1 Introduction

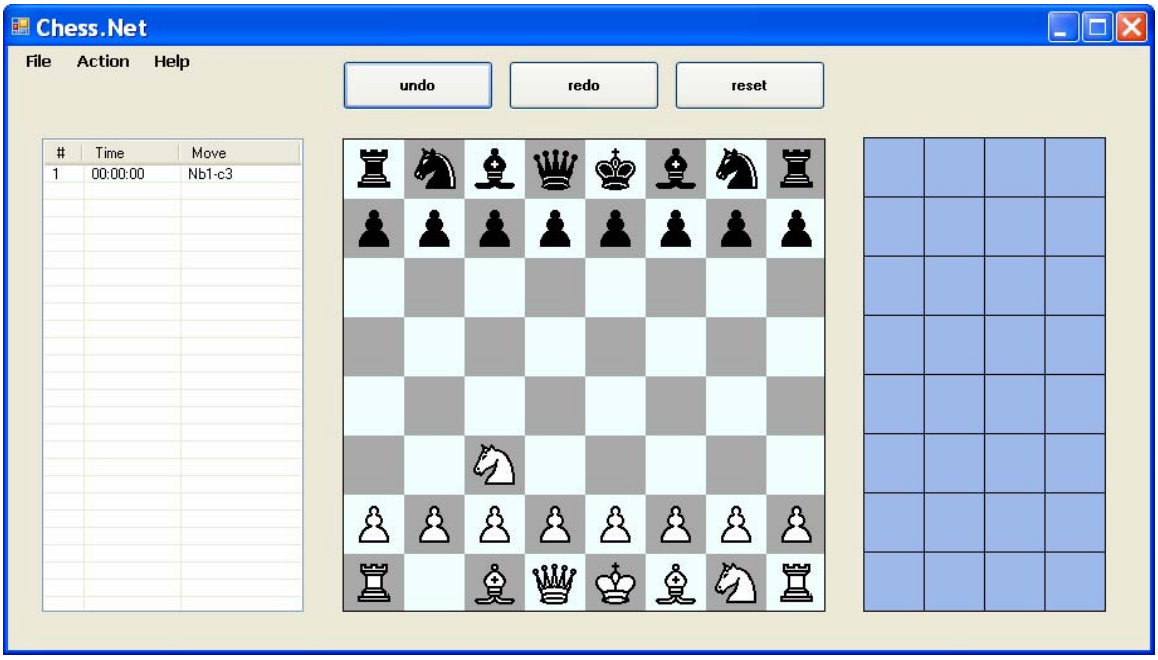
In this assignment, you are required to extend the results of first assignment to implement a fully workable chess game, which allows human being (the white player) to play with the computer (the black player). The goal of this assignment is to exercise your skills in WinForms, event-based and multithread programming.

You will have to implement the following features of the game:

- Provide a graphical chess board so that each piece can only be moved to a legal position on the board.
- Implement the computer player component that can give the move of black player at each step.
- Implement a multithread program.
- Allow players to redo/undo any moves.

2 GUI requirement

The GUI of the chess program looks similar to the following one:



As seen in the above picture, the GUI should include a chess board, an area to show the captured pieces (blue squares in above picture), a history list and some common commands (for example, redo, undo and reset).

The item "**File**" in the menu includes the following subitems:

- **New**
  - This item starts a new chess game and renders the initial positions of the pieces on the board

- **Load**
  - Open a position file recorded in the Forsyth notation, calculate the next move of the black player and render the result on the board.
  - Do not consider the moves depending on history (such as En passant, castling, etc).
- **Open**
  - Open a game file "*.chess". The format of the game file is introduced in section 2.1.
  - Set the piece positions and the move history.
  - Start to play from the current position.
- **Save as…**
  - Save the current game into a game file
- **Exit**
  - Exit the game without saving the state.

The item "**Action**" in the menu includes the following subitems:

- **Redo**
  - Reverse the latest undo command. This item has an effect if there exists an undo command to reverse only.
- **Undo**
  - Undo the latest move
- **Reset**
  - Reset the board to the initial state of the game.

