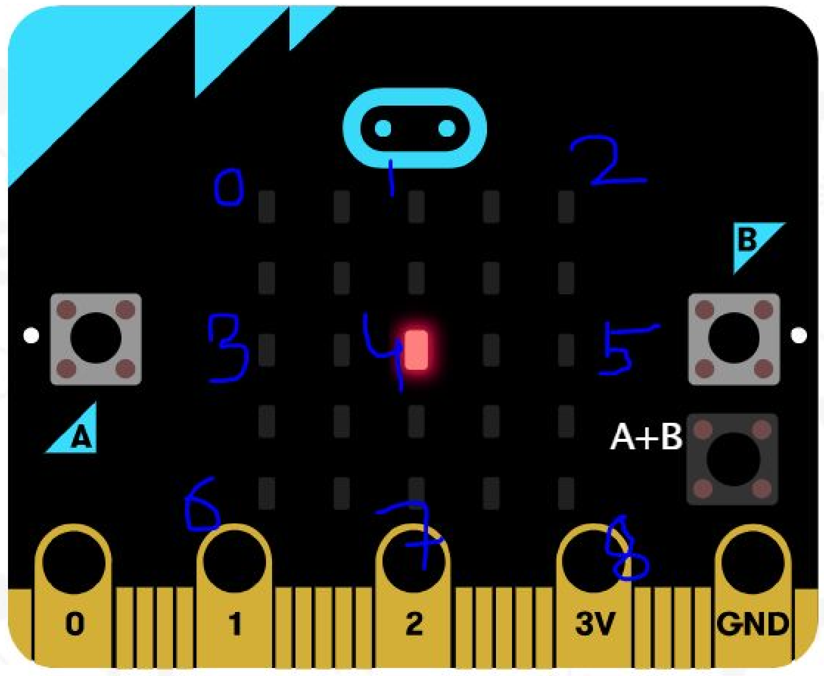
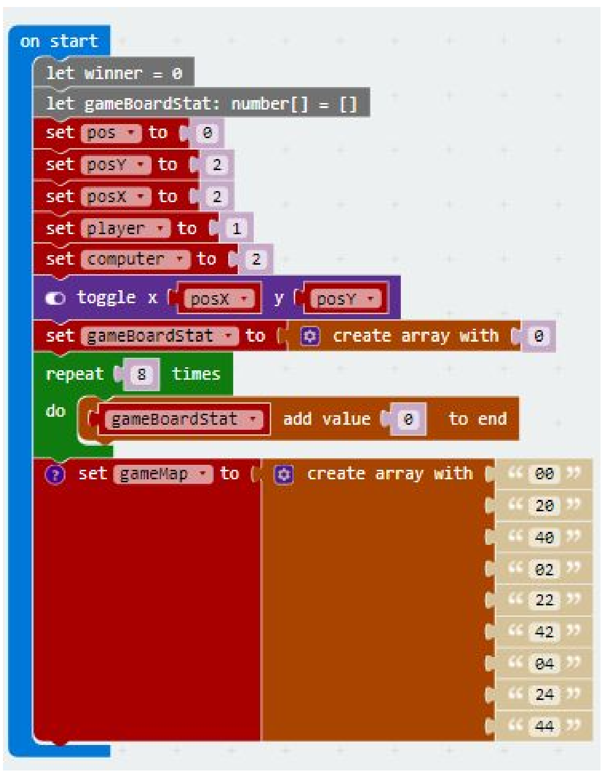
About the game



The 3x3 tic tac toe is played on the 5x5 matrix. Where the first element has the coordinate 0.0, the second element has 2.0 and so on.
The icon for the computer has lower light brightness than the player.

On start



```
on start
    let winner = 0
    let gameBoardStat: number[] = []
    set pos to 0
    set posY to 2
    set posX to 2
    set player to 1
    set computer to 2
    toggle x posX y posY
    set gameBoardStat to create array with 0
    repeat 8 times
    do
        gameBoardStat add value 0 to end
    set gameMap to create array with "00"
                                     "20"
                                     "40"
                                     "02"
                                     "22"
                                     "42"
                                     "04"
                                     "24"
                                     "44"
```

First, we need to initialize all the variables and array we are going to use in the project.

