Monica-Pauline BLANC
Lise SPILERS
L1-INT

Black & White

Project n°2

Table of contents :

I. Introduction

a. A quick presentation of the game

This game is a strategy game with two players. We had the choice to create a game where someone plays against the computer, or against another person. We decide to play with two humans' player and don't plays with AI because we didn't arrive to do this. One player plays with Black pawns and the others with the White. A pawn is double-sided (white and black. The game board is 8x8 boxes but in our project, we have to program a variable game board so the users can play max with a 26x26 board (because we have 26 letters in the alphabet). The game is played alternately: white pawns returns black pawns and vice-versa. In our game, we didn't use colors: we replace black by "X" and white by "O".

How to win?

There are three possibilities to win:

- First, by strategy: one player manages to block the other. It means that the player returns all his opponent's pawns: the player two has no more pawns, he can't play again, player one won.
- Second, by elimination: When the board is empty, the game is finish so players count their pawns: more pawns of its colors than the opponent win. If they have the same number of pawns, there is no winner: equality!
- Third, by abandon: if one player abandon, the other win.

b. Main rules of the game

For the rest of the explication we are going to talk about an 8x8 boxes. At the beginning of the game, we have two pawns black and two pawns white places in E4 and D5 (black) and D4 and D5 (white). So, we have four pawns in the middle of the board. It's Black which starts.

When he plays, the player must place a pawn of its color in a fill box, adjacent to an opposing pawn. So, for example, if the player one, who has black pawns, can only place its pawns next to white pawns. Also, he must place his pawn enclose one or several opposing pawns between the pawn he places and a pawn of his color, already placed on the game board. He then returns the pawn(s) he has just enclosed (which become of his color). The pawns are neither removed from the game board nor moved from one box to another.

The enclosure is done according to the lines, the columns and the diagonals.

When a pawn is placed, the player returns all the opposing pawns enclosed by this pawn and his pawns already placed. In the rest of the statement, all examples will be shown on a 4x4 game board, but can be transposed on the original 8x8 board.

II. The different algorithm

a. Preparation of the Game

1- Function resume. This function display a little resum at the end of eatch turn : the number of the turn, the number of X and O and who hav to play.

```
Function resume (turn, X, O: integer; Player1, Player2: String)
Changed parameter: /
Copied parameters: turn, X, O, Player1, Player2
Local Variable: /
Begin
    Write ("turn:", turn, "\nX=", X, "\nO=", O)
    If (turn%2 = 0) then,
        return 'Player1,it is your turn. What do you want to do?'
    else then,
        return 'Player2,it is your turn. What do you want to do?'
    end if
End
```

2- Function gameboard. This function display the board.

```
Function GameBoard (Table:[][] : Two dimensional array of integer ;
size:integer ; lett[]:array of letters ; num[]:array  of string,
lett2[]: array of letters ; num2[]:array  of string)
Changed parameter:
Copied parameters: Table, size, lett, num, lett2, num2
Local Variables: i,j (integer)
Begin
    If (size<10)then
        For i ← 1 to size, do:
            Write(lett[i])
        End for
        For i ← 1 in range (size):
            Write(num[i])
            For j ←1 to size, do:
                Print (Table[i][j])
            End for
        End for
    Else
        For i ← 1 to size, do:
            Write(lett2[i])
        End for
        For i ← 1 in range (size):
            If i<10 then
                Write(num2[i])
            Else
                Write(num[i])
            End if
            For j ←1 to size, do:
                Print (Table[i][j])
            End for
        End for
End
```

We use the list lett and num when the user chooses a size bottom than 10 and the list lett2 and num2 in the other case because these two lists have different space between the string that allows to have a good alignment of board's points.

3- Function init. This function displays the position of the four pawns at the beginning of the game.

```
Function init (Table:two dimension array of integer; size:integer ;
lett[]:array of letters ; num[]:array  of string)
Changed parameter:/
Copied parameters: Table, size, let, num
Local Variables: i,j (integer)
Begin
    For i ← 1 to size+1, do:
        Table[int((size+1)/2)-1] [int((size+1)/2)]←" 0"
        Table[int((size+1)/2)] [int((size+1)/2)+1]←" 0"
        Table[int((size+1)/2)] [int((size+1)/2)]←" X"
        Table[int((size+1)/2)-1] [int((size+1)/2)+1]←" X"
    End for
End
```

