GAME PLAYING AND HEURISTICS: TIC-TAC-TOE

1. Heuristics for Tic-tac-toe game playing

Today we will implement a Flash movie for playing the Tic-tac-toe game. The program will allow you to play against the computer. Your sign will be an "X", while the computer will use a "O".
You can choose to play against a "random" computer or against an "clever" computer. In the first case, the computer chooses the next move randomly. In the case of the "clever" computer, it will use a problem solving strategy based on heuristic functions and MINIMAX search strategies. The heuristic functions allow the computer to assess the "fitness" of its tic-tac-toe moves (i.e. how close they are to the game goal of lining up three Os). When the computer has to decide its next move, it will assess the heuristic value of each "open" move. The computer will choose the one with the best heuristic value.
During the Problem Solving and Game Plying lecture we studied two heuristic functions for the Tic-tac-toe game:

Heuristic function 1

$$3X_2 + X_1 - (3O_2 + O_1)$$

where X_n is the number of rows/columns/diagonals with *n* "X" and no "O".

Heuristic function 2

$$Open_{COMPUTER} - Open_{PLAYER}$$

where $Open_{COMPUTER}$ is the number of rows/columns/diagonals that are still open for the computer. For instance, a row with only Os, or only empty cells, or a mix of both is still open because the computer can win with it.
In today's program we will implement the second evaluating function.

2. Preparing the Flash movie

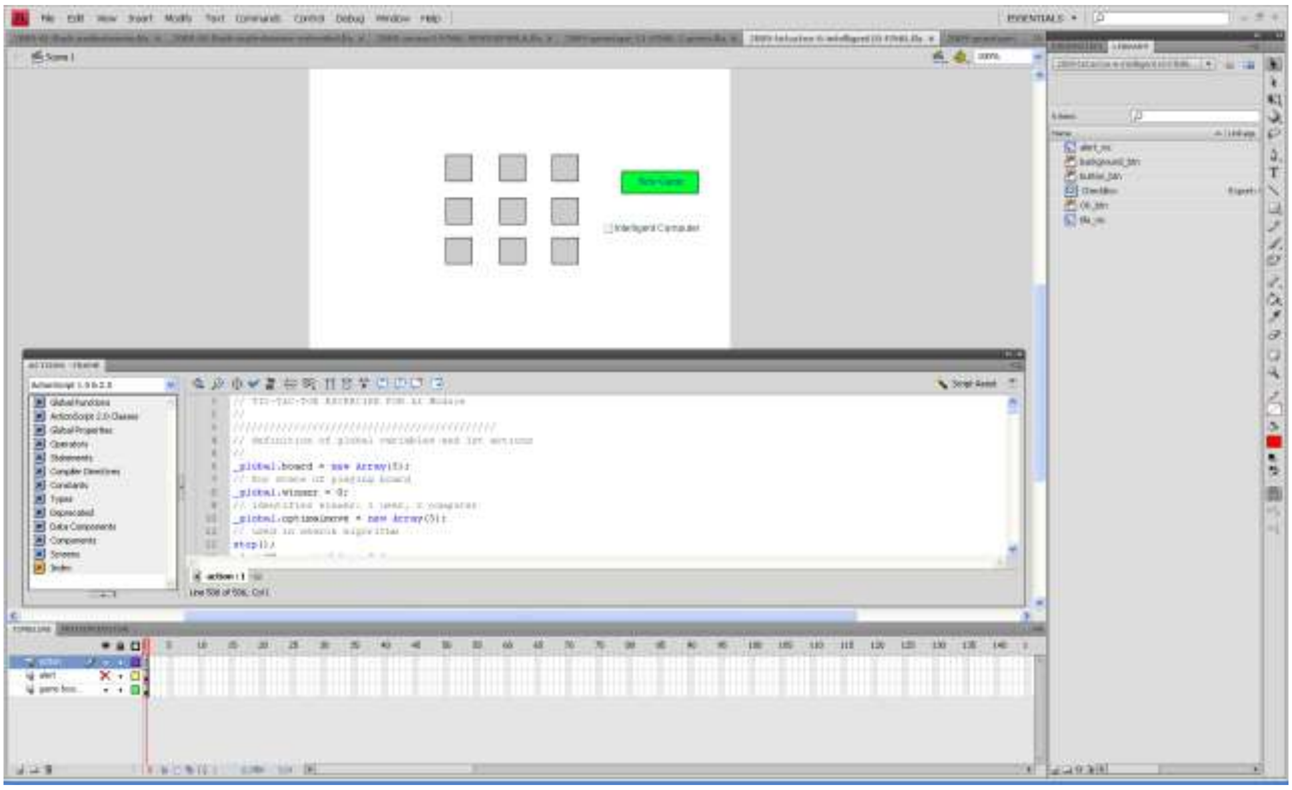
To program the Tictactoe game in Flash you will need the following components:

