

**ENCS, Computer Systems Engineering**

**ENCS3130, LINUX LABORATORY**

**Project 1 Shell script**

Prepared by:

Sahar Hmidat ID: 1202038

Instructor’s Name: Dr. Mohammad Jubran

Assistant Name: Tareq

Section No: 1

Date: 27/4/2024

Table of Contents

[**Introduction** IV](#_Toc165150150)

[**Program Implementation** V](#_Toc165150151)

[**Running the Program** XI](#_Toc165150152)

# List of figures

[Figure 1 Main V](#_Toc165150170)

[Figure 2 variables VI](#_Toc165150171)

[Figure 3 loopStart VI](#_Toc165150172)

[Figure 4 gridFunction VII](#_Toc165150173)

[Figure 5 startGame VII](#_Toc165150174)

[Figure 6 startGameFile VIII](#_Toc165150175)

[Figure 7 winFunction VIII](#_Toc165150176)

[Figure 8 winFunctionCont IX](#_Toc165150177)

[Figure 9 score IX](#_Toc165150178)

[Figure 10 moveOne X](#_Toc165150179)

[Figure 11 moveTwo X](#_Toc165150180)

[Figure 12 moveThree X](#_Toc165150181)

[Figure 13 moveFour XI](#_Toc165150182)

[Figure 14 Run1 XI](#_Toc165150183)

[Figure 15 Run2 XII](#_Toc165150184)

[Figure 16 Run3 XII](#_Toc165150185)

[Figure 17 Run4 XII](#_Toc165150186)

[Figure 18 Run5 XIII](#_Toc165150187)

[Figure 19 Run6 XIII](#_Toc165150188)

[Figure 20 Run7 XIII](#_Toc165150189)

[Figure 21 Run8 XIV](#_Toc165150190)

[Figure 22 Run9 XIV](#_Toc165150191)

[Figure 23 Run10 XV](#_Toc165150192)

[Figure 24 Run11 XV](#_Toc165150193)

# **Introduction**

This project is called X-O Game and consists of a board of NxN size and two opposing players and each one of them tries to win the game by placing his position in a certain state. The game contains five modes, the first is to add the special mark in a specific place, the second mode is to remove a specific mark from the place, the third mode is to change a row with another row, the fourth mode is to replace a column with a column, and the last mode is to change a place with another place.

# **Program Implementation**

We used shell script language to implement this game using Linux. We first start the main function for the user to enter player names and choose what they want from the directory. The grid has special character parameters. The board is a two-dimensional array, which includes only three ASCII characters: the 'X' and 'O' characters, plus the delimiter '|' With no spaces between them

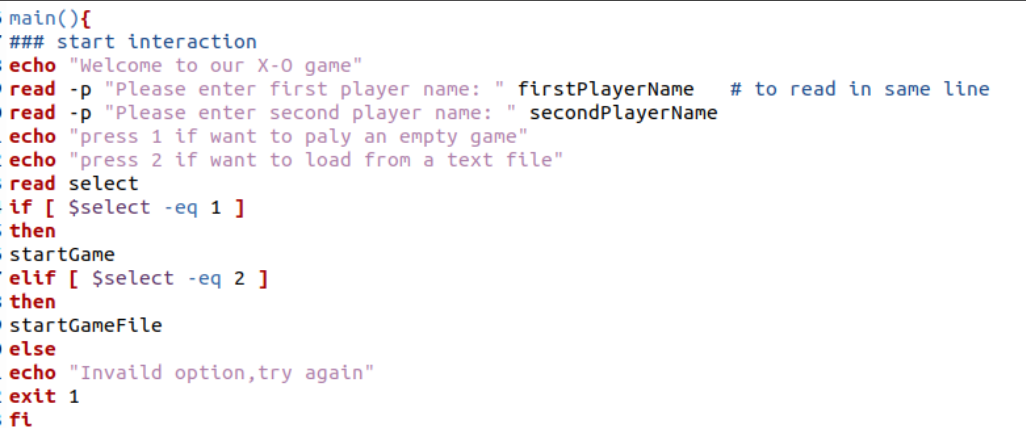


Figure 1 Main

Then we assigned these values to zero to update later when game start and each player do own action and earn points

