**----------------------------------------------------------------------------**

**Author: Zhong Zhenghang (Klaus)**

**Date: Oct, 2016 - Dec, 2016**

**----------------------------------------------------------------------------**

**EEE101 C Programming and Software Engineering 1 – Connect 4**

1. **Problem statement.**

Design a chess game called connect 4. Firstly, the program should give players choice to decide whether to play with another player or with computer. Secondly, the program should display a grid of size of 6x7 (6 rows and 7 columns) with every space blank for players to place counters. (Also it would be better to provide an option to change the matrix size and the winning length). Then the program should display the current state of board to each player, and as the matrix is perpendicular to the ground, every counter could only be placed on ground or other counters. Finally, the program could declare the winner when someone connects 4 same counters and it could send warning when players enter an unsatisfactory number to increase the robustness of the game. Moreover, some other functions could be added to enhance the interest and personality.

1. **Analysis**

Input:

* A character to represent the choice of playing with computer or another player.
* A character to represent the choice of choosing a classic mode (6x7, connect 4) or change rules.
* (When choose change rules) three integers to represent the rows, columns and winning length of matrix.
* A character to represent the columns players want to place a counter in.
* A character to represent the choice whether players want to restart the game.

Output:

* The rules of connect 4.
* Ask if players are ready to start.
* Ask player 1 to choose to play with player 2 or computer.
* Ask players whether to start with classic mode or change matrix size and winning length.
* Ask players to place counters.
* Ask players to enter a satisfactory number if they enter an incorrect integer or character.
* Place the times both players have enters in current time.
* Ask players to change a column to place counter when it is filled.
* Declare the winner when someone wins or a draw state happens.
* Ask if players want to start again.

1. **Design**

Algorithm:

* 1. Declare a function namely \_initial. This function has a second level pointer input of type of char which point to a 2D array and 2 variable of type of int and namely **ROWS** and **COLS**, return nothing. This function can initialize the 2D array and show it on the screen.
  1. Declare two variables namely **i** and **j** representing rows and columns separately. Then use tow functions of **for loop** to initialize every elements of matrix as ‘**-**‘.
  2. Use another 3 functions of **for** loop to present every element and numbers of columns on the top of matrix, which helps players to see clearly where to put counters.
* 2. Declare a function namely **\_playing**. This function has a second level pointer namely **Grid** of type char and 4 variables namely **ROWS, COLS, k** and **Turn** of type int input, and return an integer. This function can display the current state of matrix.

2.1 Declare two variables namely **i** and **j** representing rows and columns separately.

2.2 The function runs every time players enter a number, then the first step is to see whether the column is fulfilled——use a function if to judge with this condition.

2.3 If the column chosen is not fulfilled, place a counter onto other counters in the column——use two for loop function to search through the whole column from bottom to the top until a blank space is found, and then replace it with a character X or O.

2.4 Use a function called **system(“cls”)** to clear up the screen. Then use another 3 functions of for loop to present every element and numbers of columns on the top of matrix, which helps players to see clearly where to put counters.

* 3. Declare a function namely **\_rightNumber.** This function has two variables of type int and namely **k** and **COLS** as input, and a variable of type int namely **k** as return. This function could judge if number entered is satisfactory.

3.1 To see if the number input as **k** is in interval [0, **COLS**], while k is not satisfactory, ask the player change the number and enter again.

* 4. Declare a function namely **\_counters**. This function has a variable of type int namely **Turn** as input, and return nothing. This function could display the times of inputting of players.

4.1 Declare three variables of type of int namely **i,m** and **k** separately, and **i** works in for function, besides, **m** and **n** represent the input times of both players.

* 5. Declare a function namely **JUDGE.** This function has a second level pointer namely Grid of type char and 3 variables of type int namely **ROWS, COLS** and **CON** as input and a variable of type int namely found as return. This function could judge if anyone wins.

5.1 declare four variables in the type of int namely found, **i, j** and **c.** And found has two values as 0 and 1, and 1 means someone connects counters in a line of number CON and win the game, while 0 means there is still no winner. Besides, **i** and **j** representing rows and columns separately, and c means the place of a counter in four counters.

