


Report of the program of Apple Chess

Program title: Apple Chesss

Execution file name: apple.exe

Source file name: apple.pas

Time duration: About 9 hours


Objective: This project is to write a program for playing apple chess. As my teacher hasn't specified whether we should write human vs. human or computer vs. human, I chose to write a program for human vs. human. Only learning Turbo Pascal for Windows as the programming language, so I use this language to write this program. 

Analysis: I first broke the project into several parts:

1. To display the interface for the user

- i. The maximum size of the chess board is 10x10
- ii. I have to choose whether to use "半形" or "全形" symbols for displaying the chess board
- iii. It is much easier to handle the display "半形" symbols than those "全形" ones.

2. To get the input by the user

- i. Using co-ordinate systems of the chessboard: 
 1. Easier for me to handle the input
 2. Easier to code
- ii. Allowing the user to input by arrow keys:
 1. Much more user-friendly
 2. Harder to code

3. Using a function or procedure to check the chess whether is placed in a valid position

- iii. I have to check the chess in 8 directions namely N, S, E, W, NE, NW, SE and SW.
- iv. Finding the checking procedures are in similar pattern, so maybe they can be written into parametric procedure or function forms.

3. Think of the condition for the player to win, lose or draw.

Count the number of black and white chesses, and the spaces left.

Design:

1. To display the interface for the user

I finally used "全形" symbols for displaying the interface, because it is much desirable.

I used "for loop" to display the board, beginning from 0 to 10, where column 0 and row 0 are the co-ordinates, indicating the x-axis and y-axis. The other elements are in an array of 10x10, storing the condition of each spaces. The initial state of each element in the array is a blank (" "). Say if the board is with a size of n (where n is an odd number), the element of $[n \text{ div } 2, n \text{ div } 2]$, $[n \text{ div } 2 + 1, n \text{ div } 2 + 1]$ should be assigned to one of the colour while the element of $[n \text{ div } 2, n \text{ div } 2 + 1]$ and $[n \text{ div } 2 + 1, \text{mid}]$ should be assigned to another color.

2. To get the input by the user



I finally chose to use the inputting of co-ordinate system.

After getting the input, I have to check whether the input is valid or not.

- a. To check whether the entered co-ordinates are valid or not

- b. To check whether the space is occupied by another chess
- c. To check whether the space is valid or not, ie. after the chess is placed in the space, can it "eat" the other colour?

I broke down this case into smaller parts :

- (1) check whether the chess is surrounded by another color in any of the 8 directions: N, E, S, W, NE, NW, SE and SW.
- (2) if it is surrounded by another color in any of the 8 directions, then check whether it can "eat" in the 8 directions
- (3) store which direction it can eat

I think that Step (2) is the most difficult part in this project. Though the co-ordinates of N, E, S, W can be dealt in a similar way, I found it is hard for me to implement them into one parametric function. So I separate them into parts.

Say, the {Check N} part is to check the spaces from the top to the bottom to see whether there is a chess of the same colour in the same x-coordinate. Then if there are one or more than one chess of another colour exist in between the 2 chess, then this direction is valid for "eat". The S, E and W checking are in similar approach.

The NE, NW, SE and SW are in similar implementation, but the expression of coordinates are different.

Say, the NW direction:

```

{check NW}
if [x<>0] and [y<>0] then begin
  if x>y then z:=x-y else z:=0;
  for a:=z to x-1 do if board[a, a-[x-y]]=chess[turn] then alt[4]:=a;
  if [alt[4]<>-1] and [alt[4]<>x-1] then begin
    dir[4]:=true;
    for a:=alt[4]+1 to x-1 do if board[a, a-[x-y]]<>chess[abs(turn-1)] then dir[4]:=false;
  end;
end;
end;