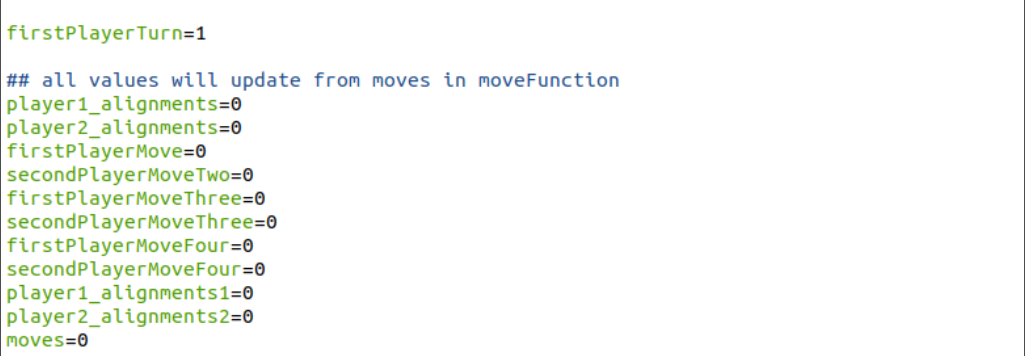


Figure 2 variables

After that, we passed the parameter (move) from the user to specify the number of rounds to finish the game, and then play the game based on this number. After reaching the last round, the score will be printed for each player and the winner will be determined.

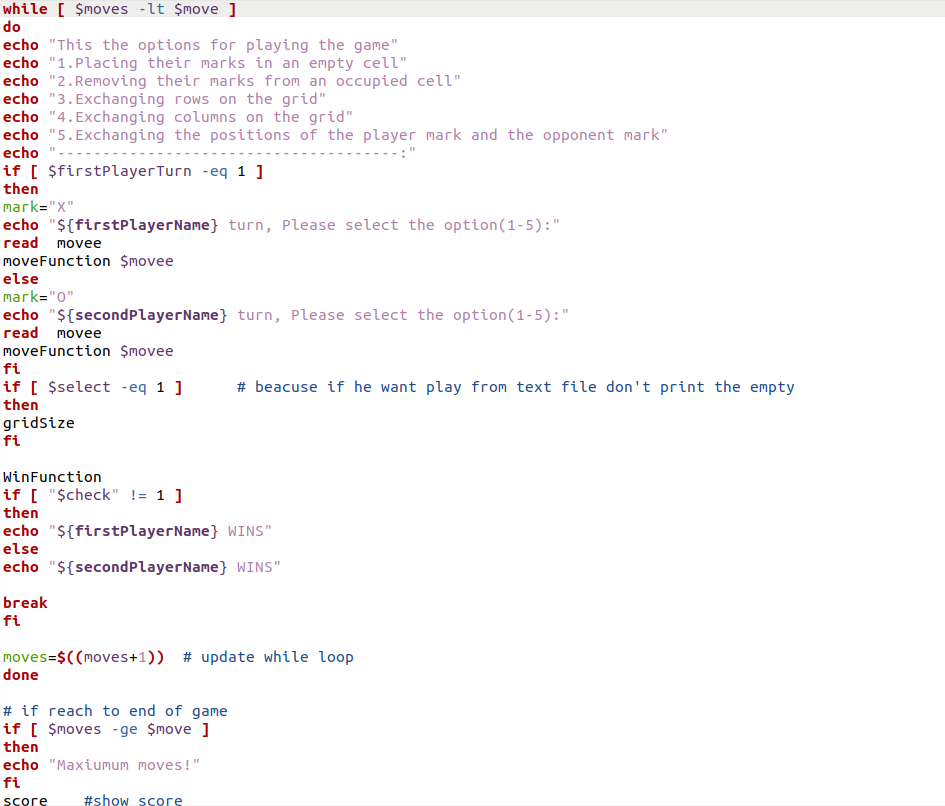


Figure 3 loopStart

The Gridsize function will print the panel shape that includes (|) The delimiter and the space between each (|) Based on user defined size.

