CS246 Winter 2021 Project – Chess

Plan of attack (Due Date 1)

April 9, 2021

Group Members:

Vrajang Parikh

Mohanpreet Narang

Geraldine Mendoza Trivino

Project Breakdown:

First, we break down overall project work into the following categories/components:

- Game Logic: It is the core of the game. Need to implement chess rules, validate moves, account for checks, and maintain the board.
- Display: Graphical Display and Text Display. Display pieces on the screen and other information. Moreover, it would be helpful if the user can also interact with the graphical display
- Computer Mechanics: Implement various difficulties for the computer player
- Input mechanism: command Interpreter and command-line Interface
- Main Game loop(controller): combine all other components and simulate the game. The main game loop for Chess is following:
    - Get user input (Input mechanism)
    - Update the game state (Game logic, Computer mechanics)
    - Draw the game (Display)

- Bonus: We intended to attempt the following bonus components in order of decreasing priority
    - Undo feature
    - Accept input in different chess notations.
    - Print a history of all moves made in various notations.
    - Next move suggestions
    - The standard book of openings
    - Chess variations: mainly focus on variations that require 2 players and do not affect the majority of rules. For example, FischerRandom (chess 360)

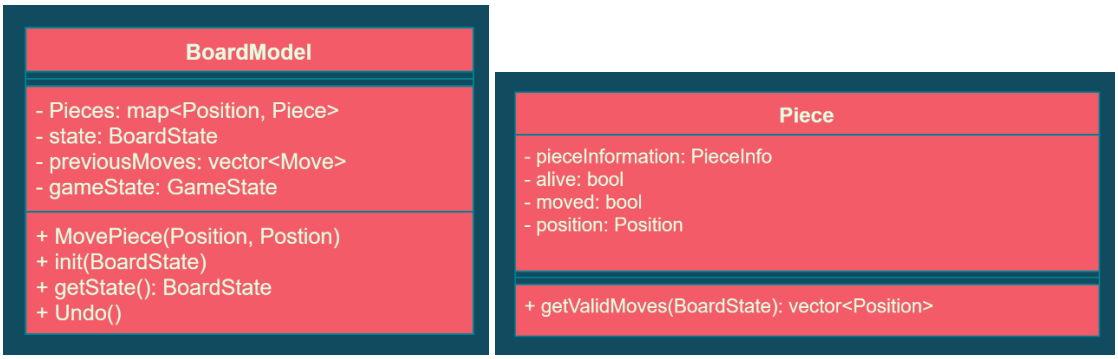
Design:

We are planning to use Model-View-Controller Architecture (MVC) along with Observer pattern for the project.

Model: **BoardModel** class will maintain the game state (chess board). It will inherit from the subject class. Internally, the **Piece** class to validate moves and changes in the state. Each piece on the board will have appropriate piece type. The purpose of Piece is to find all the possible *legal* moves for itself, given the current board state. The plan is to use the pull variation of the observer pattern. The board's **state** (information) will be stored and passed to observers (displays) as a 2-D vector.

The game logic will operate on a dictionary with position as key and the piece (an instance of Piece class) at that position as the value. This is how we keep a track of where each piece is, and if a piece is captured, it is removed from the map.

This division of responsibility and rules will help to maintain game logic and break down the implementation into smaller and manageable parts.



View(s): The purpose of the view is to display (draw) game state. The plan is to have two views: **TextDisplay** and **GraphicsDisplay**. They will derive from Observer class, and they observe the BoardModel. Whenever, they are notified of changes in game state, they will update the graphics display or output board in text form.

Presenter (Controller): The job of the presenter will be to interpret commands from the standard input (Command interpreter) and call functions in BoardModel to update/setup the state. It will also take care of keeping scores.

Computer Mechanics: Use strategy design pattern to implement algorithm for each leval.

Level 1: Use a random number generator and seed to choose a piece (that is currently on the board, and has at least 1 valid move). Then, pick a random move associated with that piece.

Level 2: Iterate over the valid moves of pieces of the colour that computer is playing and make a move that results in (1) a check, (2) a capture or (3) if neither is possible, choose a random move.

Level 3: Prioritize the moves which result in capture of important enemy pieces (ex. capture of queen is preferred over pawn). Also avoid making a move that risks our piece by checking the valid moves of all the pieces of the opponent: avoid placing a piece in one of these spots.

Ideas for Level 4: Use some sort of minimax algorithm to capture enemy pieces with the highest points while risking our own pieces with minimum points. Try to implement standard opening if we have time at the end.

