MAPÚA UNIVERSITY
SCHOOL OF ELECTRICAL, ELECTRONICS, AND COMPUTER ENGINEERING

Experiment 4: Design Patterns and Unit Testing

CPE106L (Software Design Laboratory)

Darren M. Briones [illegible] Date: 2021. 04. 10 19:41:23
Edmund Hans Tapado [illegible] Date: 2021. 04. 10 16:24:42
Justine Angelo Ariel V. Villespin [illegible] Date: 2021. 04. 09 20:38:05

Group No.: **Group 3**
Section: **B2**

PreLab

Readings, Insights, and Reflection

Insights

In these chapters we learn about building desktop applications and unit testing. We learned about building command-line interfaces which consists of the Data layer, Core Logic layer, and the User Interface. The Data layer is created to hold the state of a game so that it can be saved to be resumed on a later date. The Core Logic layer, on the other hand, is essentially the most important layer. It contains all the functions that will predict how the program will work. Lastly, the User Interface. This layer contains the code of what the User will see on their end. The User interface is also an important layer since it is what the user perceives, and it should also consider the flow of the program (since the flow will depend on the UI). Finally, unit testing. Unit testing is a way to make a program consistent over time. **(BRIONES)**

In doing LAB Report 4, we delved in understanding GUI in python and unit testing. GUI is helpful way to give user a more interactive application because not all people are familiar with console applications. Moreover, in creating a program, it is best to unit test methods because individual units of the code are put into various test to determine whether they are capable of handling different scenarios. **(TAPADO)**

When performing the assigned tasks in Laboratory report 4, I realized that when creating a GUI python program, Tkinter is required since it is the toolkit for GUI python programs. I noticed it when debugging and reviewing the programmer's code of Tic-Tac-Toe. Moreover, I also learned about Unit testing. From what I have understand, it allows you to check or test your program if it works as what you have expected of the outcome. **(VILLESPIN)**

Reflection

This is the first time I have heard some of the terms used in this chapter in my limited programming experience. Although I have experience creating GUIs, I have never heard of Command-line interfaces and GUI toolkits. Also, Unit testing was new for me as well. Knowing that all of these are readily available for every programmer out there (and there are different versions that will fit any kind of person), programming has become even more broader in my eyes. **(BRIONES)**

Unit testing is a great way of analyzing what different situation your program could be in. It helps identify all the possible errors that may occur when it is being used. Also, GUI is a very helpful way of showing your code in a presentable manner. It helps other people understand it easily. Much like the tic-tac-toe game that was first created for the console terminal. It functions the same way, however, it is not as presentable as its GUI form. **(TAPADO)**

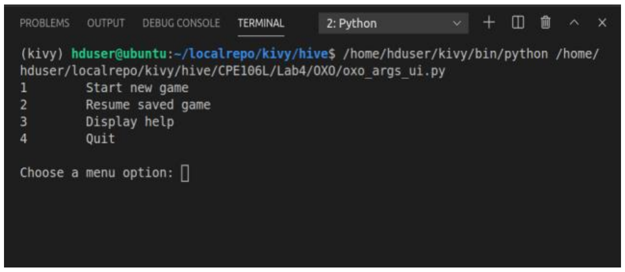
In Lab report 4, I learned that it is feasible to create a code in a console that looks like a game like Tic-Tac-Toe. When I debug the console application program of Tic-Tac-Toe, I was curious and amazed that this kind of output of the program can be done in a console application. Moreover, in the GUI application of Tic-Tac-Toe, I learned the structure of creating GUI programs. In Unit testing, I was able to determine that it could be used to test if your program has no errors. **(VILLESPIN)**

InLab

- **Objectives**
  1. **Use** Visual Studio Code of Linux in debugging the python and GUI programs.
  2. **Use** Linux terminal in running programs with unit testing.

- **Tools Used**
  - Linux Terminal
  - Visual Studio Code
- **Procedure.**

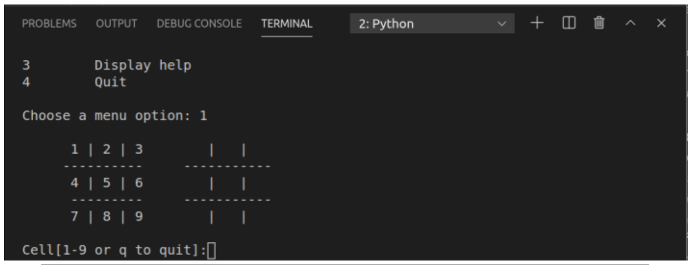
In Figure 1.1, we debug the python program oxo_args_ui.py. When the program was debugged, it outputs a menu and the program asks us to input a menu option that we want to run.



```
(kivy) hduser@ubuntu:~/localrepo/kivy/hive$ /home/hduser/kivy/bin/python /home/
hduser/localrepo/kivy/hive/CPE106L/Lab4/OXO/oxo_args_ui.py
1       Start new game
2       Resume saved game
3       Display help
4       Quit

Choose a menu option: 
```

Figure 1.1. Start of the program

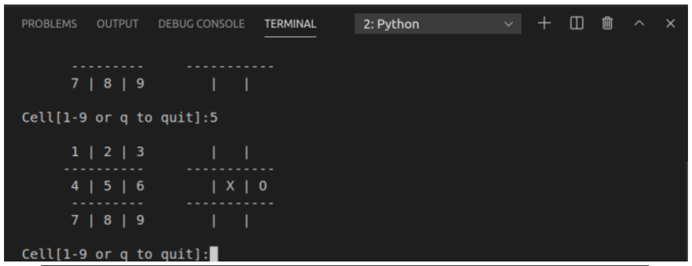
In Figure 1.2, We chose the first option. When we chose the first option, as we can see in the left, we can see the numberings for each cell. On the other hand, in the right side, it is the area where the output of the game is shown.



```
3       Display help
4       Quit

Choose a menu option: 1

      1 | 2 | 3       |   |
     ---------     -----------
      4 | 5 | 6       |   |
      ---------    -----------
      7 | 8 | 9       |   |

Cell[1-9 or q to quit]:
```

Figure 1.2. Choosing the first option.