4- Function movesonboard. This Boolean function return if the coordinates entered by the user or good or not

```
Function movesonboard (pll, placingnum, size: integers): Boolean
Changed parameter: /
Copied parameters:pll, placingnum, size
Local variable: /
Begin
    if placingnum>=0 and placingnum<= size and  pll<=size then
        return True
    if Table[placingnum][num]= Table[int((size+1)/2)-
1][int((size+1)/2)-1] or Table[placingnum][num]==
Table[int((size+1)/2)-1][int((size+1)/2)] or
Table[placingnum][num]==Table[int((size+1)/2)][int((size+1)/2)-1] or
Table[placingnum][num]==Table[int((size+1)/2)][int((size+1)/2)]:
        return False
  else:
        return False
    end if
End
```

It returns True if the coordinates are located on the board, and False there are out of the board?)

b. Invalid moves

5- Function invalidmoves

So this is our main difficulty . It contain 8 checks for all direction. We decided to do eight function for each checks because it was more simple to see if there is a problem, and it's more clear to read. Then we resume in one function all the invalid moves.

```
Function North(Table:two dimension array of integer; turn,
placingnum ,pll, size  :integer ;
Changed parameter:
```

```
Copied parameters:
Local Variables:
Begin
    Num← placingnum
    if turn%2=0 then
        if Table[num-1][pll]=” X” then
            return False
        end if
        while Table[num-1][pll] = " O" and num-1>0 do
            num← num - 1
        end while
        if num-1=0 then
            return False
        else
            if Table[num-1][pll]=" X" then
                return True
            end if
            return False
        end if
    end if
    if turn%2=1 then
        if Table[num-1][pll]=” O” then
            return False
        end if
        while Table[num-1][pll] = " X" and num-1>0 do
            num← num - 1
        end while
        if num-1=0 then
            return False
        else
            if Table[num-1][pll]=" O" then
                return True
            end if
            return False
        end if
    end if
end

Function South(Table:two dimension array of integer; turn,
placingnum ,pll, size  :integer ;
Changed parameter:
Copied parameters:
Local Variables:
Begin
    Num← placingnum
    if turn%2=0 then
        if Table[num+1][pll]=” X” then
            return False
        end if
        while Table[num+1][pll] = " O" and num<size do
            num← num + 1
        end while
        if num=size then
            return False
        else
```

```
                if Table[num+1][pll]=" X" then
                    return True
                end if
                return False
            end if
        end if
        if turn%2=1 then
            if Table[num+1][pll]=” O” then
                return False
            end if
            while Table[num+1][pll] = " X" and num<size do
                num← num +1
            end while
            if num=size then
                return False
            else
                if Table[num+1][pll]=" O" then
                    return True
                end if
                return False
            end if
        end if
end

Function East (Table:two dimension array of integer; turn,
placingnum ,pll, size  :integer ;
Changed parameter:
Copied parameters:
Local Variables:
Begin
        Pll2← pll
        if turn%2=0 then
            if Table[placingnum][pll2+1]=” X” then
                return False
            end if
            while Table[placingnum][pll+1] = " O" and pll2<size do
                pll2← pll2+1
            end while
            if pll2=size then
                return False
            else
                if Table[placingnum][pll2+1]=" X" then
                    return True
                end if
                return False
            end if
        end if
        if turn2=%1 then
            if Table[placingnum][pll2+1]=” O” then
                return False
            end if

            while Table[placingnum][pll+1] = " X" and pll2<size do
                pll2← pll2+1
```

```
        end while
        if pll2=size then
            return False
        else
            if Table[placingnum][pll2+1]=" O" then
                return True
            end if
            return False
        end if
    end if
end
```

```
Function West(Table:two dimension array of integer; turn, placingnum
,pll, size  :integer ;
Changed parameter:
Copied parameters:
Local Variables:
Begin
    Pll2← pll
    if turn%2=0 then
        if Table[placingnum+1][pll2-1]=" X" then
            return False
        end if
        while Table[placingnum][pll2-1] = " O" and pll2-1>0 do
            pll2← pll2-1
        end while
        if pll2-1=0 then
            return False
        else
            if Table[placingnum][pll2-1]=" X" then
                return True
            end if
            return False
        end if
    end if
    if turn2=%1 then
        if Table[placingnum][pll2-1]=" O" then