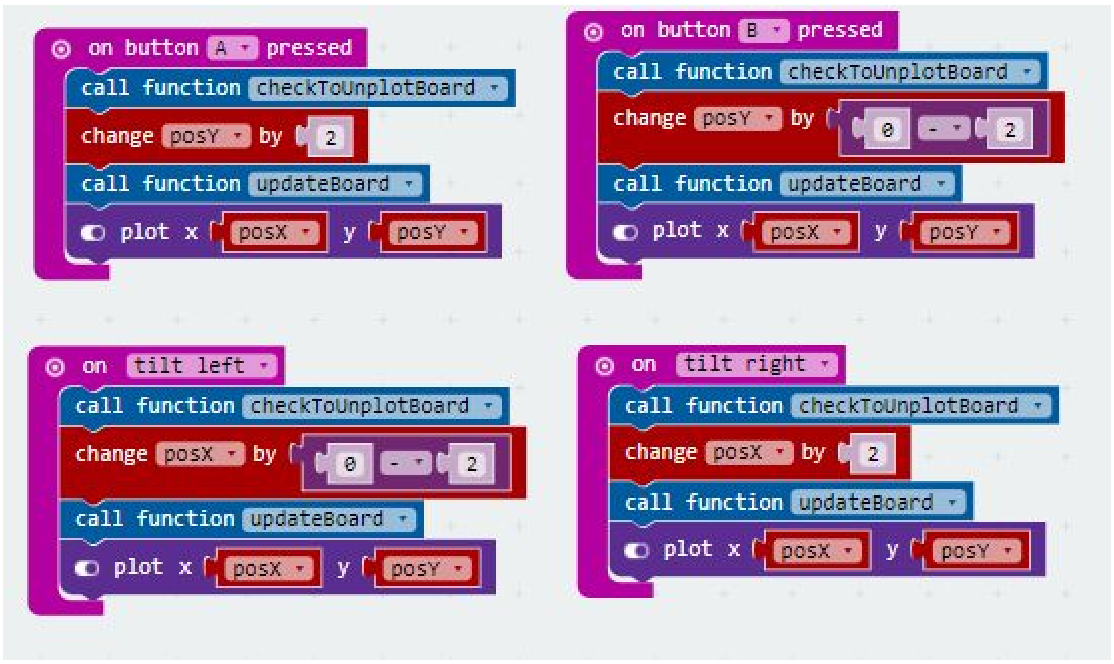
We are going to start the player's icon in the middle, therefore, we set the position to (2, 2). Therefore, posY = 2 and posX = 2. To see where the current player's icon is, the light is toggle on at the position.

To keep track on where the player and the computer place their icons, numbers are used to representing them. The player icon is 1 and computer 2.

The gameBoardStat is use to track of the board in order to check who wins. The winner is initialize to 0 and all the elements in the gameBoardStat is zero.

There is a need to transfer the coordinate of the 5x5 matrix into the 3x3 tic tac toe board. The matrix gameMap is used for that purpose. The first element is 00, indicates that the first element of the 3x3 tic tac toe board has the coordinate (0,0). The second element is 20, indicates that the second element of the 3x3 tic tac toe board has the coordinate (2,0) and so on. Notice that the coordinates are written in the form as (x,y) and so xy.

Control



```
on button A pressed
    call function checkToUnplotBoard
    change posY by 2
    call function updateBoard
    plot x posX y posY

on button B pressed
    call function checkToUnplotBoard
    change posY by 0 - 2
    call function updateBoard
    plot x posX y posY

on tilt left
    call function checkToUnplotBoard
    change posX by 0 - 2
    call function updateBoard
    plot x posX y posY

on tilt right
    call function checkToUnplotBoard
    change posX by 2
    call function updateBoard
    plot x posX y posY
```

To control the icon and where the icon goes, there are these input handlers.