5.2 The are four **for** function with double loops. The first one judges in rows if identical counters are in a horizontal line in number of **CON**. I applied an algorithm that only number of (**COLS-CON**) counters need to be considered in every row; the second one judges in columns if identical counters are in a vertical line in number of CON. And only number of (**ROWS-CON**) counters need to be considered in every column; the third one judges in slash direction line if identical counters of size (**ROWS-CON**) x (**COLS-CON**) counters need to be considered; the fourth one judges in backslash direction line if identical counters of size (**ROWS-CON**) x (**COLS-CON**) in the location of top right corner or lower right corner. Final for loop function is to see if the matrix is fulfilled and return value 2.

* 6. In main function. Declare variables namely **i, P2, k , ROWS, COLS** and **CON** and only **P2** is not declared before, which indicates a state whether player 2 is controlled by human (1) or computer (0).

6.1 Then rules need to be explained clearly. Then give players choices between a classic modes (6x7) and change the size of matrix.

6.2 While chosen to change the size, a **scanf** function is used to store three variables, let k equal to the function **scanf**, and only when the k equal to 3 (three variables are entered correctly) then could go to next step, otherwise the system ask players to change the inputting way.

6.3 Creates a block of memory space with a size of **ROWS** **\*** the byte size of a pointer to a char pointer and assigning it to the second level pointer **Grid**. Then creates a block of memory space with a size of **COLS** **\*** the byte size of a pointer to a char variable and assigning it to the second level pointer Grid[i]. And free these memories at the end of program, using free function.

6.4 Using the variable Turn to remember the times players play the chess, and simply mathematic method could be used to find separate times of each one.

6.5 While playing with computer, create a seed according to time function and apply the rand function to get a **k** which is between 1 and **COLS**.

6.6 Before set counters into the matrix, the number k of column need to be judged whether it’s satisfactory. Use the **\_playing** function to see if the preferred column is fulfilled and then display the matrix.

6.7 Show players their inputting times as a reminder.

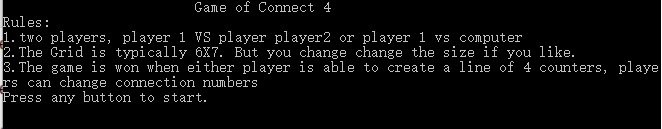
6.8 Use the **JUDGE** function to judge if who wins. If nobody wins and then back to the step where players input numbers, else if someone wins and then declare the winning and end the whole loop. Ask if players want to restart the game again. Press ‘N’ to start again or press any other buttons to end the game.

* 7. Free the memory created before and restart or end the game.

1. **Implementation**: see the C code in file 1508686\_4.c with comments.
2. **Testing**

The C program was tested by carrying out a set of experiments; and the C program output was verified successfully. For instance,

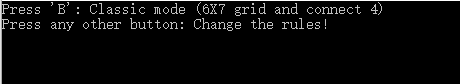
* 1. program interface



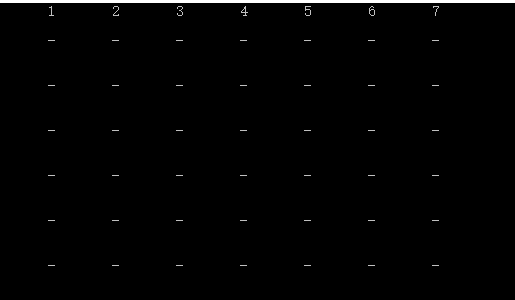
* 1. choosing the mode

C:\Users\ADMINI~1.QQO\AppData\Local\Temp\5A04.tmp.png

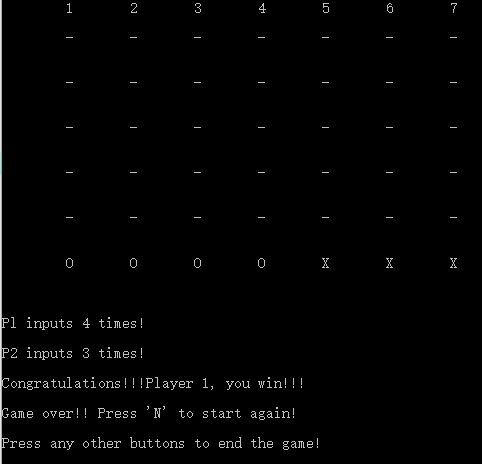
* 1. ‘Player1 VS Player2’ mode
     1. then choose matrix size



