CMPUT 274 — Tangible Computing I **Version 1.2-20**
Weekly Exercise #XYZ — Object-Oriented Minimax

Introduction

The purpose of this exercise is to gain experience in writing an object-oriented program (OOP), by modifying an existing non-OO Python program. You are **not** adding any new functionality to the original program, just converting it into an OOP. Other tasks in this exercise will also require you to gain some experience with git-based version control and debuggers.

This exercise involves modifying a substantial amount of code that you have been given. Therefore, the use of a debugger to understand the given code, and to debug your modifications to that code, is essential for time-efficiency. Using a debugger to understand the original code will likely save you hours of time.

There may be unspecified details for this exercise for which a **reasonable design decision** can be made (i.e., does **not** contradict an existing specification), stated explicitly (e.g., in a comment in the code or input data), and implemented.

This assignment will require you to know something about:

1. Python, both OO and non-OO
2. Classes, Attributes
3. Minimax
4. Recursion
5. Debuggers
6. git, for version control

The main challenge of this assignment is **not** in the total number of lines of code that has to be implemented, but rather in understanding the original code (`minimax.py`) well enough to be able to adapt it to OOP.

Task I: Git, Git Log (10/100 marks)

Create a clone of the git repository at:

`https://github.com/paullu-ualberta/tic-tac-toe-minimax.git`

It is a fork of the **original** git repository by **Clederson Cruz**, with some key changes to make the program run "deterministically" (i.e., with a random number seed) and to simplify the output, for easier testing. A Makefile and some test inputs (and outputs) have also been added. Therefore, you must use the version from `paullu-ualberta`.

Note that Clederson Cruz has no known affiliation with the University of Alberta, but has made this code available under the open-source GPL 3.0 licence. This code was selected because it is the right balance

between easy-to-understand and functionality, especially in its use of recursion for minimax. Also, there is documentation about the program and the minimax algorithm at the GitHub page.

Note that the code, by Cruz and modified by us, **may** have bugs in it. Therefore, the main goal of Task III is to match the output of the original program, so-called "feature-to-feature, bug-to-bug" compatibility.

Resources for git are on the eClass page, in the block titled *Resource Documents by Topic*. The commands for cloning and commit are purposefully not given here so that you can find and learn them yourself; the commands are not complicated, but learn them, please. A previous video on git from CMPUT 274 can be found at `https://youtu.be/nNJBc3IndcA`

For the rest of this exercise description, it is assumed that we are discussing the program and file `minimax.py` from `tic-tac-toe-minimax/py_version` in that repository.

Now, add your name and CCID to the docstring at the beginning of the program, in preparation for the screenshot in Task II. For example, like this:

```
1  from math import inf as infinity
2  from random import choice
3  from random import seed as randomseed       # Paul Lu
4  import platform
5  import time
6  from os import system
7
8  """
9  An implementation of Minimax AI Algorithm in Tic Tac Toe,
10 using Python.
11 This software is available under GPL license.
12 Author: Clederson Cruz
13 Year: 2017
14 License: GNU GENERAL PUBLIC LICENSE (GPL)
15
16 Example:
17 Paul Lu
18 CCID:  paullu
19 """
```

As you work on this exercise, make at least 5 git commits of your changes. I strongly advise you to include meaningful commit comments, and make commits often (i.e., you can make more than 5 git commits; even 200 git commits is not too much).

If you want, and if you have a GitHub account, you can optionally create a fork of the repository. Please make your fork a private repository to avoid exposing your Solo Effort work visible to the world. And, you can push your commits out to your fork of the original repository. If you do not fork the repository, do NOT try to push out your commits. No pull requests to the original repository will be accepted.

Capture the git log with `git log > submit.git.log.txt` and **submit file `submit.git.log.txt`** (see details below). Here's what one example of a **partial** git log with 5 commits looks like:

```
commit f65dd3cddb6403c8feadea5b6e847af0781a3260
Author: Paul Lu <paullu@cs.ualberta.ca>
Date:   Wed Nov 4 21:37:52 2020 -0700

    First automated test

commit c68303a8ec6fe9ffec62e1ca5b3e3693cffb35e9
Author: Paul Lu <paullu@cs.ualberta.ca>
Date:   Wed Nov 4 21:31:08 2020 -0700

    Simplify output and do not sleep.

commit dc72103fcc35c359210b2fa61e7fbe1ab572321d
Author: Paul Lu <paullu@cs.ualberta.ca>
Date:   Wed Nov 4 21:29:22 2020 -0700

    Mod to prevent clearing of the screen

commit 2521a0220ec9147a2409d67efef1f6ef3ae85716
Author: Paul Lu <paullu@cs.ualberta.ca>
Date:   Wed Nov 4 21:20:09 2020 -0700

    Easier testing

commit f882fa95c819d8813f6494588f987110d0a6a538
Author: Paul Lu <paullu@cs.ualberta.ca>
Date:   Wed Nov 4 21:18:59 2020 -0700

    Make random numbers deterministic with random seed.
```

Task I is worth 10/100 of your mark for a proper `submit.git.log.txt`.

