**S225 PRT582 SOFTWARE ENGINEERING: PROCESS AND TOOLS**

**Assessment 02**

**Lecturer name: Muhammad Rana**

**Student Name: Susmita Sharma**

**Student ID: S394867**

**Campus Sydney**

**Submitted on 05/09/2025**

Contents

[**Introduction** 2](#_Toc208002533)

[**Process** 3](#_Toc208002534)

[Step 1: Wring the First test 3](#_Toc208002535)

[Step 2: Implement the test cases code. 7](#_Toc208002536)

[Requirement of the project showing all the relevant screenshot 7](#_Toc208002537)

[Refactoring of code 17](#_Toc208002538)

[Automate Unit Testing 18](#_Toc208002539)

[Code Quality and Static Analysis 19](#_Toc208002540)

[**Conclusion** 22](#_Toc208002541)

[**References** 23](#_Toc208002542)

[Appendices 24](#_Toc208002543)

[Appendix 1: Code snippet 24](#_Toc208002544)

[Appendix 2: Unit test result 30](#_Toc208002545)

[Appendix 3: Screenshot 30](#_Toc208002546)

# **Introduction**

The main objective of this project is to create Hangman Game in Python Programming following a Test Driven Development (TDD) approach, making sure of robust functionality through automated unit testing. Hangman is a popular old school favorite a word guessing game where the players guess a hidden word by suggesting letters. The player has a limited number of life that is seven, where every wrong guess reduces their chances of winning the game. The project has the requirement of the game having two levels that is Basic and Intermediate levels where basic refers to choosing single word randomly and intermediate refers to choosing multi word phrase randomly where the letter that have not been guessed are hidden as underscore and guessed letters are reveled to the players. The player has 15 seconds to guess the word if time runs out one life is lost and also each incorrect guess deducts the life the game continues until the player guesses the word correctly for their seven chances. After each game, the player can choose to play again or terminate the game. Python was chosen for this project due to its easy to read and is widely used in software development and education. For automated testing, pytest was used, which makes it easy to write small, clear test and check that each part of the game works correctly. Using TDD with pytest ensures that the code is reliable and bugs are caught early.

# **Process**

For the completion of this assignment, Test Driven Development Approach(TDD) is used. In simple terms, first small unit test that described what Hangman game was meant to do. Only after that worked on actual code to make the tests pass. In this way keeping the track of code weather each requirement was properly working as program was build.

Pytest testing tool is used as its easy to setup, runs fast and clearly shows which tests pass and which ones fail. Whenever the changes were made to the game pytest command was run throughout the development to check if something was broken by mistake.

## Step 1: Wring the First test

Started by writing a test to check if a word is generated correctly in the basic level, at this stage the test failed because there was no game logic yet. This is expected in TDD, fist we write the test code, check if its failing then write the minimum code to make it pass.