- Layer 1 called "actionscript" for the ActionScript code, as in previous exercises.
- Layer 2 called "interaction" for the dialog box (as in the genetic algorithm exercise), an information text field, a "new game" button and another button to switch between a clever and a random computer playing strategy
- Layer 3 called "playing board" for the 9 tiles (3x3 grid) of the playing board
- Create a movieclip symbol called "tile_mc" that will be used to make the game playing board. This symbol will contain 3 keyframes, each having a graphics for the 3 states of the tictactoe tile: empty (use a square shape), a cross, a circle. Create nine instances of this symbol and organise them in a 3x3 grid. Name the 9 instances exactly as in the following table (*rc* stands for row and column, the first number for row position 1st to 3rd and the second for column position 1st to 3rd):

rc00_mc	*rc01_mc*	*rc02_mc*
rc10_mc	*rc11_mc*	*rc12_mc*
rc20_mc	*rc21_mc*	*rc22_mc*

- Create a button symbol with a label “New Game”. Drag an instance on the movie and name it “newgame_btn”.
- Create a text field for feedback to the user and call it “info_field”.
- For the dialog box, use the same movieclip and instance names as in the genetic algorithm excecise
- Add a movieclip in the Library of symbols and name it “clever_mc”, as this will be used to switch on/off the intelligent algorithm. Add two keyframes, the first containing the text “Clever Computer” and the second “Random Computer”.

The Flash movie layout will be similar to the Figure below.



```
// TIC-TAC-TOE EXERCISE FOR AI Module
//
////////////////////////////////////////////
// definition of global variables and [illegible]
//
_global.board = new Array(3);
// [illegible]
_global.winner = 0;
// identifies winner: 1 user, 2 computer
_global.optimalmove = new Array(3);
// used in search algorithm
stop();
```

3. General Functions[1]

Add the following initialisation instructions at the beginning of the “actionscript” Layer 1.

```
// definitions of global variables
clever_mc.gotoAndStop(2);  // by default, starts random
var board:Array=[["-","-","-"],["-","-","-"],["-","-","-"]];
var winner:int=0; // identifies winner: 1 user, 2 computer
var optimalmove:Array=new Array(3); // used in search algorithm
alertOK_mc.visible=false;
```

[1] In the following sections there will be many functions to type. Therefore, if you want to avoid too many spelling mistakes, copy and paste the scripting code from the Portal. In each section you will be told WHAT they do and WHERE to put them in. There are only two handlers that are not available in the portal, i.e. `choose_this_tile` and `calculateHeuristic2`. You will have to type them in the “actionscript” Layer. You are asked to type these two functions as they are the core part of the movie. During the typing you will be able to pay attention to their content and analyse and understand them better. In any case, you are advises to READ ALL functions and try to understand what they do!

```
// link between the 9 tiles and choose_this_tile function
rc00_mc.addEventListener(MouseEvent.MOUSE_UP, choose_this_tile);
rc01_mc.addEventListener(MouseEvent.MOUSE_UP, choose_this_tile);
rc02_mc.addEventListener(MouseEvent.MOUSE_UP, choose_this_tile);
rc10_mc.addEventListener(MouseEvent.MOUSE_UP, choose_this_tile);
rc11_mc.addEventListener(MouseEvent.MOUSE_UP, choose_this_tile);
rc12_mc.addEventListener(MouseEvent.MOUSE_UP, choose_this_tile);
rc20_mc.addEventListener(MouseEvent.MOUSE_UP, choose_this_tile);
rc21_mc.addEventListener(MouseEvent.MOUSE_UP, choose_this_tile);
rc22_mc.addEventListener(MouseEvent.MOUSE_UP, choose_this_tile);
//
// actions linked to other buttons/instances
clever_mc.addEventListener(MouseEvent.MOUSE_UP, change_clever);
alertOK_mc.alertOK_btn.addEventListener(MouseEvent.MOUSE_UP,
⇔alertOK_acknowledged);
newgame_btn.addEventListener(MouseEvent.MOUSE_UP, newgame);
```

