CS246 Chess Group Project Plan of Attack

Project Overview & Breakdown

From reading the project specifications there seems to be three stages of completeness for Chess.

> Stage 1) A basic stripped down version of Chess playable only by humans. This means an implemented game with a text display without the following rules: pawn capturing, pawn initial movement (two squares forward), en passant, pawn promotion, castling and King move validity. Creating a test suite for this basic implementation of Chess should be done as well.
>
> Stage 2) Implementing rest of the formal rules included in the game, the computer player's behavior and the graphical interface for the program and the test suite for this implementation.
>
> Stage 3) Any enhancements beyond the required specifications and an test suite for this.

Documentation should be completed as parts are completed.

Stage 1 Timelines

Judging from the project's UML the basic game should be broken down into the following classes.

- An Game Controller: Which handles I/O and overall control over the game
- A Game State: Keeps track of the board and controls is responsible for making movement calls for the game
- Pieces(King, Queen, Bishop, Knight, Rook, Pawn): Define the behavior of pieces in the game
- Players(Human and Computer): Defines users and how they interact with the game
- Move: Used throughout the program.

The Game Controller has dependencies on most other parts of the project some methods can be completed in the beginning though. This section should be started first but should be completed last.

The Game State depends on the piece classes and should be completed after the pieces are completed some methods and fields such as the board can be completed first.

Pieces should be started and completed first since most other classes use it.

Players should be completed last since they come into play after the board is set up.

Move is used throughout the program and should be completed first.

Individual responsibilities:

Game Controller- Group Effort

Game State- Daniel + (Group Contribution)

Pieces- Allen + Mike

Players- Group

Move- Group

Deadline without Test Suite and Bug Fixes: July 18th

Deadline with Test Suite and Bug Fixes: July 19th

Stage 2 Timelines

Stage 2 can be broken down into 3 sections:

- Additional Chess Rules: pawn capturing, pawn initial movement (two squares forward), en passant, pawn promotion, castling, King move validity.
- Graphical Interface: Interface using graphics
- Computer players: Behavior of computers as well as their interactions with humans and other computers

Additional Rules will need to be implemented first in this case as the computer behavior relies on these rules(these rules will be split among group members but yet to be decided who does what)

Graphical Interface should be implemented last as this does not affect other components of the game

Computer Players should be implemented after the game rules are completed.

Additional Rules- Group (Pawn Rules- Mike, Castling- Allen, King Moves- Daniel)

Graphical Interface- Group

Computer Behavior- Group

Deadline without Test Suite and Bug Fixes: July 23rd

Deadline with Test Suite and Bug Fixes: July 24th

Stage 3 Timelines

Possible enhancements planned include a player move logs, and starting moves for computer players. These will only be done if time permits them.

Chess Questions

Q1: Add a private static array of vectors of Move objects, containing the opening sequences (with move order in reverse), to the AI class, as well as a non-static vector of vectors of move objects, containing the currently valid sequences (with the last move at the front). Additionally, add a lastMove field and getLastMove function to Game, which will store a move when executeMove() is called, and return the last move executed, respectively. Then, when the AI is taking its turn, it will compare Game.getLastMove() to the back of the vector of move sequences. If they don't match, remove the move sequence, and if they do, remove the move from the sequence. After processing every move sequence, chose a random move sequence, and repeat the process using the next move in the chosen sequence instead of getLastMove(), then return the move.

Q2: Add a vector of pairs of Moves and piece pointers to Game, with the piece pointer's representing any piece captured. Then, when undoing a move, (with a call to Game.undo()) set the board at Move.origin to the piece at Move.destination, and set Move.destination to the piece captured (which, if there was none will be null)

Q3: the size of board would need to be enlarged, and checks performed to ensure that pieces never move to the 2x2 squares in each corner. Additionally, everywhere a bool is used to represent colour, we would need to change to an int (from 1-4). We would probably have a constant, static 2x4 array used to convert the player number to an x,y direction. Lastly, there would be a couple of minor tweaks, such as increasing Controller.players from size 2 to size 4, and changing loop conditions