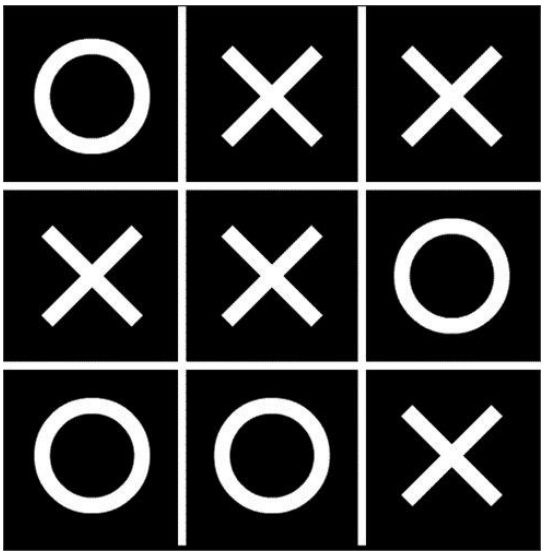
תיכון ניסויי ע"ש ב. אוסטרובסקי

סמל ביה"ס: 440024

אסמבלר

פרוייקט מסכם בנושא:

איקס – עיגול



מגיש: ירין סמואל

ת.ז.: 322605296

כיתה: י'

מורה מנחה: בן-דוד דורית

מאי, 2017

אייר, תשע"ז

מה התוכנית עושה?

פרויקט הסיום שלי הינו משחק איקס עיגול בין אדם לבינה מלאכותית. הבינה המלאכותית מחשבת את כל המהלכים האפשריים עד לסוף המשחק ויוצרת מיין עץ תרחישים, לאחר מכן היא נותן לכל תוצאה סופית ניקוד (ניצחון הכי הרבה נקודות, אחר כך תיקו ואז הפסד) ובכל תור מניחה שהשחקן יבצע את המהלך הטוב ביותר בשבילו.
ניתן גם להוריד את רמת הקושי על ידי הגבלת מספר הצעדים קדימה שהמחשב מחשב.

כיצד היא עושה זאת?

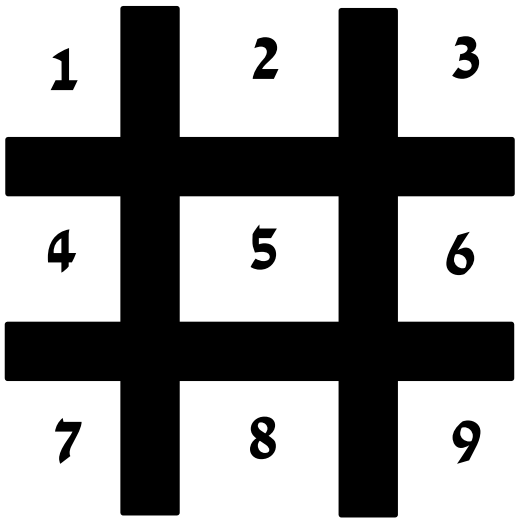
מימשתי את אלגוריתם זה בעזרת רקורסיה, קיימת פונקציה ai_main אשר מנהלת את הרקורסיה והיא קוראת לפונקציה ai_o תשעה פעמיים וכל פעם מעבירה לה את המספר המתאים בין 1 ל9 (כולל).

הפונקציה ai_o מנסה למקם עיגול במקום שהעבירה לה ובמידה והצליחה היא בודקת אם המשחק נגמר ואם כן היא מחזירה כיצד המשחק נגמר (ניצחון הפסד או תיקו) ואם לא קוראת לפונקציה ai_x (אשר פועלת בצורה דומה אך לאיקס במקום עיגול) אז הפונקציה ai_o בודקת מה הערך הגרוע ביותר שai_x החזירה ומחזירה אותו (היא יוצאת מנקודת הנחה שהשחקן השני יבצע את המהלך הטוב ביותר בשבילו שהוא בעצם התוצאה הגרועה ביותר לעיגול, בניגוד לכך ai_x בוחר את התוצאה הטובה ביותר מאותה סיבה), הפונקציה גם סופרת כמה תורות עברו ובמידה ומספר זה גדול המספר שהוגדר לפי רמת הקושי היא מחזירה ערך הזהה לתיקו.