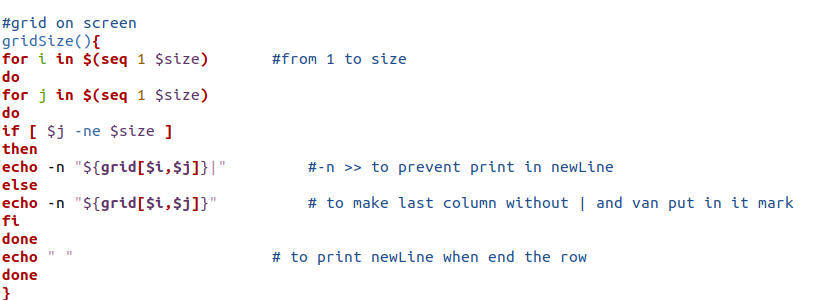
-n >> we used to prevent print in new line

Figure 4 gridFunction

The user can choose the option to play in a empty board or load a board from a text file.

If he chooses the first option, we configured each index in the array to be printed with an empty character so we put a space and then the user will fill this panel with X-O marks.

Here the user is asked to choose the board size and maximum number of game finishes

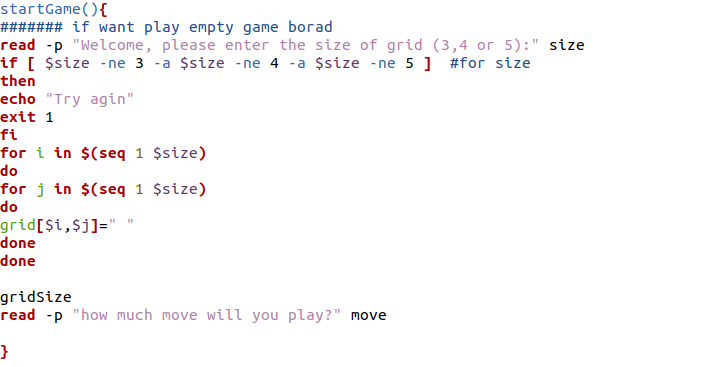


Figure 5 startGame

If he chooses the second option, will load a text file depends on size.



Figure 6 startGameFile

winFunction will check rows, columns and diagonals, it iterates for each row and column and diagonal and check if the place contains own mark or not

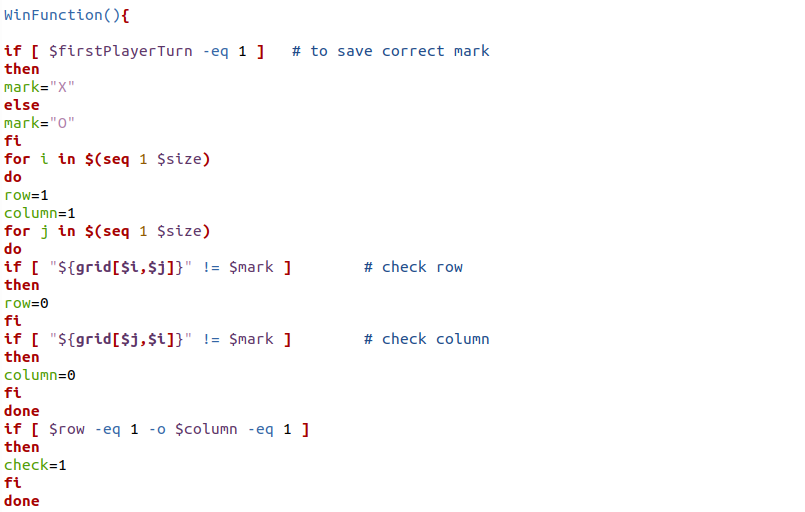


Figure 7 winFunction

Here will check left and right diagonal.

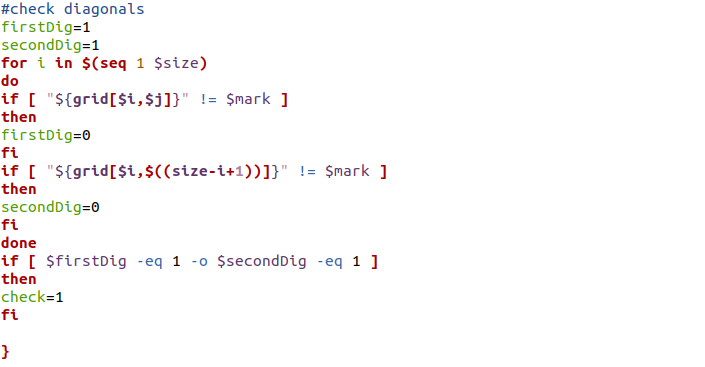


Figure 8 winFunctionCont

score function will calculate the score for each player, the value of variables will update from moveFunction and then print each score.

