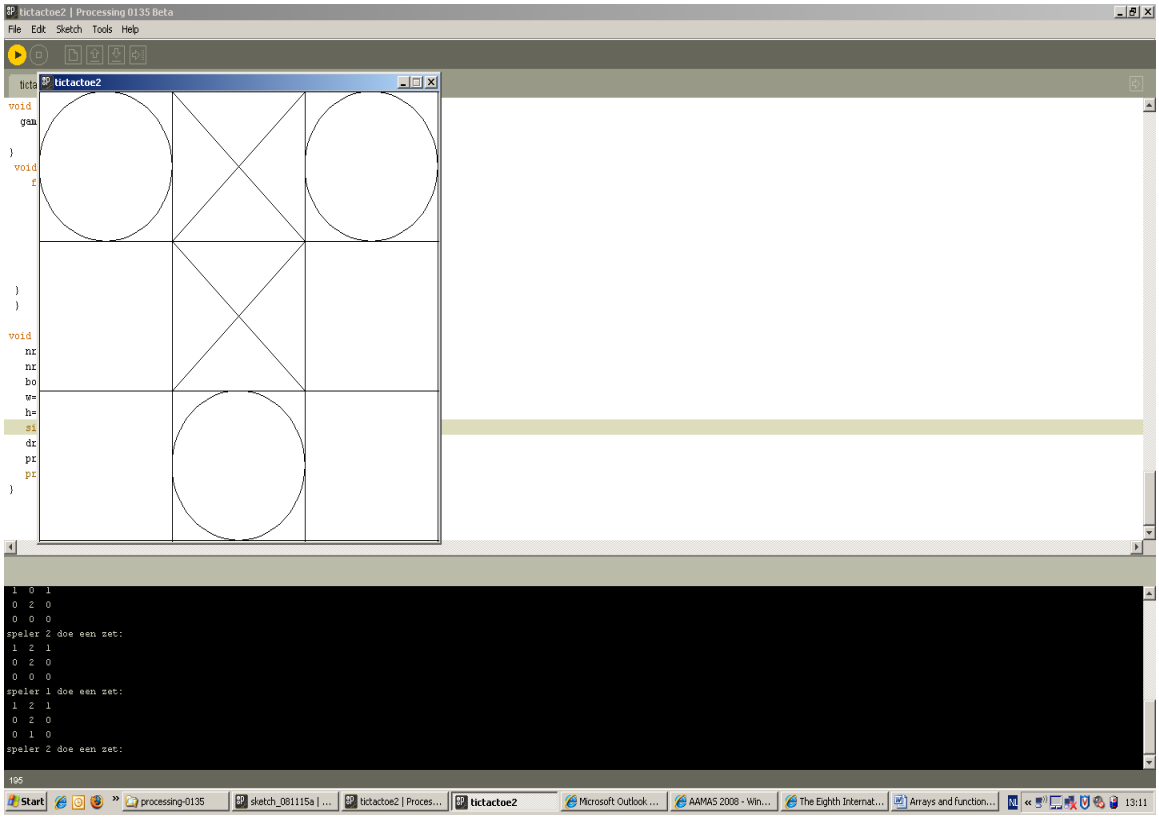
Arrays and functions: Tic Tac Toe

So far you have learned about variables, loops, arrays and functions. This enables us to implement a very simple 2-player board game, i.e. Tic Tac Toe. So that is what you will be doing during this exercise. Below you see a print screen of the game and what you will eventually produce:



The stage displays the graphical representation of the game. The black window indicates whose turn it is and also prints the board numerically. The rules of the game are simple: The object is to get three in a row. You play on a three by three game board. The first player is known as O and the second is X. Players alternate placing Os and Xs on the game board until either opponent has three in a row or all nine squares are filled. O always goes first, and in the event that no one has three in a row, the game is a tie.

If you are discouraged by now, no worries, this is easier than you think and part of the source is made available. Now start executing the following steps:

1) Let's first think of a possible data representation for the 3x3 board. It is two-dimensional and therefore needs to be an array of different arrays. We will keep things simple and declare such an array of arrays of type integer. The integers in the array represent the moves of player 1 respectively player 2. Recall that if we want to declare an array of integers we write the following statement: *int[] anarray;*

To declare our board (the array of arrays) do the following: *int[][] board;* This will be our representation of the tic tac toe board. It contains one "[]" for the rows and one "[]"for the columns. Write this statement in the Processing environment. Now also declare the following two variables: *nrofrows* and *nrofcolls*; they are meant to initialize how large our board will be, i.e., how many rows and columns it contains. Ok, we know that tic tac toe is a 3x3 board, however, we want to write code which is reusable later on, for instance for another game or for a larger board.

2) Now write the setup() function. If you cannot remember what it is meant for please look it up in the book. In the setup() function define how large you want the stage to be; it does not matter a lot how big you choose it, but please make it a square for convenience. Use the *size* function for this. Furthermore, initialize the *nrofrows* and *nrofcolls* variables with the integer 3. By now you should have something that looks like this:

```
// global variables
int[][] board;
int nrofrows;
int nrofcolls;

// the setup function

void setup(){
   nrofcolls = 3;
   nrofrows = 3;
   board = ?
}
```

The book explains how to initialize an array of integers, e.g., *items = new int[50];* Now discover yourself how you can initialize your board with the two variables for the number of rows and columns you defined. Do this and compile your sketch to see if your program does not contain errors.

3) Now you have defined your basic representation for the board it would be nice to have a function that can print this board for us in the text output box (the black zone). Later on we write a function that graphically displays the board in the stage. You should add a function to your sketch as follows:

```
Void printMatrix(){
//statements
}
```

When we call this function it prints our board of integers as follows:

0 0 0
0 0 0
0 0 0

Hint: Use a “for loop” in another “for loop”. Test your function. Also try your function for larger boards, i.e., when the number of rows and columns increases it should still work.

