BK
TP.HCM
BK - OISP

Tic-tac-toe with MIPS

Truong Thinh Do - 2052725

Semester 211

Tic-tac-toe with MIPS

Truong Thinh Do - 2052725

Semester 211

1 Introduction

Tic-tac-toe (American English), noughts and crosses (Commonwealth English), or Xs and Os (Irish English) is a paper-and-pencil game for two players who take turns marking the spaces in a three-by-three grid with X or O. The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row is the winner. It is a solved game, with a forced draw assuming best play from both players. (Wikipedia Contributors, 2004)

2 The program

2.1 The idea

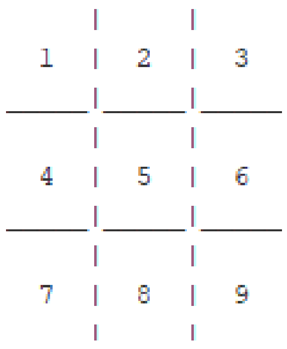
The tic-tac-toe game we will be creating is a text-based game. In order for the game to be played, we begin by creating a function to draw the board to the console. Next, we should keep track of which player is playing and their corresponding symbol (X or O). The player is prompted to enter their desired move on the board. If it's an invalid move, the program should ask them to re-enter until a valid input is read. The game is then updated by setting the correct symbol on the cell chosen by the player and the board should be redrawn to display the latest move. There are 9 cases which will end the game: 3 cases for a full row, 3 cases for a full column, 2 cases for a full diagonal row, and a draw. The checking of the wining cases occurs after each move and if one of the cases exists, we will conclude the winner and end the program. On the other hand, if no more move can be play and none of the player reaches the winning cases, it will be concluded as a draw.

2.2 The algorithm

2.2.1 Drawing the grid

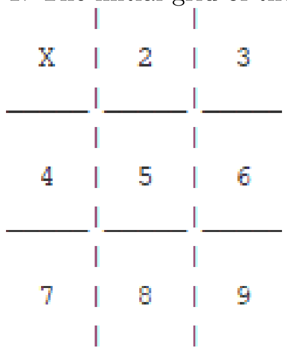
Tic-tac-toe requires a 3x3 grid for the game to be played. By using ASCII character "|" for the vertical line and "_" for the horizontal line, we should be able to form a complete 3x3 grid. An array of character from 1 to 9 is also used to mark each cell so that player know which cell to choose for their next move. After each move, the number character at the chosen cell will be replaced with the player's symbol (X or O).
We begin by printing out the each row at a time. The first row will consists of 2 strings: " | | " and " 1 | 2 | 3 ". The number we print out will be the one in

the array so that we could easily update it and redrawn later. To end the row, we print out the strings ”___|___|___” to separate it from other row. The same rule apply for the other rows with the general form ” a[i] | a[i + 1] | a[i + 2] ” with a is the array {1,2,3,4,5,6,7,8,9} and we should get a complete 3x3 grid.



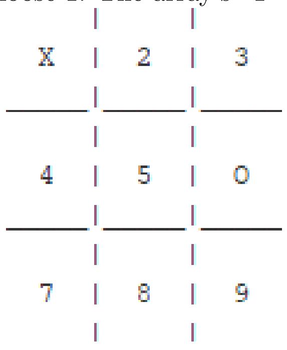
```
     |     |
  1  |  2  |  3
_____|_____|_____
     |     |
  4  |  5  |  6
_____|_____|_____
     |     |
  7  |  8  |  9
     |     |
```

Figure 1: The initial grid of the game



```
     |     |
  X  |  2  |  3
_____|_____|_____
     |     |
  4  |  5  |  6
_____|_____|_____
     |     |
  7  |  8  |  9
     |     |
```

Figure 2: Player 1 choose 1. The array’s ”1” is replaced with ”X”



```
     |     |
  X  |  2  |  3
_____|_____|_____
     |     |
  4  |  5  |  O
_____|_____|_____
     |     |
  7  |  8  |  9
     |     |
```

Figure 3: Player 2 choose 6. The array’s ”6” is replaced with ”O”

2.2.2 Determine the player

In order to determine which player is playing, a variable is used to resolve this. This variable will have 2 values: 1 for the X player and 2 for the O player. After the turn ends, the value will be toggled and the process is run again.

