CS 246 A5 Plan of Attack

Breakdown

The first step in our project is to write the minimal amount of code to get a working command loop up. This will involve bare bones implementations of HumanPlayer, GameMaster, GameState, GameBoard, and one or two BoardElements. James will do this by March 29th, at which point we will both be done all of our other assignments and be ready to work on the rest of the tasks in parallel.

The next few components can all be worked on in parallel, but they're listed in the order we expect to work on them. Of course at each step we'll be writing tests to ensure everything is functional.

We'll implement the custom game setup command loop so that it's easy to write tests. James will do this.

We'll get the actual rules of chess going. That will involve implementing each piece and generating their possible moves. Garo will do this.

These two should be done by March 31st.

We'll implement the AI levels 1-3. Garo will do this.

We'll implement the graphics display. James will do this.

These two should be done by April 2nd.

At this point we'll have implemented the core requirements of the project, and have a couple days left for testing and polish. If we're ahead of schedule, we'll work on a more advanced AI and some of the extensions described below.

Answers to Questions

Our answer to the first two question is largely based on how we decided to represent the game state: since a lot of the extensions in the document derive their difficulty from being able to look arbitrarily far back into the game history, we made game states immutable. Whenever you perform an operation on a game state, it returns a new game state in which that operation has been performed, and the new game state contains a reference to the old one. So the game states form a linked list (or potentially tree). With that design in mind:

How would you implement opening books?

Since the game state contains the game's move history, it's a simple matter of looking up the move history in a dictionary of good opening moves and picking one.

How would you implement unlimited undos?

Each undo simply sets the game state to the previous game state. It would even be easy to add redos if we stored the old head of the linked list.

How would you implement four handed chess?

This would be the most involved change:

- to account for the modified board, we'd add a new type of BoardEntity which is a space that can never be moved into, and that would require no changes to our existing code outside of board setup.
- some of the move semantics change, specifically for pawns in any case and for every piece depending on whether or not it's a team game. this part is annoying because it will involve heavily specializing a lot of our class hierarchy for 4 player chess while our old model specialized based on the piece type. we can ease some of this pain by taking advantage of our class hierarchy and providing default behaviour that works most of the time, and the only overriding in the classes with different behaviour.

Possible extensions

Some of the extensions are quite easy in our design, so we may focus on these first: undoing moves, checking for repeated board states, printing move history, etc.

Bigger ideas include modifying the rules of chess in weird ways (what if there's an NPC dragon on the board?) or bug house chess.

Ideas for the level 4 algorithm are a basic search through the game tree up to a certain depth, or maybe a Monte Carlo Tree Search if we have a lot of time, using material as a heuristic.