4) Lets now write a function *drawBoard()* to draw the board graphically in the stage. This is how you proceed:

- Declare variables “w” and “h” below the other variables you already declared in your program as an *int*. They will hold the width and the height of the rectangles in the board. In the setup function now initialize both variables: *h* as the *height* of the stage divided by *nrofrows* and *w* as the *width* of the stage divided by the *nrofcolls*:

  ```
  w= width/nrofcolls;
  h= height/nrofrows;
  ```

- With the two variables “w”and “h” we now know the measures of our rectangles. Note that in our 3x3 tic tac toe game they will be equal (at least if you defined the stage as a square, e.g.: size(200,200)). Now you can write the function *drawBoard*:

  ```
  void drawBoard(){
  }
  ```
  Hint: Think of a nested *for loop* and use the *rect* function.
  Test your function with multiple values for *nrofrows* and *nrofcolls*

5) In this step we prepare the user actions that will be done by using the mouse (put an X or put an O on the board). Therefore we need to be able to draw X’s and O’s in the rectangles of the board. To do this you need to write the following three functions with the return types and parameters as described below:

- *void drawwithX(int x, int y){}* : this function will draw an X in the square with origin *(x,y)*

- *void drawwithO(int x, int y){}* : this function will draw an O in the square with origin *(x,y)*. Hint: use the build in ellipse function for that (look up in the reference).

- Next we will implement a function that transforms the mouse coordinates of the player’s move to the corresponding row and column of the rectangle that contains the mouse coordinates. To do this you have to implement the following function:

  int[] playerMove(int mx,int my){} : This function takes as parameters the coordinates of the mouse pointer and returns an array of integers which represent the move the player actually made. An example clarifies what is meant; suppose we have the following board:

X		
	O	

Suppose I just made the move which placed the O in the middle square; I did this by pointing the mouse pointer in the middle square and clicking. Our function then takes the coordinates of the mouse pointer, e.,g. (50,50), and then has to transfer these coordinates to (1,1), which corresponds to the second row and second column in our board. Note that we start counting from 0 and not from 1. So therefore the move corresponds to (1,1) and not (2,2). Returning to our function this means that the mouse coordinates are the parameters (50,50) and the function returns an array (1,1). Write this function and test it. To help you we provide the skeleton of the function with a few missing lines (indicated in red), try to complete these lines:

```
int[] playerMove(int mx,int my){
  int[] result = new int[2];
  //search y coordinate
  for(int i =0;i<nrofcolls;i++){
  // complete the code in this for loop
    }
  }
  //search x coordinate
  for(int j =0;j<nrofrows;j++){
  // complete the code in this for loop
   }
  }
  return result;
}
```

6) Next we can write a function that actually does the move in the graphical board. It is defined as follows:

```
void doMove(int xpos, int ypos, int player){}
```

This function will do an actual move in the board and will use drawwithX and drawwithO. Each player (we have two) will have a number, i.e., either “1” or “2”. This number is also passed as a parameter to this function, indicating which player is doing the move. Parameters “xpos” and “ypos” indicate the row and column where the player wants to do his move, which are obtained by using the playerMove function from the previous step. Test your *doMove* function.

7) What we still miss now is an actual game-loop, the engine of our game that runs the game and alternates between both players. Below you find the code for this loop, add this

to your program as it appears here. Note that we use the draw() function, lookup again how it works.

```
void gameLoop(){

    // Player1 moves
    if(mousePressed && playerturn){
      noLoop();
      int[] coord = playerMove(mouseX,mouseY);
     // if(legalMove(coord[0],coord[1])){
        doMove(coord[0],coord[1],1);
        board[coord[0]][coord[1]]=1;
     //   countmoves++;
        printMatrix();
        mousePressed=false;
        playerturn = false;
     //   checkforWinner(1);
     //   if(!winner){
           println("speler 2 doe een zet: ");
     //     }
     //   else{ exit();}
        loop();
      //}
    //  else{
    //    println("illegal move player 1, gelieve een nieuwe zet te doen");
    //    loop();
    //  }
    }
    // Player2 moves
    if(mousePressed && playerturn==false){
      noLoop();
      int[] coord = playerMove(mouseX,mouseY);
     // if(legalMove(coord[0],coord[1])){
        doMove(coord[0],coord[1],2);
        board[coord[0]][coord[1]]=2;
    //    countmoves++;
        printMatrix();
        mousePressed=false;
        playerturn = true;
    //   checkforWinner(2);
    //  if(!winner){
        println("speler 1 doe een zet: ");
    //  }
    //    else{ exit();}
        loop();
    //  }
    //   else{
    //    println("illegal move player 2, gelieve een nieuwe zet te doen");
    //    loop();
    //  }
    }
}
void draw(){
  gameLoop();
}
```

Besides these two functions also add the following variable declaration to your program:
`boolean playerturn = true;`

Add the following three lines to the setup() function:

```
drawBoard();
printMatrix();
 println("speler1 doe een zet: ");
```

If everything went fine you can now start playing the game. In the black zone the computer indicates whose turn it is and also prints the board numerically. You will notice however that it is still possible to make illegal moves and that the program does not indicate a winner or a tie (if there is no winner). The calls to the code that takes care of this is commented in the game loop (ignore the noLoop() and loop() calls). Try to understand and explain what these calls and functions are meant for (they are commented by "//" in the gameLoop function). Of course you are welcome to try to implement them (it is not required for this task). Hint: if you want to implement these additional functions then use these global variable declarations:

```
int nrofmoves;
int countmoves;
boolean winner = false;
```

After you delivered your source code for this assignment you can get a complete working version of this assignment on request.

Good luck!