            return False
        end if
        while Table[placingnum][pll2-1] = " X" and pll2-1>0 do
            pll2← pll2-1
        end while
        if pll2-1=0 then
            return False
        else
            if Table[placingnum][pll2-1]=" O" then
                return True
            end if
            return False
        end if
    end if
end
```

```
Function Northeast(Table:two dimension array of integer; turn,
placingnum ,pll, size  :integer ;
Changed parameter:
Copied parameters:
Local Variables:
Begin
    Num← placingnum
    Pll2←pll
    if turn%2=0 then
        if Table[num-1][pll2+1]=” X” then
            return False
        end if
        while Table[num-1][pll2+1] = " O" and num-1>0 and
    pll2<size do
            num← num - 1
            pll2 ← pll2+1
        end while
        if num-1=0 or pll2=size then
            return False
        else
            if Table[num-1][pll2+1]=" X" then
                return True
            end if
            return False
        end if
    end if
    if turn%2=1 then
        if Table[num-1][pll2+1]=” O” then
            return False
        end if
        while Table[num-1][pll2+1] = " X" and num-1>0 and
    pll2<size do
            num← num - 1
            pll2←pll2+1
        end while
        if num-1=0 or pll2=size then
            return False
        else
            if Table[num-1][pll2+1]=" O" then
                return True
            end if
            return False
        end if
    end if
end

Function Northwest(Table:two dimension array of integer; turn,
placingnum ,pll, size  :integer ;
Changed parameter:
Copied parameters:
Local Variables:
Begin
    Num← placingnum
    Pll2←pll
```

```
    if turn%2=0 then
        if Table[num-1][pll2-1]=” X” then
            return False
        end if
        while Table[num-1][pll2-1] = " O" and num-1>0 and pll2-
    1>0 do
            num← num – 1
            pll2 ← pll2-1
        end while
        if num-1=0 or pll2-1=0 then
            return False
        else
            if Table[num-1][pll2-1]=" X" then
                return True
            end if
            return False
        end if
    end if
    if turn%2=1 then
        if Table[num-1][pll2-1]=” O” then
            return False
        end if
        while Table[num-1][pll2-1] = " X" and num-1>0 and pll2-
    1>0 do
            num← num – 1
            pll2←pll2-1
        end while
        if num-1=0 or pll2-1=0 then
            return False
        else
            if Table[num-1][pll2-1]=" O" then
                return True
            end if
            return False
        end if
    end if
end

Function Southeast(Table:two dimension array of integer; turn,
placingnum ,pll, size  :integer ;
Changed parameter:
Copied parameters:
Local Variables:
Begin
    Num← placingnum
    Pll2←pll
    if turn%2=0 then
        if Table[num+1][pll2+1]=” X” then
            return False
        end if
        while Table[num+1][pll2+1] = " O" and num<size and
    pll2<size do
            num← num + 1
            pll2 ← pll2+1
        end while
```

```
        if num=size or pll2=size then
            return False
        else
            if Table[num+1][pll2+1]=" X" then
                return True
            end if
            return False
        end if
    end if
    if turn%2=1 then
        if Table[num+1][pll2+1]=” O” then
            return False
        end if
        while Table[num+1][pll2+1] = " X" and num<size and
    pll2<size do
            num← num + 1
            pll2←pll2+1
        end while
        if num=size or pll2=size then
            return False
        else
            if Table[num+1][pll2+1]=" O" then
                return True
            end if
            return False
        end if
    end if
end

Function Southwest(Table:two dimension array of integer; turn,
placingnum ,pll, size  :integer ;
Changed parameter:
Copied parameters:
Local Variables:
Begin
    Num← placingnum
    Pll2←pll
    if turn%2=0 then
        If Table[num+1][pll2-1]=” X” then
            return False
        end if
        while Table[num1][pll2-1] = " O" and pll2-1>0 and
    pnum<size do
            num← num +1
            pll2 ← pll2-1
        end while
        if num=size or pll2-1=0 then
            return False
        else
            if Table[num+1][pll2-1]=" X" then
                return True
            end if
            return False
        end if
    end if
```

```
      if turn%2=1 then
            if Table[num+1][pll2-1]=” O” then
                  return False
            end if
            while Table[num+1][pll2-1] = " X" and pll2-1>0 and
      num<size do
                  num← num + 1
                  pll2←pll2-1
            end while
            if num=size or pll2=0 then
                  return False
            else
                  if Table[num+1][pll2-1]=" O" then
                        return True
                  end if
                  return False
            end if
      end if
end
```