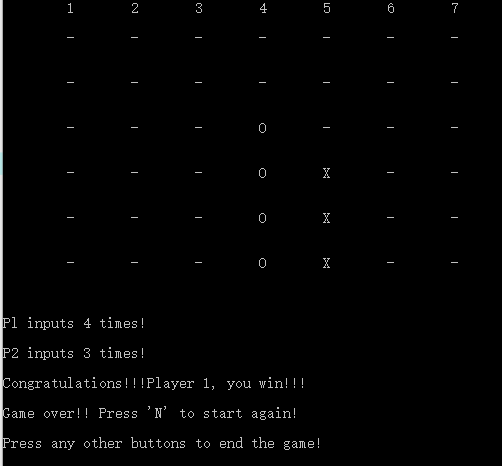
* + - 1. Classic mode
         1. the matrix



* + - * 1. win in horizontal direction



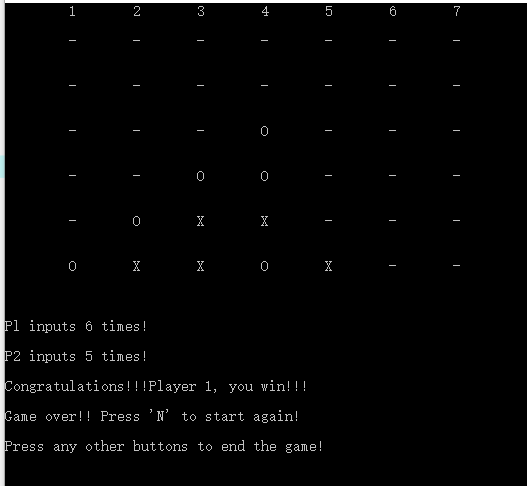
* + - * 1. win in vertical direction



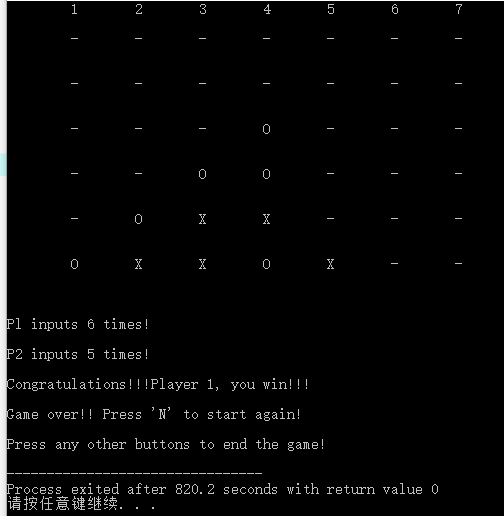
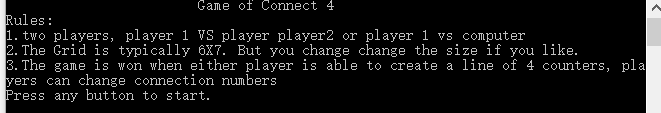
* + - * 1. win in slash direction (‘\’)



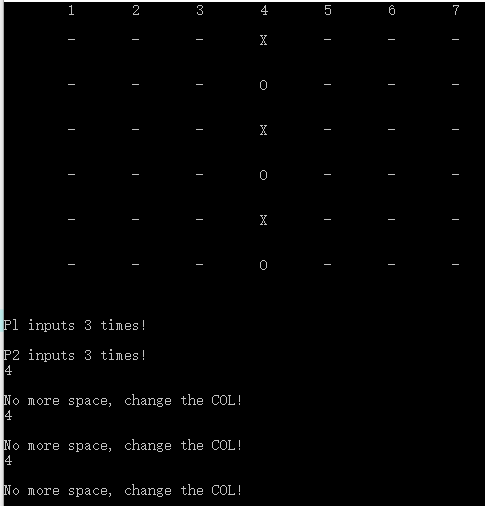
* + - * 1. win in backslash direction (‘/’)