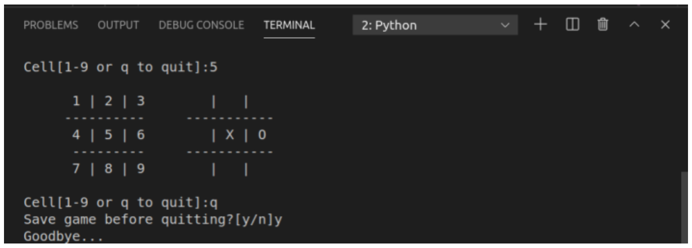
In Figure 1.3, we can see that we are playing ‘X’ and the computer is playing ‘O’ so we get to choose first the cell area that we want. As you can see in Figure 1.3, we chose 5 to be the area of our ‘X’ and after inputting 5, the chosen area of the computer is shown as well and we can see that the computer placed it in cell number 6.



```
      ---------     -----------
      7 | 8 | 9       |   |

Cell[1-9 or q to quit]:5

      1 | 2 | 3       |   |
     ---------     -----------
      4 | 5 | 6       | X | O
      ---------    -----------
      7 | 8 | 9       |   |

Cell[1-9 or q to quit]:
```

Figure 1.3. Playing the game.

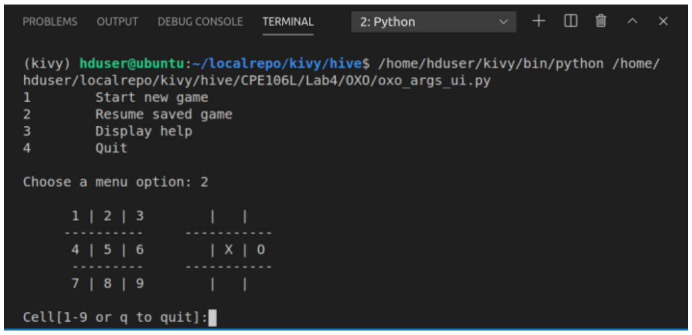
In Figure 1.4, we chose the quit the game. However, after typing q, an output was shown if we want to save the game and I chose y for yes to save the game and the program ended.



```
Cell[1-9 or q to quit]:5

      1 | 2 | 3        |   |
     ---------     -----------
      4 | 5 | 6        | X | O
      ---------     -----------
      7 | 8 | 9        |   |

Cell[1-9 or q to quit]:q
Save game before quitting?[y/n]y
Goodbye...
```

Figure 1.4. Typing q to quit the game.

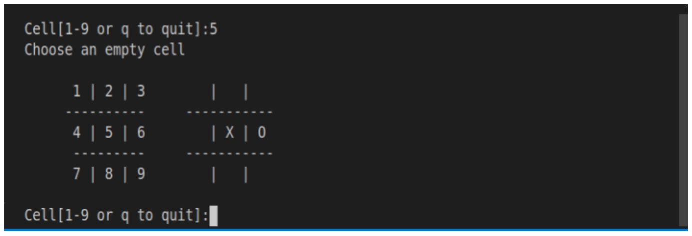
In Figure 1.5, we debug again the program since we quitted the game as shown in Figure 1.4. When we debug the program, we chose menu option 2 since we saved our previous progress of the game. After inputting it, it showed the saved progress of the game.



```
(kivy) hduser@ubuntu:~/localrepo/kivy/hive$ /home/hduser/kivy/bin/python /home/
hduser/localrepo/kivy/hive/CPE106L/Lab4/OXO/oxo_args_ui.py
1         Start new game
2         Resume saved game
3         Display help
4         Quit

Choose a menu option: 2

      1 | 2 | 3        |   |
     ---------     -----------
      4 | 5 | 6        | X | O
      ---------     -----------
      7 | 8 | 9        |   |

Cell[1-9 or q to quit]:
```

Figure 1.5. Debugging again the program.

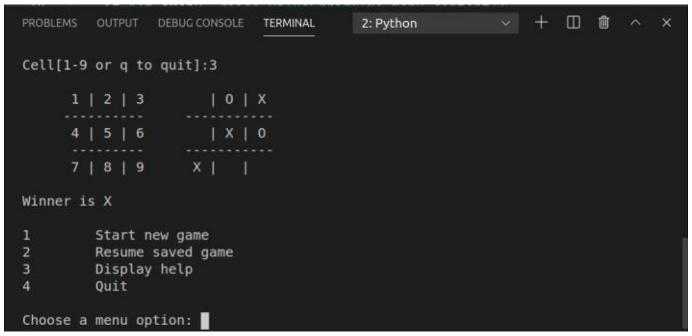
In Figure 1.6, we are trying to know the outcome of the program when inputting a cell number to an occupied cell. When we input 5, an output was shown saying that we should choose an empty cell since 5 is already occupied. After showing that, the program wants us to input again.



```
Cell[1-9 or q to quit]:5
Choose an empty cell

      1 | 2 | 3        |   |
     ---------     ----------
      4 | 5 | 6        | X | O
      ---------    ----------
      7 | 8 | 9        |   |

Cell[1-9 or q to quit]:
```

Figure 1.6. Testing the outcome of the program when inputting a cell number to an occupied cell.

