Plan of Attack

Chun Li
Weichen Zhou

November 22, 2013

Summary

For this deliverable, we will complete the project in the following steps:

1. Draw the game board
2. Create the command interpreter
3. Create the pieces
4. Implement all the basic moves of the pieces
5. Implement all the advanced moves, i.e. castling, *en passant*, promotion
6. Implement the rudimentary level of AI for computer player
7. Extend the game to include graphics
8. Finish the advanced AIs

We will try our best to adhere to the following schedule:

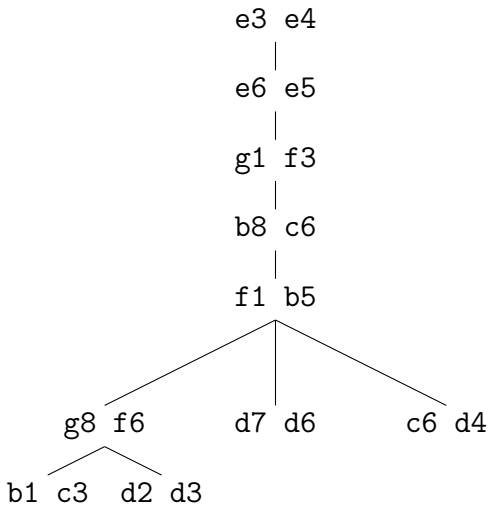
Due Date	**Component**	**Assigned**
November 21 (Wednesday)	UML Plan of Attack	Weichen
November 21 (Thursday)	Create `Board`	Chun
November 22 (Friday)	Implement command interpreter	Chun
November 23 (Sunday)	Implement moves for pieces	Weichen
November 25 (Monday)	Testing	Weichen
November 26 (Tuesday)	Level 1 AI	Weichen
November 27 (Wednesday)	Graphics	Weichen
November 28 (Thursday)	Level 2 AI	Chun
November 29 (Friday)	Level 3 AI	Chun
November 30 (Saturday)	Level 4 AI	Chun

In a nutshell, Weichen will be responsible for implementing all the moves for the pieces of chess, level 1 AI, and graphics, while Chun will be responsible for creating the `Board`, the command interpreter, and all the advanced AI. Both partners will be responsible for designing the program, although Weichen will be the one actually drawing the UML and writing the plan of attack out.

Questions

1. **Chess programs usually come with a book of standard opening move sequences, which list accepted opening moves and responses to opponents moves, for the first dozen or so moves of the game. Although you are not required to support this, discuss how you would implement a book of standard openings if required.**

Assuming that we are following the book exactly, we will simply implement a tree of moves (the `Move` class in our program). For example, take the Ruy Lopez starting position tree, which would look something like this:



Assuming that white starts, the even-numbered level (starting with the zeroth) would be interpreted as white's move, and the old-numbered level would be interpreted as black's move. The three branches at end are the possible defenses that black can take (the Berlin defense, Steiniz defense, and Bird defense, respectively). In response to the Berlin defense, white can take the move `b1 c3` or `d2 d3`. Whenever we reach the end of a branch (an empty tree), we will know that we have depleted the possible moves that we can take.

2. **How would you implement a feature that would allow a player to undo his/her last move? What about an unlimited number of undos?**

   We will implement unlimited undo using a 2D array and a stacks of moves. The 2D array will be the board created after setup. Whenever a move is made, it will be pushed onto the stack. When a player decides to undo their last move, the topmost move will be popped from the stack, and the board will be created by applying all the moves from the stack.

3. **Variations on chess abound. For example, four-handed chess is a variant that is played by four players (search for it!). Outline the changes that would be necessary to make your program into a four-handed chess game.**

   Referring to UML, we currently have a `Board` ctor that accepts the rows and columns of a board as its parameters. We will use that to create a 14×14 board and block off the corners that will not be used. The basic rules for four-player chess is the same as that for two-player chess, but we would need to update the command interpreter so that it will recognize the existence of four different colours and players instead of just two.