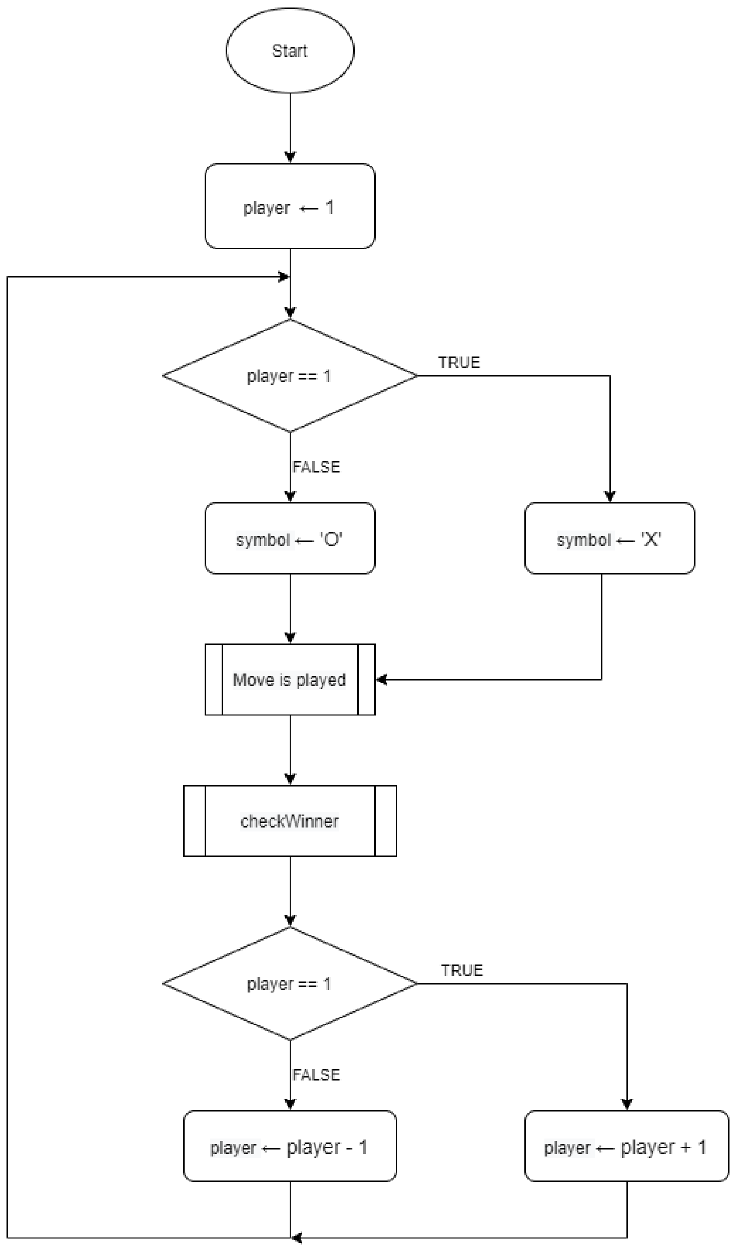


Figure 4: The flowchart for the process

2.2.3 Choose your move

At the beginning of every turn, we will ask the player to choose their desired move by enter a number from 1 to 9 corresponding to the array of characters from 1 to 9 which we use to draw our grid. After the input, we perform a checking to see if it is valid or not. If the input is less than 1 and greater than 9, it is an invalid input and player must enter it again. If player choose a position in the array where the character is "X" or "O", the grid is already occupied and we must ask them to re-enter it. Else if the input is valid, we will replace the number character in the array with the symbol of that player and redrawn the grid.