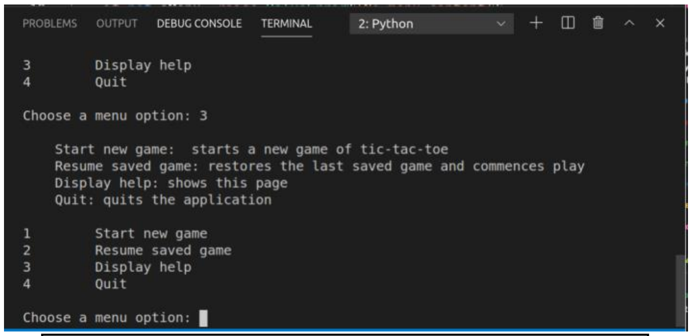
In Figure 1.7, after inputting the cells numbers that we want in order for us to win the game, the program outputs the winner of the game and showed again the menu option. So, when the program reaches its end, it showed again the menu option and the program will not end since you need to type 4 for us to quit the game and for the program to end.



```
Cell[1-9 or q to quit]:3

      1 | 2 | 3        | O | X
     ---------     ----------
      4 | 5 | 6        | X | O
      ---------    ----------
      7 | 8 | 9      X |   |

Winner is X

1       Start new game
2       Resume saved game
3       Display help
4       Quit

Choose a menu option:
```

Figure 1.7. Winning the game.

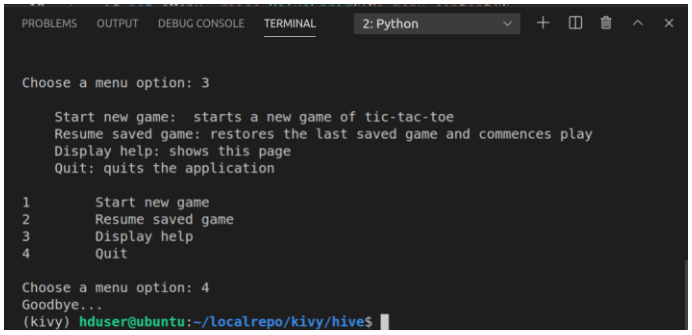
In Figure 1.8, we chose option 3 for us to see the contents of the game. After that, it showed the menu option again.



```
3       Display help
4       Quit

Choose a menu option: 3

    Start new game:  starts a new game of tic-tac-toe
    Resume saved game: restores the last saved game and commences play
    Display help: shows this page
    Quit: quits the application

1       Start new game
2       Resume saved game
3       Display help
4       Quit

Choose a menu option: 
```

Figure 1.8. Choosing option 3.

In Figure 1.9, we chose option 4 for us to quit the game and end the program.



```
Choose a menu option: 3

    Start new game:  starts a new game of tic-tac-toe
    Resume saved game: restores the last saved game and commences play
    Display help: shows this page
    Quit: quits the application

1       Start new game
2       Resume saved game
3       Display help
4       Quit

Choose a menu option: 4
Goodbye...
(kivy) hduser@ubuntu:~/localrepo/kivy/hive$ 
```

Figure 1.9. End of the program.

In Figure 2.1, we started to debug the program oxo_gui_complete.py and it showed a GUI output. We can see 2 tabs which is the File and Help. Below it, we can see the cells for us to play Tic Tac Toe.

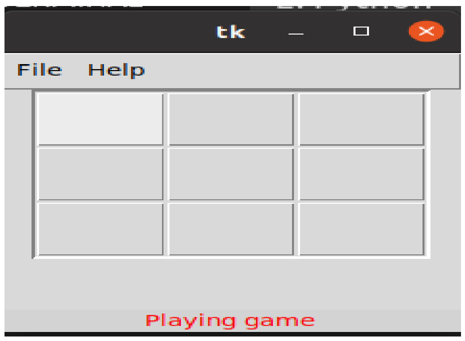


Figure 2.1. Start of the program.

In Figure 2.2, we can the options in the File tab. It has 4 options which are New, Resume, Save, and Exit.

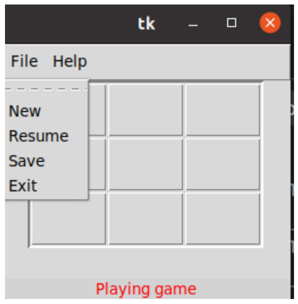


Figure 2.2. The File tab.

In Figure 2.3, we can the options in the Help tab. It has 2 options which are Help and About.

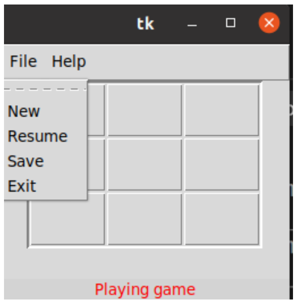


Figure 2.3. The Help tab.

In Figure 2.4, we choose the first option in the Help tab which is Help. When we clicked it, it showed the description for the options of the File and Help tab.

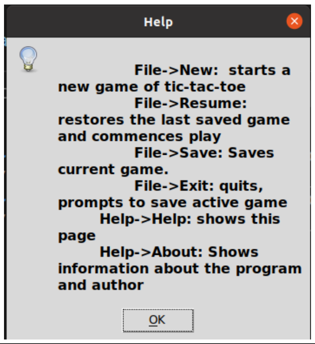
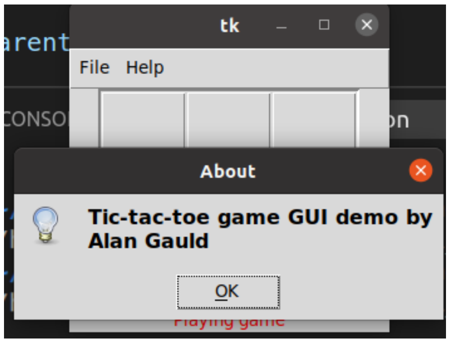

```
Help
        File->New:  starts a
new game of tic-tac-toe
        File->Resume:
restores the last saved game
and commences play
        File->Save: Saves
current game.
        File->Exit: quits,
prompts to save active game
     Help->Help: shows this
page
     Help->About: Shows
information about the program
and author
OK
```

Figure 2.3. The contents of Help option.