firstPlayerScore=$((2 \* player1\_alignments + player1\_alignments1+ firstPlayerMove - firstPlayerMoveThree - - 2 \* secondPlayerMoveFour))

secondPlayerScore=$((2 \* player2\_alignments + player2\_alignments2 + secondPlayerMoveTwo - secondPlayerMoveThree - 2 \* secondPlayerMoveFour))

if player choose move one here will either earn tow point for any alignment or lose three points for any alignment for opponent’s mark so we put (2 \* player1\_alignments) if earn two points because (player1\_alignments) will increment one, but (player1\_alignments1) will decrement three

if user chooses move two will earn one point (firstPlayerMove).

If user chooses move three or four will lose one point, so we declared the variable (-firstPlayerMoveThree) to assign any lose from these moves.

If user chooses move five will lose two points. (- 2 \* secondPlayerMoveFour)

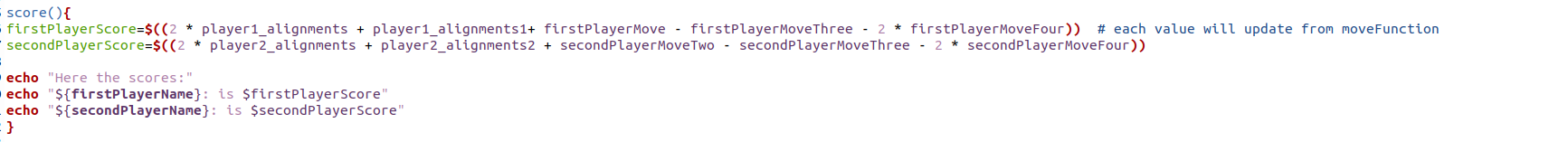


Figure 9 score

Here if player chooses move one, will ask player to determine the position of row and column to put the mark and check if this position in range and it null then will update the values for score function.



Figure 10 moveOne

if player chooses move 2 will asked the player to enter position to remove the mark and check if the position in range and contains the mark then update the values for score function.

