

## 1. Specification

Processor: QuadCore Intel Core i5-4570R, 2.70 GHz

Memory: 8 GB of 1333MHz DDR3 memory

OS: Windows 10 教育版 64 bits

Graphics: Intel Iris Pro Graphics 5200

IDE: Code Blocks 16.01

Compiler: C++11 5.3.0 - GNU C++ Compiler with options: -lm -lcrypt  
-O2 -std=c++11 -pipe -DONLINE\_JUDGE

Phone:0911177474

Email:wuhhoward75621@kimo.com

Name:張哲魁

## 2. How to run the project

Due to the implement consisting of API SFML and I built it on the IDE

Code::Blocks, please follow the series of step to set up the proper environment.

### I. Download the IDE Code::Blocks.

<http://www.codeblocks.org/downloads/26>

Code::Blocks - The IDE with all the features you need, having a consistent look, feel and operation across platforms.

Home Features Downloads Forums Wiki

**Main**

- Home
- Features
- Screenshots
- Downloads
  - Binaries
  - Source
- SVN
- Plugins
- User manual
- Licensing
- Donations

Please select a setup package depending on your platform:

- Windows XP / Vista / 7 / 8.x / 10
- Linux 32 bit
- Linux 64 bit
- Mac OS X

NOTE: For older OS'es use older releases. There are releases for many OS version and platforms on the [Sourceforge.net](http://Sourceforge.net) page

NOTE: There are also more recent nightly builds available in the [forums](http://forums) or (for Debian and Fedora users) in [Jesse's Debian repository](http://Jesse's Debian repository) and [Jesse's Fedora repository](http://Jesse's Fedora repository). Please note that we consider nightly builds to be stable, usually.

NOTE: We have a [Changelog](http://Changelog) for 16.01, that gives you an overview over the enhancements and fixes we have put in the new release.

**Windows XP / Vista / 7 / 8.x / 10:**

File	Date	Download from
codeblocks-16.01-setup.exe	28 Jan 2016	Sourceforge.net or FossilHub
codeblocks-16.01-setup-nonadmin.exe	28 Jan 2016	Sourceforge.net or FossilHub
codeblocks-16.01-nosetup.zip	28 Jan 2016	Sourceforge.net or FossilHub
codeblocks-16.01mingw-setup.exe	28 Jan 2016	Sourceforge.net or FossilHub
codeblocks-16.01mingw-nosetup.zip	28 Jan 2016	Sourceforge.net or FossilHub
codeblocks-16.01mingw_fortran-setup.exe	28 Jan 2016	Sourceforge.net or FossilHub

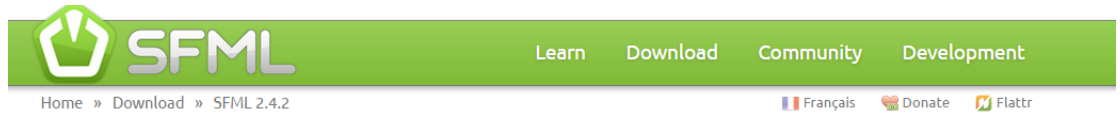
NOTE: The codeblocks-16.01-setup.exe file includes Code::Blocks with all plugins. The codeblocks-16.01-setup-nonadmin.exe file is provided for convenience to users that do not have administrator rights on their machine(s).

NOTE: The codeblocks-16.01mingw-setup.exe file includes additionally the GCC/G++ compiler and GDB debugger from TDM-GCC (version 4.9.2.32 bit, 5.1.1). The codeblocks-16.01mingw\_fortran-setup.exe file includes additionally to that the GFortran compiler (TDM-GCC).

NOTE: The codeblocks-16.01mingw-nosetup.zip files are provided for convenience to users that are allergic against installers. However, it will not allow to select plugins / features to install (it includes everything) and not create any menu shortcuts. For the "installation" you are on your own.