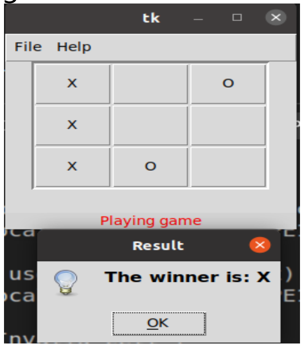
In Figure 2.5, we choose the second option in the Help tab which is About. When we clicked it, it showed the name of the program and the programmer's name who programmed the GUI program.



Tic-tac-toe game GUI demo by Alan Gauld

Figure 2.5. The contents of About option.

In Figure 2.6, the program is played by clicking the unoccupied cells and it will display the winner of the game.



The winner is: X

Figure 2.6. Playing the game.

In Figure 2.7, the program is played by clicking the unoccupied cells and it will display the winner of the game.

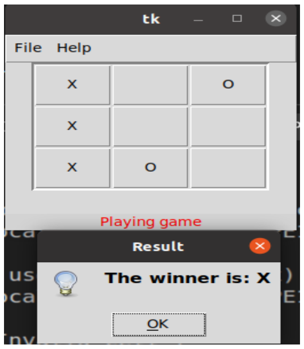


Figure 2.7. Playing the game.

In Figure 2.8, We clicked the New option to start a new game and for us to try the save and resume options.

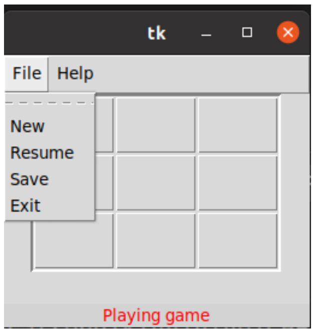


Figure 2.8. Clicking the New option.

In Figure 2.9, we clicked the Save option to save the game that is occurring, and we exited the GUI program to test if the Save option would work if we tried to click the Resume option.

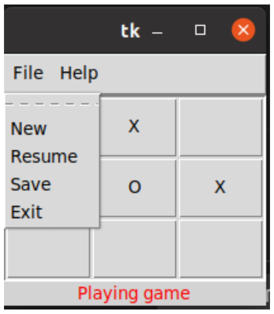


Figure 2.9. Clicking the Save option.

In Figure 2.10, we clicked the Resume option and it showed the saved progress that we did in Figure 2.9.

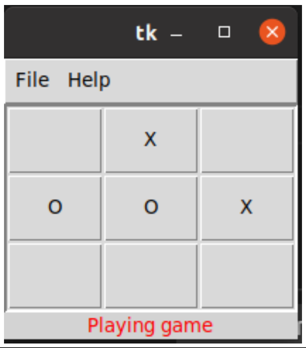
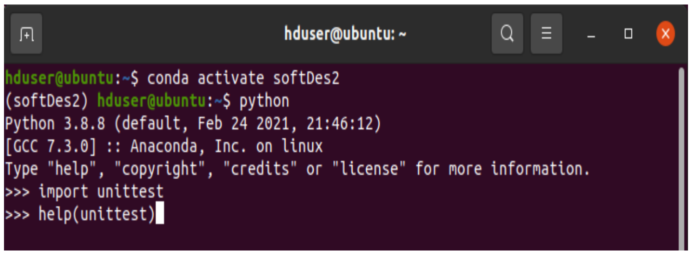


Figure 2.10. Clicking the Resume option.

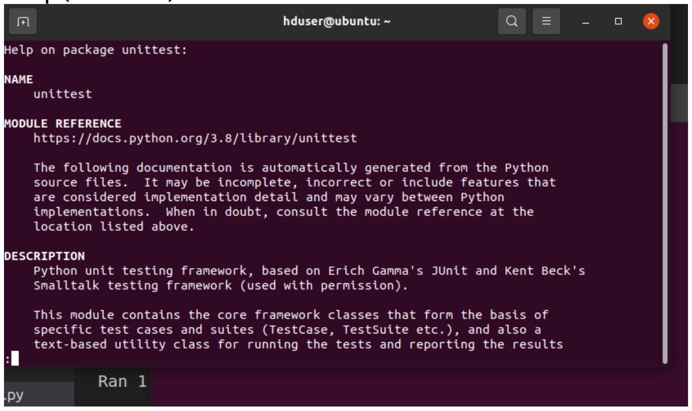
In Figure 3.1, it shows the two commands for us to see the Help on package unittest.



```
hduser@ubuntu:~$ conda activate softDes2
(softDes2) hduser@ubuntu:~$ python
Python 3.8.8 (default, Feb 24 2021, 21:46:12)
[GCC 7.3.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import unittest
>>> help(unittest)
```

Figure 3.1. Commands for running Help on package unittest.

