19348276 - COMP10050 - Assignment 2

Part 1 - Creating main components of game and initialisation

Creating Components:

I edited the player struct by adding variables for all the relevant information related to the player(e.g. name, colour, no. reserved pieces, etc.).

I edited the print board function to print axes along the edges of the board to help the players know the coordinates of the squares, I also edited the function so that along with printing "R" or "G" depending on the colour of the top piece on the square, it would print in brackets beside this, the size of the stack.

Initialisation:

I declared and initialised(to zero) two global int variables, one to keep track of how many squares have a red top piece and one for how many squares have a green top piece. These variables are incremented and decremented throughout the program and are used to help determine when a player wins.

The initialisePlayers function saves the player's colour, name, current no. pieces captured and current no. pieces saved/reserved to the player struct array. The initialiseBoard function sets the appropriate values to each square of the 2D board array(e.g. valid/invalid, no. pieces, the pointer to the piece on square, the colour of the piece, etc), in the functions that initialise the colour of the pieces on the square the appropriate global variable that holds the total number of squares with that colour is incremented.

Part 2 - The main logic of the game

I decided to have a number associated with each player/colour that I would use throughout the program to differentiate between the players, 0 for Red and 1 for Green.

I created an int variable called round(initialised to zero) that increments after each turn, the result of this variable modulo 2 determines which players' turn it is. After each turn the board and the respective numbers of squares with either red or green top pieces are printed, the variable that determines if there is a winner or not is given a value and the round is incremented.

Turn Function:

The turn function begins by asking the player to enter an int value depending on their choice, moving a piece already on the board or placing one of their reserved pieces on the board. This choice then leads to a switch statement that ensures the validity of their choice through the use of if statements or a while loop with appropriate parameters(e.g. Coordinates given are in the range of the board, the square is valid, same colour,etc.).

After a value choice is determined, the function moves on to a second switch statement which in a similar way to the first switch statement determines that the destination square of the piece/stack is valid. The destination coordinates are calculated by looping through a user inputted string character by character and incrementing/decrementing the destination row and

column variables accordingly depending on the letters in the string (u(up,^), d(down,v), l(left,<), r(right,>)). Loops until the end of the string but stops incrementing/decrementing the coordinate variables once the amount of moves made is equal to the size of the stack.
Once the destination is determined and validated, a function that handles the movement of pieces onto squares or stacks is called. Each stack is a linked list. The function called depends on the player's choice but both functions work very similarly. They ensure that all the variables that reflect the movement are changed appropriately(e.g. no. pieces on the square or numGreenSquares) and that the piece/stack is linked/joined to the square/stack.

Part 3 - Maintaining stack size equal to five

Both the functions that are used to move the piece/stack to another square/stack have an if statement that checks if the stack size is greater than five, if it is they call the function removeGrFive, this function keeps that stack size equal to five
removeGrFive:
This function receives as part of its input the pointer to the fifth element(from the top) of the stack. It then sets the current piece to the next(6th) element of the stack and enters a loop where it moves through the rest of the stack until it reaches the null pointer. On each iteration it checks the colour of the piece, and either increments the player's counter of reserved pieces or increments their counter of pieces captured, depending on that colour. Once the loop finishes it sets the piece pointed to by the fifth piece to NULL.

Part 4 - Winning Condition

Loser is the name of the variable used to determine if a winner has been found. If loser=0, green wins, if loser=1, red wins and if loser=2, no winner has been found.
Winner is the name of the function that determines the value of loser, it first checks if any player has zero reserved pieces, once it finds a player for which this is true it breaks from the for loop. It then enters a switch statement, and checks if this player also has zero pieces of their colour on the board(using either the numRedSquares or numGreenSquares variable). If this is not true the function will return 2, if this is true the function returns either 0 or 1, depending on the losing player.

The main body of the function in main.c occurs in a while loop, this loop occurs as long as loser != 0 && loser !=1(indicating no winner has been found), once this is not true, the loop ends and a message, indicating the winner, their colour and the amount of pieces they captured is printed.