(Be aware of you exactly downloading the fourth row. It is critical for SFML)

- II. Download SFML 2.4.2(I have put it in the rar)  
<https://www.sfml-dev.org/download/sfml/2.4.2/>

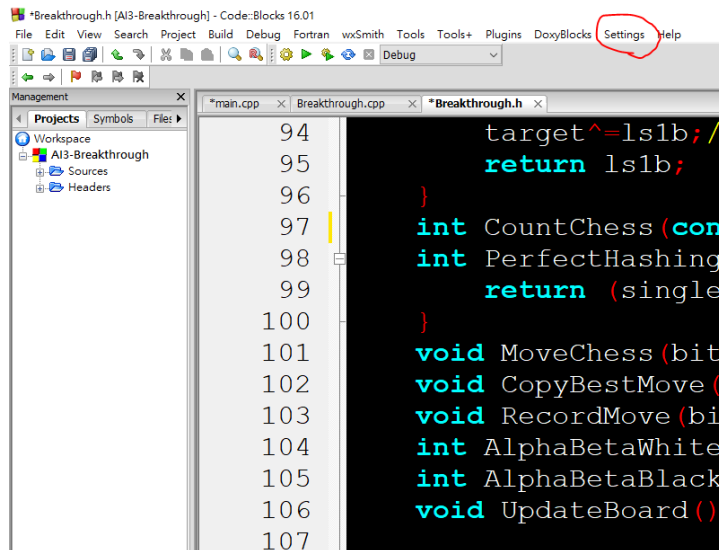


## Download SFML 2.4.2

Windows	Visual C++ 11 (2012) - 32-bit	<a href="#">Download   16.7 MB</a>	Visual C++ 11 (2012) - 64-bit	<a href="#">Download   18.5 MB</a>
	Visual C++ 12 (2013) - 32-bit	<a href="#">Download   16.1 MB</a>	Visual C++ 12 (2013) - 64-bit	<a href="#">Download   17.8 MB</a>
	Visual C++ 14 (2015) - 32-bit	<a href="#">Download   16.0 MB</a>	Visual C++ 14 (2015) - 64-bit	<a href="#">Download   17.6 MB</a>
	GCC 4.9.2 TDM (SJLJ) - 32-bit	<a href="#">Download   13.8 MB</a>	GCC 4.9.2 TDM (SJLJ) - 64-bit	<a href="#">Download   15.8 MB</a>
	GCC 6.1.0 MinGW (DW2) - 32-bit	<a href="#">Download   15.3 MB</a>	GCC 6.1.0 MinGW (SEH) - 64-bit	<a href="#">Download   16.2 MB</a>
<p>On Windows, choosing 32 or 64-bit libraries should be based on which platform you want to compile for, not which OS you have. Indeed, you can perfectly compile and run a 32-bit program on a 64-bit Windows. So you'll most likely want to target 32-bit platforms, to have the largest possible audience. Choose 64-bit packages only if you have good reasons.</p> <p><b>The compiler versions have to match 100%!</b> Here are links to the specific MinGW compiler versions used to build the provided packages: TDM 4.9.2 (32-bit), TDM 4.9.2 (64-bit), MinGW Builds 6.1.0 (32-bit), MinGW Builds 6.1.0 (64-bit)</p>				
Linux	GCC - 64-bit	<a href="#">Download   11.7 MB</a>		
	<p>On Linux, if you have a 64-bit OS then you have the 64-bit toolchain installed by default. Compiling for 32-bit is possible, but you have to install specific packages and/or use specific compiler options to do so. So downloading the 64-bit libraries is the easiest solution if you're on a 64-bit Linux.</p> <p>If you require a 32-bit build of SFML you'll have to <b>build it yourself</b>.</p>			

(This selection matches the corresponding IDE you had just downloaded.)

