Charlie Cao
Jack Liu
Simon Huang

CS 247 Project: Chess Plan

Breakdown of Project

Tasks	Exp. Completion Date
Setup GitHub and version control, discuss plans with teammates	July 15, 2022
Develop test cases and a respective test suite that we will use to evaluate the correctness of our project	July 16, 2022
Create and populate header files that model UML design	July 17, 2022
Create .cc implementation files, as well as makefile	July 17, 2022
Formulate skeleton of implementation files to house appropriate functions/constructors/destructors	July 17, 2022
Work on GameInstance class and command interpreter to take in user input/configure the game correctly ● setup() allows user to place pieces on the board and specify whose turn is next ● startGame() enters a loop of asking each Player for input, asking GameBoard whether that move is valid, then updating the board and associated flags.	July 18, 2022
Start coding GameBoard which is a part of a GameInstance to set up the board ● Has methods to set a square to a piece, make a move, update text and graphical displays, check if a player is in check, check if a player has any valid moves left	July 19, 2022
Create private fields and methods for abstract GamePiece base class ● Each piece has a char for type, char for white/black, a pointer to the actual board, and an overridden method for calculating all valid moves	July 19, 2022

Develop methods for concrete game pieces such as Bishop, Queen, Pawn, etc… ● Implement method for calculating valid moves for each of the different piece types ● Implement special rules (promotion, en passant, castling)	July 20, 2022
Create private fields and methods for abstract Player base class so that HumanPlayer and ComputerPlayer can inherit from it ● getMove() returns the move to be made, for a human player simply wait for input and parse ● If the move is a promotion, call promoteTo() to find what piece to promote to ● Each player has a pointer to the GameBoard	July 21, 2022
Develop computer player levels 1-3 ● getMove() for a computer player needs to find all possible moves and select one based on difficulty level, need to check whether each move is a capture, check, if a piece is under threat, etc	July 23, 2022
Implement TextDisplay class and develop the update function ● Keep a local copy of the current board, compare with updated board to only redraw squares that are changed	July 24, 2022
Using ideas/logic from TextDisplay, implement GraphicalDisplay class which draws and *moves* the pieces accordingly ● Similar to TextDisplay, only redraw squares as needed	July 25, 2022
Test the functionality of chess game with test suite along with the TextDisplay along with debugging GraphicalDisplay	July 26, 2022
Work on computer player level 4+	July 26, 2022
Work on extra credit features such as standard openings, move undoing, chess variations, etc… (optional)	July 26, 2022

<u>Responsibilities of each member</u>

Simon: Develop test cases for pieces, work on creating skeleton for pieces/board files, implementing code for concrete classes regarding individual pieces, develop TextDisplay class

Charlie: Implement the GameBoard class and its methods, implement getting input from human player to setup the board and play moves, implement code for concrete classes of pieces

Jack: Implement the computer player and the various levels of difficulty, implement the graphical display class

Questions

Question: 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.

- A method of implementing this would be to have a set of accepted opening board states, and if the computer is one move away from reaching one of those board states, then it makes the move to reach that board state.

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

- A simple undo would be to cache the piece that was last moved, the piece that was captured (if required) and the original position. When asked to undo, the piece last moved is restored to its original position, and the captured piece (if any) is placed in the new position.
- For an unlimited number of undos, we would need a vector of Moves objects. Each Move stores a former position, a new position, the piece moved, and the piece captured. After a valid move has been made by piece P from position A to position B and capturing nullable piece Q, we emplace Move{A, B, P, Q} into the vector of moves. When required to undo a move, we pop the back of the vector and follow the procedure outlined in the simple undo.

Question: 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.

- Assume that we are incorporating FFA four-handed chess. One of the main changes that have to be made is the addition of cells on the board to support the four 8x3 extensions where the four players begin with their pieces. Furthermore, a point system class should be implemented to keep track of the four players' points to determine the winner at the end. We would have to override functions such as checking, capturing, etc… so that it incorporates point addition to respective players.