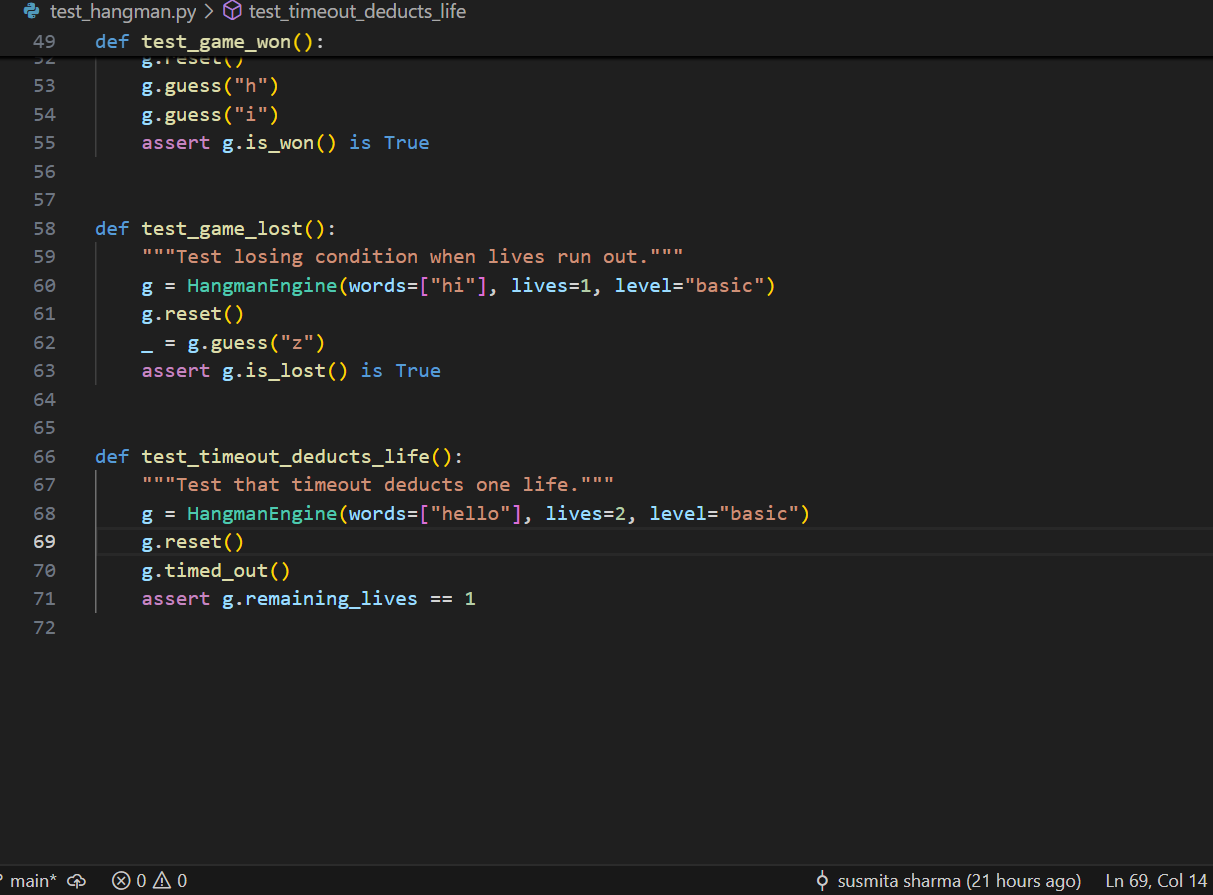
Screenshot showing all the test cases



*Figure 1: Code showing test cases of hangman*

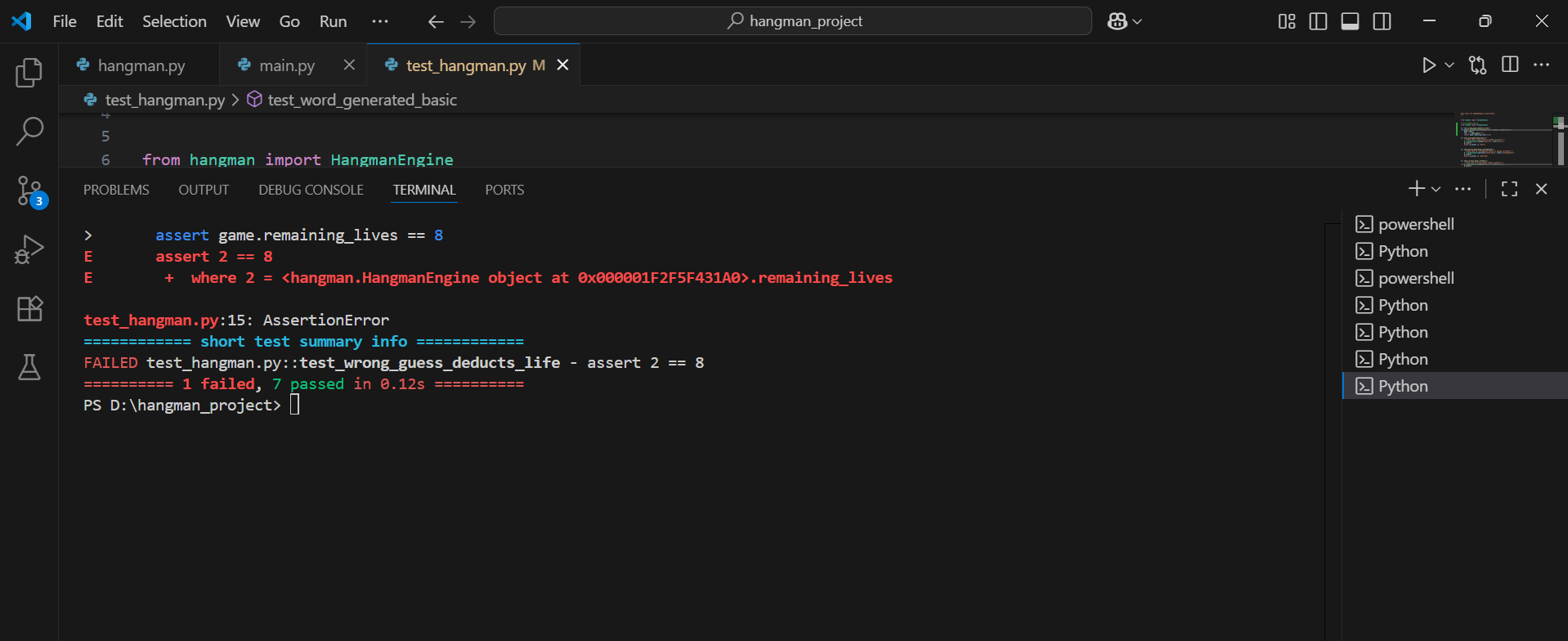


*Figure 1.1: Code showing test cases of hangman*