This array `board` is a global matrix of dimensions 3x3, and it corresponds to the 9 tiles of the 3x3 playing board/grid. It contains strings with the following 3 characters "`-`", "`0`", "`X`", i.e. respectively for indicating a tile that is empty, used by human player or used by the computer. The variable `winner` contains the following integer numbers `0`, `1` or `2` respectively to indicate that the winner is none, or the human player, or the computer. The vector array `optimalmove` will be used in the `playIntelligent` function to memorise the best move found. The first two positions of this vector are used to memorise the x and y coordinates of the best move. The third position is used as a tag (-9 when there is not an optimal move, 100 for a winning move, or the numerical result of the heuristic function in the `calculateheuristc2` function)
Now type (or Copy&Paste the code from the portal) the function **`newGame`** in the scripting window of the "actionscript" layer. This function will initialise the playing board and reset the global variables.

```
function newgame(event:MouseEvent) {
    var row:int,column:int;
    var tilename:String;
    board=[["-","-","-"],["-","-","-"],["-","-","-"]];
    // "-" = free, "X" = human player, "O" = computer
    winner=0;
    for (row=0; row<3; row++) {
         for (column=0; column<3; column++) {
              tilename="rc"+String(row)+String(column)+"_mc";
              this[tilename].gotoAndStop(1);
         }
    }
    info_field.text="Game started";
}
```

Note the use of the code `this[tilename].gotoAndStop(1)` to change the property of an instance whose name has been saved in the string variable `tilename`.

4. Main playing function (to choose the tile which has been clicked on)

This is the core interaction engine of the game playing, so you are asked to **type** this handler in the "actionscript" Layer 1 window.

```
function choose_this_tile(event:MouseEvent) {
   var clickedtilename:String=event.target.name;
   var myrow:int,mycolumn:int;
   if (this[clickedtilename].currentFrame==1) {// tile is free
        if ( (winner>0) || (boardcheck() == 9) ) {
             // Game over - tie; no free tiles
             alertOK_mc.visible=true;
             alertOK_mc.alert_text.text="Game over! To start
   ⇔a new game, click on the NewGame button";
             return;
        } else {
             myrow=Number(clickedtilename.substr(2,1));
             mycolumn=Number(clickedtilename.substr(3,1));
             board[myrow][mycolumn]="X";
             // the human player has used this cell
             this[clickedtilename].gotoAndStop(2);
             if (checkWinner()==1) {//Human player wins
                  alertOK_mc.visible=true;
                  alertOK_mc.alert_text.text="Congratulations,
⇔you won! Press NewGame button to play again";
                  winner=1;// the human player
                  return;
             }
             if (clever_mc.currentFrame==1) {
                  if (boardcheck()<9) {
                       playIntelligent();// computer uses heuristics
                  }
             } else {
                  if (boardcheck()<9) {
                       playRandomly();// computer plays randomly
                  }
             }
             if (checkWinner()==2) {//Computer wins
                  alertOK_mc.visible=true;
                  alertOK_mc.alert_text.text="I won!!!";
                  winner=2;// the computer
```

```
                } else {
                        if (boardcheck()==9) {// Game over - tie
                                alertOK_mc.visible=true;
                                alertOK_mc.alert_text.text="Game over! To
⇔start a new game, click on the NewGame button";
                                return;
                        }
                }
        }
    }
}
```