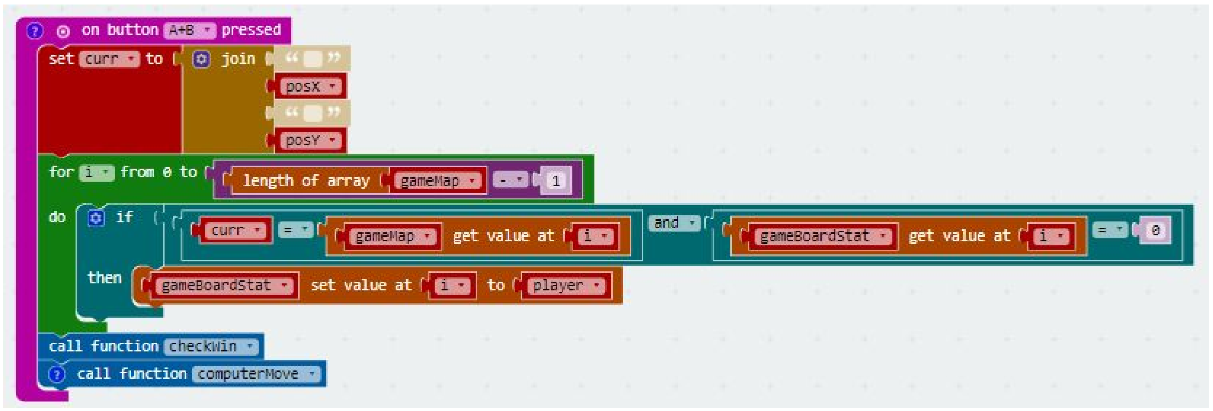
To move the icon left or right, simply tilt them to that direction. Therefore, only x coordinates are changed. To move the icon to the left, x coordinate is changed by subtracting 2 places from the current position. To the right, then adding 2 places to the current position.

To move the icon up simply press A and down by pressing B. Therefore, only y coordinates are changed. To move the icon up, y coordinate is changed by subtracting 2 places from the current position. To move down, then adding 2 places to the current position.

Before moving the icon, there need to be a check to see if a position has already been occupied by either the player or the computer. The function ***checkToUnplotBoard*** sees about that. *We explain this later.*

After changing the location of the player's icon, the board is updated through the function **updateBoard**. *We explain this later.*

Finally, plot the light according to the positions.



```
on button A+B pressed
  set curr to join " " posX " " posY
  for i from 0 to length of array gameMap - 1
  do  if curr = gameMap get value at i and gameBoardStat get value at i = 0
      then gameBoardStat set value at i to player
  call function checkWin
  call function computerMove
```

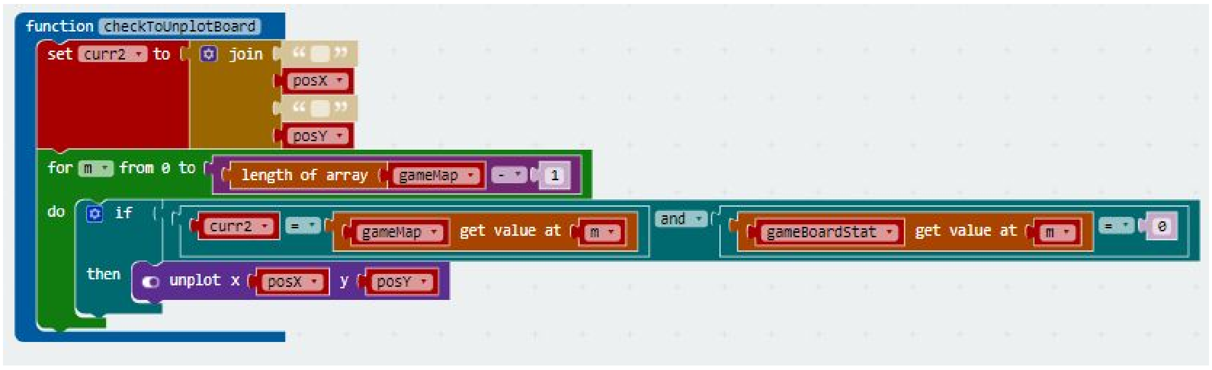
To confirm the position where the player is going to place the icon, both button A and B are pressed.

The next thing is to map the position from a 3x3 tic tac toe matrix into the 5x5 microbit matrix. We call this the mapping process.