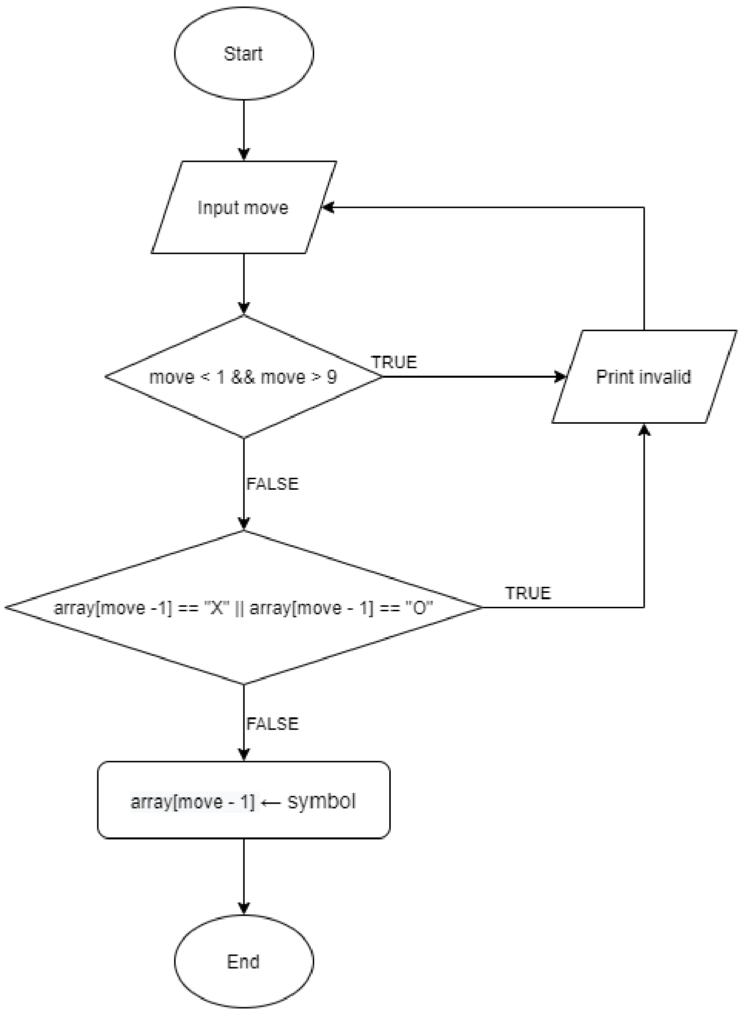


Figure 5: The flowchart for processing input

2.2.4 We have a winner

After every move, we need to check whether the game has ended or not. There are 8 cases where a winner is determined and we need to check all of them: 3 rows, 3 columns and 2 diagonal. If any of them is filled with the same symbol, we will end the game and announce the winner. We don't need to check which symbol is it to determine the player because the winner will always be the player who play the latest move.