Task II: Debugger, Call Stack Screenshot (10/100 marks)

Run the program multiple times (e.g., 20 times or more). For example, using:

```
python3 minimax.py
```

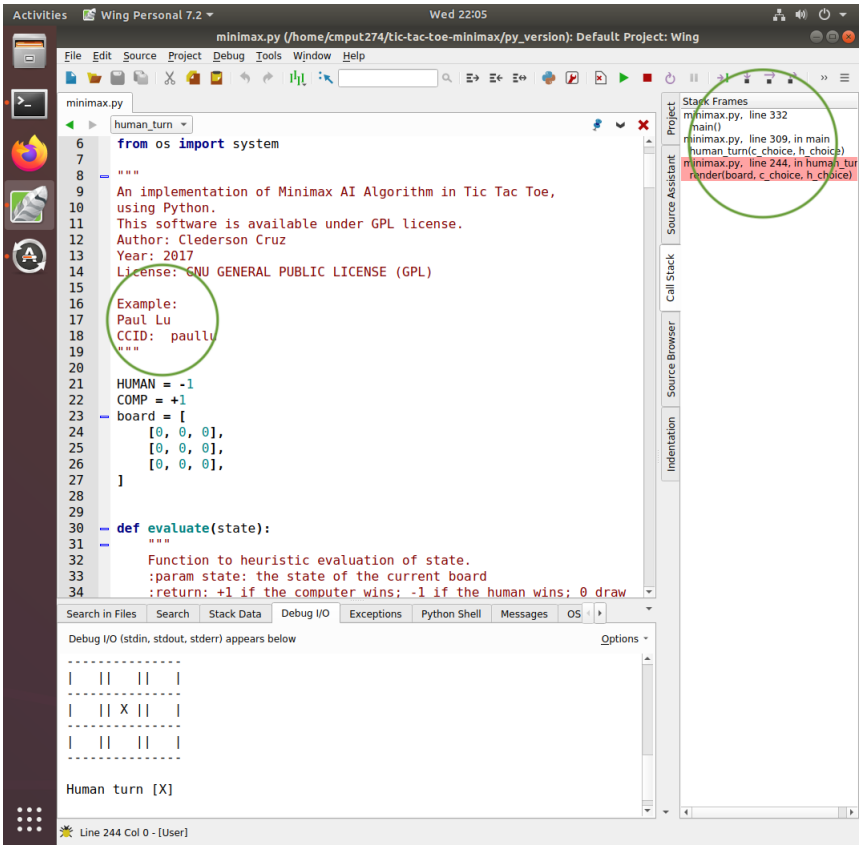
or using a debugger. Look at the Makefile and see how it automates some testing.

Learn how the game of Tic-Tac-Toe is played. Use a debugger (any debugger, such as the Wing IDE, `pdb`, or VSCode) to see how function **`minimax()` is a recursive function.**

Use a debugger to examine the execution stack (also known as the call stack, of stack frames) of the program. See how recursion results in multiple stack frames with the same function name.

Submit a screenshot of your debugger (filename: `debugger.png`, **Typo fixed:** November 9, 2020) running the program with at least 5 stack frames in the call stack. Make sure that the screenshot includes an image of your name in the code being debugged (i.e., name in docstring from Task I). Be sure to **annotate** the screenshot image (e.g., with a box, circle, or hand-drawn shape) to highlight the stack frames and your name.

Here is an example screenshot showing only 3 stack frames on the call stack (your screenshot needs to show 5 stack frames):



Task II is worth 10/100 of your mark for a proper `debugger.png` (**Typo fixed:** November 9, 2020).

Task III: Create OO Version of Minimax Program (60/100 marks)

Now that you understand the code as written, rewrite the code to use classes and elements from OO programming in Python. You are **not** adding any new functionality to the original program, just converting it into an OOP. In fact, your new program should produce exactly the same output, for the exact same input, as the original program. Restructuring and reorganizing a program, without adding new functionality, is called **code refactoring**. Your new program should be in file `oominimax.py`.