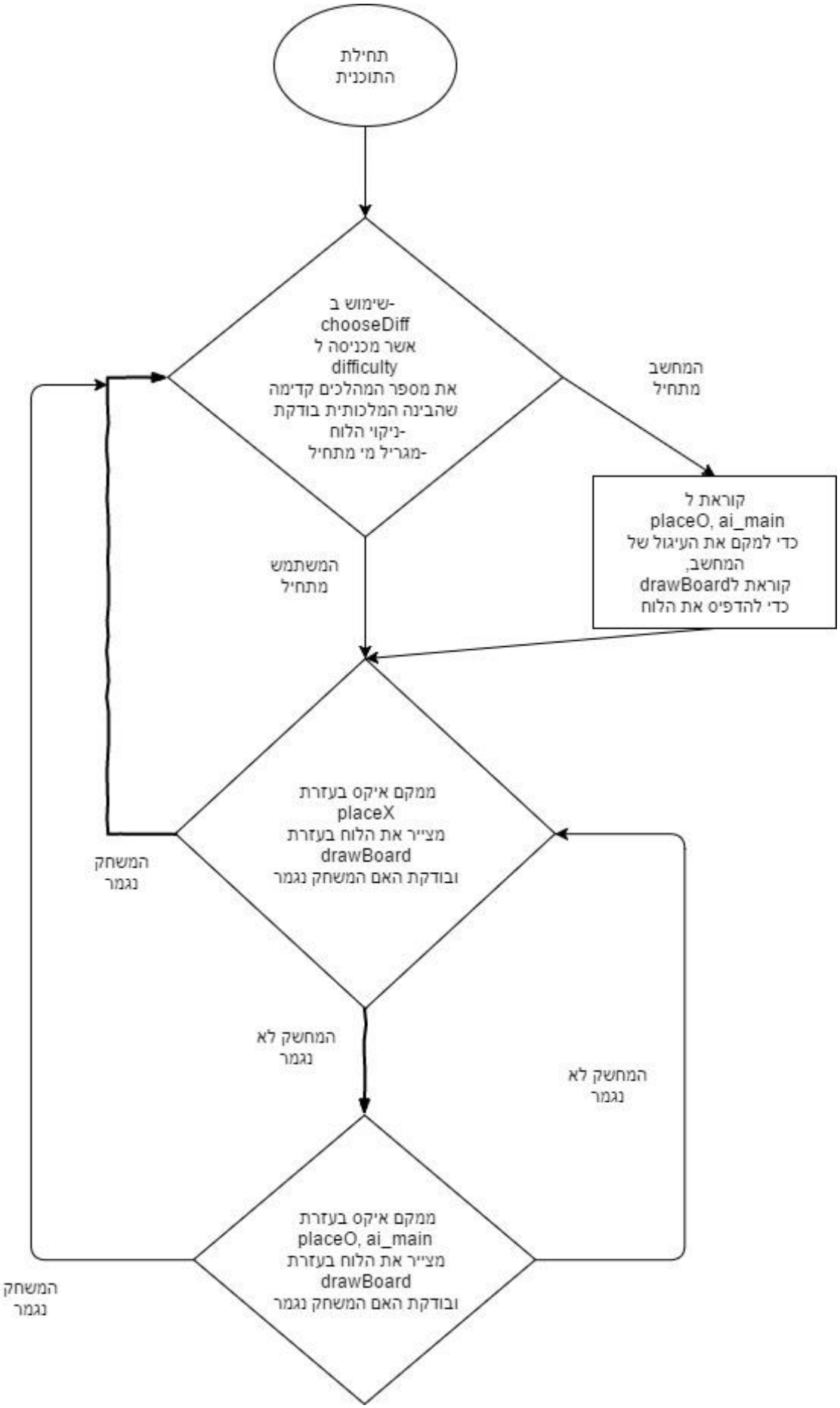
לאחר מכן הערכים האלו חוזרים לai_main אשר בוחר את המהלך הטוב ביותר מבניהם (ובמידה ויש כמה שקולים אז אחד אקראי) ומחזיר את המיקום של המהלך.