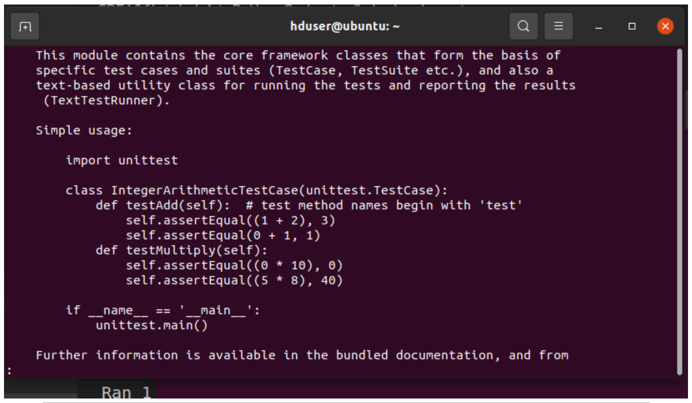
In Figure 3.2, it shows the Help on the package unittests after executing the command help(unittest).



```
Help on package unittest:

NAME
    unittest

MODULE REFERENCE
    https://docs.python.org/3.8/library/unittest

    The following documentation is automatically generated from the Python
    source files.  It may be incomplete, incorrect or include features that
    are considered implementation detail and may vary between Python
    implementations.  When in doubt, consult the module reference at the
    location listed above.

DESCRIPTION
    Python unit testing framework, based on Erich Gamma's JUnit and Kent Beck's
    Smalltalk testing framework (used with permission).

    This module contains the core framework classes that form the basis of
    specific test cases and suites (TestCase, TestSuite etc.), and also a
    text-based utility class for running the tests and reporting the results
:
```

Figure 3.2. Display after running help(unittest).

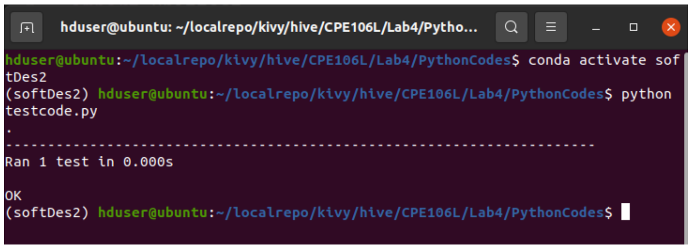
In Figure 3.3, it shows a sample simple code of the usage of unittest TestRunner.



```
    This module contains the core framework classes that form the basis of
    specific test cases and suites (TestCase, TestSuite etc.), and also a
    text-based utility class for running the tests and reporting the results
     (TextTestRunner).

    Simple usage:

        import unittest

        class IntegerArithmeticTestCase(unittest.TestCase):
            def testAdd(self):  # test method names begin with 'test'
                self.assertEqual((1 + 2), 3)
                self.assertEqual(0 + 1, 1)
            def testMultiply(self):
                self.assertEqual((0 * 10), 0)
                self.assertEqual((5 * 8), 40)

        if __name__ == '__main__':
            unittest.main()

    Further information is available in the bundled documentation, and from
:
```

Figure 3.3. Sample simple usage of unittest TestRunner.

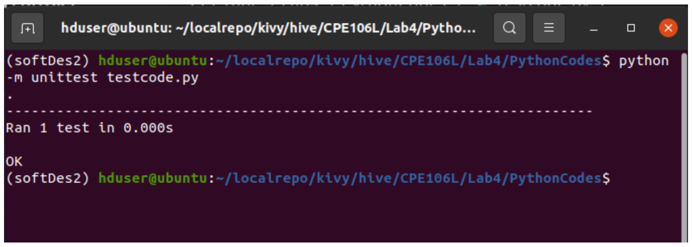
In Figure 3.4, we are running testcode.py with unittest.



```
hduser@ubuntu:~/localrepo/kivy/hive/CPE106L/Lab4/PythonCodes$ conda activate sof
tDes2
(softDes2) hduser@ubuntu:~/localrepo/kivy/hive/CPE106L/Lab4/PythonCodes$ python
testcode.py
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
(softDes2) hduser@ubuntu:~/localrepo/kivy/hive/CPE106L/Lab4/PythonCodes$ 
```

Figure 3.4. Running preliminary unittest code.

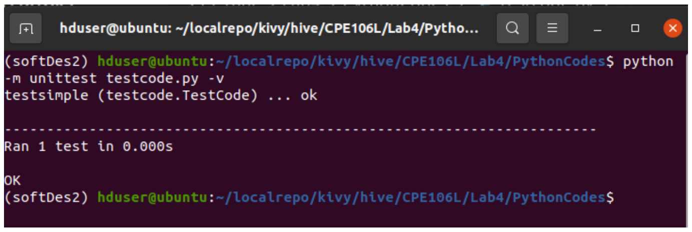
In Figure 3.5, we run the command python -m unittest testcode.py to see the output difference. Comparing it to Figure 3.4, there is no difference that can be seen for the output.



```
(softDes2) hduser@ubuntu:~/localrepo/kivy/hive/CPE106L/Lab4/PythonCodes$ python
-m unittest testcode.py
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
(softDes2) hduser@ubuntu:~/localrepo/kivy/hive/CPE106L/Lab4/PythonCodes$
```

Figure 3.5. Running with the command python -m unittest testcode.py.

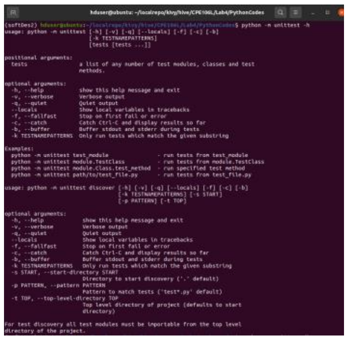
In Figure 3.6, we run the command python -m unittest testcode.py -v to see the output difference. Comparing it to Figure 3.4 and 3.5, there is a difference that can be seen for the output. The output testsimple (testcode.TestCode) ... ok was seen in this figure.



```
(softDes2) hduser@ubuntu:~/localrepo/kivy/hive/CPE106L/Lab4/PythonCodes$ python
-m unittest testcode.py -v
testsimple (testcode.TestCode) ... ok

----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
(softDes2) hduser@ubuntu:~/localrepo/kivy/hive/CPE106L/Lab4/PythonCodes$
```

Figure 3.6. Running with the command python -m unittest testcode.py -v.

