PRT582 SOFTWARE ENGINEERING: PROCESS AND TOOLS

REPORT OF SOFTWARE UNIT TESTING OF HANGMAN PROJECT (CDU SYDNEY CAMPUS)

STUDENT ID : S394325

STUDENT NAME : BIBEK KC

INTRODUCTION

The aim of this project is to implement the classic Hangman game using Python, applying Test-Driven Development (TDD) principles and automated unit testing. Here the project includes both a command-line interface (CLI) and a graphical user interface (GUI) built with framework, Tkinter. Python was chosen as the programming language because of its simplicity, readability, and strong ecosystem for testing. For automated unit testing, the unittest framework is used, as it is part of Python's standard library and integrates seamlessly with TDD.

PROCESS

We followed a TDD approach, which means writing unit tests before implementing new features in the game engine (engine.py). TDD is an iterative development process where tests are written before the actual code, ensuring that every functionality is validated as soon as it is implemented. Here by doing this, ensures correctness and prevents regressions. The process is implemented by following steps:

1. Writing the first failing test
   The development started by identifying the core functionalities required for the game engine, such as word masking, guessing letters, life deduction, and timeout handling. For each feature, a unit test was first written using the unittest module in Python. For example, before implementing the guess() method, a test was created to verify that a correct guess reveals the appropriate letters and an incorrect guess deducts a life.

2. Implementing the Minimum Code
   After writing a failing test, the minimum amount of code was implemented in the engine.py file to make the test pass. For instance, logic was added to deduct a life when the guessed letter was incorrect, or to update the masked word when the guessed letter was correct.

3. Running Automated Unit Tests
   The unittest framework is being used to automate the execution of all test cases. Running the test suite after each change provided immediate feedback on whether the new code met the expected requirements. For example, tests confirmed that the game ends when lives reach zero, that repeated guesses do not deduct extra lives, and that the timer deducts a life if no input is made within 15 seconds.

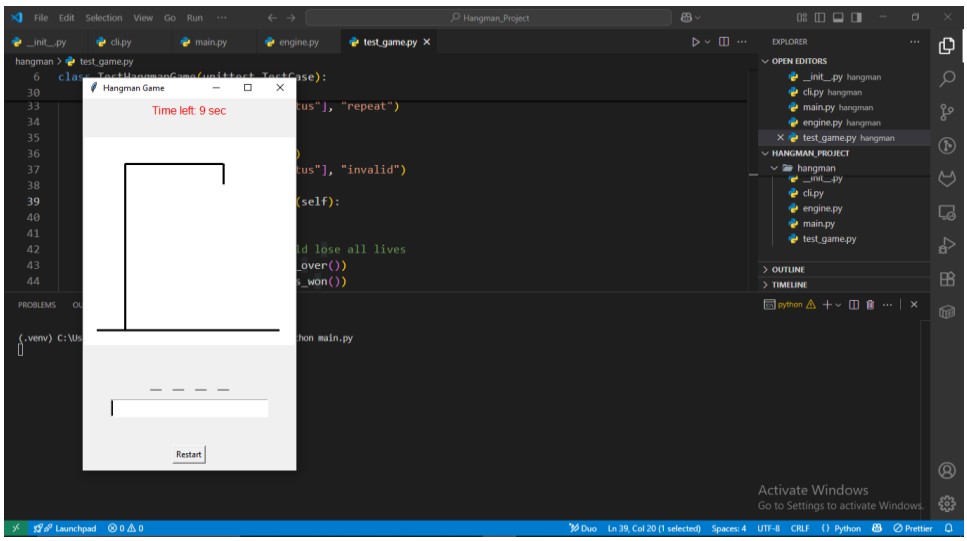
4. Refactoring and Extending
   Once all tests for a feature passed, the code was refactored to improve readability and maintainability without altering functionality. Additional tests were then added to cover edge cases, such as handling invalid input (e.g., numbers or multiple characters). This process was repeated until all the game requirements were covered.

5. Integration with GUI
   After the game engine was fully tested, it was integrated with a Tkinter-based GUI. The GUI interacts with the engine by calling its methods and updating the display, while the correctness of the game logic continues to be guaranteed by the test suite.

