**ENGR 102**

**Team Project**

This project is meant to be open-ended, requiring team participation, and incorporating all the aspects of the course we have covered so far. This project will count for the equivalent of 4 homework/lab assignments (i.e. about 2 weeks’ worth of regular work).

Your team should put together a program that lets a person play a 2-person turn-based (think board-style) game, either against another person, or against the computer. Examples of the game you choose could range from the very simple (tic-tac-toe) to the much more complex (chess). Other examples of games could include Connect 4, Reversi (Othello), Mancala, Checkers, Dots and Boxes, etc.

The project has some basic requirements. You will earn more points for going beyond the minimum requirements, including handling more advanced games.

**Requirements**

Here are minimum requirements your games should have:

* There should be some sort of display of the board itself each turn. This can be text output to the screen if you wish.
* The rules of the game should be enforced. For example, if your team chose to implement chess, then you should ensure that each piece can only make valid moves (and that all valid moves are possible). Also, you should be able to detect and report when there is a winner (or if the game ends tied).
* You should allow the user to select whether they are playing a 2-person game or playing against the computer. For a 2-person game, just alternate asking for moves.
* The computer should, at minimum, make a valid move if the user is playing against the computer. The minimum requirements are just that the computer always makes some valid move if one is available.
* You should include try-except blocks to handle user input, and possibly other aspects of your program. That is, you should ensure that users enter valid moves, using try-except to catch cases when they might enter their move in an invalid way.
  + If for some reason your input is not amenable to using try-except statements, you can use try-except statements in different areas of your code, but they should be included to catch exceptional situations.
* You must provide design documents and a statement of group work for your team’s project.

The minimum requirements will account for 80% of the overall grade. Earning more than 80% will require adding additional functionality or features beyond the minimum. Examples of additional functionality (you are not limited to these – these are just some ideas) could include:

* A more advanced game (i.e. chess is more complex than a basic game like Connect 4). Games with randomness (like Backgammon) would also be more advanced. A game like ordinary tic-tac-toe would lose some credit in this category.
* Computer AI behavior. Making “smarter” moves, etc. Even making a random move from the set of all available moves is a slight improvement over the minimum. Somehow trying to determine a “best” move is even better.
* A win/loss record-keeping system, high score, etc.
* A better user interface (e.g. a graphical user interface)

You are certainly not expected to include all of these; the idea is that some of these improvements can be included

You may use external modules to assist in your program. However, be sure to use the following guidelines regarding external modules:

* Be sure that the module is publicly available (e.g. via PyPi) and works with Python 3
* Be sure that license restrictions do not prevent you from using the module)
* Realize that the module should not solve the problem for you. For instance, a “Checkers game” module would not be acceptable to use in your checkers game. A “Checkers AI” module that let a computer play checkers would be OK to include, and you would get some bonus for including that over the minimum, but not as much as if you came up with your own computer program.
* Clearly document in your code (and in your final document) what you used, and how.

**Approach**

Your team should begin by choosing a game that you would like to implement.

You should use the techniques we have discussed in class when designing and building your program. This includes:

* Top-down and/or Bottom-up design processes
* Incremental (pyramid-style) development
* Generating a full set of tests, and ensuring the program meets all tests
* Using methodical debugging approaches to find and handle bugs.

Note: it is critical that you approach this task using an incremental development approach. That is, ensure that one (small) part of your program is working before adding another, small, part on. It is very common when people face a task like this to try to “bite off more than you can chew” – imagining a program with far more complexity and features than is feasible to implement in a given time frame. If you follow an incremental development approach, you should be able to ensure that you always have a working, bug-free program.

To help ensure that your team gives sufficient thought to the project, you will need to provide a design description. Stated below.

When working as a team, it is common for different team members to assume different roles and responsibilities for the project. Not every person needs to write the same amount of code, but each person should contribute equally to the project. For example, one team member might focus primarily on writing test cases for your team’s functions, another might focus only on the interface, another might do less coding but handle all the design documentation and ensure that everyone else on the team is doing their part, etc. You are free to organize your team however you see fit, but you will need to document everyone’s role at the end.

Please note that your team will not be able to sit together during class time to develop most of this program – the actual programming will need to be done outside of class time.

**Design Document and Work Statement**

Your team should put together a design document, laying out the way that your program will be set up. This should include the following:

* The game you will be using.
* An initial design breakdown.
  + You should follow a top-down and/or bottom-up design process to determine what functionality you want to implement. You should end up with a list of core functions, each broken down to a r
  + In addition, your team should describe any “important” variables you will use and how they will be defined. An example of an “important” variable would be one holding the information about where pieces are on the board at a particular time. An example of an “unimportant” variable (that you don’t need to include in the design document) would be an iterator for some loop in your program.
* An initial plan of work: what responsibilities does each team member have on the project, and what is the timeline you expect to follow.

After your team has finished your program, your team should put together a statement of work. In this document, you should do the following:

* List what was accomplished from your original plan
* List any items not accomplished from your original plan
* List any external modules (if any) that you included (outside of the Python Standard Library) that someone would need to install to run your program.
  + Provide a very short description of how that library was used in your code.
* Provide a brief (1-3 paragraph) summary of your team’s work. What problems did you encounter along the way, and what incorrect assumptions did you have that led you to being able to accomplish less or more than originally planned?
* Provide a list of what each team member contributed to the project. Be specific: it is not enough to say “team member X helped code”. You should identify exactly which parts each person did on the project.
* Provide a brief (1-2 paragraph) “reflection” statement. Given what you now know from the project, how would have approached it differently from the beginning.

**Oral Presentation**

On the final class day, Wednesday December 5, your team should be prepared to present a **brief** overview of your project in class. During this overview, you should be prepared to state:

* What game you are implementing
* A brief demonstration of your program working
* A summary of any “extras” you added on to your code.

Your presentation should take a **maximum** of 5 minutes. It is OK to have a significantly shorter presentation.

**Grading:**

Design Document – 10%

Reflection Document – 10%

Commenting, Naming, and Clear Structure of your program – 10%

Basic Game-playing rules enforced, including win conditions – 30%

Allowing both 2-player and vs. Computer play – 5%

Computer always chooses a valid move – 5%

Use of try-except statements – 10%

Additional Functionality – 20%