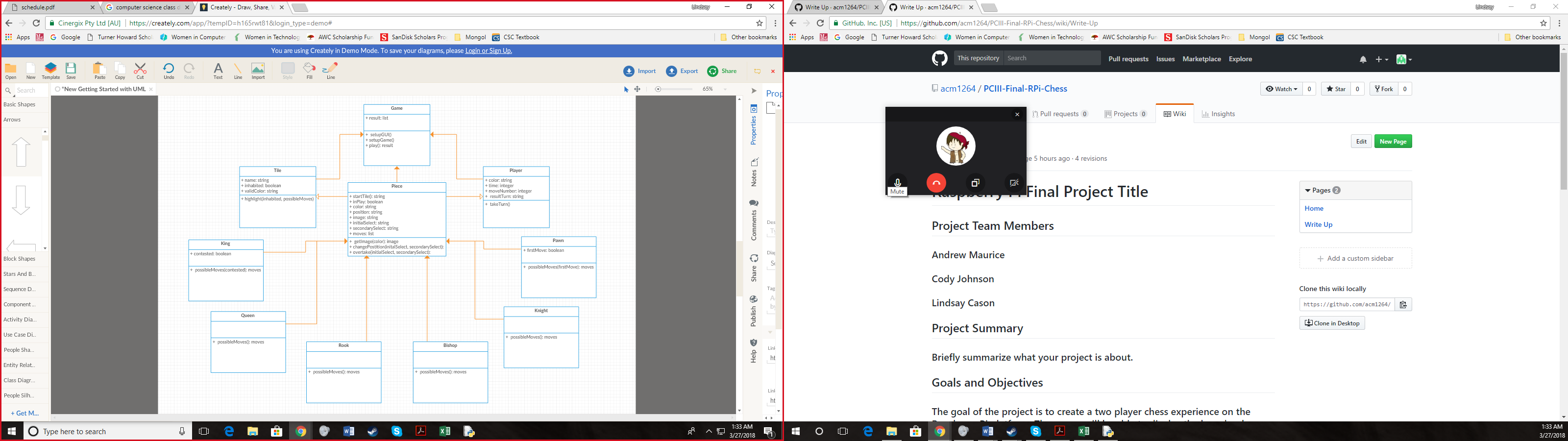
Project Proposal

Chess in Python

Everything in this document can be changed. This is simply a possible outline to organize a complex program to be more readable and easily divisible amongst the three of us to avoid anyone having to do more work than necessary. If all goes well, we should be able to use the majority of the code already written to apply in a program that utilizes classes. Please feel free to express opinions, thoughts, and concerns so we can find create this program in a way we will all be happy with the result.

I look forward to working with you guys!

**Class Diagram**:

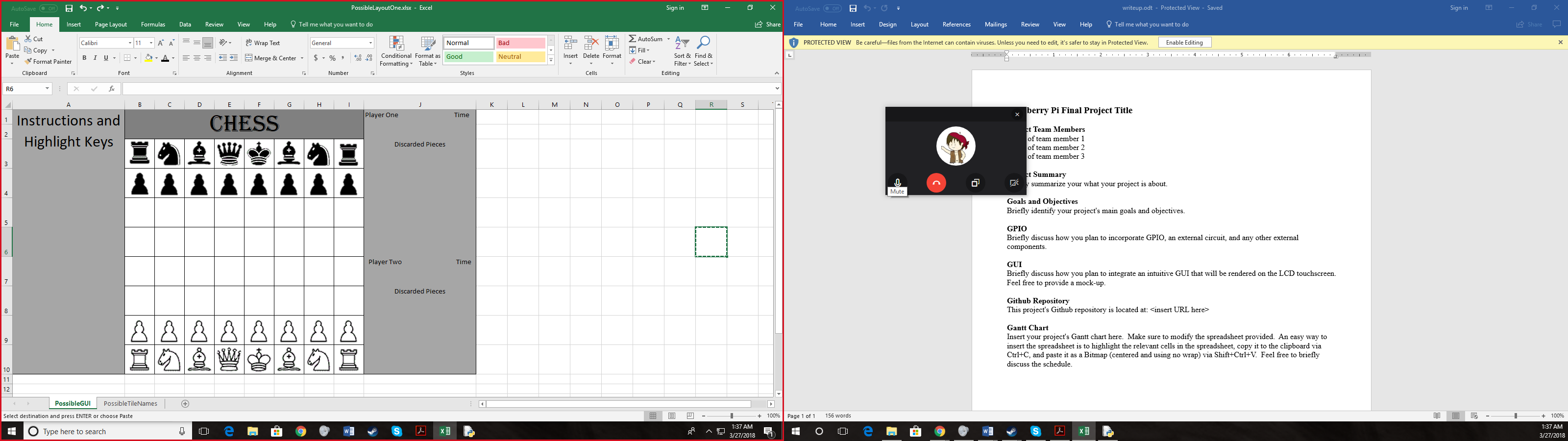


The class diagram above is a very rough outline of what a possible class diagram could be for this project. In things such as creating a list of possible moves, it could get very convoluted quickly seeing that it would have to refer to several different instance variables in different classes. This class diagram also does not consider GUI and GPIO aspects of the program which will need to be implemented as we go. I believe before we get too far ahead we will need to create an updated version of the class diagram that is more equipped to handle complex functions such as determining possible moves.

When utilizing classes for this project, I believe we will either need to heavily rely on a Piece class or a Tile class that inherits from game. This sample class diagram is simply introducing the many different types of things we will need to consider and why it will be important to have an organized structure to keep up with them. This class diagram does not take into account Check, Checkmate, or many other processes that will also need to be checked throughout the game.

In my opinion, I think it would be a good idea to discuss and partially plan out the benefits and drawbacks of both a Piece heavy program and a Tile heavy program to decide which would be more readable for us, and also most efficient. I will happily make class diagrams that are more accurate for both a Piece heavy program and a Tile heavy program for us to refer to while making the program, however I feel it is necessary to receive input on from each of you to create a structure that is pleasing for all of us.

**GUI**:



This is one possible version of the GUI. The title at the top can be removed and the sidebars narrowed to accommodate a larger board if preferred. However, if we make the board any larger, we risk all the pieces not being able to fit in their specified discard section of the right sidebar when on the Raspberry Pi. However, shrinking the left sidebar is also an option to fix this. I believe the GUI will need to be adjusted as we go, but if a completely different GUI layout is preferred then it would be best to discuss and have a mock-up ready to turn in with the write up for the project due on April 4th.

**Numbering the Tiles**:

When numbering the tiles, naming will be important in simplifying identifying possible moves. Consider the following:



With both possible numbering systems, the name of the tile will hold weight both as a string and as an integer. The two-digit numbers can be broken into two one-digit numbers representing the row and column. Both layouts also allow possible moves to be determined by handling the tile name as an integer.

In the case of Possible Layout One, the numbering follows the traditional chessboard layout. Thus, to traverse the columns, adding or subtracting ten to the original tile integer will result in the tile in the next column. To traverse rows, adding or subtracting one will result in the tile in the next row.

For Possible Layout Two, the numbering follows the standard left to right, top to bottom system. To traverse columns and rows in this layout is opposite of the previous. Adding or subtracting ten will traverse rows and adding or subtracting one will traverse columns.

With both of these layouts, range checking will be necessary to ensure that no possible move is calculate as a tile that is not on the board. For example, “49” would not be found on the board. Two ways I have thought of solving this problem is either checking a list or dictionary of possible tiles, or to break the integer into two one-digit integers and ensure each value is between 1 and 8.