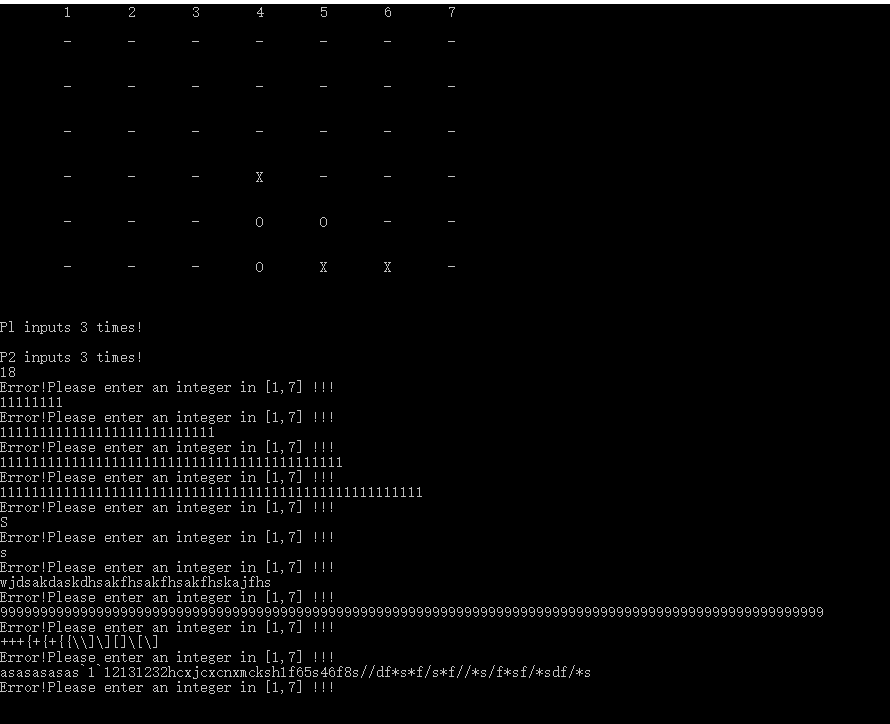
* + - * 1. press a character except ‘N’ to end the game or press ‘N’ to start again

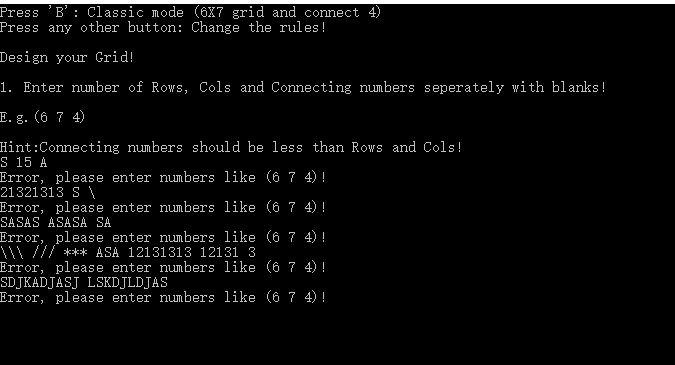
* + - * 1. when a column is fulfilled and player want to fill counter in it



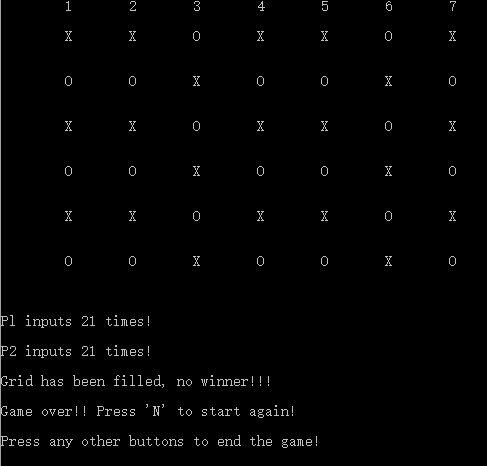
* + - * 1. when someone enter a number which is not in interval [1, COLS]