כיצד הממשק עובד?

תוכנה זו משתמשת בממשק של קונסולה ומדפיסה בכל תור את הלוח החדש. כדי לבצע מהלך המשתמש צריך להכניס את המספר של המשבצת משמאל לימין (כמו שניתן לראות בציור).



תרשים זרימה:


תחילת
התוכנית
-שימוש ב
chooseDiff
אשר מכניסה ל
difficulty
את מספר המהלכים קדימה
שהבינה המלכותית בודקת
-ניקוי הלוח
-מגריל מי מתחיל
המחשב
מתחיל
קוראת ל
placeO, ai_main
כדי למקם את העיגול של
המחשב,
קוראת לdrawBoard
כדי להדפיס את הלוח
המשתמש
מתחיל
ממקם איקס בעזרת
placeX
מצייר את הלוח בעזרת
drawBoard
ובודקת האם המשחק נגמר
המשחק
נגמר
המחשק לא
נגמר
המחשק לא
נגמר
ממקם איקס בעזרת
placeO, ai_main
מצייר את הלוח בעזרת
drawBoard
ובודקת האם המשחק נגמר
המשחק
נגמר

הקוד

```
IDEAL
MODEL small
STACK 256
DATASEG
	defualt equ '*'
	boolean equ dh
	input equ dl
	true equ 1
	false equ 2
	boardSize equ 3
	boardSpace equ 9
	winning equ 3
	tie equ 2
	losing equ 1
	illegal equ 0

	tieMessage db 13,10,'ah... ties, thoose are the
worst!',13,10,'$'
	oWinMessage db 13,10,'O won this time, don',96,'t worry,
you will beat him next time!',13,10,'$'
	xWinMessage db 13,10,'Wow good job for winning, humanity
still has hope!',13,10,'$'
	errorMessage db 13,10,'You entered an invalid value, please
try agian',13,10,'$'
	startMessage db 13,10,'Thank you for playing Tic-Tac-Toe
by Yarin Samuel,',13,10,'Plz enter the number of the square you
want to place your X in',13,10,'(they are numbered from left to
right)',13,10,'$'
	OturnMessage db 13,10, 'O',96,'s turn:',13,10,'$'
	XturnMessage db 13,10, 'X',96,'s turn:',13,10,'$'
	diffMessage db 13,10, 'Choose your opponent
diffuclty:',13,10,'a) Easy',13,10,'b) Medium',13,10,'c) Hard',13,10,'$'
	board db 10 dup (defualt)
	tempboard db 10 dup (defualt)
	address dw ?
	address1 dw ?
	tempAx dw ?
	tempBx dw ?
	tempCx dw ?
	var dw ?
	bestMove db ?
	bestScore db ?
	diffuclty dw ?
```

```
CODESEG
start:
        mov ax, @data
        mov ds,ax

        mov dx, offset startMessage
        mov ah, 9h
        int 21h

@newGammeloop:
        call clearBoard
        mov dx, offset diffMessage
        mov ah, 9h
        int 21h
        call chooseDiff
        ; Randomly desides who starts
        mov  ax, 40h
        mov  es, ax
        mov  ax,  [es:6Ch]
        and al, 00000001b
        cmp al, 1
        je @turnLoop

        mov dx, offset OturnMessage
        mov ah, 9h
        int 21h

        call ai_main
        call placeO
        call drawBoard
        je @newGammeloop
@turnLoop:
        mov dx, offset xturnMessage
        mov ah, 9h
        int 21h

        call placeX
        call drawBoard
        call boardState
        cmp boolean, true
        je @newGammeloop

        mov dx, offset OturnMessage
        mov ah, 9h
```

```
        int 21h
        call ai_main
        call placeO
        call drawBoard
        call boardState
        cmp boolean, true
        je @newGammeloop

        jmp @turnLoop

exit:
        mov ax, 04C00h
        int 21h

;-----------------------------------------------
;-- End of Main Program ... Start of Procedures
;-----------------------------------------------
proc ai_main
        push offset board
        call pushArray
        push offset tempboard
        call popArray
        push ax
        push bx
        push cx

        xor cx,cx
        mov bx, 1
@ai_main_Loop:
        call ai_o
        cmp cl, dl
        ja @notBetter
        je @same
        mov cl, dl
        mov [byte ptr offset bestMove], bl
        mov [byte ptr offset bestScore], cl
        jmp @notBetter
@same:
        ; Randomly choose from equale moves
        mov  ax, 40h
        mov  es, ax
        mov  ax, [es:6Ch]
        and al, 0000001b
        cmp al, 1
        jne @notBetter
```

```
