**Project 1(Chess)**

**CIS-17A-49287**

**October 25, 2020.**

**Antero Avalos**

**Table of Contents:**

1. Introduction
2. Summary
3. Description
4. References
5. Program

**Introduction:**

The game I chose to develop for this project was Chess. I realized about a week prior to the deadline that I was attempting to bite off more than I can chew. The finished product is incomplete -to say the least- but with enough work, my expectations are that it will become a worth-while final project.

The game is currently a two-player version of the old-time classic. In the future, a simple AI will be added to enable a single player to run the game without having to play against themselves.

The rules adhere to the classic chess entirely. The movement of pieces, and the winning and losing conditions are the same.

The board is printed using characters such as spaces “ ” and exes “X” due to the limited character output that NetBeans is constrained to (see figure below). The player enters the location of a the piece they want to move, then the location to which the piece is being moved to.

Calendar

Description automatically generated

Input validation exists to ensure that the user enters appropriate values for moving the pieces. Additional testing is needed to ensure that the user is unable to make the program crash with small input error, which still occurs.

**Summary:**

Using C++, the game was an attempt at utilizing as many of the concepts we have seen in class up to this point. The program is about 1200 lines of code. Several functions are used to evaluate when user input matches the allowed set of possible moves for any specific piece. I estimate the total amount of time spent on this project to be about 60-80 hours.

I struggled with keeping track of my goals during development since my approach to coding this project was constantly changing. Despite the game being incomplete I found the development of this current version very challenging. I even re-evaluated my decision to pursue a career in computer science due to the amount of time spent with such little progress made. Ultimately, I realize that I can only succeed if I continue to try…

**Description:**

Initially I programmed the board and left the pieces for later. I created my first version of the board using nothing but cout statements which was a waste of time that I would not realize until later.

I thought that making the most basic version of the board I could think of would expedite the creation of a board that was created using a 2D Array of characters, I was wrong. Overall, the first few days spent on the board problem helped me understand the importance of designing my application’s logic before moving on with the code. After fixing the issues with the board creation, I ran into some output issues that were fixed by just changing the chars I used for “black” and “white” squares on the board.

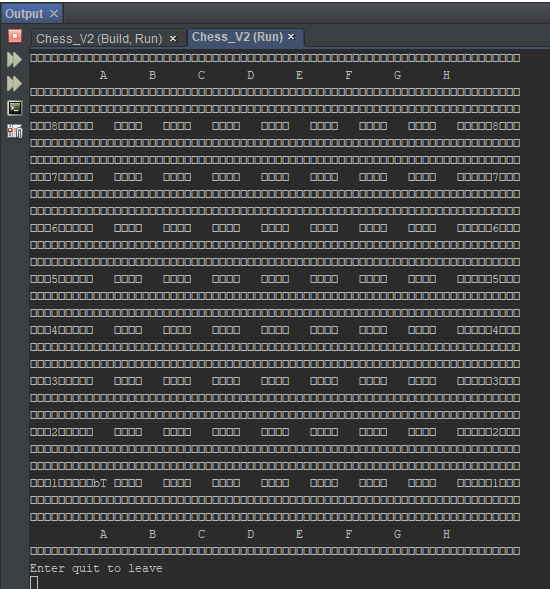
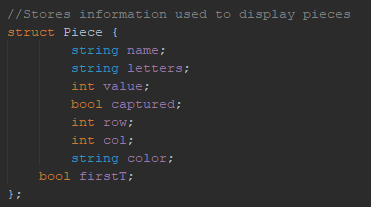
****

Figure 2. Version of the board used in submitted version of Project 1

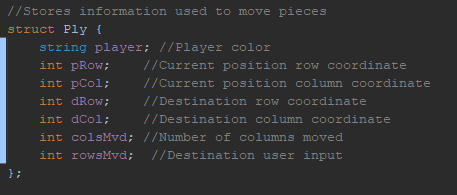
Figure 1. Initial Output, NetBeans was not displaying the chars I wanted it to... so I switched to the chars seen on Fig.2

After making the board, I added the pieces to the game. Each piece is a struct which contains the following member variables:

* **Name**(string): Used to identify the piece with a string during gameplay. I was unable to add the function that would display the piece information when selected, but this is essentially what the name member’s purpose is. It was also meant to be used to display a message when a user captured a piece, something like: “You have captured a Knight!”.
* **Letters**(string): Used to store the set of characters that are displayed on the board for each piece, such as “bP” for a black pawn and “wK” for white king, etc.
* **Value**(int): an assigned value which will be used in the final project version of the program. It is meant to aid the AI in decision making by choosing which pieces to prioritize when multiple captures are available, or when it needs to decide which piece to save.
* **Captured**(bool): Used to identify when a piece has been captured, if this is set to true, the piece will no longer be seen on the board.
* **Row**(int) & **Col**(int): these integer values serve as “coordinates” to identify a piece’s “location” on the board. They are also used for movement; when these values are modified by inputting a legal location during one’s turn, the pieces are moved using the modified values.
* **Color**(string): This was meant to be used in the identification of possible captures as well. This current version of the program does not use this member anywhere.
* **FirstT**(bool): Used to identify if the current turn is a piece’s first one or not. This boolean value is meant to facilitate moves that can only be done on a piece’s first move, such as the double square move by pawns and castling.