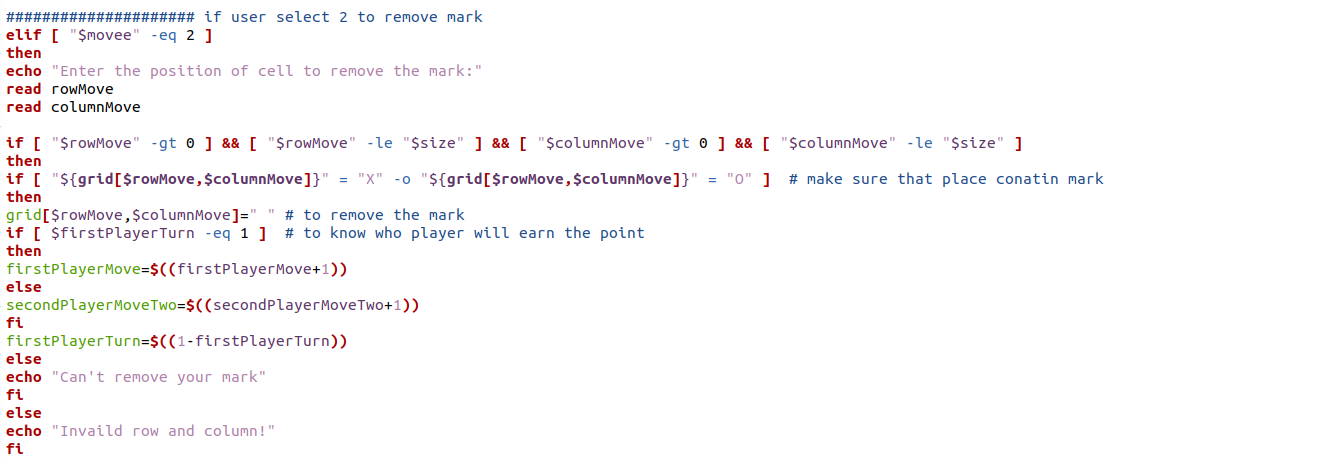


Figure 11 moveTwo

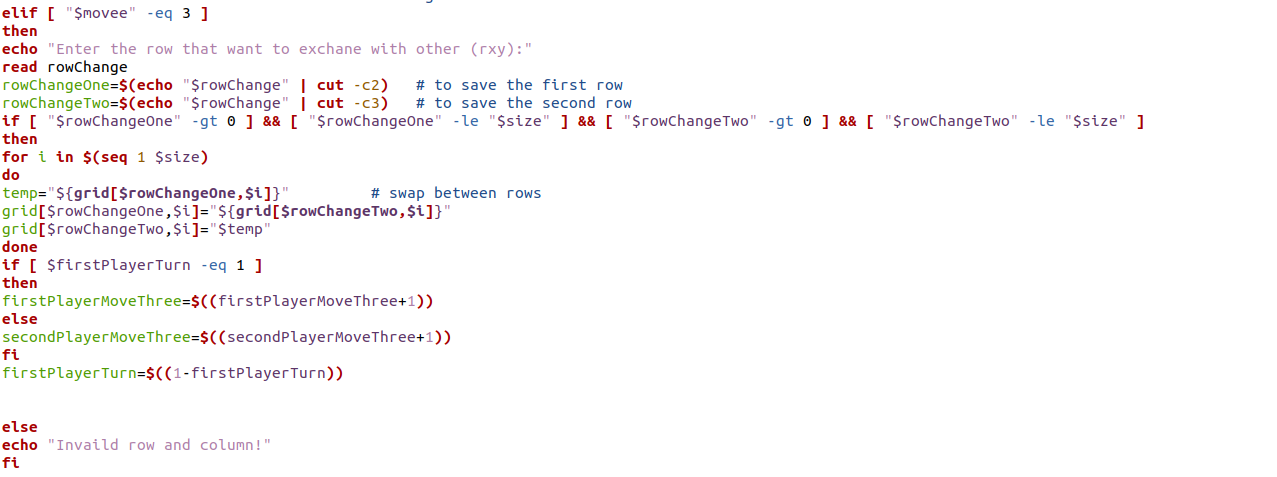
If player chooses move three, first we cut the (rxy) format to determine the row and column then check if they in range and made swap to exchange the rows after that will update the value of variables for score function.

Figure 12 moveThree

If player chooses move four, we did the same thing like previous part put with change with index of array.

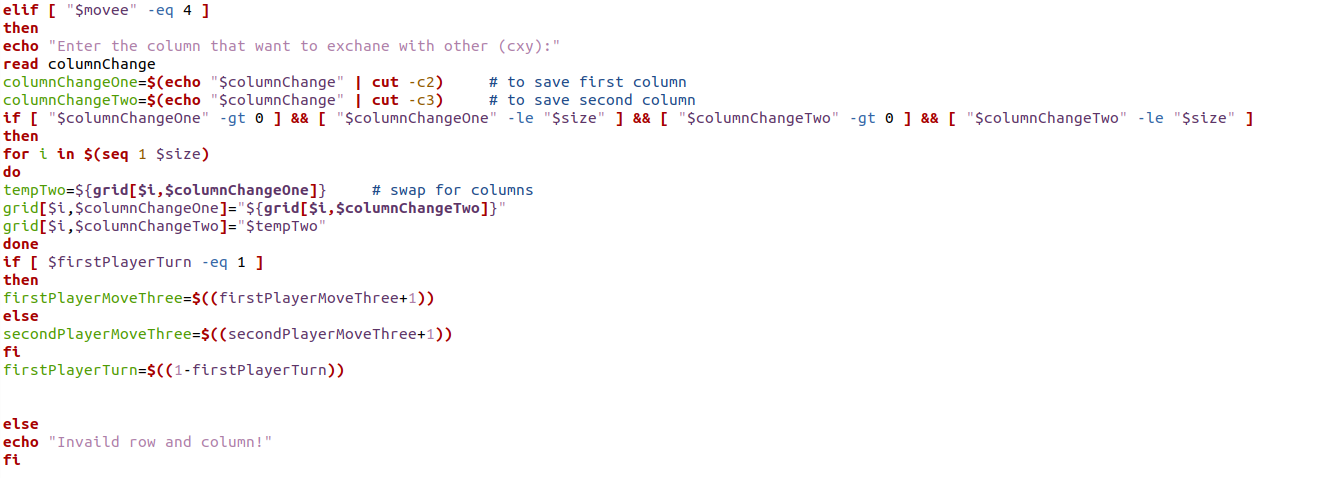


Figure 13 moveFour

If player chooses move five, will cut (exyuv) format to determine the first position that will replace with second position and check if the positions in range and update the value of variables foe score function