The item "**Help**" in the menu includes the following subitem:

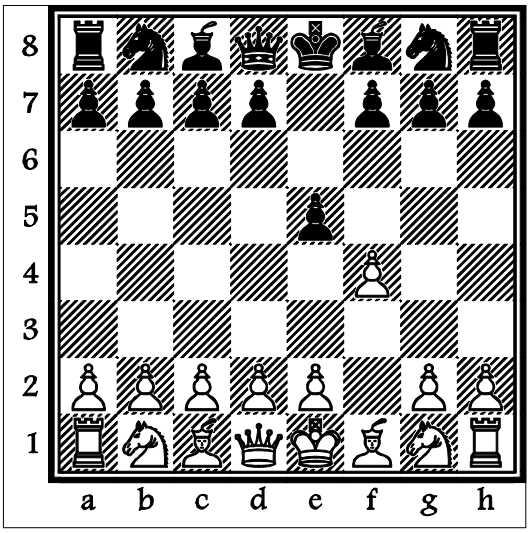
- **About**
  - Show your name and the version of your game application

2.1 Format of a game file

A game file *.chess is composed of two parts: the current position of the pieces recorded in the Forsyth notation and the history of the moves recorded in the algebraic notation. The format of the game file is defined as follows:

[position]
position area
[move history]
move area

For example, in the following board, the corresponding game file is listed on the right side.



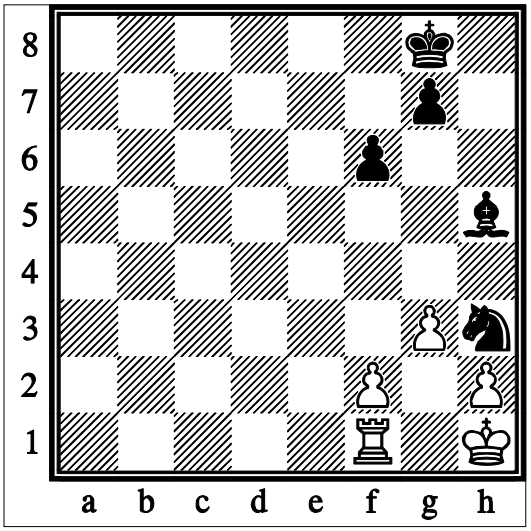
[position]
rnbqkbnr;
pppp1ppp;
8;
4p3;
5P2;
8;
PPPPP1PP;
RNBQKBNR
[move history]
1. f2-f4 e7-e5

Notice that except for the last line in the position area, a semicolon separates each description of the rows. Finally notice that each move in the move area is numbered, and is not followed by any delimiter.

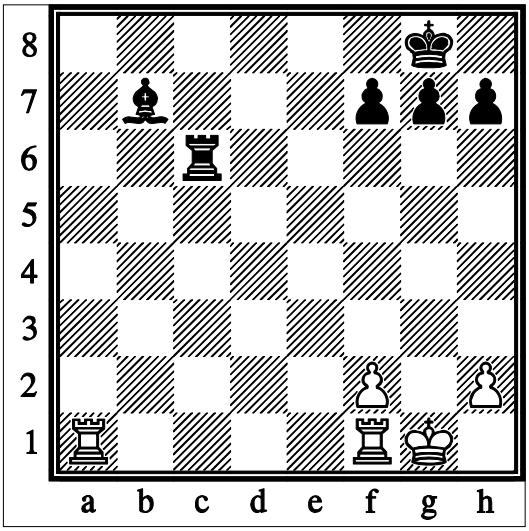
3 Functional Requirements

3.1 AI component

As a minimal requirement for the AI part, the program shall find the optimal move when the white player can be checkmate in one step:

