

# TIC TAC TOE ASSIGNMENT

## Assignment Requirements:

1. Abstraction ✓
2. Decomposition ✓
3. Pseudo code ✓
4. Python code

## Abstraction

Abstraction essentially refers to the process of simplification of a task or a problem such that all low-level, unnecessary and irrelevant details are hidden behind a layer of abstraction, ie. they are "abstracted" away. In other words, we do not worry about the meticulous implementation details, we only worry about how to effectively utilize the products of those details.

Therefore, the process of abstraction involves the following:

1. Hiding the irrelevant details
2. Generalising the common essence

## Abstraction of Tic Tac Toe

### Game Setup

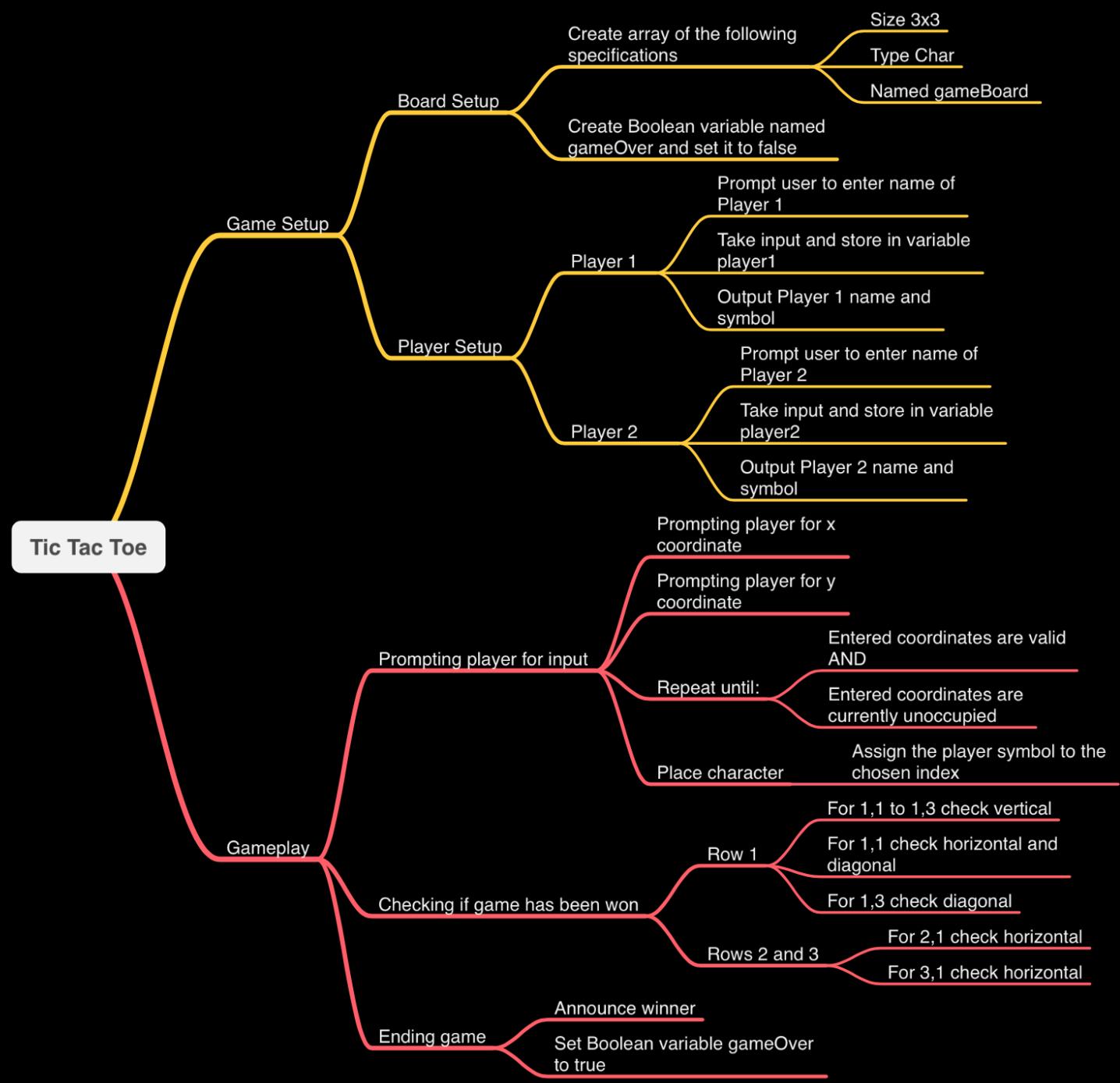
1. Prompt user for name of two players
2. Assign the X and O symbols to the two players
3. Inform the players of player 1 + their symbol and player 2 + their symbol
4. Prompt player 1 for location on game-board to place their symbol
5. Keep re-asking as long as entered location is either invalid or already occupied
6. If game has been won, end game and reset board
7. If game has not been won, go to line 8
8. Prompt player 2 for location on game-board to place their symbol
9. Keep re-asking as long as entered location is either invalid or already occupied
10. If game has been won, end game and reset board
11. If game has not been won, go to line 4

## DECOMPOSITION

Decomposition refers to the process of breaking down a particular task (in our case, the task is executing a PvP tic tac toe game) into sub-tasks, which are then broken down further into processes, which are broken down into activities.