General Timeline & responsibilities:

We will divide the project into three phases.

Phase 1: (4 days, April 9-12)

The goal for this phase is to implement first iteration of (board) Model, Controller, and Text view. By the end of this phase, we will have basics and almost all core mechanics of the game.

Work division:

Vrajang:

- Controller (Basic input harness, command line Interface)
- Implement Basic game loop that allows to player vs player gameplay
- Implementation for Queen and Rook pieces (Given the board state, find all possible moves that these types of pieces can make).

Mohanpreet:

- Work on first iteration of BoardModel
    - Implement accessors/mutators that allows to mutate state of the game
    - Perform basic move validations
- Text Display
- Implementation for King and Knight pieces

Geraldine:

- Implement abstract Piece class
- Implementation for Pawn and Bishop pieces
- Implementation for methods that allows to move pieces in BoardModel

Phase 2: (3 days, April 13 – 16)

The goal of this phase is to finish the first iteration of graphics display and computer mechanics. The main goal is to finish all the required specifications for the project by the end of this phase.

Vrajang

- Work on graphics display
- Refine controller and modify basic game loop to allow for computer gameplay
- Validate the correctness for each piece. (Make sure that given the state, all pieces correctly gives all possible valid moves)

Mohanpreet:

- Work on refining the Model class and making sure that all moves are valid, and Model works correctly. Specifically, work on castling, pawn promotion and En passant

- Start working on Computer Mechanics (Level 1, Level 2, Level 3)

Geraldine:

- Start working on Computer Mechanics (Level 4)
- Work on bonus feature undo/show history if everything goes according to plan.

Phase 3:(4 days, April 17-20)

The plan of this phase is to implement bonus features and finish working on second iterations of display and computer mechanics. By the end, we would like to have polished game.

The priority is to finish any leftover work from phase2. Afterwards, if we have time, we plan to work on the following:

Vrajang:

- Allow interaction through graphics display.
- Accept input in different notations
- Make sure game loop works correctly, commands interpreted correctly, and wrong command does not result in any crash

Mohanpreet:

- Work on standard opening
- Validate computer algorithms

Geraldine:

- Work on implementing chess variation. This will require to modify the controller, and game loop. The goal is to implement variations that does not require modification of how pieces move
    - FischerRandom (chess 960): https://en.wikipedia.org/wiki/Fischer_random_chess
    - Marseillais chess (also called Double-Move chess) https://en.wikipedia.org/wiki/Marseillais_chess

Implementation Ideas for project specifications questions:

Question: Book of standard opening

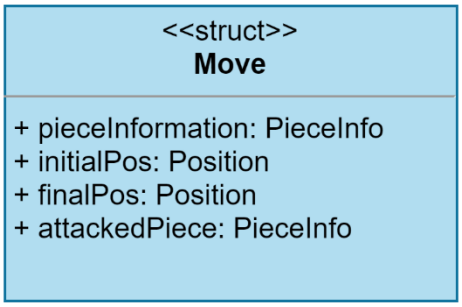
Implementation idea: Upon searching, we figured out that most chess games are stored in PGN(Portable Game Notation) format and binary books. To implement, book of opening, we will first need to download opening database or create our own small database that stores openings in tree format (Arena's book format), possibly through parsing a file.

The idea is to construct dictionary structure like trie (or hash table) that allows faster access/search through all openings. After, we have constructed data structure, we just perform search on trie to find the best move. Note that we can "follow" along the trie on every move made, allowing us to use an opening that depends on the player's moves.

To construct this data structure, we will need to use dictionary construction algorithms in cs240.

Question: Undo moves

The idea is to every time we change the state, we store the move that changed the state. We are planning to implement this feature by storing every move made by players in a vector/stack instead of storing board states. To undo an action, we just pop the last item from the stack/vector and use the information to reverse the move, bringing back any piece, if it was captured.



Question: Variations

Since, BoardState uses a vector to state the board and does not assume the size of the board, changing size of the board is possible. However, adding extra players will require changing many things and internals of Piece class and valid Move logic. For example, we need to add more piece colours and work with an "uneven" board shape. So, it would be difficult to add 4 player chess variation. However, variations like king of the hill would be easier to implement.

Discussion about other features

Print a history of all moves made: Since, for undo feature we are storing previous moves, this will also stores the history of the game. We just iterate over that vector and print each move object in a certain format.