4.1 Service functions

The following functions are used in different parts of the program. Type (or Copy&Paste) them in the “actionscript” layer 1 window.

```
function change_clever(event:MouseEvent) {
    if (clever_mc.currentFrame==1) {
        clever_mc.gotoAndStop(2);
    } else {
        clever_mc.gotoAndStop(1);
    }
}

function alertOK_acknowledged(event:MouseEvent) {
    alertOK_mc.visible=false;
}

function randomRangeInteger(min:int, max:int):int {
// returns a random integer between the two extremes min and max
    var randomNum:int = Math.floor(Math.random()*(max-min+1))+min;
    return randomNum;
}

function boardcheck():int {
// This checks if 9 tiles are used; returns number of used cells
    var row:int,column:int;
    var sum:int=0;
    for (row=0; row<3; row++) {
        for (column=0; column<3; column++) {
            if (board[row][column]!="-") {
                sum++;
            }
```

```
          }
     }
     return sum;
}

function checkWinner():int {
//this function checks if computer-or-human has won
//returns winner's number (1 user, 2 computer)
     // first check rows
     var row:int,column:int;
     var you:int,computer:int;
     var player:String;
     var index:int;
     for (row=0; row<3; row++) {
          you=0;
          computer=0;
          for (column=0; column<3; column++) {
               player=board[row][column];
               switch (player) {
                    case "X" :
                         you++;
                         break;
                    case "O" :
                         computer++;
                         break;
               }
          }
          if (you==3) {
               return 1;
          }
          if (computer==3) {
               return 2;
          }
     }
     // check columns
     for (column=0; column<3; column++) {
          you=0;
          computer=0;
          for (row=0; row<3; row++) {
               player=board[row][column];
               switch (player) {
                    case "X" :
                         you++;
                         break;
                    case "O" :
                         computer++;
                         break;
               }
          }
          if (you==3) {
               return 1;
          }
          if (computer==3) {
               return 2;
          }
     }
     // check 1st diagonals
     you=0;
     computer=0;
     for (index=0; index<3; index++) {
          player=board[index][index];
          switch (player) {
               case "X" :
                    you++;
                    break;
               case "O" :
                    computer++;
                    break;
          }
     }
     if (you==3) {
          return 1;
     }
     if (computer==3) {
          return 2;
     }
     // check 2nd diagonal
     you=0;
     computer=0;
     for (index=0; index<3; index++) {
          player=board[index][2-index];
          switch (player) {
               case "X" :
                    you++;
                    break;
               case "O" :
                    computer++;
                    break;
          }
     }
     if (you==3) {
          return 1;
     }
     if (computer==3) {
          return 2;
     }
     return 0; // detault if no winner found
}
```

5. playRandomly function

The `playRandomly` function is called when the state of the CheckBox button “intelligent_CheckBox” is `false`. It causes the computer to choose a random cell. This used the `checkWinner` function (as previous section) which is executed after each move. If checks all 8 rows/columns/diagonals combinations to see if there are three Xs or three Os lined up.
Copy this function in the Flash movie.

```
function playRandomly():void {
// Computer chooses a random free tile
   var computerrow:int,computercolumn:int;
   var tilename:String;
   var done:int=0;
   while (done == 0) {
        computerrow=randomRangeInteger(0,2);
        computercolumn=randomRangeInteger(0,2);
        if (board[computerrow][computercolumn]=="-") {
             board[computerrow][computercolumn]="O";
   tilename="rc"+String(computerrow)+String(computercolumn)+"_mc";
             this[tilename].gotoAndStop(3);// "O"
             done=1;
        }
   }
}
```