6- Function which check if we can place the pawn

```
function Invalid moves (Table : two dimensional array,
placingnum,pll,turn,size : integer ):Boolean
Changed parameter:
Copied parameters:
Local Variables:
Begin
    if north(Table,placingnum,pll,turn) = True then
        return True
    elif south(Table,placingnum,pll,turn,size) = True then
        return True
    elif east(Table,placingnum,pll,turn,size) = True then
        return True
    elif west(Table,placingnum,pll,turn) = True then
        return True
    elif northeast(Table,placingnum,pll,turn,size) = True then
        return True
    elif nortwest(Table,placingnum,pll,turn) = True then
        return True
    elif southeast(Table,placingnum,pll,turn,size) = True then
        return True then
    elif southwest(Table,placingnum,pll,turn,size) = True then
        return True
    end if
    return False
end
```

c. Valid moves

7- Function Valid moves. This function show in the board the valid moves. We didn’t know if it works, but we wanted to put it in the report in order to first verify by you if it’s

correct and also wanted to show you that we didn't stop there. It's not because we didn't arrive to do invalid moves that we stopped.

```
Fonction request1(Table : two dimensional array of integer, size :
integer ):
Changed parameter:
Copied parameters:
Local variable:
Begin
       listvalidmoves ← []
       for placingnum to size do
           for pll to size do
               if east(size,pll,placingnum,Table,turn)=True or
   north(size,pll,placingnum,Table,turn)=True or
   north_east(size,pll,placingnum,Table,turn)=True or
   south_east(size,pll,placingnum,Table,turn)=True or
   south(size,pll,placingnum,Table,turn)==True or
   north_west(size,pll,placingnum,Table,turn)=True or
   south_west(size,pll,placingnum,Table,turn)=True or
   west(size,pll,placingnum,Table,turn)=True then
                   listvalidmoves.append([placingnum,pll])
               end if
           end for
      end for
   return listvalidmoves
end

Function request2(size,placingnum,pll, : integers ; Table :  two
dimension array of integer ; lett,lett2,num2,num : array of string ;
listvalidmoves : array of integer ):
Changed parameter:
Copied parameters:
Local variable:
Begin
      Copygameboard ← Gameboard(Table,size,lett,num,lett2,num2)
      for placingnum, pll in listvalidmoves(copyboard):
           copyboard[placingnum][pll] = ' -'
      end for
      return copyboard
```

d. Flip and place pawn

7- Function which flip pawn

```
def flip (turn,placingnum,pll,size : integer, Table : two dimension
array of integer)
Begin
      if north(size,pll,placingnum,Table,turn)= True then
           if turn%2=1 then
                y←placingnum-1
                while Table[y][pll] = " X" do
                     Table[y][pll] ← " O"
                     y ← y-1
                end while
           end if
           if turn%2=0then
```

```
            y←placingnum-1
            while Table[y][pll] = " O" do
                Table[y][pll] ←" X"
                y ← y-1
            end while
      end if
end if

if north_east(size,pll,placingnum,Table,turn)=True then
      if turn%2=1 then
            y←placingnum-1
            x←pll+1
            while Table[y][x] = " X" do
                Table[y][x]← " O"
                X← x+ 1
                y ← y- 1
            end while
      end if
      if turn%2=0 then
            y←placingnum-1
            x←pll+1
            while Table[y][x] = " O" do
                Table[y][x]← " X"
                x ← x + 1
                y← y - 1
            end while
      end if
end if

if north_west(size,pll,placingnum,Table,turn)=True:
      if turn%2=1 then
            y←placingnum-1
            x←pll-1
            while Table[y][x] = ' X' do
                Table[y][x]← ' O'
                X← x- 1
                y ←y- 1
            end while
      end if
      if turn%2=0 then
            y←placingnum-1
            x←pll-1
            while Table[y][x] = " O" do
                Table[y][x]←" X"
                x ←x- 1
                y← y- 1
            end while
      end if
end if

if south(size,pll,placingnum,Table,turn)=True then
      if turn%2=1 then
            y←placingnum-1
            while Table[y][pll] = " X" do
```

```
                Table[y][pll]←" O"
                y ← y+ 1
            end while
        end if
        if turn%2=0 then
            y←placingnum+1
            while Table[y][pll]= " O" do
                Table[y][pll]← " X"
                y ← y+ 1
            end while
        end if
end if

if south_west(size,pll,placingnum,Table,turn)=True then
        if turn%2=1 then
            y←placingnum+1
            x←pll-1
            while Table[y][x] = " X" do
                Table[y][x]← " O"
                x ←x - 1
                y← y+ 1
            end while
        end if
        if turn%2=0 then
            y←placingnum+1
            x←pll-1
            while Table[y][x] = " O" do
                Table[y][x]← " X"
                X← x- 1
                y←y+ 1
            end while
        end if
end if

if south_east(size,pll,placingnum,Table,turn)= True then
        if turn%2=1 then
            y←placingnum+1
            x←pll+1
            while Table[y][x] = " X" do
                Table[y][x]← " O"
                x ←x+ 1
                y ←y+ 1
            end while
        end if
        if turn%2=0 then
            y←placingum+1
            x←pll+1
            while Table[y][x] = " O" do
                Table[y][x]← " X"
                x ←x + 1
                y← y+ 1
            end while
        end if
end if
```

```
      if east(size,pll,placingnum,Table,turn)= True then
      |     if turn%2=1 then
      |     |     x←pll+1
      |     |     while Table[placingnum][x] = " X" do
      |     |     |     Table[placingnum][x]← " O"
      |     |     |     x ← x+ 1
      |     |     end while
      |     |     Table[placingnum][x]← " O"
      |     end if
      |     if turn%2=0 then
      |     |     x←pll+1
      |     |     while Table[placingnum][x] = " O" do
      |     |     |     Table[placingnum][x]←" X"
      |     |     |     x ← x+ 1
      |     |     end while
      |     |     Table[placingnum][x]←" X"
      |     end if
      end if

      if west(size,pll,placingnum,Table,turn)=True then
      |     if turn%2=1 then
      |     |     x←pll-1
      |     |     while Table[placingnum][x] =" X" do
      |     |     |     Table[placingnum][x]←" O"
      |     |     |     x ← x- 1
      |     |     end while
      |     end if
      |     if turn%2=0 then
      |     |     x ← pll-1
      |     |     while Table[placingnum][x] = " O" do
      |     |     |     Table[placingnum][x]← " X"
      |     |     |     x ← x - 1
      |     |     end while
      |     end if
      end if
end
```