        mov [byte ptr offset bestMove], bl
        mov [byte ptr offset bestScore], cl

@notBetter:
        inc bx
        cmp bx, boardSpace + 1
        jne @ai_main_Loop

        pop cx
        pop bx
        pop ax
        ret
endp ai_main
;-----------------------------------------------------------------------------------------
; Returns in dl the outcome of the board, gets the location in bx
proc ai_o
        push offset tempboard
        call pushArray
        push bx
        push ax
        push [diffuclty]
        dec [diffuclty]

        mov input , 'o'
        call pseudoPlace
        cmp boolean, true
        je @o_successfulPlace
        mov dl, illegal
        jmp @o_endThisCase
@o_successfulPlace:

        push offset tempboard
        call isWinning
        cmp boolean, false
        je @o_notWinning
        mov dl, winning
        jmp @o_endThisCase
@o_notWinning:

        push offset tempboard
        call isTie
        cmp boolean, false
        je @o_notTie
        mov dl, tie
        jmp @o_endThisCase
```

```
@o_notTie:

        push offset tempboard
        mov dl, 'x'
        call isWinning
        cmp boolean, false
        je @o_notLosing
        mov dl, losing
        jmp @o_endThisCase
@o_notLosing:

        cmp [word ptr diffuclty],0
        jne @o_notOver
        mov dl, tie
        jmp @o_endThisCase
@o_notOver:

        mov al, winning
        mov bx, 1
@o_loopMoves:
        call ai_x
        cmp dl, illegal
        je @o_worse
        cmp al, dl
        jb @o_worse
        mov al, dl
@o_worse:
        inc bx
        cmp bx, boardSpace+1
        jne @o_loopMoves
        mov dl, al

@o_endThisCase:
        pop [diffuclty]
        pop ax
        pop bx
        push offset tempboard
        call popArray
        ret
endp ai_o
;-----------------------------------------------------------------------------------------
; Returns in dl the outcome of the board, gets the location in bx
proc ai_x
        push offset tempboard
        call pushArray
```

```
        push bx
        push ax
        push [diffuclty]
        dec [diffuclty]

        mov input, 'x'
        call pseudoPlace
        mov input, 'o'
        cmp boolean, true
        je @x_successfulPlace
        mov dl, illegal
        jmp @x_endThisCase
@x_successfulPlace:
        push offset tempboard
        call isWinning
        cmp boolean, false
        je @x_notWinning
        mov dl, winning
        jmp @x_endThisCase
@x_notWinning:

        push offset tempboard
        call isTie
        cmp boolean, false
        je @x_notTie
        mov dl, tie
        jmp @x_endThisCase
@x_notTie:

        push offset tempboard
        mov dl, 'x'
        call isWinning
        cmp boolean, false
        je @x_notLosing
        mov dl, losing
        jmp @x_endThisCase
@x_notLosing:

        cmp [word ptr diffuclty],0
        jne @x_notOver
        mov dl, tie
        jmp @x_endThisCase
@x_notOver:

        mov al, illegal
```

```
        mov bx, 1
@x_loopMoves:
        call ai_o
        cmp dl, illegal
        je @x_worse
        cmp al, dl
        ja @x_worse
        mov al, dl
@x_worse:
        inc bx
        cmp bx, boardSpace + 1
        jne @x_loopMoves
        mov dl, al

@x_endThisCase:
        pop [diffuclty]
        pop ax
        pop bx
        push offset tempboard
        call popArray
        ret
endp ai_x
;-----------------------------------------------------------------------------------------
proc pushArray
        pop [offset address]
        mov [offset tempAx], ax
        mov [offset tempBx], bx
        mov [offset tempCx], cx

        pop bx
        mov cx, 5
@pushingLoop:
        mov al, [byte ptr bx]