```

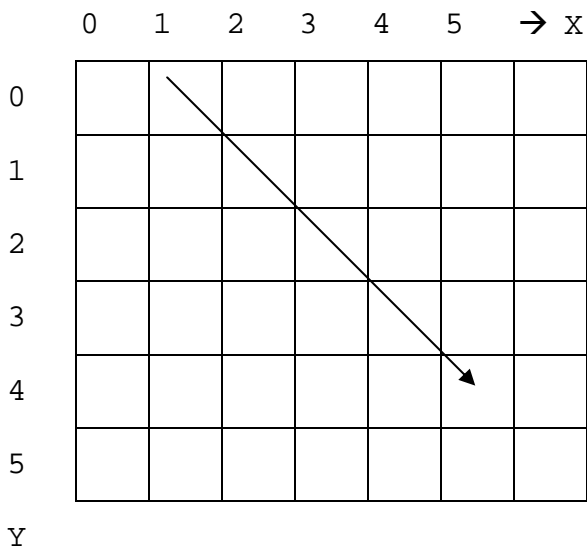
suppose it is now the turn of ,

if I want to place at (5,4), notified by , then in the program segment above, x = 5 and y = 4.

As it is now checking in a diagonal direction, I have to calculate the upper left co-ordinate ie. (1,0). The x-coordinate, z is determined by the 2nd statement: if x>y then z:=x-y else z:=0; and the y-coordinate is determined by z-(x-y).

Then I use for loop to implement the checking.

Again, if there are 1 or more than 1 chess of another colour exist in between the 2 chess, then this direction is valid for "eat".



3. Think of the condition for the player to win, lose or draw.

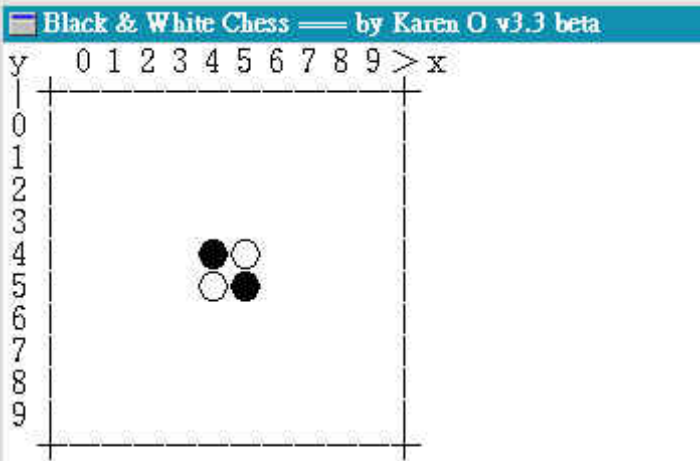
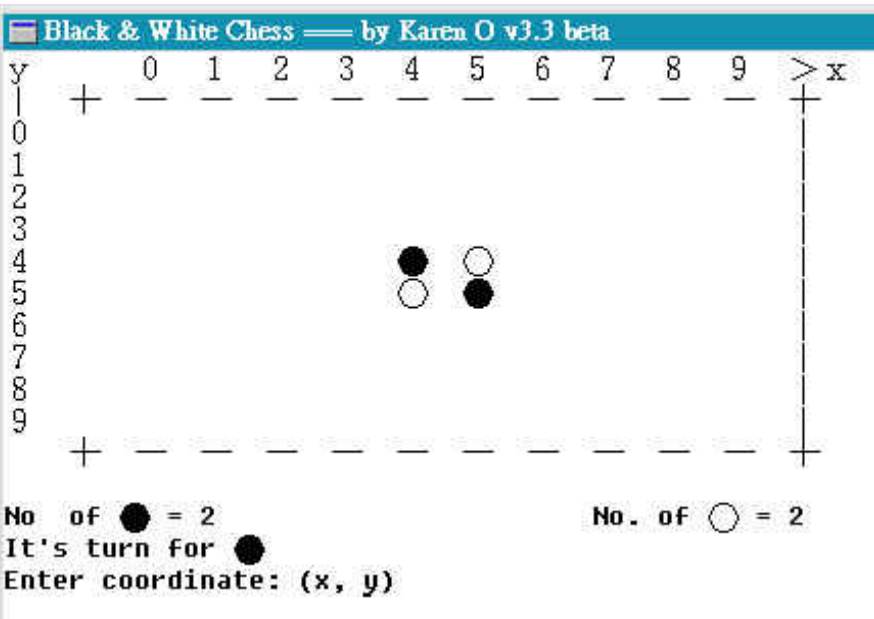
- i. If the spaces are full, then if the number of black chesses is more than that of white chesses, black wins and vice versa. If the number of chesses of both colors are the same, then they

are draw.