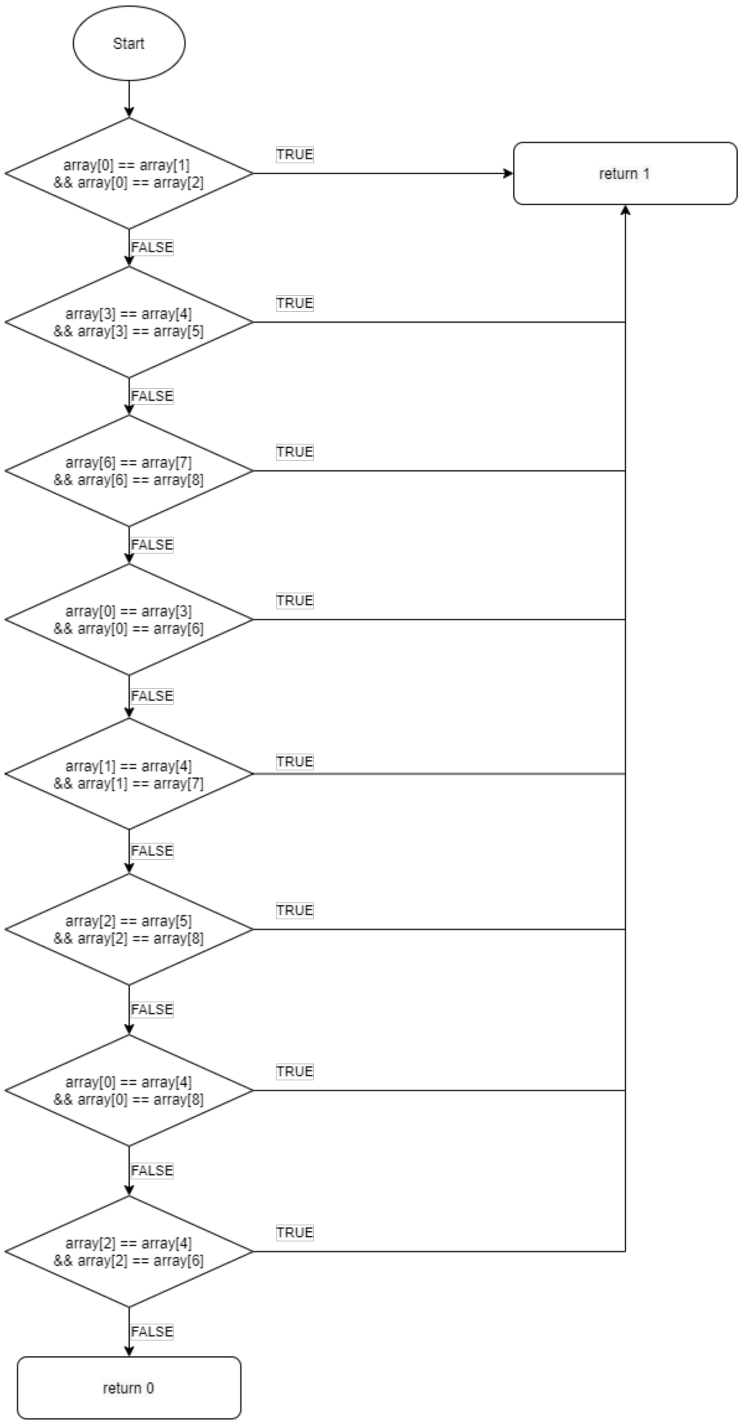


Figure 6: The flowchart for checking winner

2.2.5 It's a draw

A draw is when the 8 cases in the winner checking is still not satisfied but no more move can be made by both player. By using a variable to keep track of the number of move, we should be able to conclude when this situation occurs. After every move, the variable is incremented and if the winner checking still hasn't ended the program but the number of move reach 9, then it's a draw.