Another struct that was used was the Ply, which is used to get the user input for a move and modify the board accordingly. The Ply struct contains the following member variables:

* **Player**(string): used to identify which player this set of instructions corresponds to.
* **pRow**(int) & **pCol**(int): these two integers correspond to the piece that the player has chosen to move. They help find the piece in order to modify its values and move.
* **dRow**(int) & **dCol**(int): these two values are what the piece’s coordinates are changed to if the move was legal.



**Functionality Missing:**

Many features which were intended to be in this first version of the game were left for the final project, below is a list of all these features. (I know, it is pretty much the entire game, my apologies professor).

-Capturing

-Saving the game and exiting.

-Castling

-Check Mating

-Playing against AI

-Displaying help during each turn. After selection of a piece, a dialog with movement rules would appear.

Some of the features missing where meant to include concepts such as file input and output. Unfortunately, due to bad planning, they will only be present in the final project version.

**Pseudo Code for the Main function:**

//Declare variables:

-2d array of char pointers for the board

-2 arrays of structs which hold the pieces

//Enter do while loop(until check mate or exit):

//Display who’s turn it is

//Get user input for piece selection

//Get user input for where to move

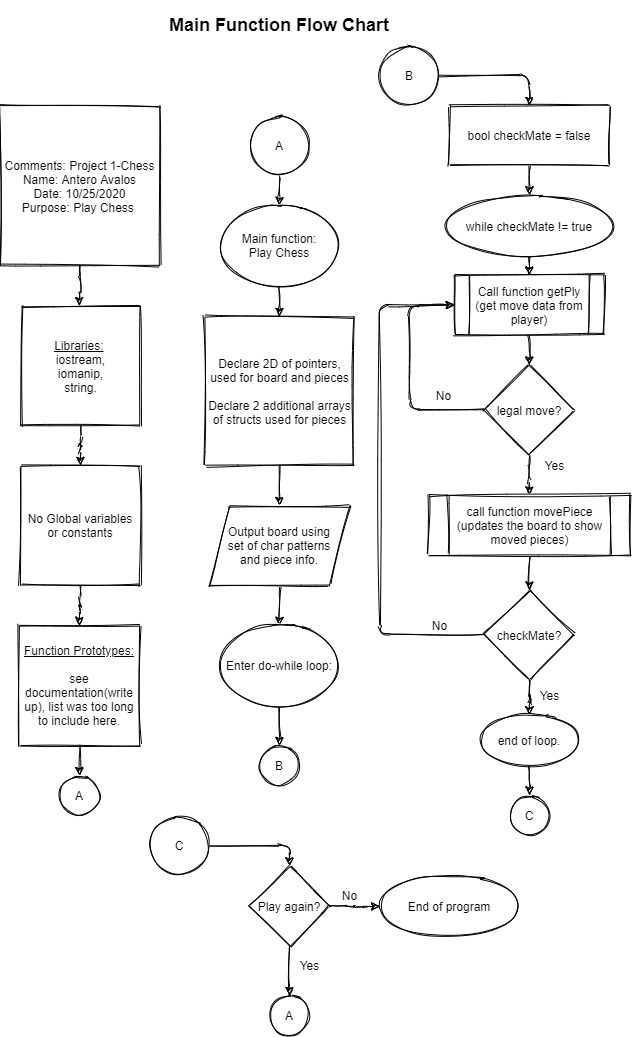
//Move the piece (movePiece function)

