Plan of Attack

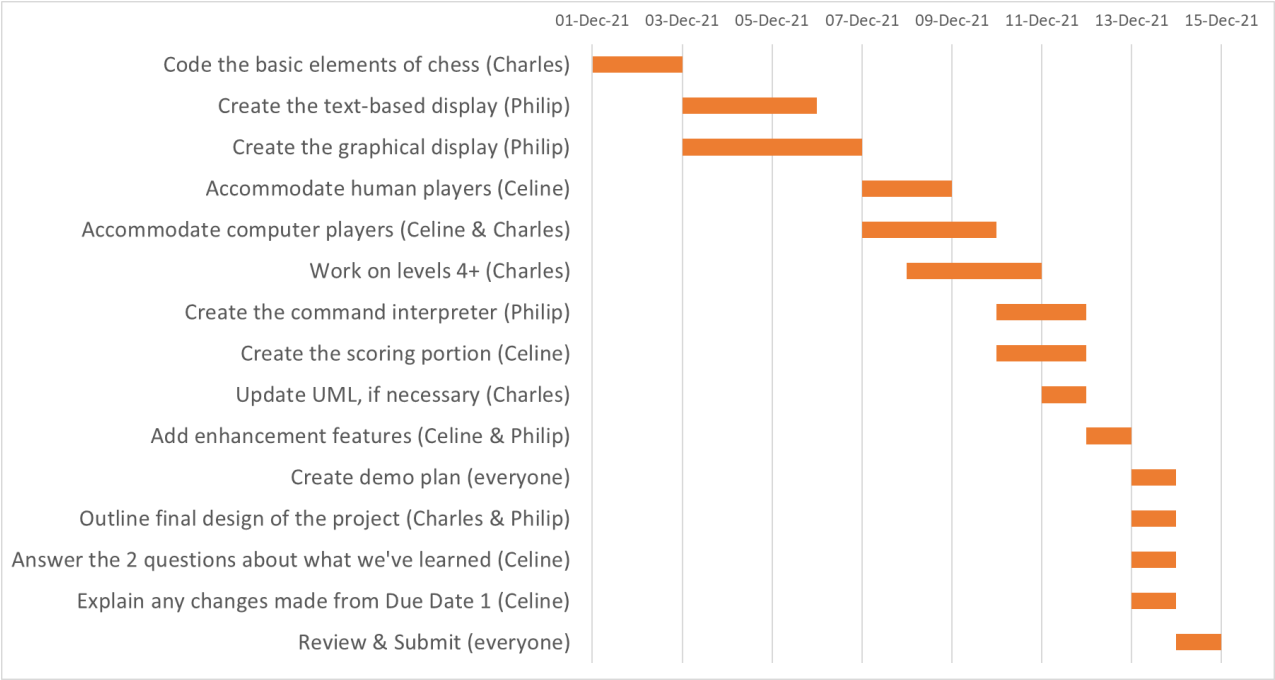
Group Meeting Dates:

1. **Thursday, December 2 at 2PM**
2. **Wednesday, December 8 at 12PM**
3. **Friday, December 10 at 9PM**
4. **Sunday, December 12 at 9PM**
5. **Tuesday, December 14 at 12PM**
6. **Wednesday, December 15 at 9AM**

Sub Tasks & Due Dates:

1. **Friday, December 3, 2021:** Code the basic elements of chess (Charles)
2. **Monday, December 6, 2021:** Create the text-based display (Philip)
3. **Tuesday, December 7, 2021:** Create the graphical display (Philip)
4. **Thursday, December 9, 2021:** Accommodate human players (Celine)
5. **Friday, December 10, 2021:** Accommodate computer players (Celine & Charles)
6. **Saturday, December 11, 2021:** Work on levels 4+ (Charles)
7. **Sunday, December 12, 2021:** Create the command interpreter (Philip)
8. **Sunday, December 12, 2021:** Create the scoring portion (Celine)
9. **Sunday, December 12, 2021:** Update UML, if necessary (Charles)
10. **Monday, December 13, 2021:** Add enhancement features (Celine & Philip)
11. **Tuesday, December 14, 2021:** Create demo plan (everyone)
12. **Tuesday, December 14, 2021:** Outline final design of the project for the design document (Charles & Philip)
13. **Tuesday, December 14, 2021:** Answer the 2 questions about what we've learned (everyone)
14. **Tuesday, December 14, 2021:** Explain any changes made from Due Date 1 (Celine)
15. **Wednesday, December 15, 2021:** Review & submit (everyone)

*All due dates are by 11:59 PM



Question 1:

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.

We'll first create an StandardOpeningTree class where the nodes have a string representing the moves made in the standard opening. The root of this tree will be the first move of the standard opening and each response for a given move will be represented as a child node. Therefore, each layer of the tree represents a turn in the game.

The book of standard opening will then be a map with the name of the opening as the key, and a StandardOpeningTree object pointer as the value. We will then traverse the tree depending on what we want to accomplish.

For example, if the player wants the computer to use a specific opening, then our program would access the StandardOpeningTree object. The computer will traverse then traverse through the tree and perform the move commands using the values returned from the nodes.

Question 2:

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

In our program, the Board class is responsible for knowing the state of the game, which is essentially the state of the board. So, it should also keep track of all the moves made during the game, which can be tracked in a stack of strings. Each move will be represented as a string in the the form of:
"[*initial_position (ex: e4)*]-[*final_position (ex: e7)*]
-[*chess_piece_taken, or null if no piece was taken*]".

Since we have to implement a move command (move e2 e4), we will have a function moving the piece from *initial_position* to *final_position*. Therefore, we can undo a move by popping a string out of the stack, and converting the popped string into a stream, splitting it at '-'. Afterwards we just have to run the function used to implement the move command with the positions switched (the final position as the first parameter and the initial position as the second parameter). If there was a piece taken, we simply add the taken piece back to the board at *final_position*. This can be repeated until the stack is empty.

Question 3:

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.

We first have to change:

1. Enable player array to also accept 4 values upon game initialization
2. Increase colour enumeration to account for 2 more players. Since our piece's state only accounts for colours and not which player it belongs to, each colour is indirectly tied to a player.
3. Our Board class uses a Map with strings (coordinates) as keys and pointers to Square Objects to keep track of the shape of the chess board. Each square on the chess board will be represented by a key, value pair in the map. To make our program into a four-handed chess game, we would have to change the map, so that each square in the four-player chess board will have a key, value pair in our map.
4. We wouldn't have to change much for the code that is responsible for checking if a move is valid or not because 4-player chess follows all the rules of normal chess in terms of valid moves. Therefore, if our original code can already determine if a move is valid or not, then it should work for the new program. The different board shape shouldn't matter because we use a map to keep track of the board's shape, so we just check if there is an existing key, value pair for a square's coordinate to see if the square exists or not.

5. Add a direction field to the Piece class since valid moves will be calculated differently depending if the player is facing up/down vs left/right