- ii. If the spaces are not full, but one of the color has no more chesses, then the other color wins.



Thinking flow chart:

1 st hour	<p>Formatting the interface</p>  <p>● begin first... Press ENTER to continue...</p> <p>I found there is a problem of displaying the “全形” characters, but I cannot solve it. Where the program rewrite the board, or when minimize or maximize the browser, there appears some spaces between the characters.</p>  <p>No. of ● = 2 It's turn for ● Enter coordinate: (x, y)</p> <p>No. of ○ = 2</p>
----------------------	---

2nd hour

Programming

3rd - 6th hr

Thinking of the function checking:

```

Black & White Chess — by Karen O v3.3 b
y  0 1 2 3 4 5 6 7 8 9 > x
|-----|
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|-----|
No. of ● = 7   No. of ○ = 2
It's turn for ○
Enter coordinate: (x, y) 5 2
Enter coordinate: (x, y) 5 3
    
```

- How to check whether the position is valid in the 8 directions
- N, S, E, W are similar
- NW, SE, NE and SW are a bit tricky.
- Turn over the chess when doing checking.
- Try to using one parametric function to check the direction N, S, E, and W.
- Haven't checked for the case of no moves. And some errors occur in the checking function.

```

Black & White Chess =
y  0 1 2 3 > x
|-----|
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|-----|
BLACK : 6
WHITE : 9
It's turn for ○
Enter coordinate:
    
```

E.g.: In fact it should be no more moves for the black chess, but unfortunately, I can place it on (3,1).

```

Black & White Chess =
y  0 1 2 3 > x
|-----|
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|-----|
BLACK : 5
WHITE : 9
It's turn for ●
Enter coordinate:
    
```

7th hour

- Finding the previous method is inappropriate
- Modify the previous method.
- Checking first, and store which direction and which position is valid.
- Turn over the chess after checking.

8th hour

```

Black & White Chess — by Karen O v3.3 beta
y  +  0  1  2  3  4  5  6  7  8  9  > x
|  |  |  |  |  |  |  |  |  |  |  |
0  |  |  |  |  |  |  |  |  |  |  |  |
1  |  |  |  |  |  |  |  |  |  |  |  |
2  |  |  |  |  |  |  |  |  |  |  |  |
3  |  |  |  |  |  |  |  |  |  |  |  |
4  |  |  |  |  |  |  |  |  |  |  |  |
5  |  |  |  |  |  |  |  |  |  |  |  |
6  |  |  |  |  |  |  |  |  |  |  |  |
7  |  |  |  |  |  |  |  |  |  |  |  |
8  |  |  |  |  |  |  |  |  |  |  |  |
9  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
+  +  +  +  +  +  +  +  +  +  +  +

```

● has no moves...press ENTER to continue...

- Debug

9th hour

- An error in checking is found.

- Debug.

```

(Inactive Black & White Chess — by Karen O v3.3 beta)
y  0  1  2  3  4  5  6  7  8  9  > x
|  |  |  |  |  |  |  |  |  |  |  |
0  |  |  |  |  |  |  |  |  |  |  |  |
1  |  |  |  |  |  |  |  |  |  |  |  |
2  |  |  |  |  |  |  |  |  |  |  |  |
3  |  |  |  |  |  |  |  |  |  |  |  |
4  |  |  |  |  |  |  |  |  |  |  |  |
5  |  |  |  |  |  |  |  |  |  |  |  |
6  |  |  |  |  |  |  |  |  |  |  |  |
7  |  |  |  |  |  |  |  |  |  |  |  |
8  |  |  |  |  |  |  |  |  |  |  |  |
9  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
BLACK : WHITE = 9 : 91
WHITE WINS!!!

```



Conclusion & Discussion: Testing can be hardly done as it sometimes takes a long time to in the case like "no more moves". And the interface is not good enough. Using arrow keys to control the cursor is a better interface but I failed to do it. And it is sometimes confusing to place the chess at the right space. And because I was not able to write those similar steps into some parametric functions / procedures, so there are lots of variables, in result of a confusion.