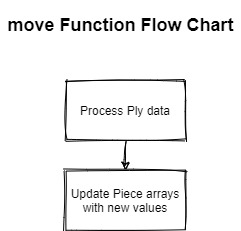
//Clear screen

//Repeat from beginning of loop for next player’s turn

//After exiting the loop, unallocate memory and close program

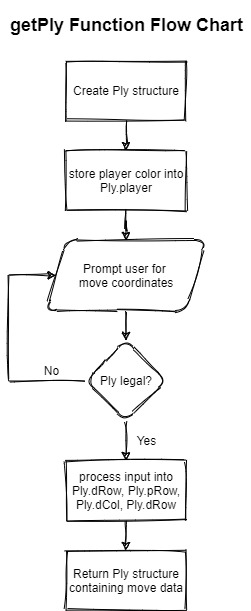
See flowchart below



**Pseudo Code for the movePiece function(moves pieces):**

//Process information from a given Ply structure

//Modify the arrays of pieces with the Ply data



**Pseudo Code for the getPly function(gets move information):**

//Create Ply struct to store turn information

//Store player color in corresponding member variable

//Get input for piece selection

//Get input for piece re-location

//Check that Ply data is “legal”, according to game rules

//Process input from user and store corresponding coordinates in Ply

//Return Ply

**Concepts:**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | |  |  |  | |  | |  |
|  |  | |  |  |  | |  | |  |
|  | | **CSC/CIS 17A Project 1 Check-Off Sheet** | | | |  | |  | |
|  |  | |  |  |  | |  | |  |
|  |  | |  |  |  | |  | |  |
|  | **Chap.** | | **Section** | **Concept** | **Points for** | | **Location in** | | **Comments** |
|  |  | |  |  | **Inclusion** | | **Code(Line #)** | |  |
|  |  | |  |  |  | |  | |  |
|  | **9** | |  | **Pointers/Memory Allocation** |  | |  | |  |
|  |  | | 1 | Memory Addresses |  | |  | |  |
|  |  | | 2 | Pointer Variables | 5 | | 30, 60, 160 | | makeBoard(); |
|  |  | | 3 | Arrays/Pointers | 5 | | 214, 190, 290 | | writePiece\_v2() |
|  |  | | 4 | Pointer Arithmetic |  | |  | |  |
|  |  | | 5 | Pointer Initialization |  | |  | |  |
|  |  | | 6 | Comparing |  | |  | |  |
|  |  | | 7 | Function Parameters | 5 | | 26-49 | | Prototypes |
|  |  | | 8 | Memory Allocation | 5 | | 328, 163, | | Create & delete pointers to variables and arrays |
|  |  | | 9 | Return Parameters | 5 | | 26-49,  160-1196 | | Seen in prototypes and definitions |
|  |  | | 10 | Smart Pointers |  | |  | |  |
|  |  | |  |  |  | |  | |  |
|  | **10** | |  | **Char Arrays and Strings** |  | |  | |  |
|  |  | | 1 | Testing |  | |  | |  |
|  |  | | 2 | Case Conversion |  | |  | |  |
|  |  | | 3 | C-Strings | 10 | | 504 | |  |
|  |  | | 4 | Library Functions |  | |  | |  |
|  |  | | 5 | Conversion |  | |  | |  |
|  |  | | 6 | Your own functions |  | |  | |  |
|  |  | | 7 | Strings | 10 | | 111, 190 | |  |
|  |  | |  |  |  | |  | |  |
|  | **11** | |  | **Structured Data** |  | |  | |  |
|  |  | | 1 | Abstract Data Types |  | |  | |  |
|  |  | | 2 | Data |  | |  | |  |
|  |  | | 3 | Access |  | |  | |  |
|  |  | | 4 | Initialize |  | |  | |  |
|  |  | | 5 | Arrays | 5 | | 282, 468 | |  |
|  |  | | 6 | Nested | 5 | | 472, 477 | |  |
|  |  | | 7 | Function Arguments | 5 | | 468, 451, 555 | |  |
|  |  | | 8 | Function Return | 5 | | 394 | |  |
|  |  | | 9 | Pointers | 5 | | 214, 190, 290 | |  |
|  |  | | 10 | Unions \*\*\*\* |  | |  | |  |
|  |  | | 11 | Enumeration | 5 | |  | |  |
|  |  | |  |  |  | |  | |  |
|  | **12** | |  | **Binary Files** |  | |  | |  |
|  |  | | 1 | File Operations |  | |  | |  |
|  |  | | 2 | Formatting | 2 | |  | |  |
|  |  | | 3 | Function Parameters | 2 | |  | |  |
|  |  | | 4 | Error Testing |  | |  | |  |
|  |  | | 5 | Member Functions | 2 | |  | |  |
|  |  | | 6 | Multiple Files | 2 | |  | |  |
|  |  | | 7 | Binary Files | 5 | |  | |  |
|  |  | | 8 | Records with Structures | 5 | |  | |  |
|  |  | | 9 | Random Access Files | 5 | |  | |  |
|  |  | | 10 | Input/Output Simultaneous | 2 | |  | |  |

References:

Gaddis, Tony. *Starting out with C++: from Control Structures through Objects*. Pearson Education Limited, 2015.