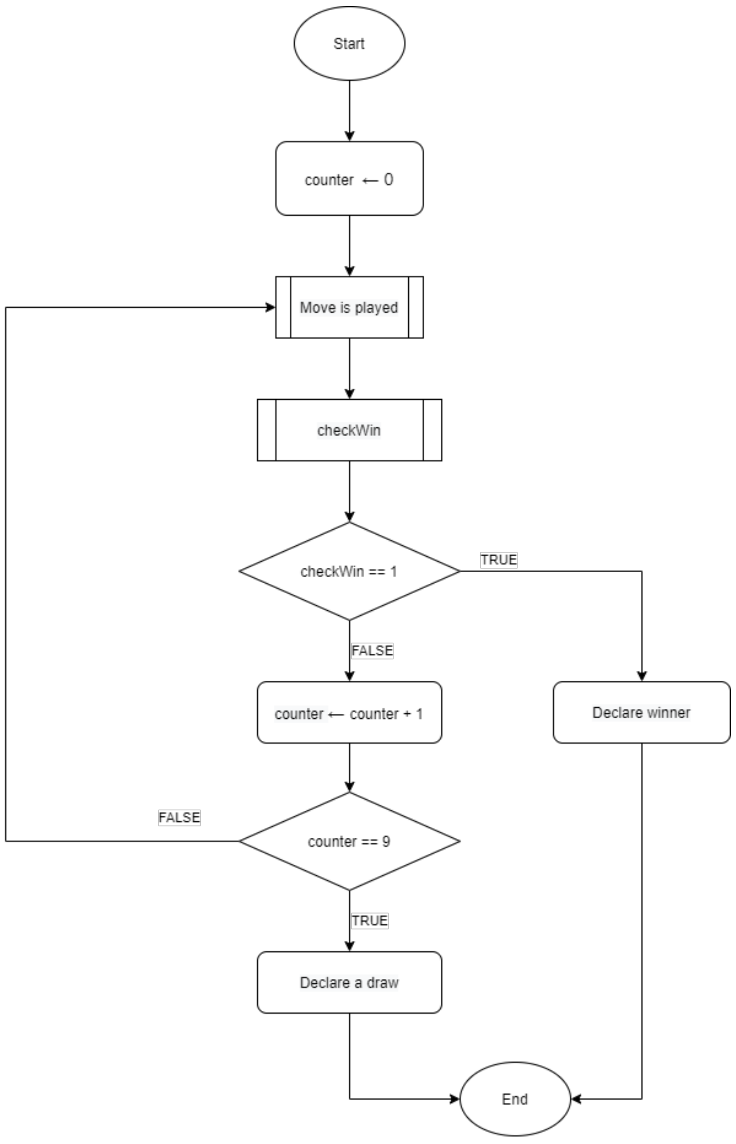


Figure 7: The flowchart for checking a draw

2.3 The code

We begin by declaring some data for our program, including the title, the array for storing symbol, the line for drawing the grid, and some notifications for the player.

```
    .data
title: .asciiz "\n\n\tTic Tac Toe\n"
player: .asciiz "\nPlayer 1 (X)  -  Player 2 (O)\n"
grid: .byte '1','2','3','4','5','6','7','8','9'
vertical: .asciiz "     |     |     \n"
horizontal: .asciiz "_____|_____|_____\n"
space: .asciiz "  "
seperator: .asciiz "|"
newline: .asciiz "\n"
enter: .asciiz ", enter your move: "
play: .asciiz "Player "
invalid: .asciiz "Invalid move\n"
win: .asciiz " wins\n"
tie: .asciiz "It's a draw\n"
```

2.3.1 The main program

```
.text
main:
la $a0, title
addi $v0, $0, 4
addi $t0, $0, 1 #variable for player turn
addi $t1, $0, 1
addi $s4, $0, 0 #variable for checking draw
addi $s5, $0, 9
syscall #print title
```

We first load the title into the \$a0 register and print it out. The \$t0 register is used for the player variable to determine who is playing and \$s4 register is used for the number of moves to check if it's a draw or not. \$t1 and \$s5 just simply store number 1 and 9.

```
Redraw:
jal draw
bne $t0, $t1, p2
addi $s1, $0, 'X'
j Play
p2: addi $s1, $0, 'O'
```

In this section, we will call the draw procedure to draw the grid (which will be implemented later), the program will use the label "Redraw" to redraw the grid with updated symbol and create the game loop. The $t0 register, which is the player variable, is compared with number 1 stored in $t1 register. If the player turn is equal to 1, it's player 1's turn and the $s1 register will store the "X" symbol and jump to "Play" for the game. If not, it will jump to p2 where $s1 register will store the "O" symbol.

```
Play:
la $a0, play
addi $v0, $0, 4
syscall #Print "Player "
addi $a0, $t0, 0
addi $v0, $0, 1
syscall #Print player number id
la $a0, enter
addi $v0, $0, 4
syscall #Print ", enter your move: "
addi $v0, $0, 5
syscall #Input move
slti $t5, $v0, 1
bne $t5, $t1, Valid1
la $a0, invalid
addi $v0, $0, 4
syscall
j Play
Valid1:
slt $t5, $s5, $v0
bne $t5, $t1, Valid2
la $a0, invalid
addi $v0, $0, 4
syscall
j Play
Valid2:
```

The "Play" labeled part is where our main function of the game work. Printing the "play " and the "enter" with the player variable in $t0 will print out the string "Player 1, enter your move: " or "Player 2, enter your move: " depending on the player. The program will then receive the input from the user.