*Figure 1.2: Code showing test cases of hangman*

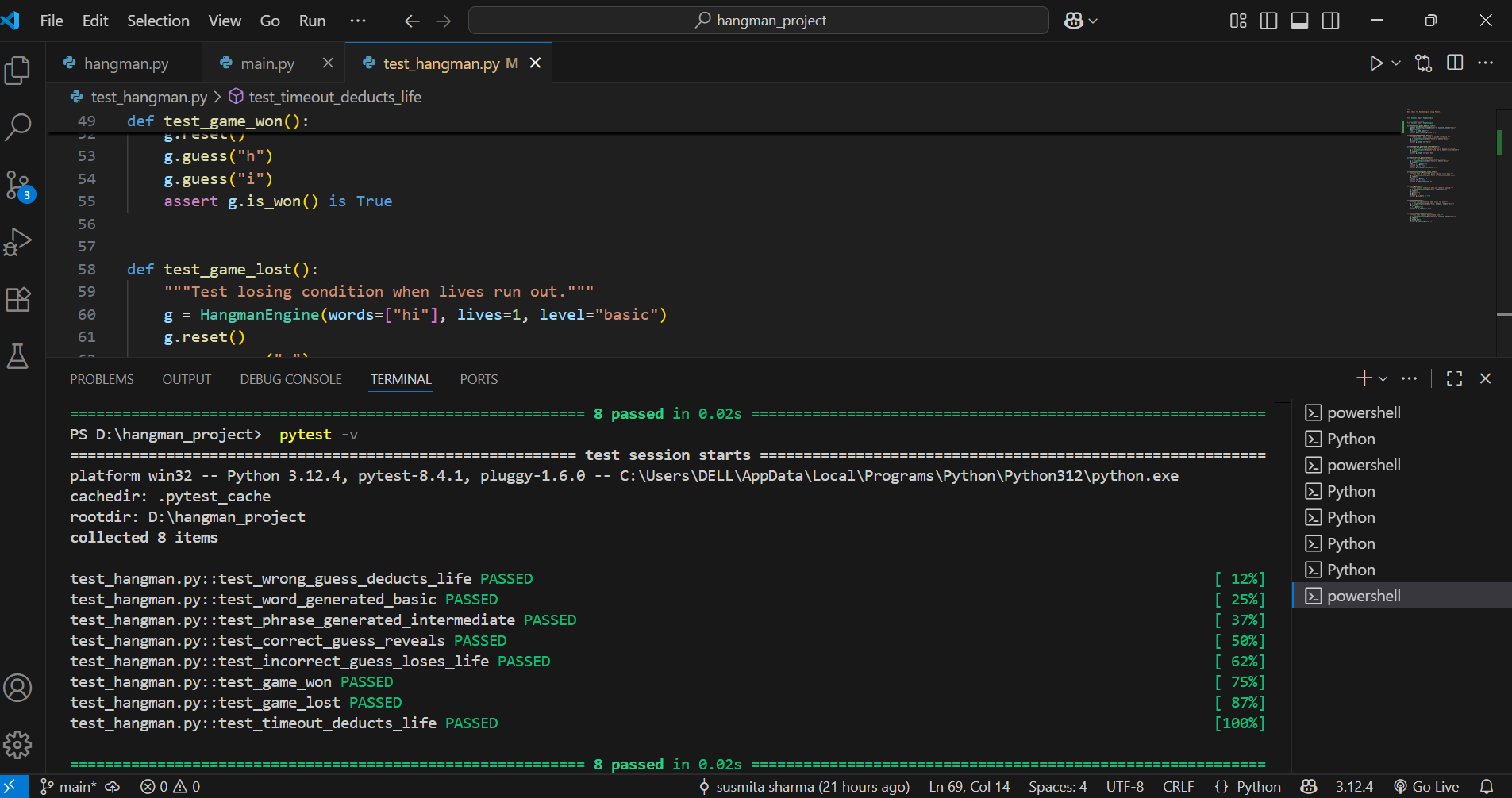
Screenshot showing Failed pytest output showing missing function or incorrect result.