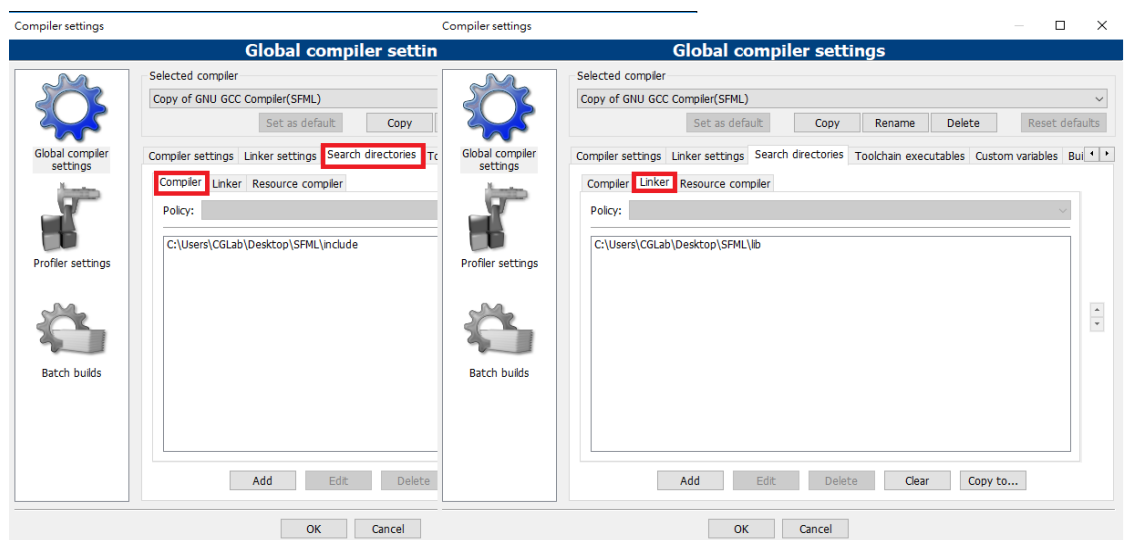
- III. Set up the IDE environment  
(More detailed information in the reference of point 5)  
A. Choose Settings→Compiler...



B. Choose “Search directories”.

Add the path of your downloading SFML\include to the “Compiler”.

Add the path of your downloading SFML\lib to the “Linker”.