We need to verify the input before moving on to the next step. By using "slt" and "slti", we check the input to see if it's greater than 1 and less than 9. If the input is less than 1, we will load the "invalid" string, print it out, and jump back to "Play" for another input, else we jump to "Valid1" as the first condition is valid. We then dcheck for the greater than 9 case and the same procedure is carryout. If it satisfies the condition then we jump to "Valid2" and continue the play.

```
la $s2, grid
Loop:
beq $v0, $t1, Found #finding the position
addi $s2, $s2, 1
addi $v0, $v0, -1
j Loop
Found:
```

The array of symbol is loaded to the $s2 register. Loop is created to iterate the array and reach the position chosen by the player's input. If it has not been reached yet, the input is decremented and the address stored in $s2 will move up 1 byte. When the element is reached, we will exit the loop via the "Found" label.

```
addi $t2, $0, 'X'
addi $t3, $0, 'O'
lb $t4, 0($s2)
bne $t4, $t2, ValidO #check if it's occupied or not
la $a0, invalid
addi $v0, $0, 4
syscall
j Play
ValidO:
bne $t4,$t3, ValidX
la $a0, invalid
addi $v0, $0, 4
syscall
j Play
ValidX:
```

When the position is found, we need to check if the position is occupied or not. We store the symbol "X" and "O" in $t2 and $t3 for later usage. $t4 is loaded with the character stored at the address $s2. The loaded character is then compared with "X" and "O" by using $t2 and $t3. If the character at the current position is "O", we will print out the invalid statement and jump back to "Play", else we jump to "ValidO". The same applies for "X".

```
sb $s1, 0($s2) #store the symbol into the array
jal checkWin
beq $v0, $0, Continue
j Win
Continue:
addi $s4, $s4, 1
beq $s4, $s5, Tie
bne $t0, $t1, decrement
addi $t0, $t0, 1
j Redraw
decrement:
addi $t0, $t0, -1
j Redraw
```

If the input position is valid, we will begin to replace the number character in that position of the array with the player's symbol in $s1 by using "sb". After that, the "checkWin" procedure is executed and return the result. If the return value is 1, that player wins the game and the program will jump to "Win". Otherwise, it continues and begin to check for a draw. The number of move store in $s4 is incremented and if it is equal to 9, the grid is now full and the game ends in a draw. Else, we will toggle the player by increasing the player variable in $t0 to 2 if it's equal to 1 and decreasing to 1 if it's number 2. Jumping back to "Redraw" will updated the grid and start the next move.

2.3.2 The draw procedure

Here is the drawing procedure. The grid is drew by using the character "|" and "_" along with the character from the array used for displaying the position and move played by player.

```
draw:
la $s0, grid
la $a0, player
addi $v0, $0, 4
syscall
la $a0, vertical
syscall
la $a0, space
syscall
lb $a0, 0($s0)
addi $v0, $0, 11
syscall
```

```
addi $v0, $0, 4
la $a0, space
syscall
la $a0, seperator
syscall
la $a0, space
syscall
lb $a0, 1($s0)
addi $v0, $0, 11
syscall
addi $v0, $0, 4
la $a0, space
syscall
la $a0, seperator
syscall
la $a0, space
syscall
lb $a0, 2($s0)
addi $v0, $0, 11
syscall
addi $v0, $0, 4
la $a0, space
syscall
la $a0, newline
syscall
la $a0, horizontal
syscall
la $a0, vertical
syscall
la $a0, space
syscall
lb $a0, 3($s0)
addi $v0, $0, 11
syscall
addi $v0, $0, 4
la $a0, space
syscall
la $a0, seperator
syscall
la $a0, space
syscall
lb $a0, 4($s0)
addi $v0, $0, 11
syscall
```

```
addi $v0, $0, 4
la $a0, space
syscall
la $a0, seperator
syscall
la $a0, space
syscall
lb $a0, 5($s0)
addi $v0, $0, 11
syscall
addi $v0, $0, 4
la $a0, space
syscall
la $a0, newline
syscall
la $a0, horizontal
syscall
la $a0, vertical
syscall
la $a0, space
syscall
lb $a0, 6($s0)
addi $v0, $0, 11
syscall
addi $v0, $0, 4
la $a0, space
syscall
la $a0, seperator
syscall
la $a0, space
syscall
lb $a0, 7($s0)
addi $v0, $0, 11
syscall
addi $v0, $0, 4
la $a0, space
syscall
la $a0, seperator
syscall
la $a0, space
syscall
lb $a0, 8($s0)
addi $v0, $0, 11
syscall
addi $v0, $0, 4
la $a0, space
syscall
```

```
la $a0, newline
syscall
la $a0, vertical
syscall
jr $ra
```