6. PlayIntelligent function

The `playIntelligent` function is used when the state of the CheckBox button "intelligent_CheckBox" is `true`. The intelligent playing strategy is based on the use of two functions: `searchWinningMove` and `tryHeuristics`.
The function `searchWinningMove` looks in the 8 rows/columns/diagonals. It searches for a combination of one empty cell + 2 Xs or + 2 Os. These are winning lines because they allow the player, or the computer, to win. As soon as a winning line is found, the computer plays an O in the empty cell. It will allow the computer to win, or it will avoid the player to do it.
When no winning lines are found, the program will call `tryHeuristics`, and it will execute the `CalculateHeuristic2` function to assess the heuristic function of each move (using the second evaluating function in Page 1). The move with the highest heuristic value will be saved in the `optimalmove vector` array. This vector memorises the row and column indices in the first two positions, and the heuristic value in its third position.

```
function playIntelligent():void {
// it first looks for a winning move or for avoding opponent's win
// if not found, a heuristics is used
   var tilename:String;
   optimalmove=[0,0,-9];
   // -9 for worst value before move search and evaluation
   searchWinningMove();
   if (optimalmove[2]>-9) {
        // a winning/winning-avoidance move was found
        board[optimalmove[0]][optimalmove[1]]="O";
        tilename =
⇔"rc"+String((optimalmove[0]))+String(optimalmove[1])+"_mc";
```

```
            this[tilename].gotoAndStop(3);
            // "O"
        } else {
            // winning move not found, now try heuristics
            tryHeuristics();
            if (optimalmove[2]>-9) {
                // a good heuristic move was found
                board[optimalmove[0]][optimalmove[1]]="O";
                tilename =
⇔"rc"+String((optimalmove[0]))+String(optimalmove[1])+"_mc";
                this[tilename].gotoAndStop(3);
                // "O"
            } else {
                playRandomly();
            }
        }
    }

    function searchWinningMove():void {
        // it checks if there is a move for winning,
        // or a move for avoiding player's victory
        // if found, the move is executed
        // First check rows
        var row:int,column:int;
        var you:int,computer:int;
        var player:String;
        var index:int;
        var winningmove:Array=new Array(2);
        for (row=0; row<3; row++) {
            you=0;
            computer=0;
            winningmove=[-1,-1];
            for (column=0; column<3; column++) {
                player=board[row][column];
                switch (player) {
                    case "-" :
                        winningmove[0]=row;
                        winningmove[1]=column;
                        break;
                    case "X" :
                        you++;
                        break;
                    case "O" :
                        computer++;
                        break;
                }
            }
            if ( ( (you == 2) || (computer == 2) ) &&
⇔(winningmove[0]>-1) ) {
                optimalmove[0]=winningmove[0];
                optimalmove[1]=winningmove[1];
                optimalmove[2]=999;
                return;
            }
        }
        // check columns
        for (column=0; column<3; column++) {
            you=0;
            computer=0;
            winningmove=[-1,-1];
            for (row=0; row<3; row++) {
                player=board[row][column];
                switch (player) {
                    case "-" :
                        winningmove[0]=row;
                        winningmove[1]=column;
                        break;
                    case "X" :
                        you++;
                        break;
                    case "O" :
                        computer++;
                        break;
                }
            }
            if ( (you == 2 || computer == 2) && winningmove[0]>-1) {
                optimalmove[0]=winningmove[0];
                optimalmove[1]=winningmove[1];
                optimalmove[2]=999;
                return;
            }
        }
        // check 1st diagonal
        you=0;
        computer=0;
        winningmove=[-1,-1];
        for (index=0; index<3; index++) {
            player=board[index][index];
            switch (player) {
                case "-" :
                    winningmove[0]=index;
                    winningmove[1]=index;
                    break;
               case "X" :
                    you++;
                    break;
               case "O" :
                    computer++;
                    break;
          }
     }
     if ((you == 2 || computer == 2) && winningmove[0]>-1) {
          optimalmove[0]=winningmove[0];
          optimalmove[1]=winningmove[1];
          optimalmove[2]=999;
          return;
     }
     // check 2nd diagonal
     you=0;
     computer=0;
     winningmove=[-1,-1];
     for (index=0; index<3; index++) {
          player=board[index][2-index];
          switch (player) {
               case "-" :
                    winningmove[0]=index;
                    winningmove[1]=2-index;
                    break;
               case "X" :
                    you++;
                    break;
               case "O" :
                    computer++;
                    break;
          }
     }
     if ((you == 2 || computer == 2) && winningmove[0]>-1) {
          optimalmove[0]=winningmove[0];
          optimalmove[1]=winningmove[1];
          optimalmove[2]=999;
          return;
     }
}
```

```
  function tryHeuristics():void {
     // it uses the heuristic to decide next move
     // if nothing is found, plays randomly
     var computerrow:int,computercolumn:int;
     for (computerrow=0; computerrow<3; computerrow++) {
          for (computercolumn=0; computercolumn<3; computercolumn++) {
               if (board[computerrow][computercolumn]=="-") {
                    // tile is free
                    calculateHeuristic2(computerrow, computercolumn);
               }
          }
     }
  }
```