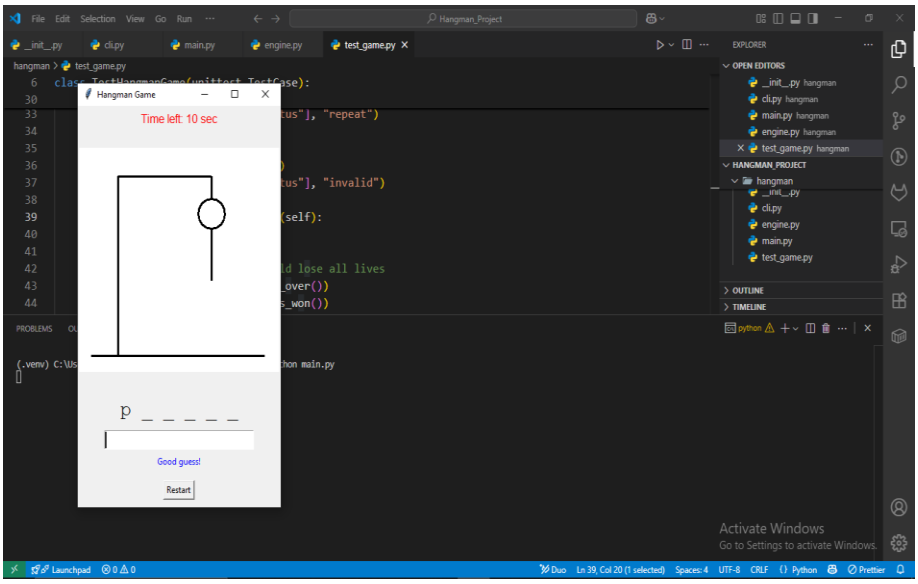
Relavant Screenshots:

Here to program is run from the GUI from the main.py file. I have also implemented the testin framework and can be test by calling the testing file (test_game.py)

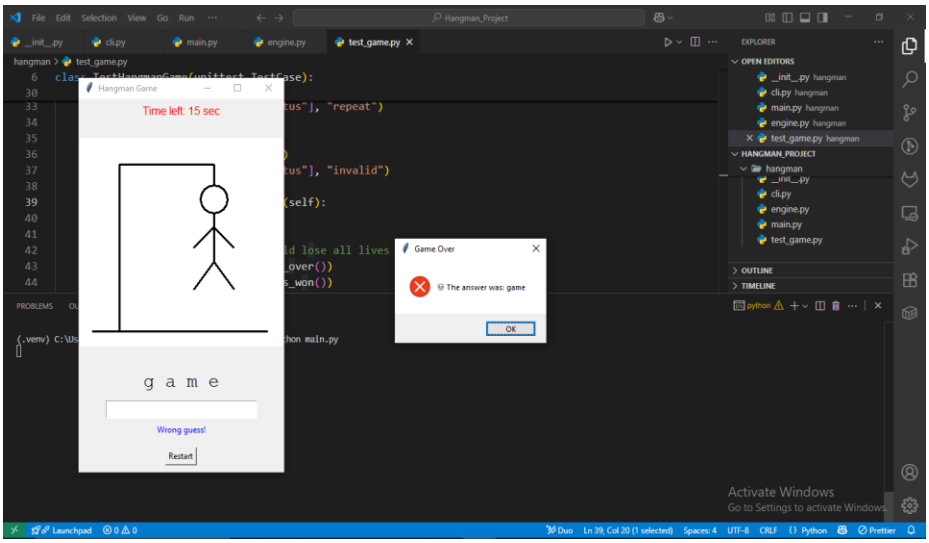
1. When program runs



2. Game Running



3. When game ends



4. Overall Testing of the program using Unitest Framework

```
(.venv) C:\Users\ACER\Desktop\Hangman_Project\hangman>python main.py

(.venv) C:\Users\ACER\Desktop\Hangman_Project\hangman>python test_game.py
.........
----------------------------------------------------------------------
Ran 9 tests in 0.001s

OK
```

CONCLUSION

The development of the Hangman game using Test-Driven Development (TDD) and Python's unittest framework provided me a valuable lesson in structured and reliable software development. By writing tests before implementation, the project maintained a strong alignment between requirements and code functionality. This approach ensures that each feature such as letter guessing, word masking, life deduction, and timer functionality was validated as soon as it was developed.

One of the key takeaways was the importance of incremental development. Implementing the game in small, testable units reduced complexity and made debugging easier. Automated unit testing created confidence that changes or refactoring did not break existing functionality.

Additionally, integrating the Tkinter-based GUI assisted to see the proper outcomes of the program. The use of TDD also revealed areas for improvement.

Overall, the project successfully met the requirements, implementing Hangman with both word and phrase levels, enforcing a 15-second timer, deducting lives for wrong or late guesses, and providing a functional graphical user interface.

GITHUB LINK: https://github.com/Wiwek09/HangMan_Report

REFERENCES

1. Beck, K. (2003). Test-Driven Development
2. George, B., & Williams, L. (2004). A structured experiment of test-driven development. Information and Software Technology, 46*(5), 337–342.*
3. Meszaros, G. (2007). xUnit Test Patterns: Refactoring Test Code*. Addison-Wesley.*