*Figure 2: Pytest output showing failing tests*

## Step 2: Implement the test cases code.

Screenshot showing all the test cases passed after implementing the hangman code.



*Figure3: Pytest output showing all test cases passing after implementing hangman code*

## Requirement of the project showing all the relevant screenshot

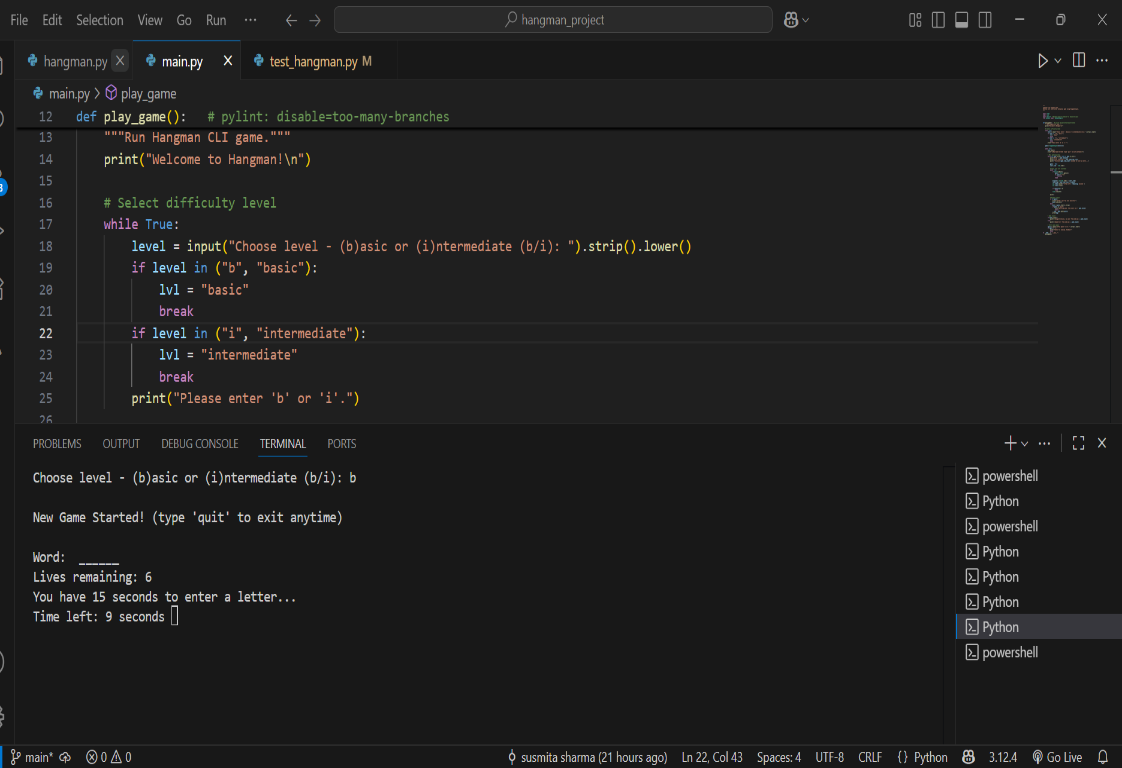
The following set of requirements for this program:

