# Progress Report

**Layers completed:** One and a half layers have been completed.

**The Main Program (Main Driver):**
The main cpp file is called *CheckerGame.cpp*. It contains the main function. It is also where all other function calls are performed to get the game functional.

It has the following variables:

```
// Variables
int mode;
    Player player1("Player 1");
Player player2("Player 2");
//AI player2;
    bool player1WinCheck = false, player2WinCheck = false;
Board board;
```

- int mode
- Player player1(“Player 1”)
- Player player2(“Player 2”)
- bool player1WinCheck
- bool player2WinCheck
- Board board;

**Player object and how it works:**
The player object is an object that is created for the player to keep track of all of their pieces and to check if they have won or not.

It has the following variables:

```
private:

    // Variables for player
    bool turn;
    bool win;
        string id;
        vector<Piece> playerPieces;
```

- bool turn: This variable is a bool variable that is constantly being updated between true or false in order to determine if it’s player 1’s turn or player 2’s turn.

- bool win: This variable is a bool variable that is constantly being checked between players' turns in order to determine if player 1 has won or player 2 has won.
- string id: This variable is a string variable that is set when a player object is created to determine if the object is player 1 or player 2.
- vector<Piece> playerPieces: This vector of objects piece holds the 12 piece objects that each player has along with their associated values and variables.

It also has the following functions:

```
// Player class
class Player {
public:

        // Class constructor
        Player();
        Player(string ids);

        // Class destructor
        ~Player();

        // Player functions
        Board userInput(Board board);
        void playerTurn();
        bool playerWin();
        string playerID();
        void createPieces(bool playerOrder);
        int charToInt(char columnChar);
```

- Board userInput(Board board): This function is of type Board that returns a Board object and accepts a board object as a parameter. This is the main function in the player object which it calls all other functions. It is responsible for printing out the messages for the user to follow. It then calls the corresponding functions to update the pieces and board respectively, and then returns the board object.
- void playerTurn(): It alternates the *bool turn* value of the object from true to false and vice versa.
- bool playerWin(): Returns the *bool win* value that is either true or false.
- string playerID(): Returns the *string id* value which distinguishes the players.

- void createPieces(bool playerOrder): This function accepts a bool value of true or false as a parameter to determine if the player is player 1 or player 2. The function calls the *playerID* function and saves it to a string. It then creates the 12 piece objects and stores them all in the *playerPieces* vector for each respective player.

- int charToInt(char columnChar): This function accepts a char variable as a parameter. It converts any char into its upper char counterpart from A to H using the *toupper* function

**Piece object and how it works:**
The piece object is an object that is created for the piece. It keeps the data for the piece and marks the piece for Player One or Player Two.

<u>It has the following variables:</u>

```
private:
    int id;
    string label;
    int row;
    int column;
};
```

- int id: It is an integer variable that keeps the id of the piece.
- string label:  It is a string variable that marks the piece as Player One or Player Two.
- int row: It is an integer variable that keeps the row of the piece.
- int column: It is an integer that keeps the column of the piece.

<u>It has the following functions:</u>

```
class Piece {
public:
    int returnID(int row, int column, vector<Piece> pieces);
    vector<Piece> updatePiece(int row, int column, int id, vector<Piece> pieces);
    int returnRow(int pieceID, vector<Piece> pieces);
    int returnColumn(int pieceID, vector<Piece> pieces);
    string returnLabel(int pieceID, vector<Piece> pieces);
```

- int returnID (int row, int column, vector<Piece> pieces): This function takes a row, a column, and a vector of Piece objects as arguments. It iterates through the vector and returns the ID of the Piece that matches the specified row and column.

- vector<Piece> updatePiece(int row, int column, int id, vector<Piece> pieces): This function updates the position (row and column) of a Piece with a specific ID. It takes the new row, column, the ID of the piece to be updated, and a vector of Piece objects. It iterates through the vector, finds the piece with the specified id, and updates its position. It returns the updated vector of Piece objects.

- int returnRow(int pieceID, vector<Piece> pieces): This function takes a pieceID and a vector of Piece objects and returns the row of the Piece with the specified pieceID.

- int returnColumn(int pieceID, vector<Piece> pieces): This function takes a pieceID and a vector of Piece objects and returns the column of the Piece with the specified pieceID.

- string returnLabel(int pieceID, vector<Piece> pieces): This function takes a pieceID and a vector of Piece objects and returns the label of the Piece with the specified pieceID.

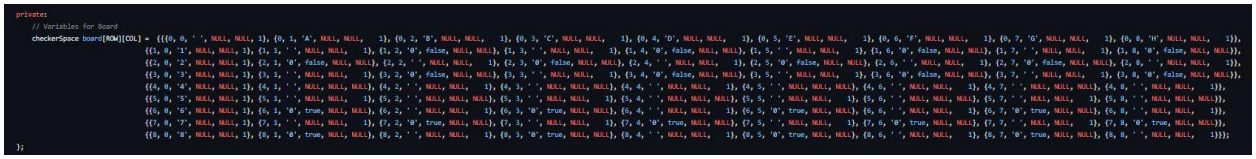
**Board object and how it works:**
The board object is an object that holds the structure of the board along with the ability to update it constantly.

<u>It has the following variables:</u>

- struct checkerSpace: This struct includes the following variables and holds all the information of a space on the board.
    - int row: An int value that represents the row of the space.
    - int column: An int value that represents the column of the space.
    - char value: A char value of the space that is printed out to the command console.
    - bool player: A bool value representing which player's piece occupies the space.
    - bool status: A bool value that represents whether the space is occupied.
    - bool nullSpace: A bool value that differentiates the nonplay spaces and the playable spaces.

```
struct checkerSpace {
    int row;
    int column;
    char value; //this is displayed on the board
    bool player; //which player's piece is on the space (true = player 1, false = player 2)
    bool status; //taken = 1 or empty = 0
    bool nullSpace; //grey spaces (nonplay zones)
};
```

- checkerSpace board[ROW][COL]: A 9 by 9 two-dimensional array of the struct checkerSpace. This initializes the board and the multiple values of each space.



It has the following functions:

```
    // Board functions
    void printBoard();
void updateBoard(int row1, int column1, int row2, int column2, string label);
    void positionCheck();
    void removePiece(int r, int c);
```

- void printBoard(): This function prints out the board and the char values of the spaces on the board. The user would never have to directly interact with this function.

- void updateBoard(int row1, int column1, int row2, int column2, string label): This function updates the position of pieces on the board. It's a given the previous location, the new location, and string to determine which player's piece is being updated. The user would never have to directly interact with this function.

- void removePiece(int r, int c): This function removes a piece from the board by accessing the location with the given parameters. The user would never have to directly interact with this function.

**What has proved to be harder (or easier):**

- We encountered several challenges and design revisions. One of the significant aspects that was harder than expected was the Piece related information. In phase one of our project, we focused on implementing the move function for moving the piece. However, as the development process progressed we realized that additional functionalities were required. To address these challenges we implemented the following new functions:  vector<Piece> Piece::updatePiece, Piece::returnID, int Piece::returnRow, int Piece::returnColumn and string Piece::returnLabel.
- We also came across the issue of the checkerboard looking too plain and the spaces being indistinguishable from one another. This required us to search for a method to colorize the board and make it look more presentable.
- Another challenge we encountered was carrying the different data into different files and making sure it was updated correctly in the main function to reflect those changes