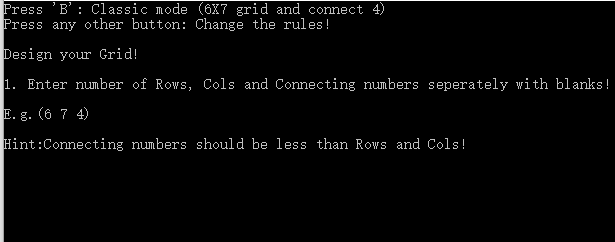
* + - * 1. someone enters unsatisfactory number when designing the rules



* + - * 1. when the grid is fulfilled

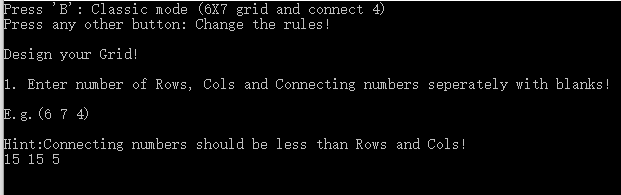


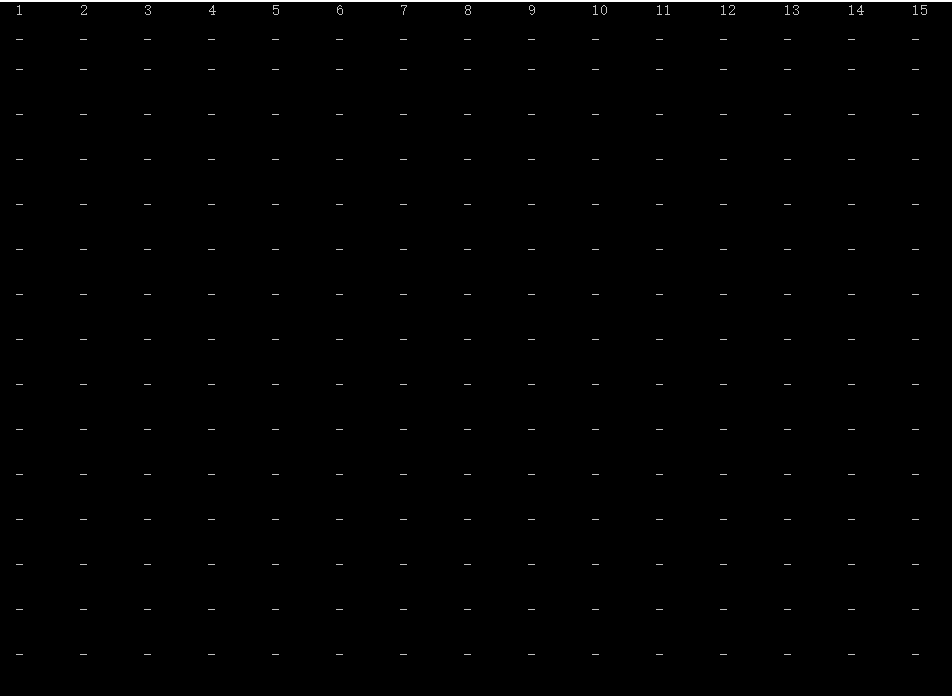
* + - 1. change the rules



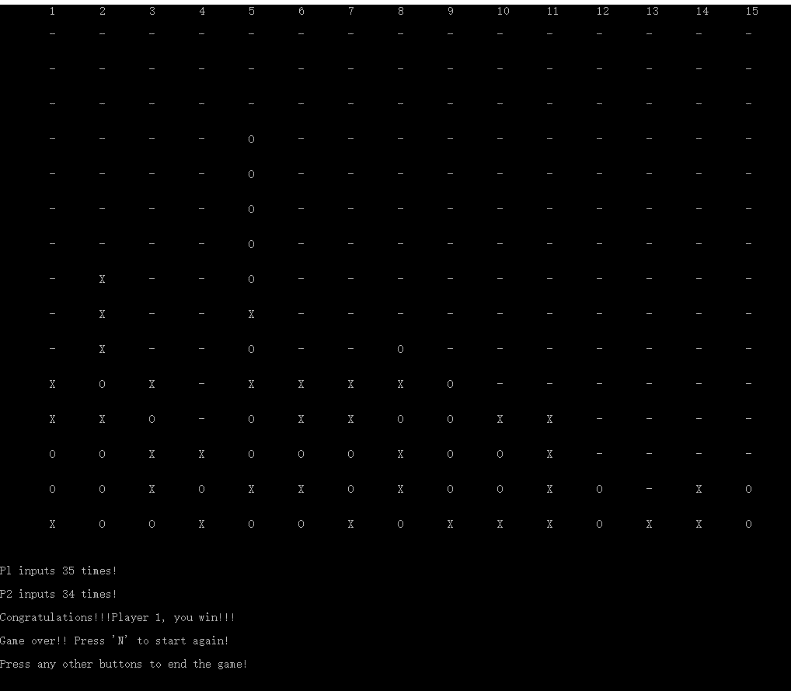
Hint is just a suggestion, but players can do in this way.