6.1 calculateHeuristic2 function for $Open_{COMPUTER} - Open_{PLAYER}$

You will have to type this function as this is the most important part of the AI program applying the heuristic function $Open_{COMPUTER} - Open_{PLAYER}$

```
function calculateHeuristic2(possiblerow, possiblecolumn):void {
     // heurist function 2 = OpenX - OpenO
     // the computer (player X) looks for max value
     // extra check for choices with 2 Os
     // when 2 Os are found, chosen move will stop it
     // results saved into optimalmove[]
     var row:int,column:int;
     var you:int,computer:int;
     var player:String;
     var index:int;
     var boardtest:Array=new Array(3);
     var OpenX:int,OpenO:int;
     boardtest[0]=board[0].slice();
     boardtest[1]=board[1].slice();
     boardtest[2]=board[2].slice();
// needs slice() to make independent copy of original board array
// with 2-dmensional arrays, slice only for works for 1D-arrays
     boardtest[possiblerow][possiblecolumn]="O";
     var openX:int=0;
     var openO:int=0;
     // check rows
     for (row=0; row<3; row++) {
          you=0;
          computer=0;
          for (column=0; column<3; column++) {
               player=boardtest[row][column];
               switch (player) {
                    case "-" :
                         you++;
                         computer++;
                         break;
                    case "X" :
                         you++;
                         break;
                    case "O" :
                        computer++;
                        break;
                }
            }
            if (you==3) {
                openX++;
            }
            if (computer==3) {
                openO++;
            }
        }
        // check columns
        for (column=0; column<3; column++) {
            you=0;
            computer=0;
            for (row=0; row<3; row++) {
                player=boardtest[row][column];
                switch (player) {
                    case "-" :
                        you++;
                        computer++;
                        break;
                    case "X" :
                        you++;
                        break;
                    case "O" :
                        computer++;
                        break;
                }
            }
            if (you==3) {
                openX++;
            }
            if (computer==3) {
                openO++;
            }
        }
        // check 1st diagonal
        you=0;
        computer=0;
        for (index=0; index<3; index++) {
            player=boardtest[index][index];
            switch (player) {
                case "-" :
                    you++;
                    computer++;
                    break;
                case "X" :
                    you++;
                    break;
                case "O" :
                    computer++;
                    break;
            }
        }
        if (you==3) {
            openX++;
        }
        if (computer==3) {
          openO++;
     }
     // check 2nd diagonal
     you=0;
     computer=0;
     for (index=0; index<3; index++) {
          player=boardtest[index][2-index];
          switch (player) {
               case "-" :
                    you++;
                    computer++;
                    break;
               case "X" :
                    you++;
                    break;
               case "O" :
                    computer++;
                    break;
          }
     }
     if (you==3) {
          openX++;
     }
     if (computer==3) {
          openO++;
     }
     if ((openO-openX)>optimalmove[2]) {
          // move better than previous
          optimalmove[0]=possiblerow;
          optimalmove[1]=possiblecolumn;
          optimalmove[2]=openO-openX;
     }
}
```

7. Play!

Now that you have written all functions you should be able to play the Tic-tac-toe game against the computer. Remember that to play against an “intelligent” computer, you have to select “intelligent_CheckBox”.

ASSESSMENT EXERCISE 2

Redesign the user interface of the present game. For example, you can add a function for showing the player’s and computer’s scores, improve the graphics, and so on.

Alternatively, you can write a new function for computing the first heuristic function (see the evaluating function 1 in the first page).

Assessment criteria

This exercise is worth 15% of marks.
To mark this exercise, the following questions will be used:

a) Does the program work correctly?
b) Is the heuristic 2 working?
c) Has the interface been improved?
d) Are players' scores shown?

Alternatively, only this criterion will be considered

a) Was the first heuristic implemented?