The position is an integer and the value stored in the map array (gameMap) is a string. Therefore, we convert it to string using " " and join them so that they will be in a form of xy. We then loop through the gameMap array to check if there is a i-th position matched from the array. If there is and the position is not occupied by the player or computer, then we marked the position, in the gameBoardStat array, as player.

Finally, we call the **checkWin** function to check if there is a winner. *We explain this later.* If there is no winner, then we make computer does some moves by the **computerMove** function. *We explain this later.*

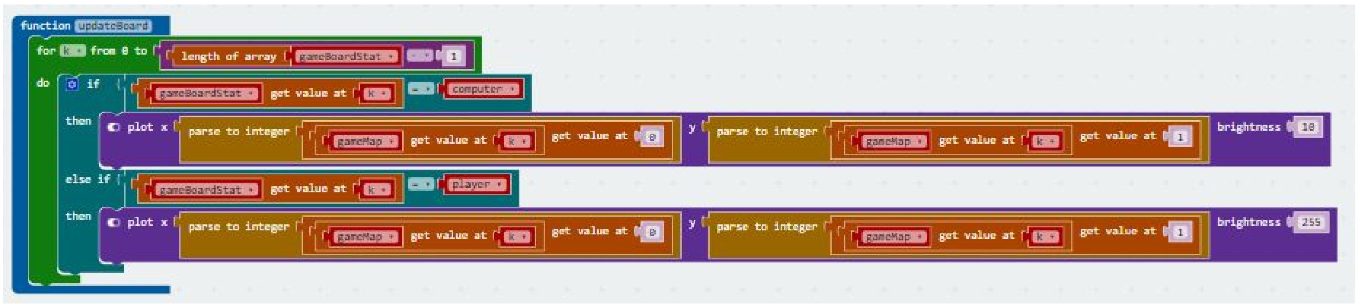
Unplot



```
function checkToUnplotBoard
  set curr2 to join " "
                    posX
                    " "
                    posY
  for m from 0 to length of array gameMap - 1
  do if curr2 = gameMap get value at m and gameBoardStat get value at m = 0
     then unplot x posX y posY
```

This is another mapping process as described above. However, instead of marking the position, we unplot it.

Update the board

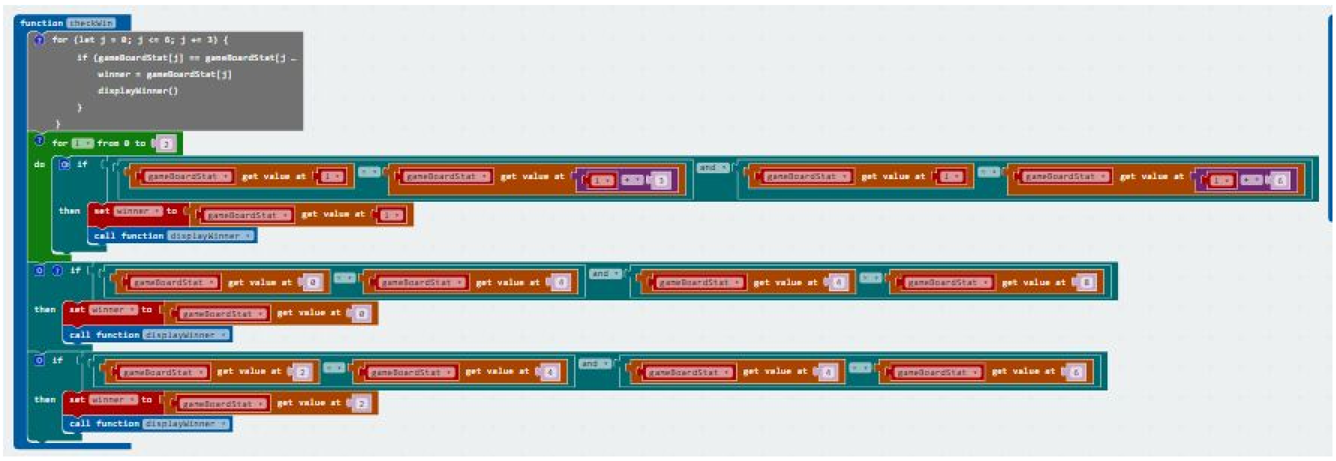


```
function updateBoard
  for k from 0 to length of array gameBoardStat - 1
  do if gameBoardStat get value at k = computer
     then plot x parse to integer gameMap get value at k get value at 0 y parse to integer gameMap get value at k get value at 1 brightness 10
     else if gameBoardStat get value at k = player
     then plot x parse to integer gameMap get value at k get value at 0 y parse to integer gameMap get value at k get value at 1 brightness 255
```