        mov ah, [byte ptr bx + 1]
        push ax
        add bx, 2
        loop @pushingLoop

        push [offset address]
        mov cx, [offset tempCx]
        mov bx, [offset tempBx]
        mov ax, [offset tempAx]
        ret
endp pushArray
;-----------------------------------------------------------------------------------------
```

```
;Resives in stack the address
proc popArray
        pop [offset address1]
        mov [offset tempCx], cx
        mov [offset tempBx], bx
        mov [offset tempAx], ax

        pop bx
        add bx, 8
        mov cx, 5
@poppingLoop:
        pop ax
        mov [byte ptr bx], al
        mov [byte ptr bx + 1], ah
        sub bx, 2
        loop @poppingLoop

        mov ax, [offset tempAx]
        mov bx, [offset tempBx]
        mov cx, [offset tempCx]
        push [offset address1]
        ret
endp popArray
;-----------------------------------------------------------------------------------------

proc clearBoard
        push bx

        xor bx, bx
@clearingLoop:
        mov [byte ptr offset board + bx], defualt
        inc bx
        cmp bx, boardSpace
        jne @clearingLoop

        pop bx
        ret
endp clearBoard

;-----------------------------------------------------------------------------------------
proc drawBoard
        push di
        push dx
        push bx
        push ax
```

```
        xor di, di
@mainPrintLoop:
                xor bx, bx
        @nestedPrintLoop:
                mov dl, [byte ptr offset Board + bx + di]
                mov ah, 2h
                int 21h
                inc bx
                cmp bx, boardSize
                jne @nestedPrintLoop

        mov dl, 0ah
        mov ah, 2h
        int 21h

        add di, boardSize
        cmp di, boardSpace
        jne @mainPrintLoop

        mov dl, 0ah
        mov ah, 2h
        int 21h

        pop ax
        pop bx
        pop dx
        pop di
        ret
endp drawBoard
;-----------------------------------------------------------------------------------------
proc placeO
        push bx
        xor bh, bh
        mov bl, [byte ptr offset bestMove]
        mov [byte ptr offset board + bx - 1], 'o'
        pop bx
        ret
endp placeO
;-----------------------------------------------------------------------------------------
proc placeX
        push ax
        push bx

@tryAgian:
```

```
        mov ah, 1
        int 21h
        sub al,'0'

        cmp al, boardSpace
        ja @invalid
        cmp al, 0
        je @invalid

        mov bl, al
        xor bh, bh
        cmp [byte ptr offset board + bx - 1], defualt
        jne @invalid
        mov [byte ptr offset board + bx - 1], 'x'
        jmp @skipInvalid

@invalid:
        mov dx, offset errorMessage
        mov ah, 9h
        int 21h
        jmp @tryAgian
@skipInvalid:
        mov dl, 0ah
        mov ah, 2h
        int 21h
        pop bx
        pop ax
        ret
endp placeX
;-----------------------------------------------------------------------------------------
; Resives the player in input, and board in stack
proc iswinning
        pop [offset address]
        pop [offset var]
        push cx
        push bx
        push di

        mov di, [offset var]
        mov boolean, false

        ; Check for a matching line
        xor bx, bx
        mov cx, 3
@lineLoop:
```

```
        cmp [byte ptr di + bx], input
        jne @skipThis
                cmp [byte ptr di + 1 + bx], input
                jne @skipThis
                        cmp [byte ptr di + 2 + bx], input
                        jne @skipThis
                        mov boolean, true
@skipThis:
        add bx, 3
        loop @lineLoop

