Assignment 2 Brief Description – Bomte Emuchay – 1945954

Link : <https://github.com/ebomte/COMP10050A>

The main purpose of this project is to play the Focus/Domination board game. In my project I have four source files accompanied with their header file. Then I have the main function which calls the functions in the source file.

In the “**input\_output**” source file, I printed out the board and players when the game starts by looping through its respective arrays. Then I started the game. Inside the “managePlayers” function, I asked the user the co-ordinates they want to move a piece from using the variables “userRow” and “columnRow” to hold the row and column of the square they want to move a piece from. Then I checked if the top piece of the square is the same colour as theirs, and if it was, then they were allowed to make the move. This was then by calling the “moveDirection” function.

Below the “managePlayers” function, I created a function called “findWinner”. This main purpose of this function was to create a condition that would allow the players array to continuously loop until a player won the game. To do, I created counters for squares with pieces on them, and counters for red and green pieces. The logic of the is function was, each time the loop got to a square with a piece on it, I would increment the square counter. Then I would check if the top colour of the current square is either green or red, then I would increment the counter of the colour that matched by 1. Once the board array has looped through, I checked which colour counter had the same value as the square value. This meant that every top piece on a square that had a piece was either red or green. Whichever colour counter matched, the player that owned that colour was the winner.

Below the “findWinner” function, I created a “moveDirection” function. This main purpose of this function is to determine the location the user wants to move (left, right, up or down), then determine if the squares in that direction is valid. Then I create variables that calculate the movements based on the number of pieces on the square. I do this because a user can only move stacks to squares based on the number of pieces on the square/in the stack. Then I have a switch statement which check to see of if the user's choice correlate with the calculations I did above to determine if the movement is a valid movement based on the number of pieces in the stack/on the square. If their movement is valid then their corresponding moving function is called, either moveItemsLeft, moveItemsRight, moveItemUp or moveItemsDown. Once all the information in the moving functions are done, then the board and players are printer again to see the changes on the board and to see if any player captured or reserved a piece.

In the **“stackInit”** source file, I have four functions that all move either left, right up or down. In each function, I loop through the stacks until it gets to the last piece. Since the last piece of the stack. next is already NULL, when making a move, I set the last piece.next to equal the top piece of the stack I am moving to. Then I set the top piece of the square I move to, to equal the top piece from the square I moved from. Then I set the stack I moved from to NULL. Once I make a move, I check to see if the number of pieces on the square moved to is greater than five, because we only have a max of 5 pieces in a single stack. If it is greater than five, it calls it respective function, either, “removeLeft”, “removeRight”, “removeUp” or “removeDown”.

In the **“stackEnd”** source file, I have four functions that all remove pieces from the bottom when the user movies either left, right, up or down. In each function I create a counter and set it to 1. Then I create a while that will loop until the piece \*next is not NULL. This continues until the counter reaches five. Once it reaches five, I store the contents of curr->next in a pointer called last and set curr->next to NULL, in other word setting it to nothing. Then I set the number of pieces on the current square to five. Then I get the colour of the pieces that were popped. I check to see if any colour is the same as the current players colour or their opponents’ colour. If it is their colour, I increment “numberOfReservedTokens” by 1 and if it not, then I increment “numberOfOpponentsTokensCaptured” by 1. If it is their colour, I as the user to select one of the squares outside the main game parameters to place the reserved piece for later use. Then I move the piece onto that square updating the “num\_piece” and the top piece of that square moved to. All of this is done inside a while loop. The while loop, loops through each piece popped individually.

Finally, the **“startGame”** source file initialises the players and the board. It also creates function that decides whether a square is valid or empty. It then has functions that set square to either green or red.