This function is made to ensure that the light for the positions placed by the player and the computer are always on. We loop through the gameBoardStat array to check if there is position marked by the computer or player. If there is, then we toggle the led on and set the right brightness level. For the player, we set it to the maximum; for the computer, we set it to a lower level (in this case is 10).

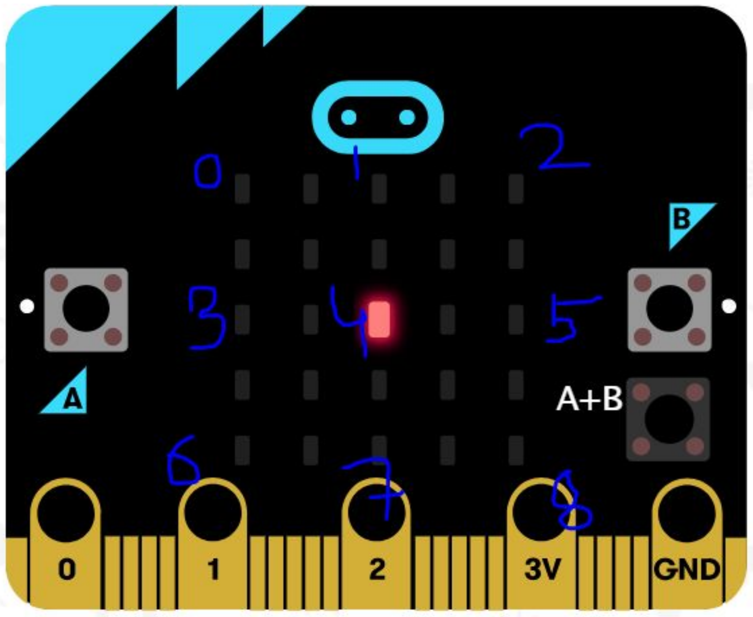
Note that because the elements in the array are string, therefore, we need to convert it into integer to use the built in led toggle function. To convert into integer, simply parse it into integer. In JavaScript, you can code parseInt(*num)*, where *num* is a number written as a string.

Check win



```
function checkWin
    for (let j = 0; j <= 6; j += 3) {
        if (gameBoardStat[j] == gameBoardStat[j …
            winner = gameBoardStat[j]
            displayWinner()
        }
    }
    for i from 0 to 2
    do  if  gameBoardStat get value at i  =  gameBoardStat get value at i + 3  and  gameBoardStat get value at i  =  gameBoardStat get value at i + 6
        then  set winner to gameBoardStat get value at i
              call function displayWinner
    if  gameBoardStat get value at 0  =  gameBoardStat get value at 4  and  gameBoardStat get value at 4  =  gameBoardStat get value at 8
    then  set winner to gameBoardStat get value at 0
          call function displayWinner
    if  gameBoardStat get value at 2  =  gameBoardStat get value at 4  and  gameBoardStat get value at 4  =  gameBoardStat get value at 6
    then  set winner to gameBoardStat get value at 2
          call function displayWinner
```

This is a large function where we have to check for all possible winning conditions.



As observed from the 3x3 tic tac toe matrix.