This process of simplification continues until the tasks can no longer be simplified while keeping the high-level implementation in mind

## Decomposition of Tic Tac Toe (Using a tree diagram)



## PSEUDOCODE

```
DECLARE gameOver : BOOLEAN
DECLARE turn : INTEGER
gameOver ← FALSE
turn ← 1
DECLARE gameBoard : ARRAY [1 : 3, 1 : 3] OF CHAR
DECLARE victor : STRING
```

OUTPUT "Enter name for Player 1"

INPUT player1

OUTPUT player1, "'s symbol is X"

OUTPUT "Enter name for Player 2"

INPUT player2

OUTPUT player2, "'s symbol is O"

REPEAT

IF turn MOD 2 <> 0

THEN

currentPlayer ← player1

playerSymbol ← 'X'

ELSE

currentPlayer ← player2

playerSymbol ← 'O'

ENDIF

// Player executes his turn

REPEAT

OUTPUT currentPlayer, " enter your row number"

INPUT y

OUTPUT currentPlayer, " enter your column number"

INPUT x

UNTIL (gameBoard [y, x] <> 'X' OR gameBoard [y, x] <> 'O')

AND ((x >= 1 AND x <= 3) AND (y >= 1 AND y <= 3))

gameBoard [y, x] ← playerSymbol

// Check for a win

FOR i ← 1 TO 3

IF i = 1

THEN

FOR j ← 1 TO 3

IF j = 1

THEN

IF (gameBoard [i, j] = gameBoard [i + 1, j + 1] AND  
gameBoard [i, j] = gameBoard [i + 2, j + 2]) OR  
(gameBoard [i, j] = gameBoard [i, j + 1] AND  
gameBoard [i, j] = gameBoard [i, j + 2])  
THEN

gameOver ← true

victor ← currentPlayer

ENDIF

ELSE

IF j = 3

THEN

IF gameBoard [i, j] = gameBoard [i + 1, j - 1] AND  
gameBoard [i, j] = gameBoard [i + 2, j - 2]

```

        THEN
        gameOver ← true
        victor ← currentPlayer
    ENDIF
ENDIF
ENDIF
IF gameBoard [i,j] = gameBoard [i+1,j] AND
gameBoard [i,j] = gameBoard [i+2,j]
THEN
    gameOver ← true
    victor ← currentPlayer
ENDIF
NEXT j
ELSE
IF gameBoard [i,j] = gameBoard [i,j+1] AND
gameBoard [i,j] = gameBoard [i,j+2]
THEN
    gameOver ← true
    victor ← currentPlayer
ENDIF
ENDIF
NEXT i
// Check complete
turn ← turn + 1
UNTIL gameOver = true
OUTPUT "The winner of this game is ", victor

```

# Some of the working behind the checking mechanism:

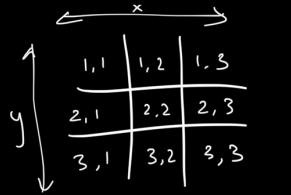
7:55 PM Sat 9 Jan



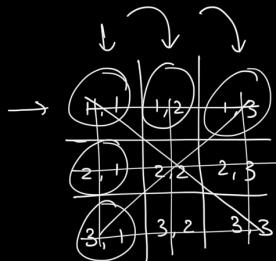
< Notes



Suggested Title: 131 [Edit](#)  
9 January 2021 at 6:27 PM



Too lengthy



[1, 3]

```
FOR (i = 1 ; i <= 4 ; i++)  
FOR (j = 1, j <= 4 ; j++)  
IF board (i, j) is occupied  
① [ check if i, j+1 and i, j+2 also  
contain the same char  
② [ check if i+1, j and i+2, j also  
contain the same char  
③ [ IF i == j AND i > 2  
check if MOD(i+1)3, MOD(j+1)3  
and MOD(i+2)3, MOD(j+2)3 also  
contain the same char  
else if  
i > 2 AND j > 2 THEN  
check if
```

## Checking Mechanism

```
FOR i ← 1 TO 3  
IF i = 1 THEN  
FOR j ← 1 TO 3  
IF j = 1 THEN  
check i+1, j+1 and i+2, j+2  
check i, j+1 and i, j+2  
ELSE IF j = 3 THEN  
check i+1, j-1 and i+2, j-2  
ENDIF  
check i+1, j and i+2, j  
ENDIF  
ELSE  
check i, j+1 and i, j+2  
ENDIF
```