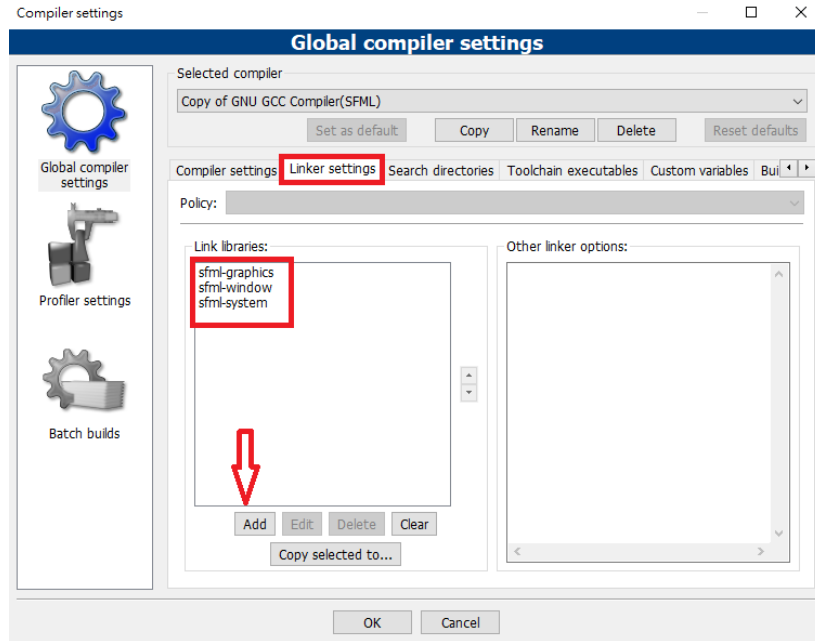


C. Choose “Linker settings” and add three link libraries.

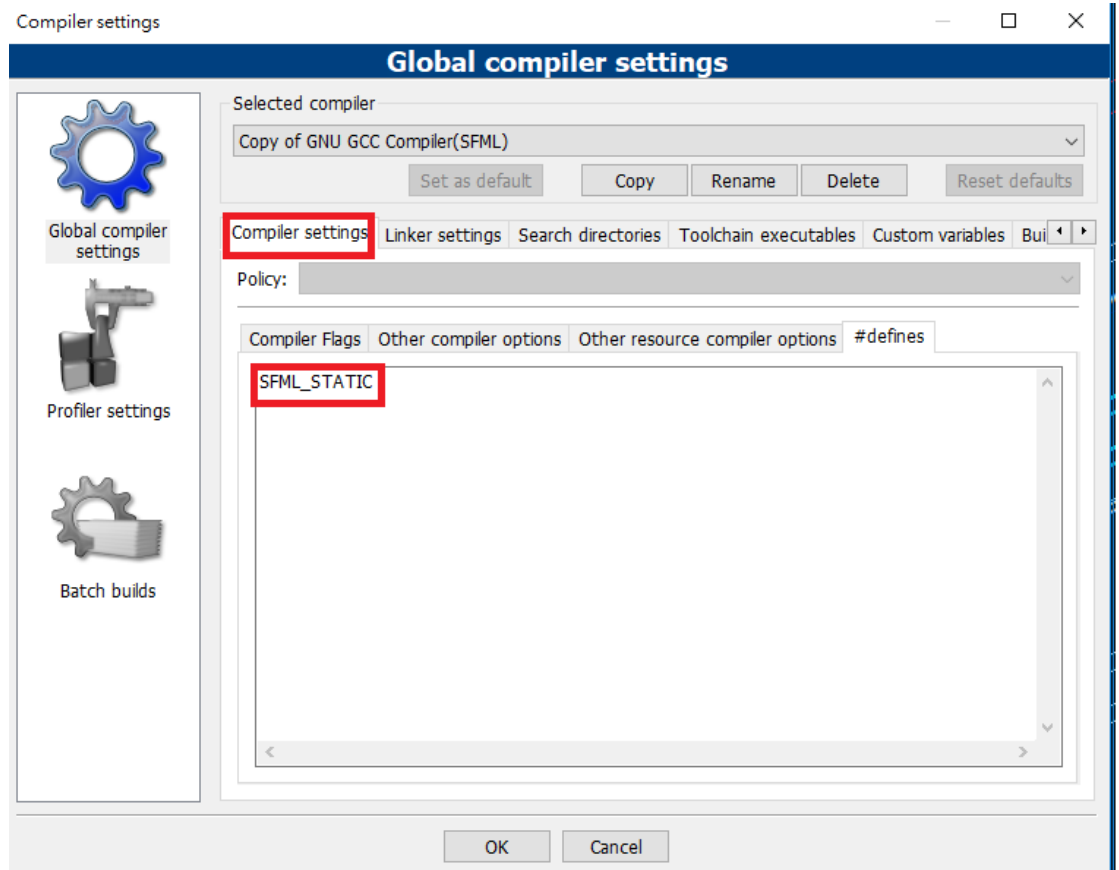
sfml-graphics

sfml-window

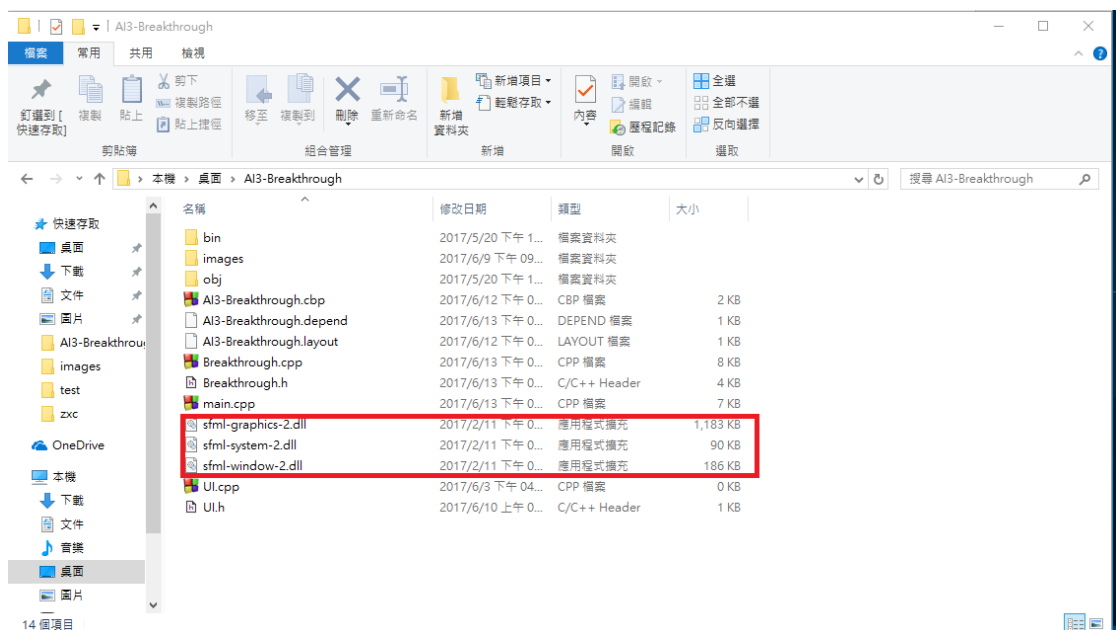
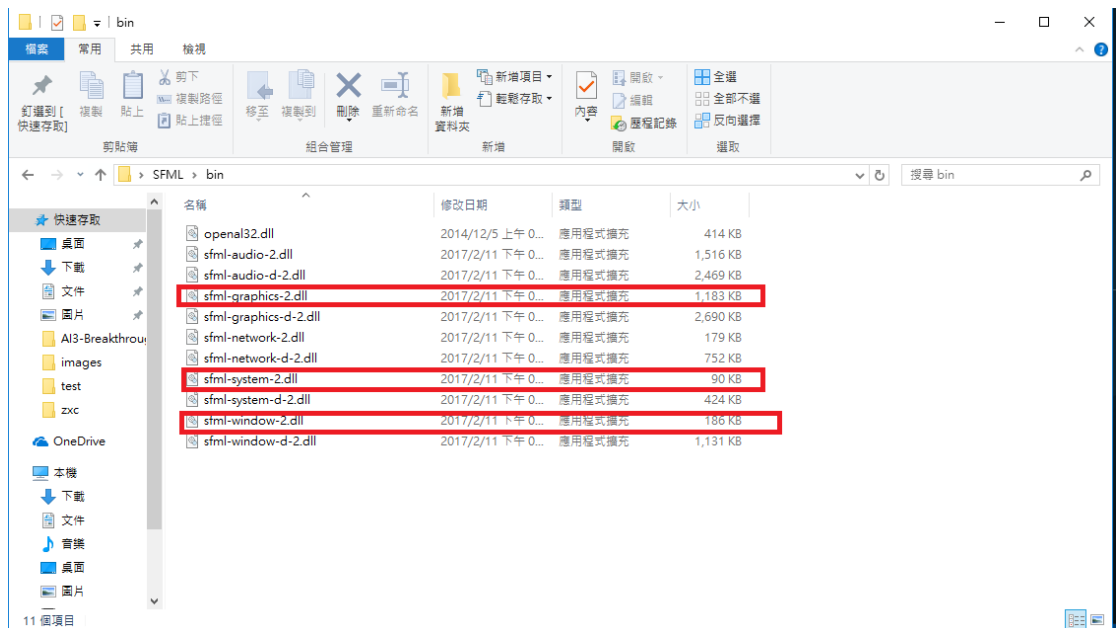
sfml-system



D. Final, choose "Compiler settings" and add "SFML\_STATIC".



- IV. Go to SFML\bin and copy three .dll files consisting of sfml-graphics-2.dll, sfml-window-2.dll and sfml-system-2.dll into the project file.(I have put it in the project)