Hint 1: Re-use as much of the code as possible. Some of the existing functions can become methods for your new classes. Some functions might remain functions outside of classes.

Avoid writing new code as much as possible. You will have to modify the existing code (e.g., converting function calls to method invocations), but avoid writing new code.

Hint 2: After investing 2 hours using the debugger with the program, you can complete Task III in between 2 to 5 hours. If you do not use the debugger to help you understand the program, then Task III can easily take between 5 and 15 hours.

After your rewrite, for the same input, the program should **generate the exact same output** as the original, non-OO program. Every line, every character, every whitespace of output should be exactly the same between `minimax.py` and `oominimax.py` for the exact same input.

Your clone of the repository includes a Makefile and 3 pairs of input and output files to help you test your code. However, you can generate your own test cases for your OO minimax program by comparing its output with the original `minimax.py`.

Your new OO program `oominimax.py` **must include**:

1. At least 2 classes. More than 2 classes are allowed, but are not required. **Hint:** Consider a class State. Consider a class Board.
2. Each of these classes must have **at least one method, in addition** to `__init__()`, `__str__()`, and `__repr()__` (i.e., at least 4 methods in total). You will likely have more than 4 methods, in a good design.
3. Inheritance may be used, but it is not required.
4. You can change any aspect of the code (e.g., parameters, identifiers) as long as it produces the same output as the original program, for the same input. **Hint:** Most of the code can be re-used (e.g., cut-and-paste).
5. The new `main()` function must include at least one object instantiation.
6. Every object created must have at least one method invocation call-site, in addition to the implicit call to `__init__()`. In other words, you must call at least one method for each object.

Task III is worth 60/100 of your mark for a correct `oominimax.py` (i.e., exact same output as original `minimax.py`)

Style and OO Design (20/100 marks)

In addition to the requirements of Code Submission and Style Guide, we will look for and mark:

1. Proper use of classes for encapsulation and abstraction
2. Proper use of attributes
3. Proper use of getters/setters for attributes
4. Proper use of other OO mechanisms and design ideas

Testing and Other Hints:

In addition to any hints already given elsewhere, please consider the following:

Design, write, and use your own test programs. We cannot provide a comprehensive test suite. Do not rely just on our tests. When we mark your solution, we will be using tests that are consistent with the specifications but have not been released to you in advance.

One benefit of writing new tests is that it forces a close re-reading of the problem description to reveal any false assumptions (which can then be fixed) or additional assumptions (which can be then be documented as design decisions, if appropriate).

Double-check your submission, before and after you submit. If you submit the wrong file(s), or if eClass does not receive your submission properly, we will not accept a submission afterwards. Re-download what you submit on eClass. Double-check it. Ideally, you will submit and double-check your submission before the deadline. But, at least check after the deadline and take advantage of the up-to-6-hours late (with penalty) option to fix any mistakes.

Submission Guidelines:

Submit all of the required files (and no other files) as **one** properly formed compressed archive called either `oominimax.tar.gz`, or `oominimax.tgz`, or `oominimax.zip` (for full marks, please do **not** use `.rar`):

- when your archive is extracted, it should result in exactly *one directory* called `oominimax` (use this exact name) with the following files in that directory:
- `submit.git.log.txt`
- `debugger.png`
- `oominimax.py`
- your `README` (use this exact name) conforms with the Code Submission Guidelines
- No other files should be submitted.

Note that your files and functions must be named **exactly** as specified above.

A new tool has been developed by the TAs to help check and validate the format of your `tar` or `zip` file *prior* to submission. To run it, you will need to download it into the VM, and place it in the same directory as your compressed archive (e.g., `oominimax.zip`).

You can read detailed instructions and more explanation about this new tool in Submission Validator: Instructions (at the top of the Weekly Exercises tab), or run:

```
python3 submission_validator.py --help
```

after you have downloaded the script to see abbreviated instructions printed to the terminal.

If your submission passes this validation process, and all validation instructions have been followed properly, you will not lose any marks related to the format of your submission. (Of course, marks can still be deducted for correctness, design, and style reasons, but not for submission correctness.)

When your marked assignment is returned to you, there is a 7-day window to request the reconsideration of any aspect of the mark. After the window, we will only change a mark if there is a clear mistake on our part (e.g., incorrect arithmetic, incorrect recording of the mark). At any time during the term, you can request additional feedback on your submission.