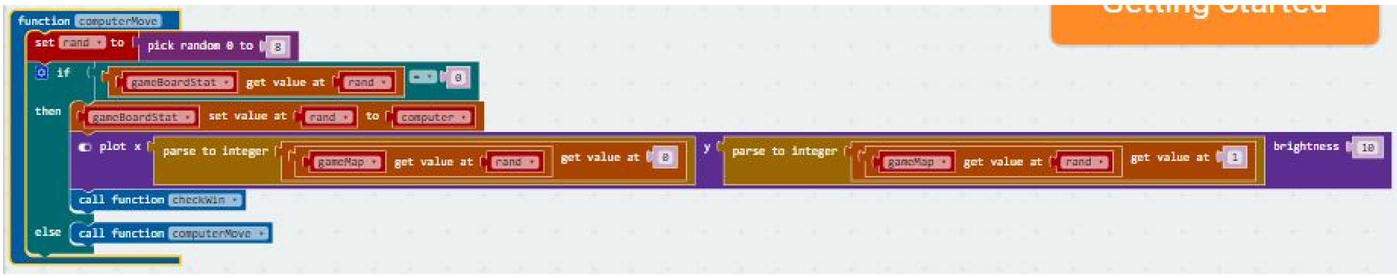
For rows:
There are three rows in the game. We simply check if the first element of the row is equal to the second one and then the last one. That is i = i+1 and i = i+2. You can also make it as i = i+1 and i+1 = i+2.
The loop increment is 3. This is because, as observed from the first column, the elements (0, 3, 6) are all different by 3.

For columns:
Apply the same logic as rows with minor changes. The checking is i = i+3 and i = i+6. The loop is increment by 1 this time.

For diagonals:
We can also do loop here. However, there are only 2 possible diagonals to check, therefore, we write this straight in.

For all of the checkings, if a condition is met, then we set the winner as the current number of the board. Then we call the function **displayWinner.**

Computer move



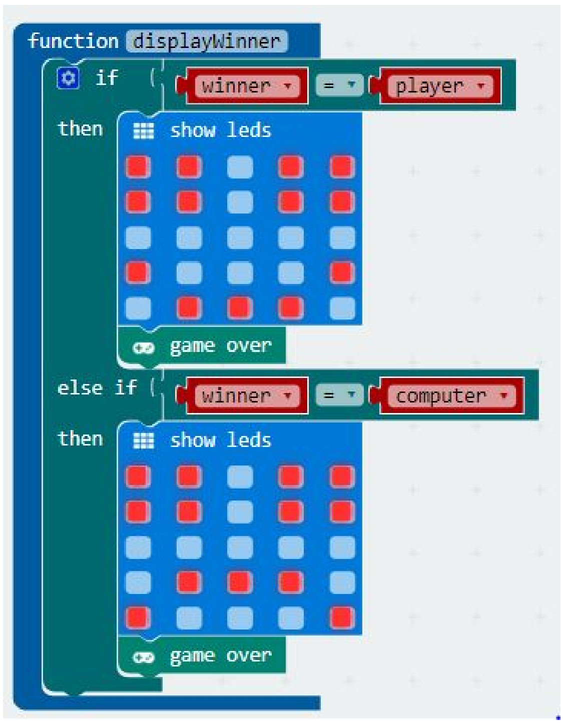
```
function computerMove
  set rand to pick random 0 to 8
  if gameBoardStat get value at rand = 0
  then
    gameBoardStat set value at rand to computer
    plot x parse to integer gameMap get value at rand get value at 0 y parse to integer gameMap get value at rand get value at 1 brightness 10
    call function checkWin
  else
    call function computerMove
```

The computer is doing the moves randomly. We can do this using the Math.random function The function works as in the example from JavaScript:

```
Math.random(3)
```
Output: 0, 1, 2

Because the tic tac toe matrix has 9 elements in total, but counting for 0. Therefore, our math random will be `Math.random(9)` in JavaScript. However, when we are building using block, we can just pick random 0 to 8.
If there is no element in the position which has been randomized, then the computer marks it and set the correct light level, and check for win. If not, which mean the position has already been occupied by the computer previous move or player, then we call the function again, so the computer can randomly pick another position.

Display the winner



```
function displayWinner
  if ( winner = player )
  then show leds
         # # . # #
         # # . # #
         . . . . .
         # . . . #
         . # # # .
       game over
  else if ( winner = computer )
  then show leds
         # # . # #
         # # . # #
         . . . . .
         . # # # .
         # . . . #
       game over
```

If the winner is the player, then we create LED shows the smiley face. Else we shows the sad face as the player has lost and the computer wins. In both the scenario, the game ends and the player needs to press the reset button on the back to reset the game.