# **Running the Program**

Here the shape of board with 3,4 and 5 size

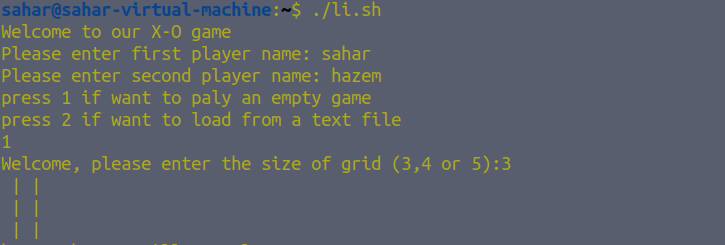


Figure 14 Run1

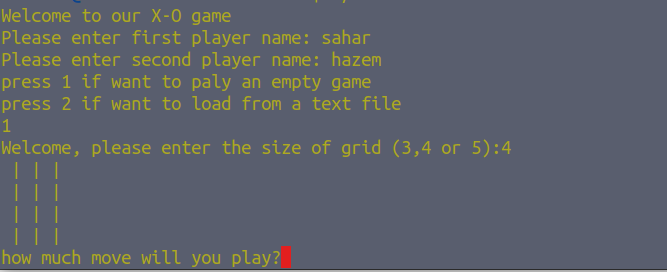


Figure 15 Run2

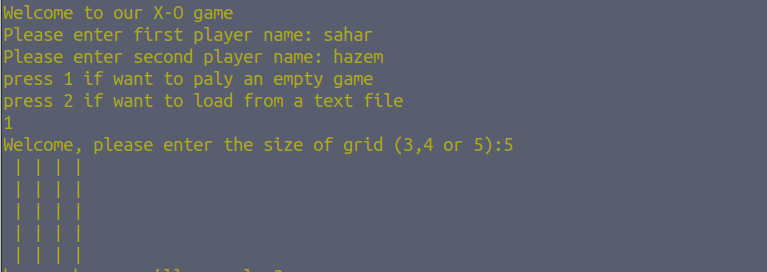


Figure 16 Run3

Here if chooses the option of load a text file

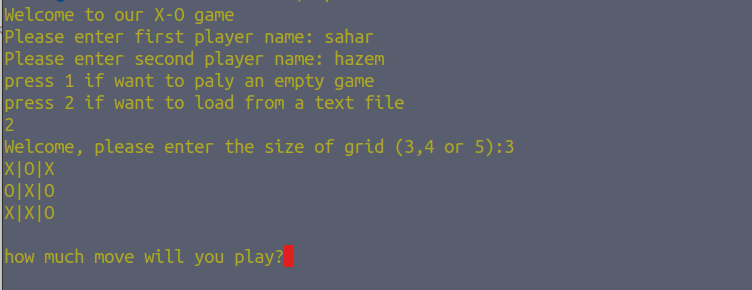