2.3.3 Winner checking procedure

The winner checking procedure will check for all possible cases of winning the game: 3 cases for a full row, 3 cases for a full column and 2 cases for a full diagonal. Each case, all 3 element will be compared. If they're equal then the winner is determined and the procedure return 1. If no case is satisfied, the return value is 0.

```
checkWin:
la $s3, grid
addi $v0, $0, 0
Case1:
lb $t2, 0($s3)
lb $t3, 1($s3)
lb $t4, 2($s3)
bne $t2, $t3, Case2
bne $t2, $t4, Case2
addi $v0, $0, 1
jr $ra
Case2:
lb $t2, 3($s3)
lb $t3, 4($s3)
lb $t4, 5($s3)
bne $t2, $t3, Case3
bne $t2, $t4, Case3
addi $v0, $0, 1
jr $ra
Case3:
lb $t2, 6($s3)
lb $t3, 7($s3)
lb $t4, 8($s3)
bne $t2, $t3, Case4
bne $t2, $t4, Case4
addi $v0, $0, 1
jr $ra
```

```
Case4:
lb $t2, 0($s3)
lb $t3, 3($s3)
lb $t4, 6($s3)
bne $t2, $t3, Case5
bne $t2, $t4, Case5
addi $v0, $0, 1
jr $ra
Case5:
lb $t2, 1($s3)
lb $t3, 4($s3)
lb $t4, 7($s3)
bne $t2, $t3, Case6
bne $t2, $t4, Case6
addi $v0, $0, 1
jr $ra
Case6:
lb $t2, 2($s3)
lb $t3, 5($s3)
lb $t4, 8($s3)
bne $t2, $t3, Case7
bne $t2, $t4, Case7
addi $v0, $0, 1
jr $ra
Case7:
lb $t2, 0($s3)
lb $t3, 4($s3)
lb $t4, 8($s3)
bne $t2, $t3, Case8
bne $t2, $t4, Case8
addi $v0, $0, 1
jr $ra
Case8:
lb $t2, 2($s3)
lb $t3, 4($s3)
lb $t4, 6($s3)
bne $t2, $t3, Else
bne $t2, $t4, Else
addi $v0, $0, 1
jr $ra
Else:
jr $ra
```

2.3.4 Announce the winner

If the winner checking procedure return 1, the program will be diverted here. We first redraw the latest move, then we start to determine who is the winner by checking the player variable in $t0 and store the corresponding player id into $s1. Finally, we print out the winner announcement and end the program.

```
Win:
jal draw
bne $t0, $t1, player2
addi $s1, $0, '1'
j winner
player2:
addi $s1, $0, '2'
winner:
la $a0, play
addi $v0, $0, 4
syscall
addi $a0, $s1, 0
addi $v0, $0, 11
syscall
la $a0, win
addi $v0, $0, 4
syscall
j Exit
```

2.3.5 Announce a draw

When the winner checking procedure is still not satisfied but the number of move has reach 9, it means no more move can be made and there's no winner. That's when the program jump to this part. We also need to redraw the latest move and finally, we print out the draw announcement and game over.

```
Tie:
jal draw
la $a0, tie
addi $v0, $0, 4
syscall
Exit:
```

References

Wikipedia Contributors. (2004). *Tic-tac-toe.* Retrieved from `https://en.wikipedia.org/wiki/Tic-tac-toe` ([Accessed 26-October-2021])