8- Function placingpawns. This function places the new pawns.

```
Function placingpawns (size, pll, placingnum, turn: integers; Table
: two dimensional array of integer)
Changer parameter: Table
Copied parameters: size, pll, placingnum, turn
Local variable: /
Begin
      If flip(turn,Table,placingnum,pll,size)=True then
            If (turn%2 = 0) then
                  if east(size,pll,placingnum,Table,turn)=True or
north(size,pll,placingnum,Table,turn)=True or
north_east(size,pll,placingnum,Table,turn)=True or
south_east(size,pll,placingnum,Table,turn)=True or
south(size,pll,placingnum,Table,turn)=True or
north_west(size,pll,placingnum,Table,turn)=True or
```

```
south_west(size,pll,placingnum,Table,turn)=True or
west(size,pll,placingnum,Table,turn)=True then
                Table[placingnum][pll]← " X"
     Else
          if east(size,pll,placingnum,Table,turn)=True or
north(size,pll,placingnum,Table,turn)=True or
north_east(size,pll,placingnum,Table,turn)=True or
south_east(size,pll,placingnum,Table,turn)=True or
south(size,pll,placingnum,Table,turn)=True or
north_west(size,pll,placingnum,Table,turn)=True or
south_west(size,pll,placingnum,Table,turn)=True or
west(size,pll,placingnum,Table,turn)=True then
                Table[placingnum][pll]← " O"
     End if
End
```

a. End of the game

9- Function Continue. This function allows to stop or continue the game. It stop the game if a player abandon, if there is no more "X" or no more "O", or if the board is empty.

```
Function Continue (X,O, size: integers; command: string):Boolean
Changed parameter:/
Copied parameters: X, O, size, command
Local variable:/
Begin
    If (X=size*size) or (O=size*size) then
        Return False
    End if
    if (X+O = size*size) then,
        Return False
    End if
    if (command = "A") or (command="a") then
        Return False
     End if
     Return True
End
```

10- Function winner. It just displays who is the winner.

```
Function winner (O, X, turn: integers; Player1, Player2, command :
string)
Changed parameter:/
Copied parameters: O, X, Player1, Player2, turn, command
Local variable:/
Begin
     If (command = A) then,
          If (turn%2=0) then,
               Write (player1, "you choose to abandon the game, you
               loose.", Player2, "is the winner ! See you soon !")
```

```
        Else then,
            Write (player2, "you choose to abandon the game, you
            loose.", Player1, "is the winner ! See you soon !")
        End if

    If (O<X) then,
        Write ("Congratulation", Player1, "! You won !")
    Else if (X=O) then,
        Write ("No winner, play again !")
    Else then,
        Write ("Congratulation", Player2, "! You won !")
    End if
End
```