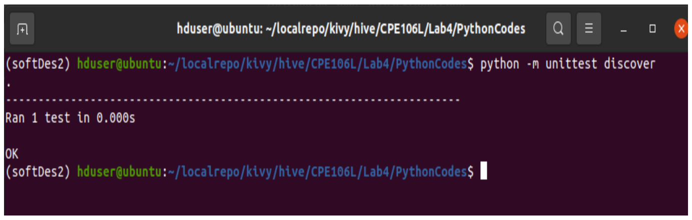
In Figure 3.7, it only showed the help commands.



```
hduser@ubuntu: ~/localrepo/kivy/hive/CPE106L/Lab4/PythonCodes
(softDes2) hduser@ubuntu:~/localrepo/kivy/hive/CPE106L/Lab4/PythonCodes$ python -m unittest -h
usage: python -m unittest [-h] [-v] [-q] [--locals] [-f] [-c] [-b]
                          [-k TESTNAMEPATTERNS]
                          [tests [tests ...]]

positional arguments:
  tests                 a list of any number of test modules, classes and test
                        methods.

optional arguments:
  -h, --help            show this help message and exit
  -v, --verbose         Verbose output
  -q, --quiet           Quiet output
  --locals              Show local variables in tracebacks
  -f, --failfast        Stop on first fail or error
  -c, --catch           Catch Ctrl-C and display results so far
  -b, --buffer          Buffer stdout and stderr during tests
  -k TESTNAMEPATTERNS  Only run tests which match the given substring

Examples:
  python -m unittest test_module               - run tests from test_module
  python -m unittest module.TestClass          - run tests from module.TestClass
  python -m unittest module.Class.test_method  - run specified test method
  python -m unittest path/to/test_file.py      - run tests from test_file.py

usage: python -m unittest discover [-h] [-v] [-q] [--locals] [-f] [-c] [-b]
                                   [-k TESTNAMEPATTERNS] [-s START]
                                   [-p PATTERN] [-t TOP]

optional arguments:
  -h, --help            show this help message and exit
  -v, --verbose         Verbose output
  -q, --quiet           Quiet output
  --locals              Show local variables in tracebacks
  -f, --failfast        Stop on first fail or error
  -c, --catch           Catch Ctrl-C and display results so far
  -b, --buffer          Buffer stdout and stderr during tests
  -k TESTNAMEPATTERNS   Only run tests which match the given substring
  -s START, --start-directory START
                        Directory to start discovery ('.' default)
  -p PATTERN, --pattern PATTERN
                        Pattern to match tests ('test*.py' default)
  -t TOP, --top-level-directory TOP
                        Top level directory of project (defaults to start
                        directory)

For test discovery all test modules must be importable from the top level
directory of the project.
```

Figure 3.7. Running with the command python -m unittest -h.

In Figure 3.8, we can see that the output is the same as Figure 3.4 and 3.5.



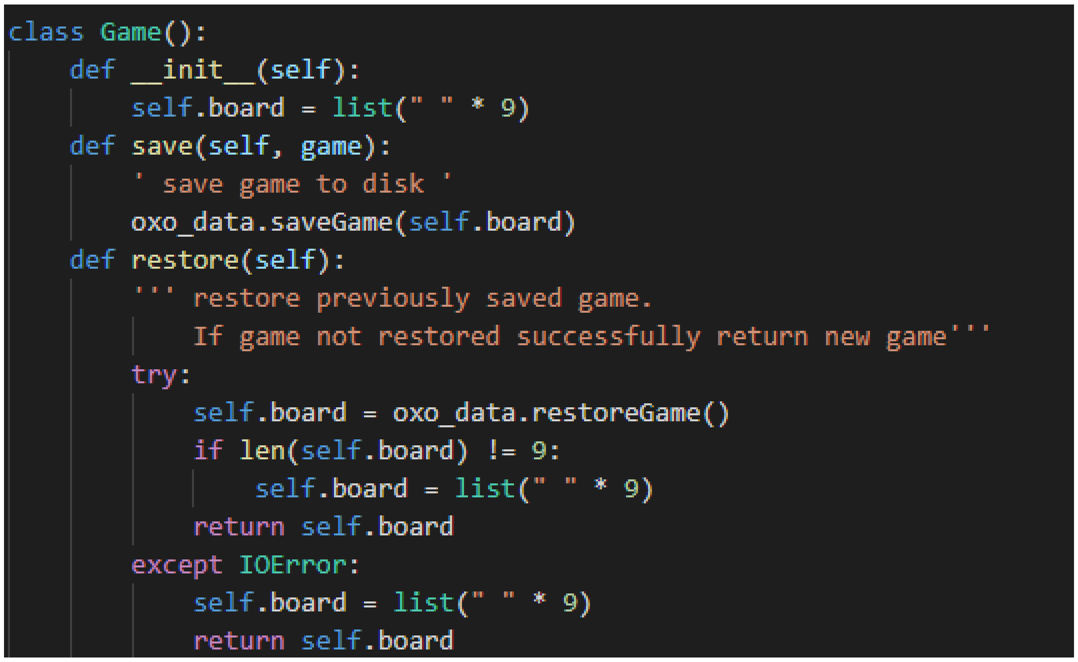
```
hduser@ubuntu: ~/localrepo/kivy/hive/CPE106L/Lab4/PythonCodes
(softDes2) hduser@ubuntu:~/localrepo/kivy/hive/CPE106L/Lab4/PythonCodes$ python -m unittest discover
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
(softDes2) hduser@ubuntu:~/localrepo/kivy/hive/CPE106L/Lab4/PythonCodes$
```

Figure 3.8. Running with the command python -m unittest discover.