        ; check for a matching column
        xor bx, bx
        mov cx, 3
@colLoop:
        cmp [byte ptr di + bx], input
        jne @skipThis2
                cmp [byte ptr di + 3 + bx], input
                jne @skipThis2
                        cmp [byte ptr di + 6 + bx], input
                        jne @skipThis2
                        mov boolean, true
@skipThis2:
        inc bx
        loop @colLoop

        cmp [byte ptr di], input
        jne @skipThis3
                cmp [byte ptr di + 4], input
                jne @skipThis3
                        cmp [byte ptr di + 8], input
                        jne @skipThis3
                        mov boolean, true
@skipThis3:

cmp [byte ptr di + 2], input
        jne @skipThis4
                cmp [byte ptr di + 4], input
                jne @skipThis4
                        cmp [byte ptr di + 6], input
                        jne @skipThis4
                        mov boolean, true
@skipThis4:

        pop di
```

```
        pop bx
        pop cx
        push [offset address]
        ret
endp iswinning
;-----------------------------------------------------------------------------------------
proc isTie
        pop [offset address]
        pop [offset var]
        push bx
        push di

        mov boolean, false

        mov di, [offset var]
        mov boolean, true;
        xor bx, bx
@tieLoop:
        cmp [byte ptr di + bx], defualt
        jne @skipTie
        mov boolean, false
@skipTie:
        inc bx
        cmp bx, boardSpace
        jne @tieLoop

        pop di
        pop bx
        push [offset address]
        ret
endp isTie
;-----------------------------------------------------------------------------------------
; Resives the player in input and the position in bx, return true if
successful
proc pseudoPlace
        mov boolean, true

        cmp [byte ptr offset tempboard + bx - 1], defualt
        jne @invalid1
        mov [byte ptr offset tempboard + bx - 1], input
        jmp @skipInvalid1

@invalid1:
        mov boolean, false
@skipInvalid1:
```

```
        ret
endp pseudoPlace
;-----------------------------------------------------------------------------------------
proc chooseDiff
@setDif:
        mov ah, 1
        int 21h
        cmp al, 'a'
        jne @notA
        mov [diffuclty], 2
        jmp @doneSettingDif
@notA:
        cmp al, 'b'
        jne @notB
        mov [diffuclty], 4
        jmp @doneSettingDif
@notB:
        cmp al, 'c'
        jne @notC
        mov [diffuclty], 9
        jmp @doneSettingDif
@notC:
        mov dx, offset errorMessage
        mov ah, 9h
        int 21h
        jmp @setDif
@doneSettingDif:
        ret
endp chooseDiff
;-----------------------------------------------------------------------------------------
proc boardState

        mov input, 'x'
        push offset board
        call isWinning
        cmp boolean, true
        jne @notWinning
        mov dx, offset xWinMessage
        mov ah, 9h
        int 21h
        mov boolean, true
        jmp @endOfChecks
@notWinning:

        mov input, 'o'
```

```
        push offset board
        call isWinning
        cmp boolean, true
        jne @notLosing
        mov dx, offset oWinMessage
        mov ah, 9h
        int 21h
        mov boolean, true
        jmp @endOfChecks
@notLosing:

        push offset board
        call isTie
        cmp boolean, true
        jne @endOfChecks
        mov dx, offset tieMessage
        mov ah, 9h
        int 21h
        mov boolean, true
        jmp @endOfChecks

@endOfChecks:
        ret
endp boardState
;-----------------------------------------------------------------------------------------
End start
```