Plan of Attack – Chess Project

Project Breakdown

See UML diagram in uml.pdf.

All major components of the chess program outlined in the UML shown in uml.pdf, and important classes are described the following section.

Game

This class will serve as an interface to the chess game for main(). The main program will ask the Game object to start a game, and all further interactions between the user and the program will be handled by methods in the Game class. This organizes and modularizes the program better, which allows for better parallelization of tasks, and encapsulates implementation details for the chess game. For instance, the command interpreter and initialization mode are both part of the Game class.

ChessBoard

This class will be the most central and probably largest class of the program. It will contain (composition):

- A 8x8 board of Cells
- White and black vectors of 16 Pieces (which are discussed below)
- Pointers to TextDisplay and GraphicDisplay
- A list (vector) of moves conducted in a game
- Chess board statistics (e.g. which turn is it)

Players will interact with the ChessBoard to register a move. The Pieces will interact with the ChessBoard's methods in order to determine whether it can make a legal move to a position on the board. In addition, the computer player will use friend functions in ChessBoard to determine what move it should make. Finally, calling operator<< on ChessBoard will be equivalent to calling the operator<< on TextDisplay.

Cell

Each Cell will contain a pointer to Piece, so in essence, a 8x8 array of Cells is a representation of the chess board. Cell will call the draw methods in GraphicDisplay when a piece has been moved. Also, Cell may contain statistics to keep track of its state.

Piece

Piece is the abstract superclass for the chess piece classes King, Queen, Bishop, Rook, Knight and Pawn. The superclass will contain methods common to all chess pieces while the subclasses will contain unique implementations for each piece type. For instance, Pawn will have methods for promotion and en-passant.

Player

The two Player classes will inherit from an abstract superclass that will contain a pointer to ChessBoard. Therefore, the Player will register a move with ChessBoard. The ComputerPlayer will contain AI methods that will rely on the friend functions in ChessBoard in order to access ChessBoard's data more efficiently. This way, ComputerPlayer should be able to evaluate the moves it could make by trying out every possible move and countermove for at least 2 or 3 turns.

TextDisplay and GraphicDisplay

The two display classes do not inherit from a superclass because they do not have much in common. TextDisplay will contain a char array that it will use to output the entire chess board every time a move is made. GraphicDisplay will be updated on a cell-by-cell basis when Cell calls its draw methods, in order to preserve efficiency. If time permitting, we will include additional graphics features.

Move

The ChessBoard class will contain a list of Moves that the players have made in order to allow for flexibility when implementing undo and chess opening features.

Software Design Patterns

Observer pattern is the most prevalent pattern used in this program. The displays will act as observers while the Cells and ChessBoard act like subjects because they notify the observers when they are changed. Singleton pattern, though not necessary, could be used on Game to ensure only a single game is being played. The template method will be used when Piece's move method decides whether a move is legal.

Schedule Planning

Estimated Date	Goals	Who's Responsible
Fri, March 28	Implement the Piece class and all its subclasses	Each person responsible for a subset of Piece subclasses. (e.g. Hao – King, Wenchao – Pawn)
	Header files for all classes	Wenchao
	Plan of Attack and UML	Hao
Sun, March 30	Implement basic ChessBoard methods	Collaborated
	Implement TextDisplay	Wenchao

	Implement basic Game class methods, including a simple command interpreter	Collaborated (divide work whenever possible)
	Implement HumanPlayer class	
Mon, March 31	Aim for playable Human vs Human (TextDisplay) Begin GraphicDisplay implementation. Begin Level 1 AI.	GraphicDisplay – Hao Begin AI - Wenchao
Wed, April 2	Aim for playable GraphicDisplay Complete AI Level 3	Collaborated
Fri, April 4	AI Level 4. If time allows, complete undo, implement chess openings for AI and additional graphics features.	Collaborated

Answers to Questions

1. Chess programs usually come with a book of standard opening move sequences, which lists 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.

Since ComputerPlayer already has friend functions to ChessBoard that will use ChessBoard's members to calculate what the next move it should make, it will have access to the list of Moves that each Player has made. So, a new Openings class could be created, which would simply be a trie of moves, where each unique path from the root node would represent an opening in chess. For instance, the Ruy Lopez opening, which is one of the most popular and powerful openings in chess, would have a path with nodes root-e4-e5-Nf3-Nc6-Bb5. The ComputerPlayer would have a new method that would traverse the trie based on the current list of Moves and see if a move from a chess opening is possible. The Openings class would also have an initialization method that will read from a file that will contain the openings and insert the items in to the trie. Such a file would likely have to be created manually.

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

The main idea here is not destroy or remove a Piece when it is captured or promoted. It will simply be removed from Cell, ie the Piece pointer in Cell will be set to NULL, but the Piece itself will remain in ChessBoard. When an undo command is issued, the ChessBoard will use its list of Moves to undo the most recent move. Such an operation would require resetting the Cells, changing the state the Piece in case it was promoted or captured, and updating the displays. Reverting a basic move is straight-forward. Reverting a capture would simply require adding back the Piece to the Cell to which it belongs. "Unpromoting" would require the promoted piece pointer in Pawn to be reset to NULL. An unlimited number of moves would not be a problem because we keep the entire list of Moves.

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.

This requires a substantially more modifications to each class than the previous two features. Here are some changes that would be necessary to implement four-handed chess:

- There are four instances of Players, which would require modifications to the Game class
- The coordinate system in ChessBoard and Cell would require modification. The move methods in ChessBoard would now check for illegal moves differently due to the “holes” in the corners of the board
- Both display classes would be changed to draw the new board accordingly
- The legal move checking mechanisms and the AI would now have to account for the possible moves of 3 other players, not 1
- Need to account for the team system in four-handed chess (more below)

Some things that we will do to make the transition easier include:

- Not including any code that depend on the colour of the pieces
- One way to circumvent rewriting all the code with regard to capture and checking is to have a Team superclass from which Player is inherited. This way, Player will still overload the move methods, but now the Game and ChessBoard classes will know which Team each Player is on,