PostLab

Programming Problems

Problem 1:

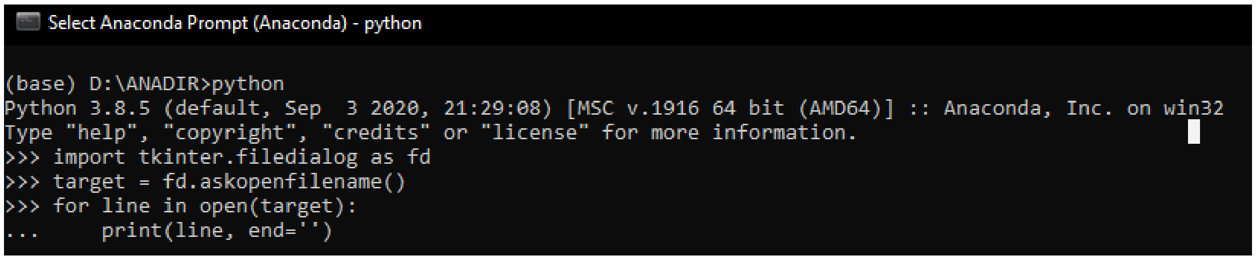


```python
class Game():
    def __init__(self):
        self.board = list(" " * 9)
    def save(self, game):
        ' save game to disk '
        oxo_data.saveGame(self.board)
    def restore(self):
        ''' restore previously saved game.
            If game not restored successfully return new game'''
        try:
            self.board = oxo_data.restoreGame()
            if len(self.board) != 9:
                self.board = list(" " * 9)
            return self.board
        except IOError:
            self.board = list(" " * 9)
            return self.board
```

Figure 1.

When converting the oxo_logic.py module into an OOP design, we added a class called Game(). The only difference between the previous version and the OOP version is that we changed the newGame(), saveGame(), and restoreGame(). The newGame() was converted into a default constructor, the saveGame() and restoreGame() both used the variable of the default constructor to perform the necessary actions.

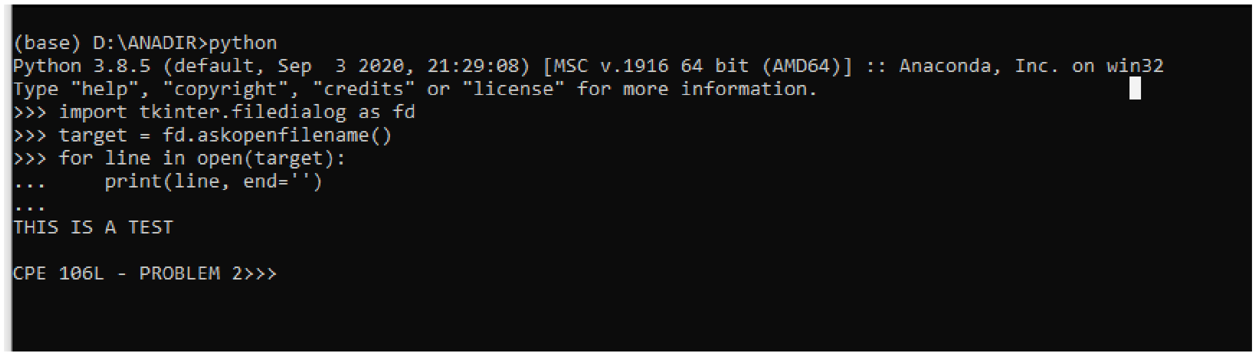
Problem 2:



```
Select Anaconda Prompt (Anaconda) - python

(base) D:\ANADIR>python
Python 3.8.5 (default, Sep  3 2020, 21:29:08) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import tkinter.filedialog as fd
>>> target = fd.askopenfilename()
>>> for line in open(target):
...     print(line, end='')
```

Figure 2.1

In Figure 2.1, you can see that we made use of the tkinter.filedialog module. First in the command line, "import tkinter.filedialog as fd" this is the part where we import the module to be used from tkinter. On the next line, "target = fd.askopenfilename()" we set a target file to be opened. After inputting this command, a prompt will appear in which the user will select the file to be opened. On the next two lines, "for line open (target): , print(line, end='') we enter commands that will output the content of the txt files into the terminal for us to view.

