# CS 246 Final Project Plan of Attack

Markus Rieg (mrieg)
William Kennedy (wkennedy)

Our final project choice is Chess.

**Project Breakdown:**

We will be working towards creating the following core components and mechanisms of chess. This is planned to be done in the order listed, by the partner delegated, and before the specified deadline.

Chessboard
The chessboard includes not only the board, but also all basic parts of the game without the gameplay. This includes pieces, their colours, coordinates, and starting locations. Moreover, the basic structure of hierarchy for inheritance and abstracting pieces and their attributes will be produced in relevant classes, including `Board`, `Piece`, and all subclasses thereof.
We hope to finish these basic elements of the chessboard by Saturday, July 18th, delegated approximately equally between Markus and William.

Text-Based UI Output
The text-based UI is done at this stage to verify that the chessboard is set up correctly, along with all future components. Relevant classes include the `AbstractUI` and `TextUI` classes, which are notified by the board, along with relevant functions such as `output` using `cout`.
We hope to finish the text-based UI output for the board by Saturday, July 18th, delegated approximately equally between Markus and William.

Input
The input allows the ability for control of the game outside what is hard-coded. Namely, we need to be able to interpret input depending on the state of the program (in-game, not in-game) and run/display a result appropriately. Relevant mechanisms include the outline of the main game loop and `input` using `cin`.
We hope to finish having different program states and being able to interpret input depending on them by Saturday, July 18th, delegated approximately equally between Markus and William.

Simple Valid Moves
Simple valid moves are the basic moves that each piece could make on any turn. We need to ensure the moves are valid, that is, not moving to an occupied square or one outside the board. Additionally, we need the ability to detect check, checkmate, and stalemate to restrict which pieces can be moved. This involves a lot of condition checking in `Board`.
We hope to finish the ability to perform basic moves by Saturday, July 18th, delegated approximately equally between Markus and William.

Game

The game includes the ability to start a game with the user playing both sides, the ability to resign, offer draws, keep track of all moves played, and keep track of the running score from wins, losses, and draws. This milestone includes the game cycle, in which different coloured pieces must alternate moves, as well as cycling through the program states for starting a game rather than setting up the board in the future. Relevant classes include `GameData` and `MoveInfo`.
We hope to finish the game cycle by Sunday, July 19th, delegated approximately equally between Markus and William.

Special Moves

Special moves include those such as castling, en passant, promotion, and three-fold repetitions that require saved context to be valid. This also includes the ability to undo moves an infinite number of times and restore the board to an original position, keeping track of these abilities.
We hope to implement the ability to play such special moves by Sunday, July 19th, delegated approximately equally between Markus and William.

Setup Mode

Setup mode is the ability to start the board from a different position and play the game forward from there, as described by the handout. This includes who is next to play, and adding/removing pieces from the board before a game has started.
We hope to implement the ability to play such special moves by Sunday, July 19th, delegated approximately equally between Markus and William.

Computer Players

There are three levels of computer players to be implemented at this stage: the random legal moves, the pro-capturing computer, and the anti-being-captured computer. All of these computers have minimal lookahead, if any, and thus will not have significant algorithmic complexity. Relevant classes include `AbstractBot` and its subclasses.
We hope to finish the computer players by Thursday, July 24th, delegated approximately equally between Markus and William.

Graphical User Interface

The graphical user interface is the representation of the text-based UI presented using the built-in vector graphics. This may be challenging as there is very limited prior experience with graphics in this course, many of which use different module formats than what we are used to. Relevant classes include `GUI`.
We hope to implement the graphical user interface by Thursday, July 24th, delegated approximately equally between Markus and William.

Good Computer Players

A highly skilled computer player that considers the results of its moves at multiple depths, assuming the same level of player, thus iteratively minmaxing the overall centipawn score. This requires a lot of recursive breadth-first search within a time limit, and performing current evaluation.

We hope to implement the ability to have a skilled computer by Thursday, July 24th, delegated approximately equally between Markus and William.

**Design**

There are 6 major components to the chess program. The board, the pieces, the text UI, the GUI, the move information, and the computer.
The text UI and GUI inherit from the AbstractUI class, which is a BoardObserver. Both the UIs have a GameData, while AbstractUI and Bot both have a Board. MoveInfo and Board have many Pieces. The Pawn, Knight, Bishop, Rook, and Queen are Pieces.

**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. See for example https://www.chess.com/explorer which lists starting moves, possible responses, and historic win/draw/loss percentages. Although you are not required to support this, discuss how you would implement a book of standard openings if required.*

A book of standard openings could be implemented in different ways, depending on how the game is being displayed. In a text-based UI, a list of the most common opening moves in algebraic notation can be provided by typing in the command line, say `openingbook`. Alternatively, this information can be provided on the side of the ASCII chessboard, toggled on with a setting (along with or replacing the move history).
A panel with a list of opening moves can also be implemented in a graphical user interface, much similar to how it would be represented using a text UI.

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

The ability to undo is a planned feature of our game. It is standard in most chess programs to keep a complete history of every move made, displayed using algebraic notation; this feature can be further built upon to store the initial location, destination, and piece captured so that any move can be undone without recalculating the board state from the start of the history. Such information is stored in a vector, which increases in length every time a move is made. This allows for infinite undos.

***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. (If it's important to your answer, state whether you're assuming free-for-all or team rules and then answer the question. You don't need to get too specific into the rule set changes in answering the question though; your focus should be more on what would need to be altered at the high level of the design)*

Changing the program to allow for chess variants such as four-handed chess would require a few modifications. For instance, the size of the board is much larger and has spaces where pieces cannot move or check through (the corners of the square). The order of play would also need to be modified to account for the additional 2 players that would be added, and the colours of the new pieces. Moreover, more must be done when it comes to tracking move history and valid moves. However, some aspects of the variant are easier to manage, such as calculating king-capture rather than checkmate. Ultimately, one of the more complex parts of the program would be tracking which variant of chess is being played and what rules to apply.