CS 246 Chess Project Planning

— Bensen Xiao, Leo Hu, Alick Wu

Working Procedure Outline:

After discussing thoroughly, all members agreed to accomplish the project in the following order:

1) **UML Designing**
    a) Discussed overall system architecture and finalized the UML diagrams for each major component.
2) **Difficulty Analysis**
    a) Identified technical challenges in implementing each class.
3) **Job Assignment**
    a) Mapped individual strengths to tasks; assigned coding, documentation, and testing roles for maximum efficiency.
4) **Collaboration Platform**
    a) Choose GitHub + Google Docs as our primary repo, communication, and issue-tracking platform.
5) **Code Implementation**
    a) Set each member's deliverable classes and due dates; agreed on PR review process and code style guidelines.
6) **Debugging**
    a) Combine and compare each member's work, locate
7) **Demo/Presentation Prep**
    a) Outlined slide deck structure; assigned sections/topics to each presenter; set rehearsal schedule.

<u>Step 1:</u> UML Diagram / Planning Stage

– This step will be completed together by all members, estimated completion date: Jul. 16th

We thoroughly discussed basic implementation ideas and the project workflow, and decided to construct a UML diagram first. In this stage, we:

a. Figured out the number and functionalities of each class
b. The dependency/inheritance relationship between these classes
c. What should the fields be in each class? Should they be public or private?
d. Got a rough idea of how the public methods can be implemented
e. Assigned coding sections to each member

<u>Step 2:</u> Difficulty Analysis

– This step will be completed together by all members, estimated completion date: Jul. 17th

1. Bot: All levels of bot should be implemented by the same person, since they share the interfaces. It would be a challenging task to implement a level 4 bot
2. Chess: All chess should be implemented by the same person, since they share the same interfaces. Especially take care of the situation of whether a move is a valid move when it is in checkmate.
3. Text/GraphDisplay: they used the Observer pattern to monitor the info of Squares, and need to correctly handle all states of Squares
4. Board: It is the core of the program, so we need to ensure correctness much earlier than others. move() is the major method, so put more effort into dealing with different situations using this method.

<u>Step 3:</u> Job Assignment

– Individual responsibilities are illustrated below, estimated completion date: Jul. 17th

Benson's job: Bot (and subclasses) / TextDisplay / GraphDisplay
Leo's job: IO (main.cc) / Board / Square
Alick's job: Chess (and subclasses)

<u>Step 4:</u> Collaboration Platform Selection

– This step will be completed together by all members, estimated completion date: Jul. 17th

a. Utilized git via GitHub to implement version control among group members throughout the whole project. It allows for distributed branching, pull requests, and issue tracking throughout the project.
b. Utilized Google Docs for planning, kickoff discussions, and real-time information sharing. Its live‑collaboration features and flexible formatting made synchronous teamwork both efficient and convenient.

<u>Step 5:</u> Code Implementation
– Each member works individually on their assigned sections, estimated completion date: Jul. 23rd

a. Upon the completion of all header files (*.h), all members gather and ensure the correctness and completeness.
b. The program shall use #include to accommodate the X11 module for the graphic display function.
c. Ensure clear and understandable documentation (both in-line comments and necessary documentation files for further explanation).

<u>Step 6:</u> Debugging
– This is an individual and collaborative task, estimated completion date: Jul. 25th

a. Write unit tests to examine whether each class/module is correct.
b. Write overall tests to ensure the robustness and correctness of the program. Gather all members to tackle the detected bugs.

<u>Step 7:</u> Demo/Presentation Preparation
– This is a collaborative task, estimated completion date: After Jul. 25th

To deliver a clear and professional presentation, our team decided to divide it into five parts: an introduction, three sections on implementation, and a conclusion. The introduction will cover the overall functionality of the project, the core algorithms and class structures used, and a brief overview of each team member's contribution. Each of the three implementation sections will be presented individually by the respective team member. The team members will explain the reasoning behind the class designs, the specific fields, methods, and algorithms implemented, and the role of each class to achieve the project properly. Presenters will also address the technical challenges they have encountered, how those were resolved, and any modifications made to the original design with their justifications. The conclusion will be delivered by all team members, summarizing the knowledge gained, the impact of the project, and potential future improvements to enhance functionality and performance.

Answers to the Questions:

1. **Q: How to apply the book of standard opening move sequences to the chess game?**
   A: Record every completed game by writing only the sequence of commands (no board graphics) to a separate "opening book" file. Summarize the data collected and find the best first move by the highest winning rate, calculated by:
   [number of games won with a particular first move divided by the total number of games that used that first move]
   In order to gain a decent amount of data samples, we can run and record the "computer-to-computer" game a number of times to get a large number of samples in a short period of time.

2. **Q: How to allow the players to undo their last moves? Even an unlimited number of undos?**
   A: To implement an "undo last move" feature, the most straightforward way is to keep a copy of the board's previous state (e.g., store the entire board vector before each move). Calling undo simply restores that saved vector. However, the shortcoming of this method is that when supporting multiple undos, it quickly occupies a lot of memory.

   A more efficient approach is to record each move and any piece captured as a command (for example, "move e2 e4"). To undo, you execute the inverse of the last command (e.g., "move e4 e2") and add back the captured piece, if any. This lets you support unlimited undos with minimal storage—just the list of move commands. As a bonus, you can save those commands to a file and analyze them later (for example, to build an opening book with win-rate statistics).

3. **Q: How to accomplish the four-handed chess option in the original chess game? What changes would be necessary to be implemented?**
   A:
   For the free-for-all version:
   Firstly, we have to change our board, since common four-handed chess has a special shape (i.e., pieces of the additional two players are placed on two sides of the board, and the board would not be a standard square). Secondly, we need to maintain two new colors and change the sequence of play to fit this rule. Thirdly, when checkmate, the game should not end. Instead, the end of the game only happens in the situation where checkmate happens and only two kings survive. Additionally, we have to take care of the situation when two or more kings are in check.
   For the 2v2 version:
   We still need to maintain two more colors, but the behaviour of these colors is different from the free-for-all version, where the four colours are grouped into two allied pairs. We still need a new board with the adjusted shape, and the sequence of moves should also be taken care of, especially on whether members of the same team should play continuously or should two teams take turns. (In this case, say team 1 has members 'A' and 'a'; team 2 has members 'B' and 'b', we have to decide if the sequence of moves is "ABab", "AaBb", or "ABba", since they all can be argued to be unfair). The game should end when the last king of a team is checkmated, which is a new feature we have to accommodate.