III. Conclusion

a. Analysis of the project

The project wasn't so rough but for Lise and I, it was hard to begin because it's wasn't guided as the first project : so we didn't know how to start. It's clear that we talked with our friend and saw how they began. We talked with a lot of groups and all the groups started differently but it gives us hints to start. When we started to code, we see that it's wasn't so hard but when we arrive to do all the checks we blocked. We arrive to do all the things without checks in about 2 weeks, and after we blocked. We are so disappointed to send you an unfinished project because we know that if we had more times maybe we will finish it.

Hypothesis: Even after this failure, we won't stop. So we decided to continue to code after the release date. Maybe, the day of the oral we will show you an new code or finished project.

b. Difficulty encountered

The first difficulty was to build the gameboard :

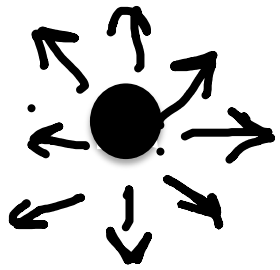
- First we know that the gameboard is variable so when the user want to play with a gameboard above an 8X8 board, we have a problem : when we pass from 9 to 10, there is an additional number. So we had to put the list (lett) on equal footing with points because when you arrive to 10, there are two number (1 and 0) so from 10 all the points are shifted. Moreover, the list (lett) have space so we didn't arrive to synchronize points and A,B,C .... How we solve it ? In the function "Gameboard" we do two different cases when size<10 and when size>=10. So in the different cases we do different list with different spaces between the letter and the "".

The second difficulty was to know where place the initialize pawns because the board is variable. For a 8X8 board, black are at e4 and d5 and two white pawns are placed in d4 and e5. Black plays first. (black = X ; white = O). But what's for 6X6 ? 10X10 ? ...

- We draw in the paper the gameboard and we understand that the X are placed in the size/2 (for column) and size/2 + 1 (for line) and the reverse, the same thing for O but the reverse. So we use a 2dimensional array called Table. The first bracket correspond to the column, the second the line : (we say size+1 because python begin from 0, so it's not +1 but -1)

```
#3 function which is place pawns in the initialize place whatever the size of t
def init(size,lett,num,Table):
    for i in range (0,size+1):
        Table[int((size+1)/2)-1][int((size+1)/2)-1]=" O"   #when it's 8X8 : D4
        Table[int((size+1)/2)-1][int((size+1)/2)]=" X"     #                E5
        Table[int((size+1)/2)][int((size+1)/2)-1]=" X"     #                D5
        Table[int((size+1)/2)][int((size+1)/2)]=" O"       #                E4
```

Our main difficulty was the invalid moves. So, we know that we have to check in all direction for each wanted placement of the user. We understand that we have to check in 8 directions : North, North-East, North-West, South, South-East, South-West, West, East.



We did several test of our checks but we don't know why it doesn't work. We think that maybe it's because of little stupid error that we don't see.
Because of the fact that our invalid moves doesn't, we were blocked for the other functions as valid moves, winner, …

c. What we didn't finish

So as we said, we didn't finish the project: we didn't arrive to build a real function for the invalid moves. We did several test, but even now we don't know why it doesn't work. When we are in 8X8 board for example, we enter A2, so it's normal that all check respond "False" because we can't place there but when we enter for example C4 it also print "False" (when it's X to play, here X = black). Other example with X, if we enter F5 the check "West" print "True". We really doesn't know where are the errors but because of that we blocked and didn't know how to solve it.
Moreover, we did not do the function for passing the turn.
But we did a function for the valid moves : it's actually two function ("request1" and "request2"). Our goal was to change point into "--" for all valid moves instead of print a list of coordinates.
We also didn't do the function to count X and O

d. Conclusion

After a month working on this project, Lise and I are surprised and disappointed to receive this project in second project of our first semester : we knew that we aren't sufficiently experienced to "actually" handle the subtleties of this game that are not mentioned in this report. It's also for that that we didn't finish the project, we know that we were very close but we blocked to the check in all direction and we didn't know how to continue without it. So we try to secure what we have to secure. Nevertheless, this project enabled us to take on multiple challenges and see several difficulties that we tried to solve.
This project was a good project to handle functions and finish this semester.