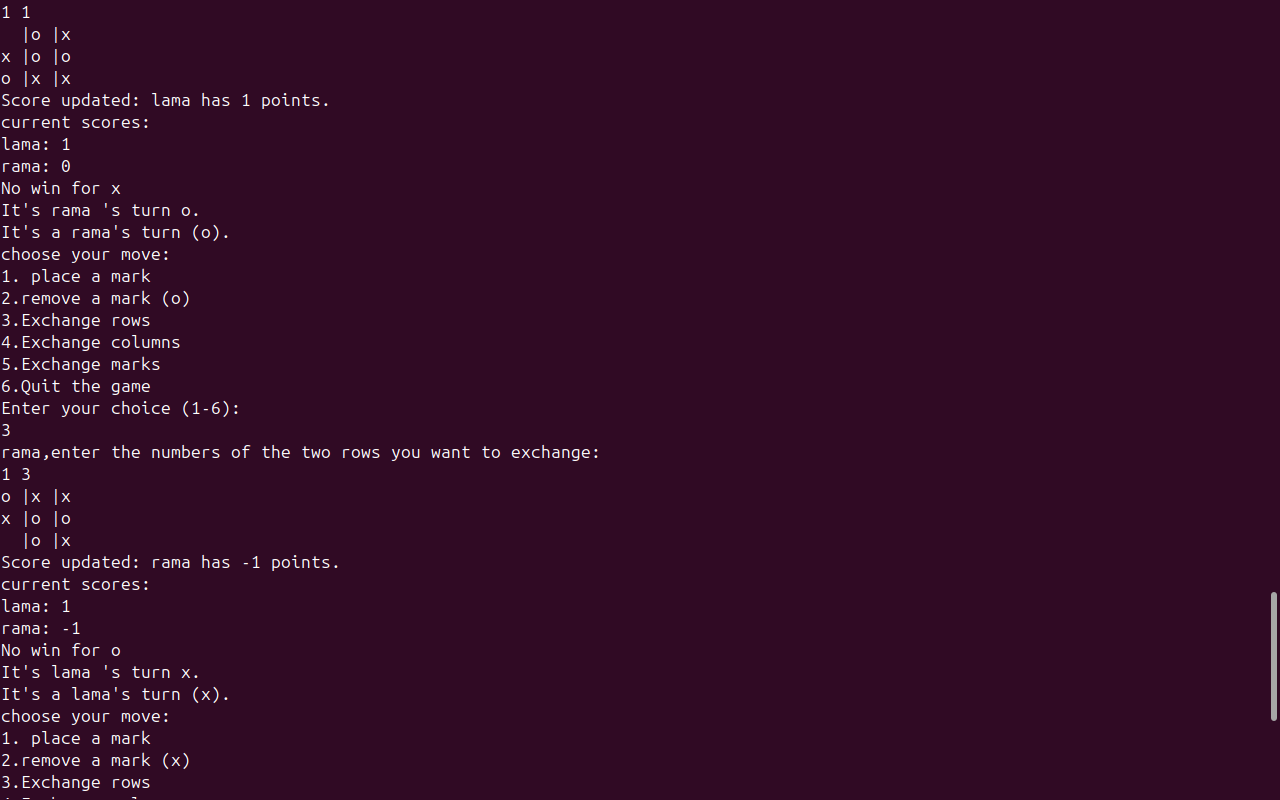
Rama chooses option 3, "Exchange rows, enters the numbers of the two rows to be exchanged and selects rows 1 and 3.

Current score:

Lama: 1

Rama: -1

“Players are penalized with one point for playing Move 3 or Move 4”

****