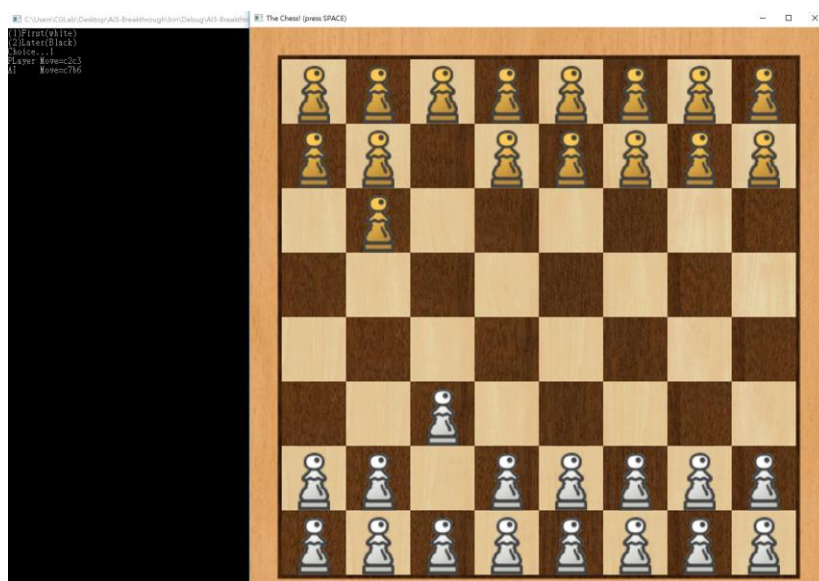


Then, all should be working well.

- V. Run the project.  
It will ask you the side. White go first then black.



When it is your turn, drag your chess to the proper place. The MS-DOS window shows every moves of AI and the player in the global form(a1~h8).  
 ✖Because of picture, I use orange chess stands for black and grey chess stands for white.



Because the program didn't detect the illegal move of players, you can cheat by move your chess to the final. You should not move the AI's chess,

it will lead to error. Be a good player.

If the game is over, the UI will halt for 5 second then exit.

### 3. Data Structures, Skills, Testing and Features

#### I. Data structures

##### A. Bitboard

I used type of unsigned long long int in total 64 bits to show the bitboard. Only one for black chess and only one for white chess because there is one kind of chess.



White: 0xFFFF000000000000

Black: 0x000000000000FFFF



White: 0x6F80002401000000

Black: 0x000042000061096E

#### II. Skills

##### A. Alpha-beta pruning with negamax

Based on the requirement of this assignment. The below is the pseudocode.

```
Function AlphaBeta(depth,alpha,beta){
  If(win)return 999999
  If(loss)return -999999
  If(depth<=0)return 0;

  While(nextMove=GenerateNextMove() )
    moveScore=evaluate(nextMove)//審局函數
    If(nextMove eat a chess)
      value=eat point+ depth*5+moveScore+opponent's threaten
      UpdateBitboard
    If(depth=1)value-= AlphaBeta(1,-beta+value,-alpha+value)
    else      value-= AlphaBeta(depth-1,-beta+value,-alpha+value)
    else //just walk , no eating
```

```

    UpdateBitboard
    value=moveScore- AlphaBeta(depth-1,-beta+moveScore,-
alpha+moveScore)

    if(value>alpha)
        alpha=value
        recordMove//the current best move
    if(value>=beta)
        return value;//cut off

return alpha;

```

#### B. Quiescence search

In the pseudocode, there is a line  
 “If(depth=1)value-= AlphaBeta(1,-beta+value,-alpha+value)”.  
 If the chess can eat the opponent’s chess and the depth is 1, we should  
 do one more depth to see whether the opponent could beat us more  
 severely by eating our chess.

#### C. Evaluation function

Every position stands for the specific point can be get if a chess at  
 this position. Make the AI occupy the more favorable place. The points  
 is organization by observing the top players’ movements. The website is  
 listed in the reference of point 3.

\* The followings are the more detailed skills operating the bitboard.

#### D. Generate the next whole moves

A chess has three directions can move. For white chess, it can go  
 left-up, up and right-up and the black chess goes the opposite  
 directions. There is a way can make all chess move to a direction in one  
 operation.

