- README.txt : contains the IDs of all team members
- REPORT.pdf : contains a report of the project. The first page should contain the IDs of all team members. The report should contain the following sections:

1. Introduction: A summary of the project
2. Project Architecture: A description of the different modules
3. Personal achievements: A sub-section per team member. Each one writes what he has implemented. and his approach.
4. Post-mortem
   What went well in the realization of the project
   What can be improved

1. INTRODUCTION

As a command line interface chess game, the main features are:
Robustness of input reception mechanism
Integration of all pieces' move logic
Cleanness of interface and interaction
Display of legal moves by adding asterisk
Support of special rule such as promotion and castling
Examination of check and checkmate before and after every move.
Record of game time via three different timers
Modularization of program's different parts

2. ARCHITECTURE

This project contains two files: Board.h and main.cpp.

The main() function in main.cpp instantiated an object of the Board class defined in Board.h, and provided an infinite loop for the game process.

Board.h contains four classes: Timer, Position, Pieces and Board.

Timer objects are able to store time data and carry out functions which allow us to start or pause a timer and get the time elapsed.

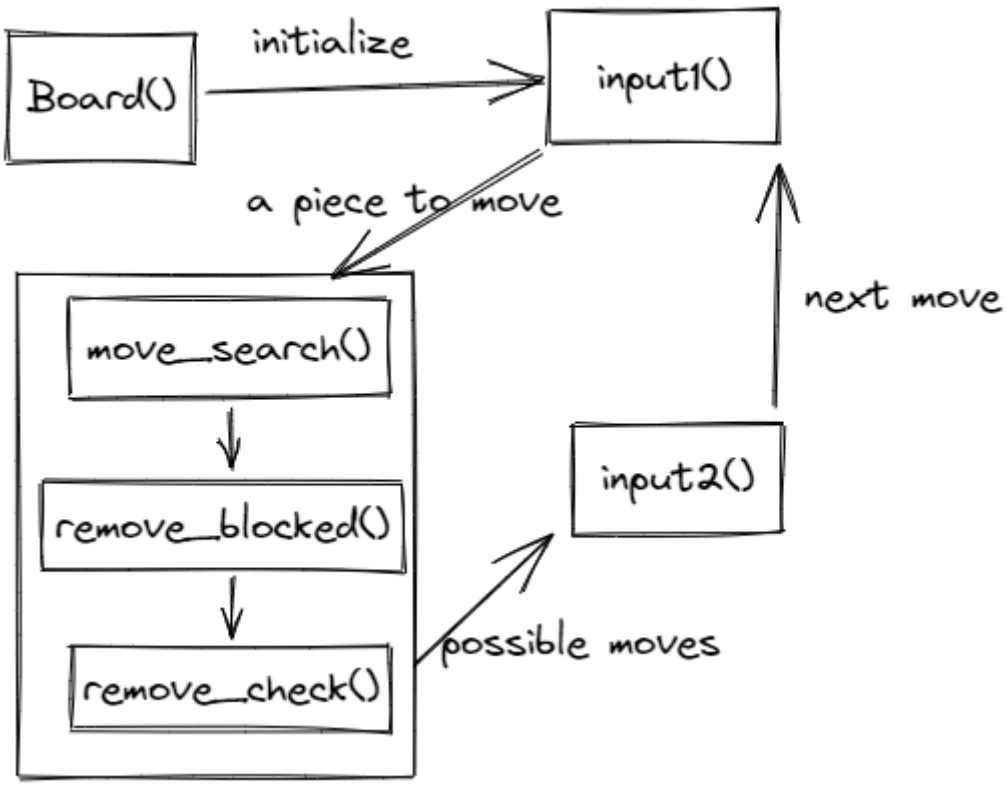
Position objects are attributions of Pieces object. A Position object stores two integers, which represents the row and column of a piece in a 2D vector object. Its member function move() provided a safe way to move a piece. That is, judging a move's validity without actually move the piece. Another member function is_empty () returns the validity of a position. Class Position is equipped with two ways of initialization: with paraments as position and without paraments.

Pieces objects contained attributes such as name, type and position. Inside class Pieces, five member functions are defined. The set() function has the ability to change a Pieces object's attributes in a simple way. The clear function will reset attributes to default, except position.
The asterisk() function can add or remove the asterisk showing on board (to indicate possible moves) accordingly. swap_pst() will exchange positions of two pieces. enemy_side() will provide enemy's side.

Board objects are in charge of the main game process. The most important attributes of Board are vboard (a 2D vector of Pieces object) and board (a vector of string). vboard is an abstraction of board which reduced the complexity to manipulate all pieces with corresponding index (position). board is the visualized game board. Their correspondence

are defined by two map variables: true_row and true_col.

The following picture shows the main process.



We will first instantiate a Board object via constructor function Board(). That is, creating all pieces and a board then putting them together. A timer is started meanwhile (to record the total game time).

After that, the input1() function is called to handle the first input. Where input's validity being examined and transferred to move_search() function.

In move_search(), a vector object of position pst_possible is created and filled with as many as possible probably valid moves. Then it is passed to remove_blocked() function to remove moves that are blocked by other pieces. Finally, possible moves are received by remove_check() function. Here moves that touch allies and moves that put allied king into check (king will be captured) are removed. If there are valid moves remain, they are supposed to be passed to input2() as paraments. Note that removed_check() may call another function is_check() which calls remove_check() indirectly. This can create an infinite loop. So remove_check() are able to receive a parament as mode, to decide whether to call is_check() and call input2().

As for input2(), possible moves are showed with asterisk and validity of input is checked here. After that, we actually move the pieces to its destination (by vboad_swap() and swap_pst() and clear() functions), then examine the game state, such as when to execute castling, promotion and game over process.

3. Personal achievements

I am in charge of the program's development. That is, I established the structure of the program and implemented the majority of this program. In addition to this, some base material for the report was written by me.

This program is developed according to the KISS principle (keep it simple, stupid) as much as possible. During the development, I always implement the most basic features first, then tackle the less important ones. For example, the board's appearance is the first to be designed. And timer is the last to be developed. To write a function, I would first clarify its role and the logic of data transmission, then try to implement it. Every time a small part was implemented, I will check if it works before continuing. Thus the development went smoothly.

4. POST-MORTEM

What went well:

The implementation of showing possible moves have once caused lots of mistakes, since searching possible moves needs to examine "check" state and examining "check" state needs to traverse all enemy' possible moves. That is an infinite loop. Eventually, I defined a special parament of move_check() function called mode, to control which parts of this function can be accessed.

Removing vector object's element with iterator was frustrating at first. As soon as an element is removed, the iterator would be out of function. Any operation on that iterator will cause an error. Thankfully, the erase() method of vector can return the next iterator. I managed to continue traversal by putting the increment of iterator on loop body and executing it according to whether iterator are changed by erase().

What can be improved:

The Board class is excessively huge and complex, harming maintainability and reusability. It could have been split into smaller parts.

To be honest, this project has the appearance of object-oriented programming, but a soul of procedural programming. We can definitely dig deeper in object-oriented programming and take advantages of its features like encapsulation, inheritance, etc.