* + - * 1. Choose a (15x15) matrix with 5 winning length

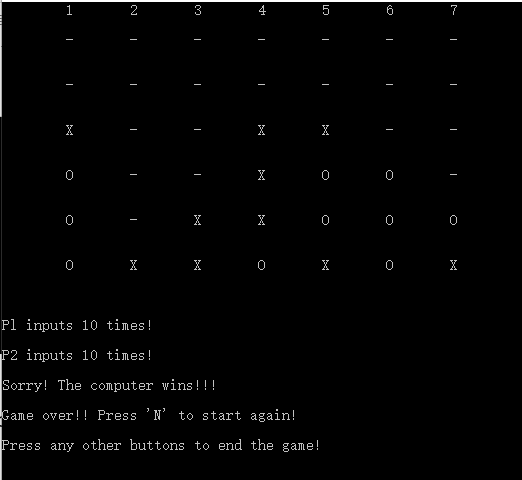




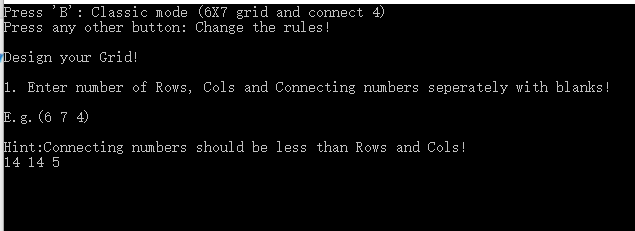
* + - * 1. Some tests



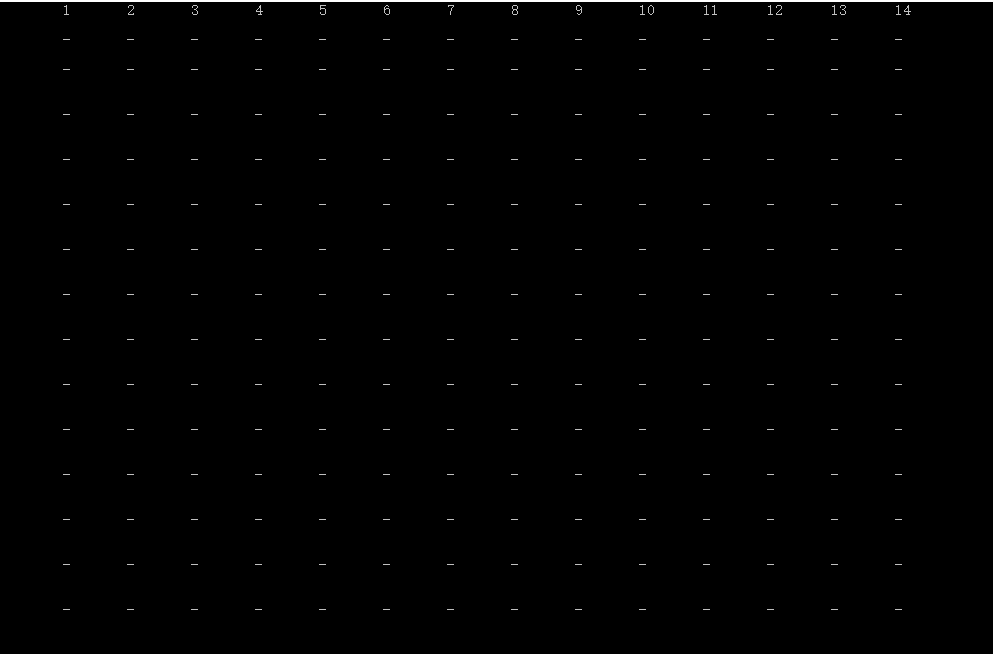
* 1. Choose the mode (Player VS computer) in classic mode
     1. Some tests