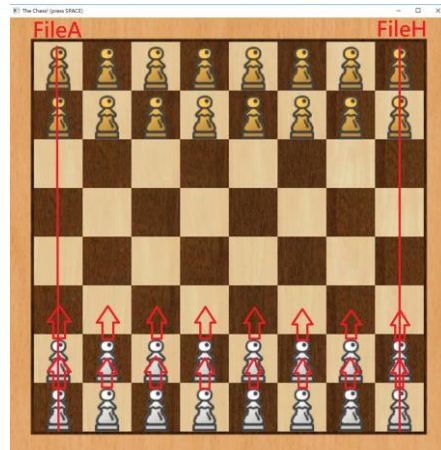
Take white chess for example.



Goal1: White goes up.

white:0xFFFF000000000000

black: 0x000000000000FFFF



The places are not be occupy can be first calculate as a bitboard.

It can filter out the illegal movements.

→Empty=0xFFFFFFFFFFFFFFFF&~black&~white

Goal=(white>>8)&Empty

Goal2: White goes up-left.

white:0xFFFF000000000000

black: 0x000000000000FFFF

Goal=(white&~FileA)>>7&(0xFFFFFFFFFFFFFFFF &~white)

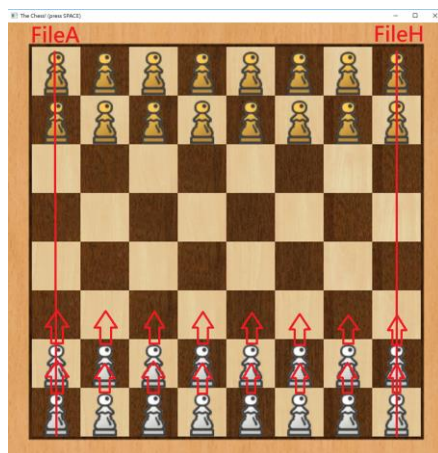
White chess cannot move to left-up from FileA. The chess will go wrongly. And it can eat black chess so delete the "&~black" limit.

White going right-up and the movements of black are quite same with above method. More detail in the project breakthrough.h file.

#### E. Extract a single move:LS1B

The above describe how to make whole movements in this step.

But we should apply one movements one by one in the alpha-beta pruning algorithm.



The movement of white going up can be expressed by a bitboard.

→0xFF00FF0000000000

The LS1B function can extract the least bit in the bitboard.

So, we can get next movement=0x0000010000000000.

(LS1B function codes also in the breakthrough.h file, implemented by bit operations.)

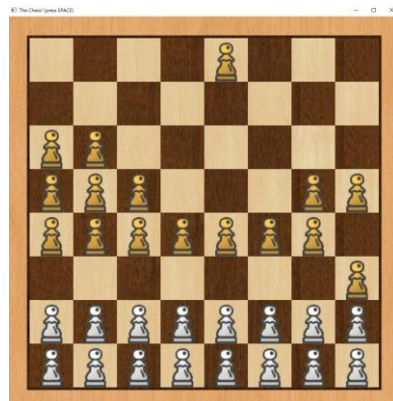
#### F. Perfect hashing

When a chess moves to a new place, we need to get the evaluation points. Now, we only know the movement bitboard like 0x0000010000000000. We can divide it by 2 to get the index of the chess position, but division is very slow with comparison to bit operation. Perfect hashing is an efficient way to get the index of the chess by bit operation and multiplication.

### III. Testing

#### A. Don't rush to the end.

What will the board be if I don't move and AI keep moving?



(AI:black)

The AI will not let the chess rush to the white side even the white side doesn't make any movements. The AI takes the most careful movements to make sure the victory even competing with an un-smart player.