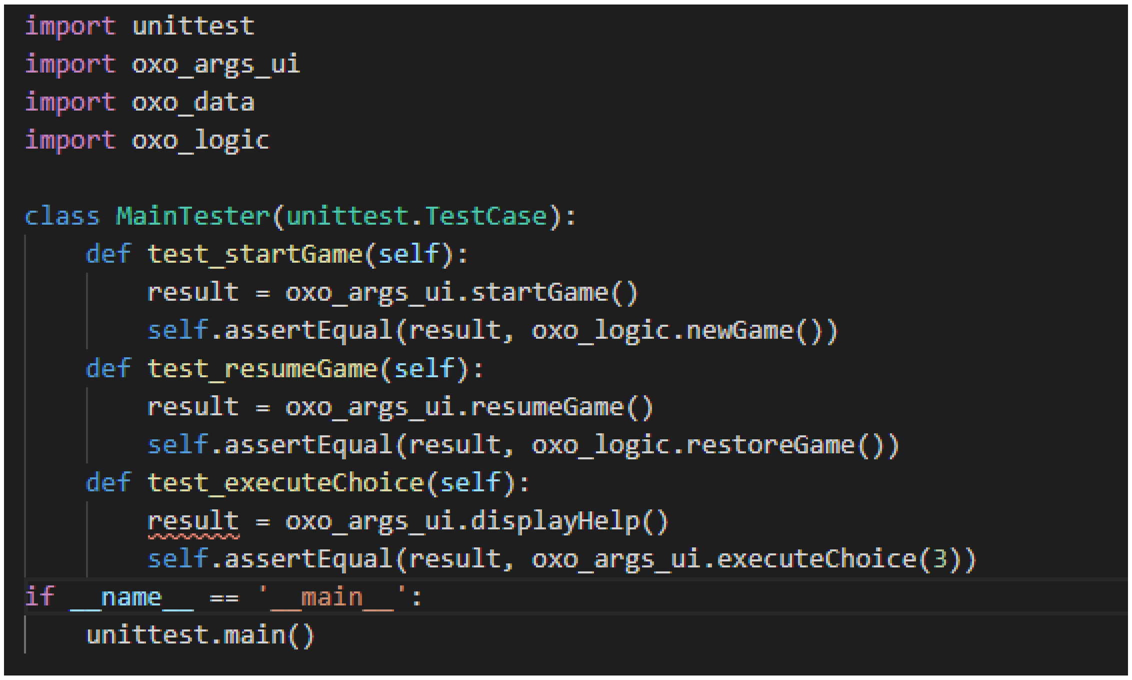


```
(base) D:\ANADIR>python
Python 3.8.5 (default, Sep  3 2020, 21:29:08) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import tkinter.filedialog as fd
>>> target = fd.askopenfilename()
>>> for line in open(target):
...     print(line, end='')
...
THIS IS A TEST

CPE 106L - PROBLEM 2>>>
```

Figure 2.2

After pressing enter a couple more times, this will show up. The contents below the commands are the contents of the test .txt file we selected when we were prompted to select a file.

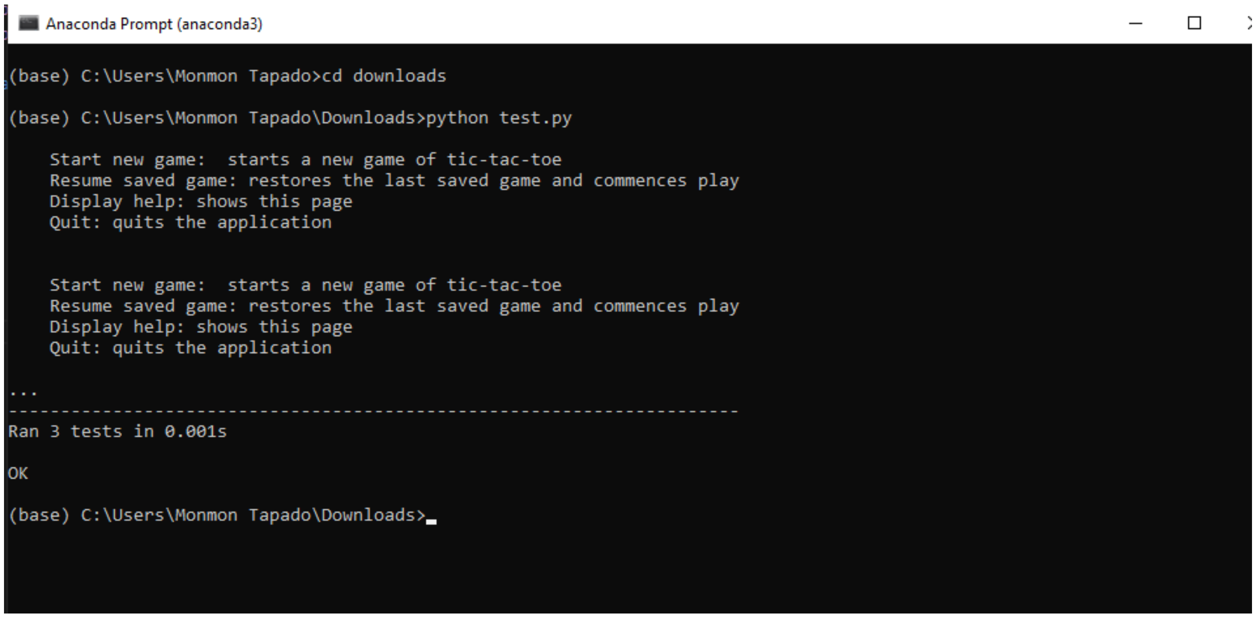
Problem 3:



```python
import unittest
import oxo_args_ui
import oxo_data
import oxo_logic

class MainTester(unittest.TestCase):
    def test_startGame(self):
        result = oxo_args_ui.startGame()
        self.assertEqual(result, oxo_logic.newGame())
    def test_resumeGame(self):
        result = oxo_args_ui.resumeGame()
        self.assertEqual(result, oxo_logic.restoreGame())
    def test_executeChoice(self):
        result = oxo_args_ui.displayHelp()
        self.assertEqual(result, oxo_args_ui.executeChoice(3))
if __name__ == '__main__':
    unittest.main()
```

Figure 3.1.

In Figure 3.1, we tested the basic functions that the user will interact at the start of the program. We tested if the user could start and resume a game, and if the instruction will be displayed properly.



Anaconda Prompt (anaconda3)

```
(base) C:\Users\Monmon Tapado>cd downloads

(base) C:\Users\Monmon Tapado\Downloads>python test.py

    Start new game:  starts a new game of tic-tac-toe
    Resume saved game: restores the last saved game and commences play
    Display help: shows this page
    Quit: quits the application


    Start new game:  starts a new game of tic-tac-toe
    Resume saved game: restores the last saved game and commences play
    Display help: shows this page
    Quit: quits the application

...
----------------------------------------------------------------------
Ran 3 tests in 0.001s

OK

(base) C:\Users\Monmon Tapado\Downloads>
```

Figure 3.2.

As shown in figure 3.2, the program ran, and it encountered no errors.