Figure 17 Run4

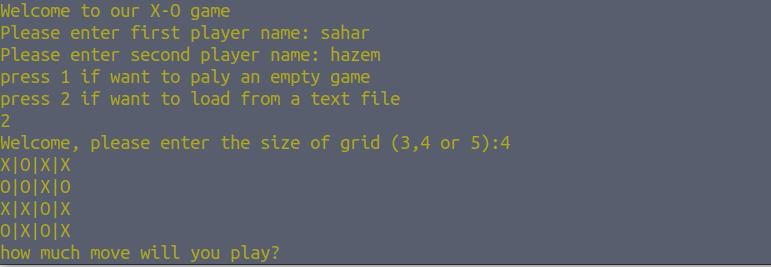
****

Figure 18 Run5

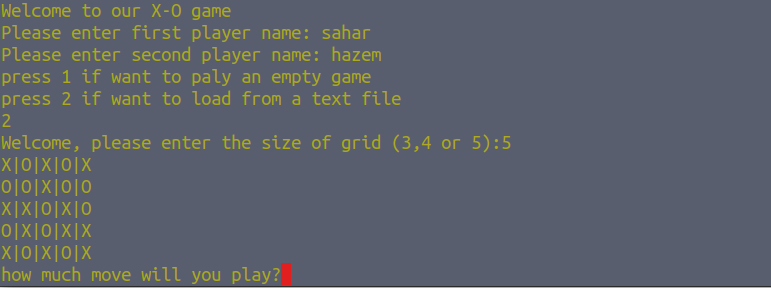


Figure 19 Run6

If chooses move one

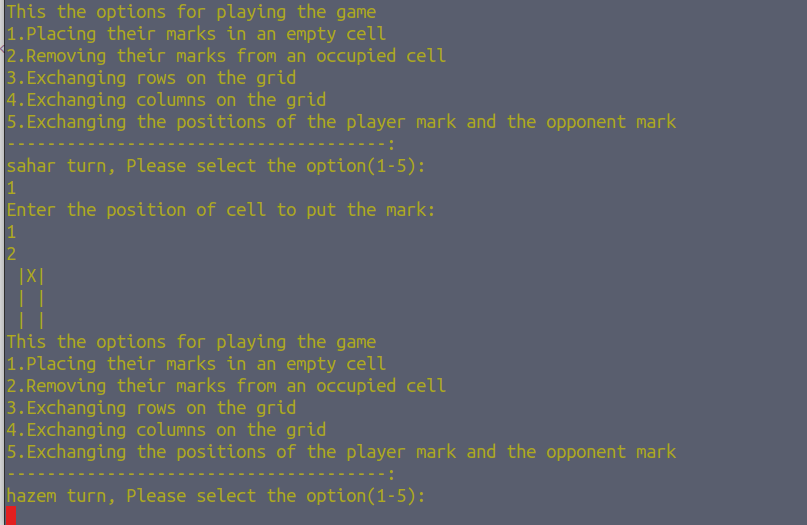


Figure 20 Run7

If chooses move two

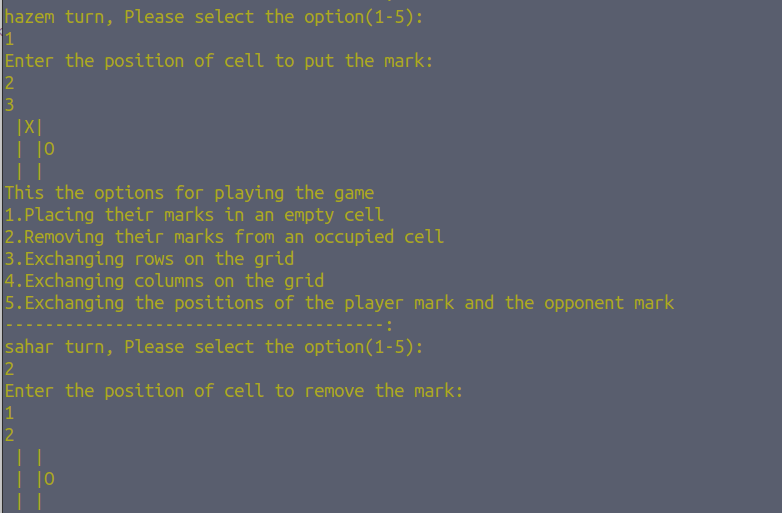


Figure 21 Run8

