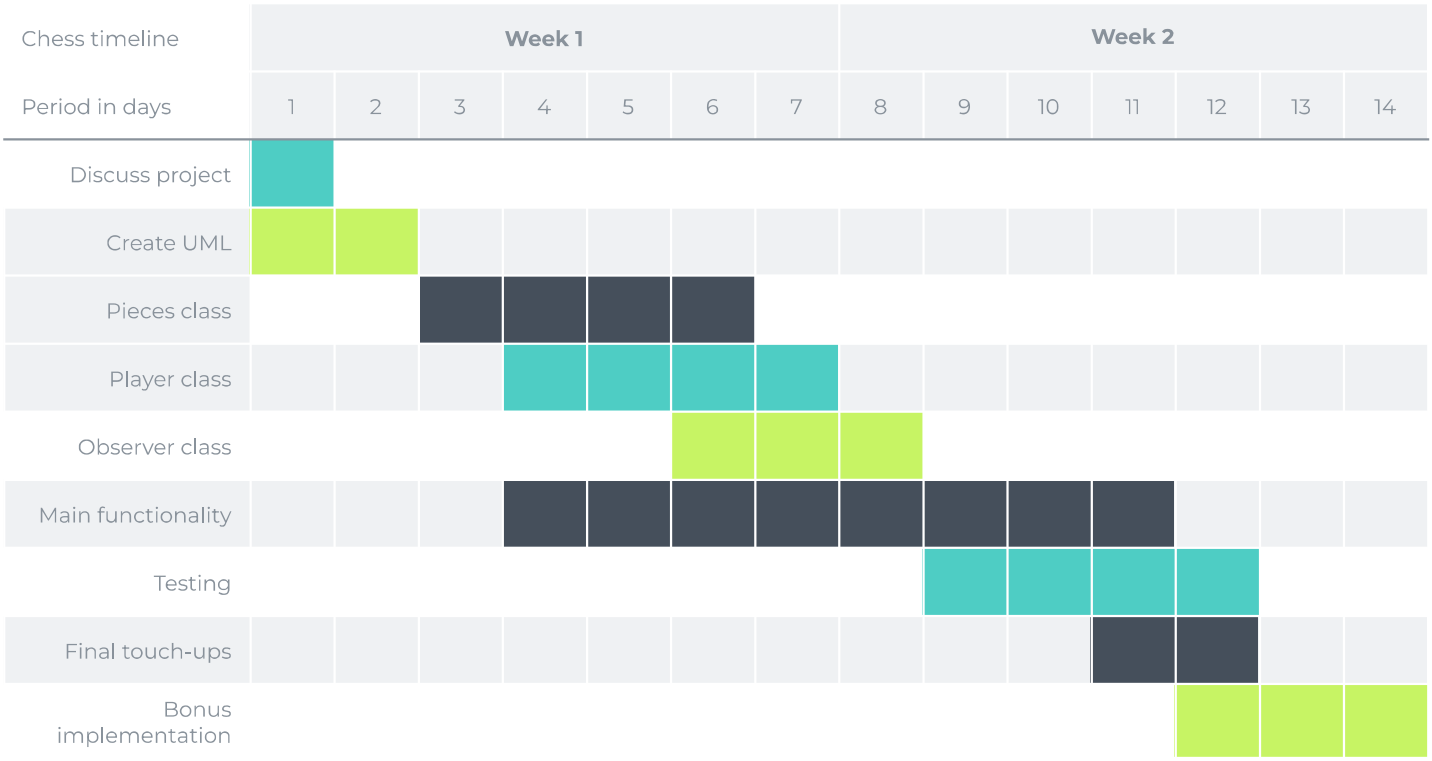
Plan of Attack:

This project can be broken into 4 general pieces:

- The Game class and main game functionality (moving pieces, setting up the board, etc.)
    - Ben Hudson will be responsible for this aspect of the project since he will also be implementing the Piece class and its subclasses; it allows for fewer communication errors between the two elements of code.
    - Given this is a critical part of the code base, it will be handled by the group's strongest programmer with support from the other members if required.
        - Having a single group member complete this has the advantage of reducing the likelihood of implementation inconsistency
    - Deadline: December 3rd
- The Piece class and its subclasses which handle moving pieces and track their location on the board.
    - Ben Hudson will also be mainly responsible for this element but the other group members will contribute to this as well.
    - Deadline: December 29th
- The Observer class and its subclasses to handle displaying the game on text or graphically.
    - Bhavya Modi will be responsible for this aspect of the project. This element of the project will directly interact with the chess class.
    - Deadline: December 1st
- The player class and its subclass which holds the command interpreter (in the human player case) and the computer-generated move (in the computer player case).
    - Pavitar Notey will be responsible for this aspect of the project. This element of the project will directly interact with the chess class.
    - Deadline: November 30th
- All group members will help each other if needed. These responsibilities are not absolute.
- The final document will be prepared equally by each group member
    - the exact roles are subject to change since they will largely depend on who implements what features which cannot be determined till after the implementation is complete

We have designed our modules following the MVC structure, ensuring any particular aspect of the game can change without affecting the classes outside of those directly implicated by the change. The Controller is the player class, the model is the GameManager and Piece classes and the viewer is the Observer class.

Timeline:



Questions and Answers:

Q: Discuss how you would implement a book of standard openings if required.
A: To store a single move, we would use a pair data structure, where the first item represents a piece to move and the second represents the tile to move it to. Then, storing these in an array would produce a list of standard moves. We can then store an array of these standard moves to get a book of standard moves (sorted by effectiveness). This can then be implemented in the Computer player class with the level of difficulty corresponding to a single set of standard moves in our array of standard moves. Additionally, if we wished to use the book of moves as a recommendation for the player, we could add a command "help" that recommends a move from the book - of course, this is only effective if the player has made previous moves (somewhat) similar to what is in the book.

Q: How would you implement a feature that would allow a player to undo their last move?
A: To implement an "undo" feature, each move that is made will be stored in a stack data structure. Once the "undo" command is invoked the principal element in the move-stack will be read, the opposite of it performed (to undo the move), and then popped off the moving stack. This requires adding an *undoMove* function in the piece class since not all undo moves will be valid (e.g. a pawn cannot move backward) so they should be handled separately. Additionally, any taken pieces will be returned to life, which is as easy as setting their isAlive field back to

true. In this, we can support an unlimited amount of undo move commands, until the stack is empty and the game is in its default state - in which case the notion of undoing a move no longer applies.

Q: Outline the changes that would be necessary to make your program into a four-handed chess game.
A: The four-handed chess board can be represented as a 14x14 board - this modification will be easily done by modifying the dimension fields in the Board class. Additionally, the 3x3 corners of the board are off-limits and so the individual pieces must have their *validMove* functions modified so that they cannot move into these corners. Lastly, the gameplay logic will remain the same except the Game class will have to check extra cases for the check, checkmate, and stalemate since there are additional players to account for.