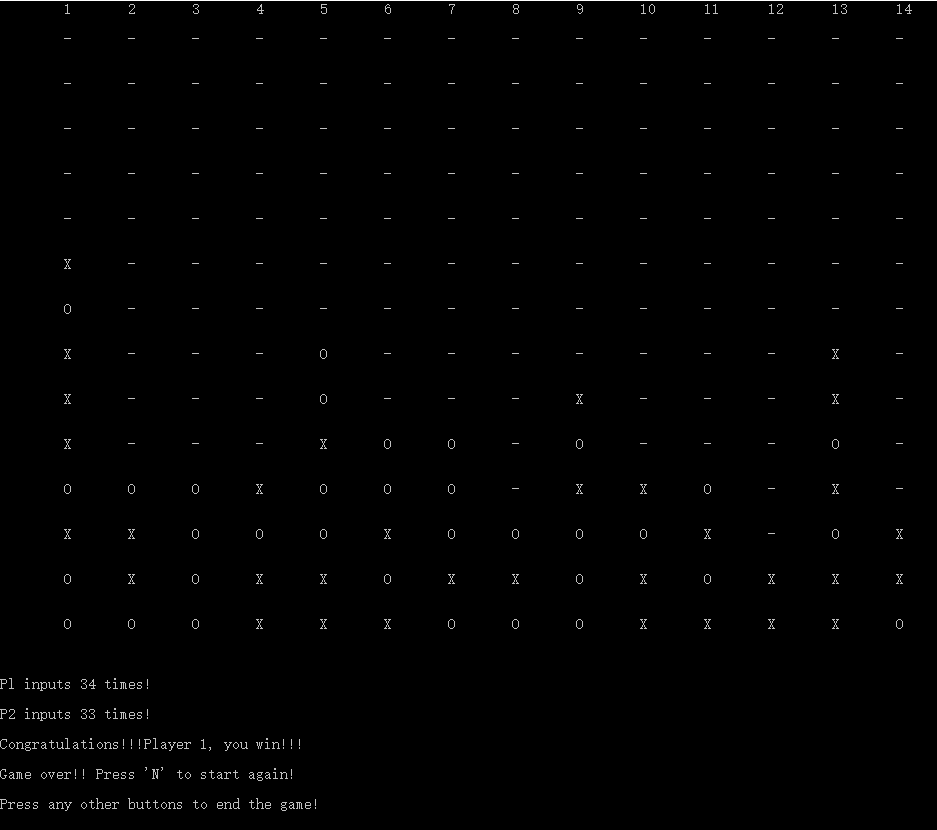
* 1. Choose the mode (Player VS computer) in self-designing mode
     1. Choose a (15x15) of size and winning length is 5



* + 1. The matrix



* + 1. Some tests



* 1. Problems I encountered when coding

In my view, the biggest problem is to judge when someone meets the winning condition as connecting 4 counters in a line. As four conditions need to be considered in four directions like horizontal, vertical, slash and backslash directions. Then I drew the matrix on a paper and tried to find something useful, then I found that only a part of the matrix points need to be considered as beginning points. For instance, when judging horizontal direction, 3 points in the last don’t need to be considered as beginning points. Then when I made a mistake when coding, as I use condition like (‘A’==’B’==’C’==’D’), which I firstly thought was right but stopped program running well, I spent a lot of time finding this bug and finally figured out why it’s wrong. As for computer, it judge the condition from left to right direction and so it firstly judge (‘A’==’B’) and returns ‘0’ or ‘1’, which caused the bug.

Some details are important to strengthen the robustness. For example, warning when players enter some unsatisfactory characters which may cause the program breaking down. Someone want to place a counter when a line is fulfilled, ask them to change a column, and the situation when the matrix is fulfilled. To sum up, many details need to be considered and also took many times to make program more solid.

Finally, the program works well in my view, but the matrix could be more beautiful if added a frame using ‘—’ and ‘|’, which makes matrix clean and comfortable. Due to the limitation of time, there indeed exist some insufficient aspects.