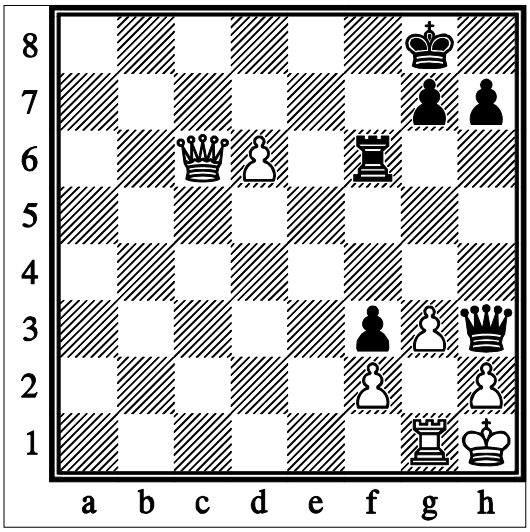


Optimal move: ... Bh5-f3



Optimal move: ... Rc6-g6

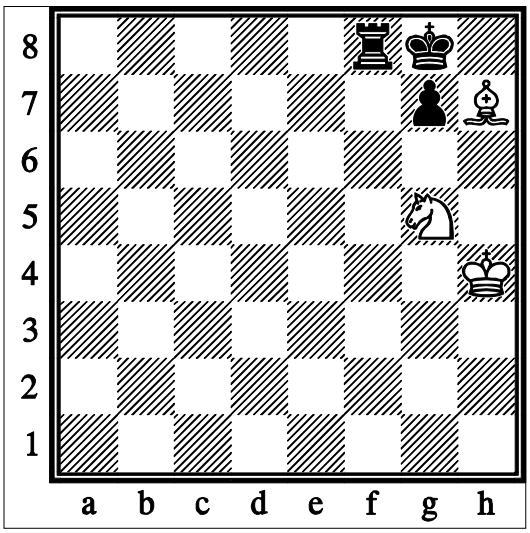
Furthermore, 5 extra points are given if the program returns the optimal next move when the white player can be defeated in two steps:



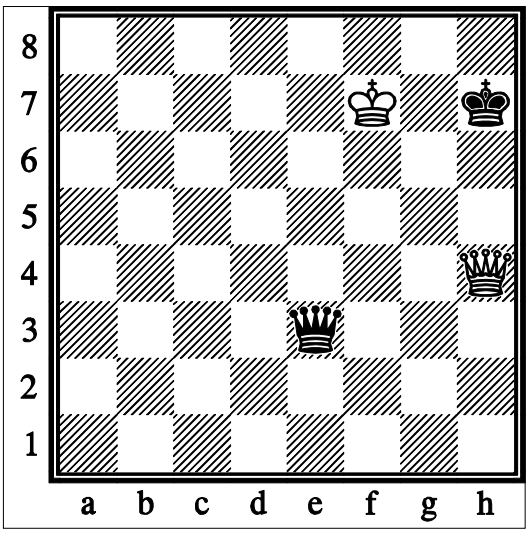
Optimal moves:

1. ... Qh3xh2
2. Kh1xh2 Rf6-h6

Another minimal requirement for the AI is that a king under attack shall escape if possible.



The next move shall be: ... Kg8-h8



The next move shall be: ... Qe3-h6

3.2 Multithreading

The application should be implemented as a multithreading application. The program should still be responsible when it is performing some lengthy task for the user. The details regarding implementing a multithreading GUI application will be illustrated in the exercise section on May 17th.

4 Program structure

We would like to suggest you to adopt the Model/View/Controller (MVC) architecture in the design of your program's overall structure. This model was first introduced with Smalltalk-80, and

includes following three parts [1]:

- **Model**: The domain-specific representation of the information on which the application operates.
- **View**: Renders the model into a form suitable for interaction, typically a graphical-interface element.
- **Controller**: Event handlers that trigger changes on the model.

Although MVC comes in different flavors, the working behavior looks as follows:

1. The user interacts with the graphical interface in some way (e.g., user presses a button)
2. A controller handles the input event from the graphical interface via a registered handler or callback.
3. The controller accesses the model possibly updating it in a way appropriate to the user's action (e.g., controller updates user's shopping cart).
4. A view uses the model to generate an appropriate graphical interface (e.g., view produces a screen listing the shopping cart contents). The view gets its own data from the model. The model has no direct knowledge of the view. (However, the observer pattern can be used to allow the model to indirectly notify interested parties – potentially including views – of a change.)
5. The user interface waits for further user interactions, which begins the cycle anew.

5 Code Delivery

Requirements:

1. The delivered code has to comply with the design guidelines [2].
2. The program has to compile with Visual Studio 2005 and the .Net Framework 2.0 under Windows XP.
3. The program has to run under Windows XP with the .Net Framework 2.0.
4. The application should give the next move in a reasonable amount of time (a few seconds).

Deadline: June 21st

Implementation tasks and point assignment:

- Graphic chess board (10 points)
  - GUI
  - File -> Load
  - File -> Open
  - File -> Save as
  - File -> Exit
  - Help -> About

- AI component (10 + 5 points, 10 points for minimal requirement, 5 points for extra requirement specified in AI requirement)

- Redo/Undo moves (10 points)
- Multithreading (15 points)

Delivered document:

- Source code

References

[1] MVC:
http://en.wikipedia.org/wiki/Model-view-controller