Project Breakdown (Chess)

July 19, 2024

Group Members

- Norman Chen (njchen)
- Roger Chen (r346chen)
- Peter Wang (p25wang)

Plan, Completion Dates, and Responsibilities

Initial Setup and Planning (July 15th - July 19th)

This is arguably the most important phase of the project, where we assign tasks between group members to ensure that everyone has an even division of responsibilities and learning opportunities. The UML diagram and the plan of attack documents are also crucial as they will save us alot of time when we actually implement the code. By this phase, we want to have all of our Dev-Ops and development environments setup to ensure smooth and enjoyable coding experience.

- Create UML design diagram and plan of attack (All members)
- Learn the general and niche rules of Chess (All members)
- Create Git repository and version control software (Peter)
- Setup Dev-Ops (CI/CD pipeline) with Docker and GitHub Actions (Norman)
- Implement boilerplate code and classes based on UML diagram (Roger)

Core Game Logic (July 20th - July 23th)

In the second phase of the project, we aim to get the core game logic of chess down, such as how to handle input/outputs from the user, how to determine possible moves for the pieces, check/checkmates/stalemate and how to overall manage the user experience. It is also in this phase that we should have implemented a minimally viable product (MVP) to support human to human interaction.

- Implement command interpreter for user I/O (Roger)
- Develop piece movement rules (ex. handling en passant) (Norman)
- Detection of check/checkmate/stalemate (Peter)
- Implement game state management (Roger)
  - Text display (Roger)
  - Graphical display (Peter & Norman)

Computer Player (July 24th - July 26th)

In this third phase of the project, we introduce a completely new aspect of the game, computer player. This will allow for humans to not only play with other humans, but to also play against a computer at one of four difficulty levels ($1 \rightarrow 4$) and eventually to also watch as two computers play against eachother automatically.

- Implement logic for basic (level 1) computer player (Roger)
- Develop more sophisticated (levels 2 and 3) computer player (All members)
  - Decision tree creation of possible moves (Peter)
  - Popular openings and variations (Norman)
  - Possible moves for checkmate/stalemate detection (Roger)
- Develop artificial intelligence based (level 4+) computer player if time permitting (All members)

Testing and Refinement (July 27th)

In the final phase of development, we would conduct extensive testing on all supported game types, such as human vs human, human vs computer and computer vs computer. The goal of this would be to utilize Google's GTest suite to conduct correctness testing and Valgrind to conduct memory failure tests. Finally, we would also create test cases that ensure edge cases are accounted for through code and branch coverage testing.

- Perform extensive testing on game functionalities (All members)
  - Computer vs Computer (Peter)
  - Computer vs Human (Roger)
  - Human vs Human (Norman)
- Test final program on Linux servers, to ensure it works during demo (All members)
- Prepare for final demo and presentation (All members)

Final Document (July 28th - July 30th)

The final document will be prepared after we have completed testing and refinement of our program. In this document, we will highlight our design choices, learning outcomes amongst features of our program.

- Final Document (All members)

Answers to Questions

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.

- To display a list of standard opening moves, we would first need to compile a list of common chess openings (and their variations) and store the first dozen or so moves inside of text file
  - The list of common chess openings can be found widely available online, and we would compile that information into a compressed format and store it inside of a text file for easy I/O at runtime.
  - Another approach we have considered is to download a large number of historical professional matches, and develop a program to analyze the most common starting moves made, and compile our own list of standard opening moves.
- At runtime, we would scan in the list of common openings and store them inside of a Trie, this would allow for us to suggest possible moves based on their own moves and their opponent's moves in $O(n)$ time where n is the number of moves that has already been played.
- This could be integrated into our game with recommended positions being shown at each move of the opening for both the text and graphics based displays. This could also be implemented before the game start, where players can browse through the command line or graphical display and see a list of the most common opening moves.

How would you implement a feature that would allow a player to undo their last move? What about an unlimited number of undos?

- The approach that we are considering is to store a vector of *Moves* that have already been made since the game started. The *Moves* object would store the starting, ending, piece type, promotion type (optional) of a specific move and we would store a starting of the board in a seperate variable. If a player ever chose to undo a move, we would then iterate through the vector and essentially rebuild the game from the start until the $i-1$th move. Even though this results in a $O(n^2)$ time complexity, we belive that this is far easier to implement and maintain as we do not need to store additional information such as piece taken, checkmates, etc. This would also allow us to efficiently undo more than one move at a time, as we are rebuilding from the start of the game anyways.
- Another appraoch that we have considered is to store a vector of board states, which would allow us to simply revert to a previous board state when a player chooses to undo. However, we believe that this approach is far too memory intensitve, as we would have to store a vector of 8x8 boards.
- Finally, the last approach was just to store a vector of moves, this approach is similar to the one we had in the first bullet point, however in this approach we would store some additional information such as pieces taken and special moves, such as castle. This would require us to reverse the steps taken in the previous move and attept to restore it to a previous state of the game. The downside to this appraoch is that if we undo more than one move at a time, it would require alot more computations compared to our first approach, the implementation of this is also much more difficult.

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.

- To implement four player chess, the core logic of the game would remain the same, as piece taking, check/checkmate and the movement of each piece still remains the same. However, this would be much more computationally expensive as for each move, we have to consider check/checkmates

by three other players, detecting stalemate for more than two players at a time and the list of possible positions is also mucuh larger.

- Moreover, the computer player for four player chess is much more complicated compared to two player chess. The decision tree for a four player chess computer player is exponentially larger than that of a two player chess, as it would consider the possible attacks for two additional players. It would also need to develop a completely new algorithm for the opening moves, as there are no longer a list of accepted openings that the computer can just follow like in two player chess. The computer player will also need to be able to analyze the board holistically, such as discovering which player is under attack, and attacking a player who is weak to eliminate opponentws in the game.
- Additional rules such as two players being check at the same time, keeping score in four player chess would also be different than that of two player chess, and these new features/rules would have to be implemented.