15-112 Term Project

Project Description:

My project is the classic game Chess with two players WITHOUT multiplayer or P2P network. The entire project will be built using basic graphics and algorithms we learned in class.

Competitive Analysis.

Most Chess games use external modules to simplify a lot of the logic involved in Chess. Since I will only be using basic graphics without relying on external modules/libraries, all of the algorithms will be implemented from scratch. I will have some additional features, such as a rotating board after each move, and a hint button, which will tell the player if they can checkmate the other player by showing a viable move. I will also have a move log on the side to document the moves.

I will also be implementing a save and load feature. For example, programs do not save on basic graphics when closed, but I will be implementing a way to save the state of an unfinished game to a file.

Structural Plan: A structural plan for how the finalized project will be organized in different functions, files and/or objects.

I will be using OOP to structure the pieces, and have each individual piece inherit from a base piece class. SInce each piece has its own set of rules, I will use a concept called abstract class to represent a base piece, and override the rules in each subclass. I will have app functions to flip the board and the pieces after each turn, and all the pieces will be stored in a list.

I will make sure all the app functions follow the MVC programming model.

As of now, I have everything in one file, as I do not have tons of code yet. I will divide them into modules later. My final project will be divided into multiple modules as follows:

- cmu_112_graphics.py, the one provided in the class
- Pieces.py, the class representation of all the pieces
- Chess.py, the game logic

Algorithmic Plan: A detailed algorithmic plan for how you will approach the trickiest part of the project. Be sure to clearly highlight which part(s) of your project are algorithmically most complex, and include details of the algorithm(s) you are using in those cases.

The trickiest part of the project would have to be verifying a checkmate. In order to handle this, I will have to check every possible valid move the player can make, and if there are no valid moves that would get the player out of the check status, then the player is in checkmate.

Another tricky part would be verifying a check (note, this is not a checkmate). In order to verify a check, the program will go through the enemy pieces, and see if the piece is in position to 'kill' the king. If there is at least one piece in such a position, then the king is in check.

Timeline Plan: A timeline for when you intend to complete the major features of the project.

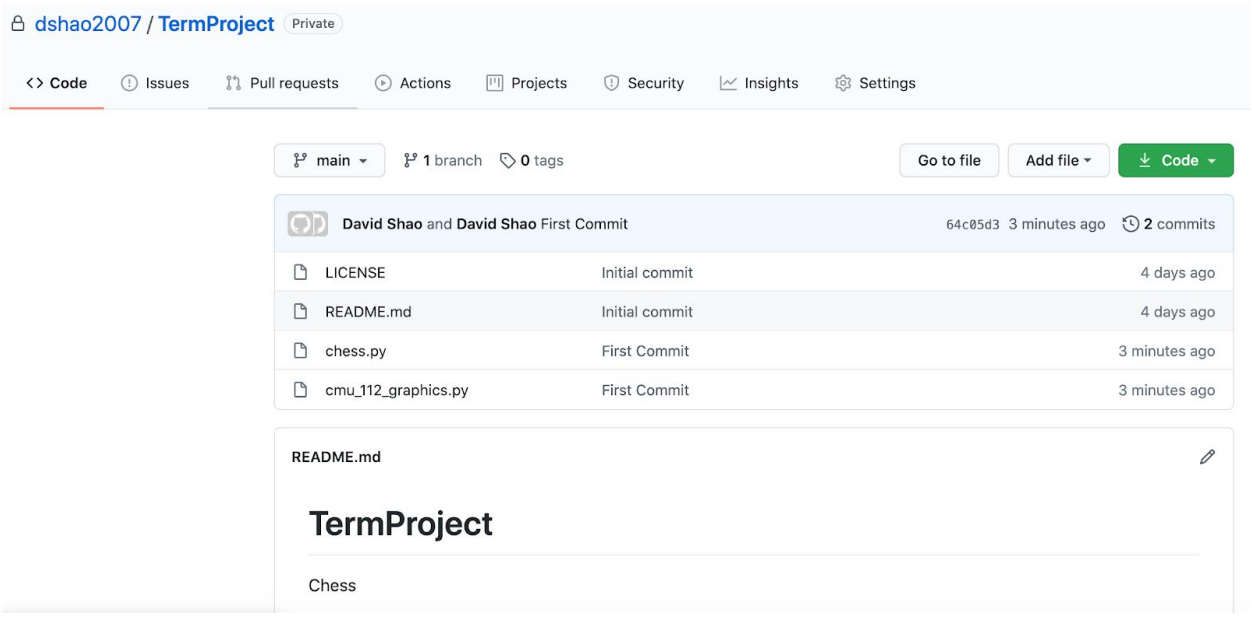
By Nov 29:
Pawns, Rooks, Queens, Bishops, Knights move
logic implemented

By Dec 5:
All of the pieces move logic implemented, along with the logic for 'check' and 'checkmate'

By the deadline:
Improved graphics with online pictures representing the pieces. Save and load features of an unfinished game implemented.

Version Control Plan

I will be using a private repository on GitHub for version control. This is my first commit:



Module List:
As of now, none aside from basic graphics.

Updated Section:

I have made a number of small, but significant changes and improvements. The user will now have the ability to choose AI Mode or a standard single player mode at the start of the program. I decided to scrap the concept of the save and load game, as that is pretty fucking useless. To elaborate on my AI mode, I use a minimax algorithm that goes 3 levels deep to run the AI. The process is quite slow, as the tree grows exponentially, but I am in the midst of a pruning algorithm that can hopefully cut the time. Structurally, I also put app functions into a class, but other than that, my structure remains the same.