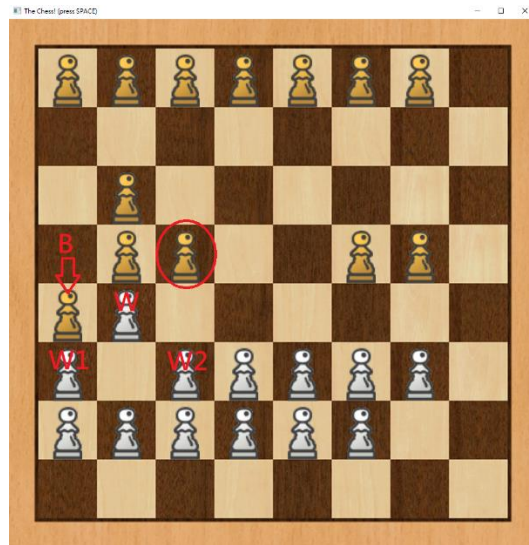
#### B. Different movements

Some classmates told that their AI had a strange temper choosing the same movement fight with different players. The mistake might occur in the evaluation function and eating points. My AI will take proper movements according to the opponents' movements.

#### C. Trade off/ Eager to win fast

The AI won't eat opponent chess in every times(the circled blacked chess doesn't eat the left-down white chess). If the AI eat my white chess, it knows that the white chess w1 or w2 will eat back,

even get the better place to the end. So, the AI move the black chess B down as a better decision.



If there are no other concerns, the UI will eat for winning soon. It credits to the proper set of value of the eating points and evaluation function and other factors.

#### IV. Features

##### A. Efficiency

The who project I never use the division and multi-dimension array except UI(it cost less than 0.1% running time).

##### B. Quite smart

AI often wins.

##### C. Delegate UI

The UI is implemented by SFML, fast and simple. This is my first time using SFML and is able to realize what is it doing in short time.

The graphic UI provides user a convenient way to interact with AI.

#### 4. Reference

##### 1.從暗棋參考盤面設計、資料結構

[http://dtim.mis.hfu.edu.tw/before/2008DTIM\\_PAPER/paper/B215.pdf](http://dtim.mis.hfu.edu.tw/before/2008DTIM_PAPER/paper/B215.pdf)

[http://www.csie.ntnu.edu.tw/~linss/Students\\_Thesis/2011\\_06\\_29\\_Lao\\_Yung\\_Hsiang.pdf](http://www.csie.ntnu.edu.tw/~linss/Students_Thesis/2011_06_29_Lao_Yung_Hsiang.pdf)

##### 2.Surakarta 棋實現技術之審局函式與 bitboard

[http://myweb.npu.edu.tw/~tcga2017/Paper\\_Submission.html](http://myweb.npu.edu.tw/~tcga2017/Paper_Submission.html)

##### 3. Evaluation setting by observing the top player's movements.

<https://www.littlegolem.net/jsp/main/>

##### 4.UI example(Using SFML)

[https://www.youtube.com/watch?v=4EuZl8Q8cs&list=PLB\\_ibvUSN7mzUffhi](https://www.youtube.com/watch?v=4EuZl8Q8cs&list=PLB_ibvUSN7mzUffhi)

[ay5g5GUHyJRO4DYr&index=14](https://www.sfml-dev.org/tutorials/2.0/start-cb.php)

## 5.SFML and Code::Blocks (MinGW) settings

<https://www.sfml-dev.org/tutorials/2.0/start-cb.php>

## 5. Difficulties

### I. Necessary for Graphic UI

In this project, it is necessary to use a graphic UI to show the board, or the user cannot realize what is the situation, especially after several movements.

It is the first time I make a UI. Fortunately, there is a good resource on the youtube that show all the detailed. This is an extra load I think I must do.

### II. Timer

In the beginning of the game, there are less legal moves so the AI running time is fast. In the middle of the game, more and more legal moves spending much more time. I should add a timer giving the AI a constant time to think, and the alpha-beta pruning should be iterative deepening. The latter is easy to achieve, but the timer need more time and exercise to make it. Next time, I will add it to the project.

### III. Thread

According to the paper on the TCGA, it suggested that the multithreads can greatly improve the speed. I also saw that implemented on a famous open source program called “stockfish”, a AI for chess not breakthrough. With the same reason, I don’t have enough time to learn about it. It is a pity.

### IV. Alpha-beta pruning is not good enough

Although we can try to optimize the data structure and algorithm to make the running time a great progress, it has the bottleneck after all. The neuron will be a better way to train the AI. Alpha-beta pruning try to give a best move in the certain depth, but the neuron network gives a most possibility to win the human.