If chooses move three

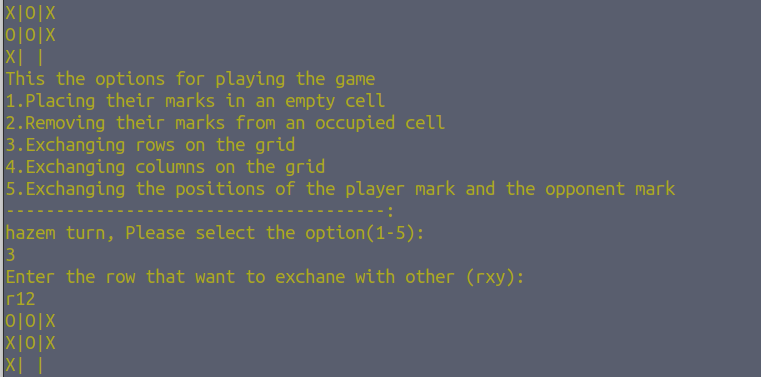


Figure 22 Run9

If chooses move four

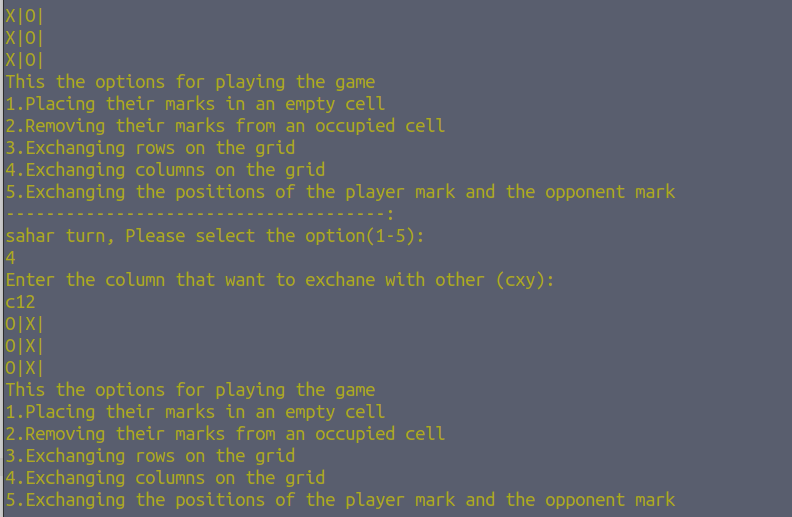


Figure 23 Run10

If chooses move five

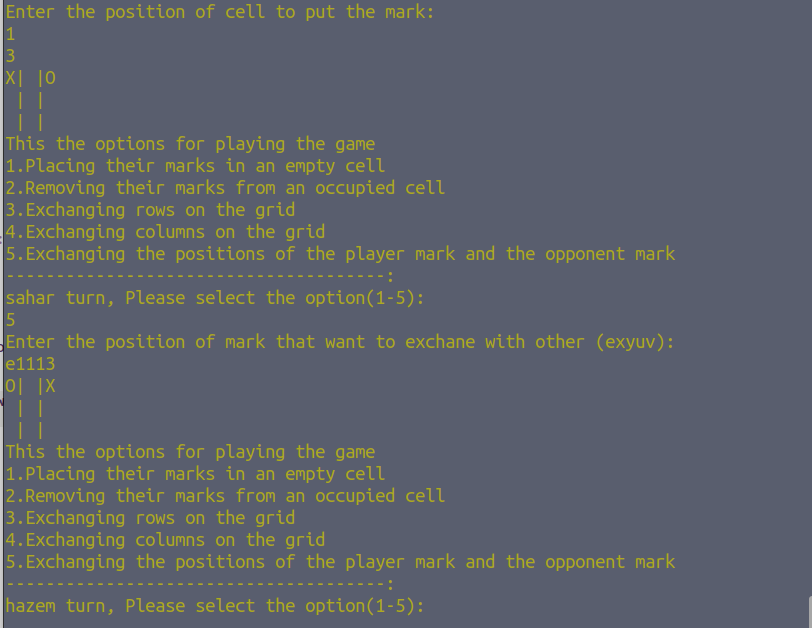


Figure 24 Run11

If choose play from text file

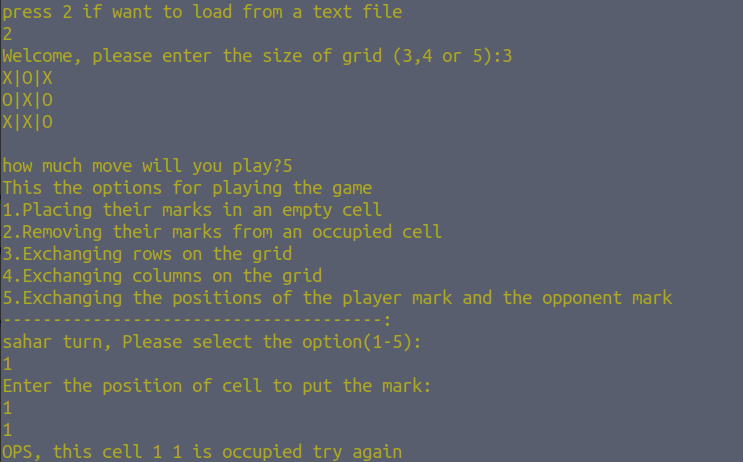


Figure 25 Run12