1. There are two levels:



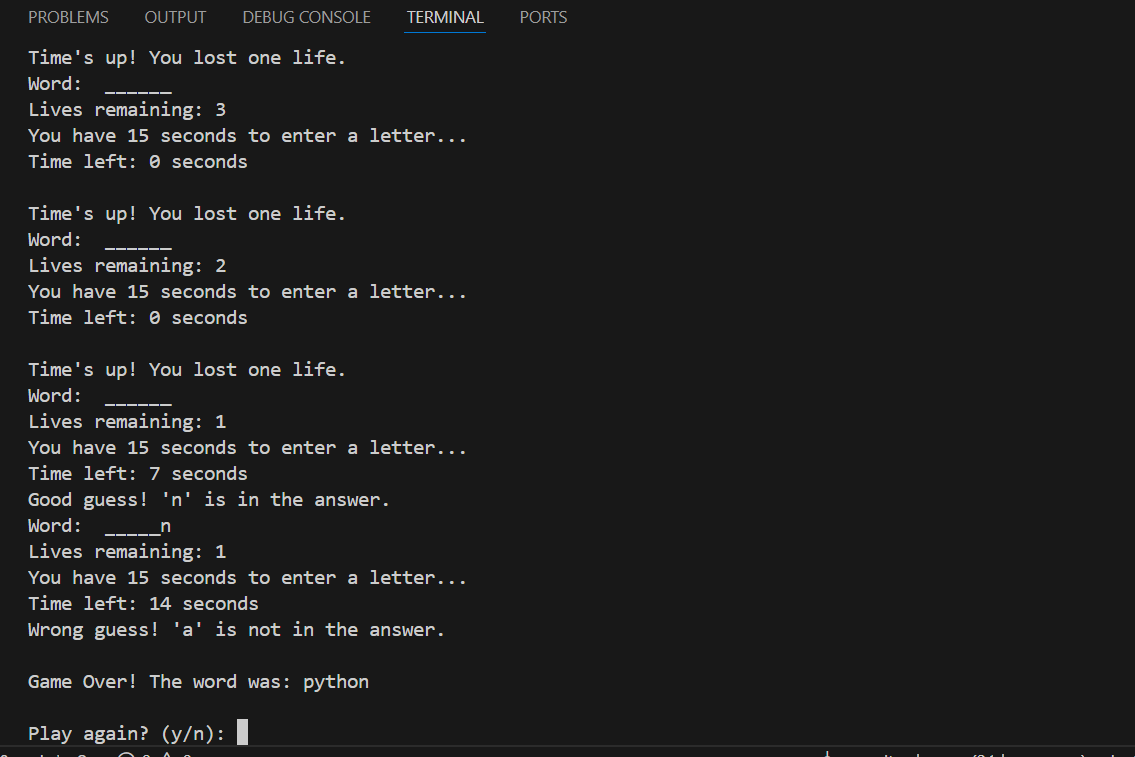
*Figure 4: Gameplay screenshot showing two levels intermediate and basic*

* 1. Basic: a word is generated randomly



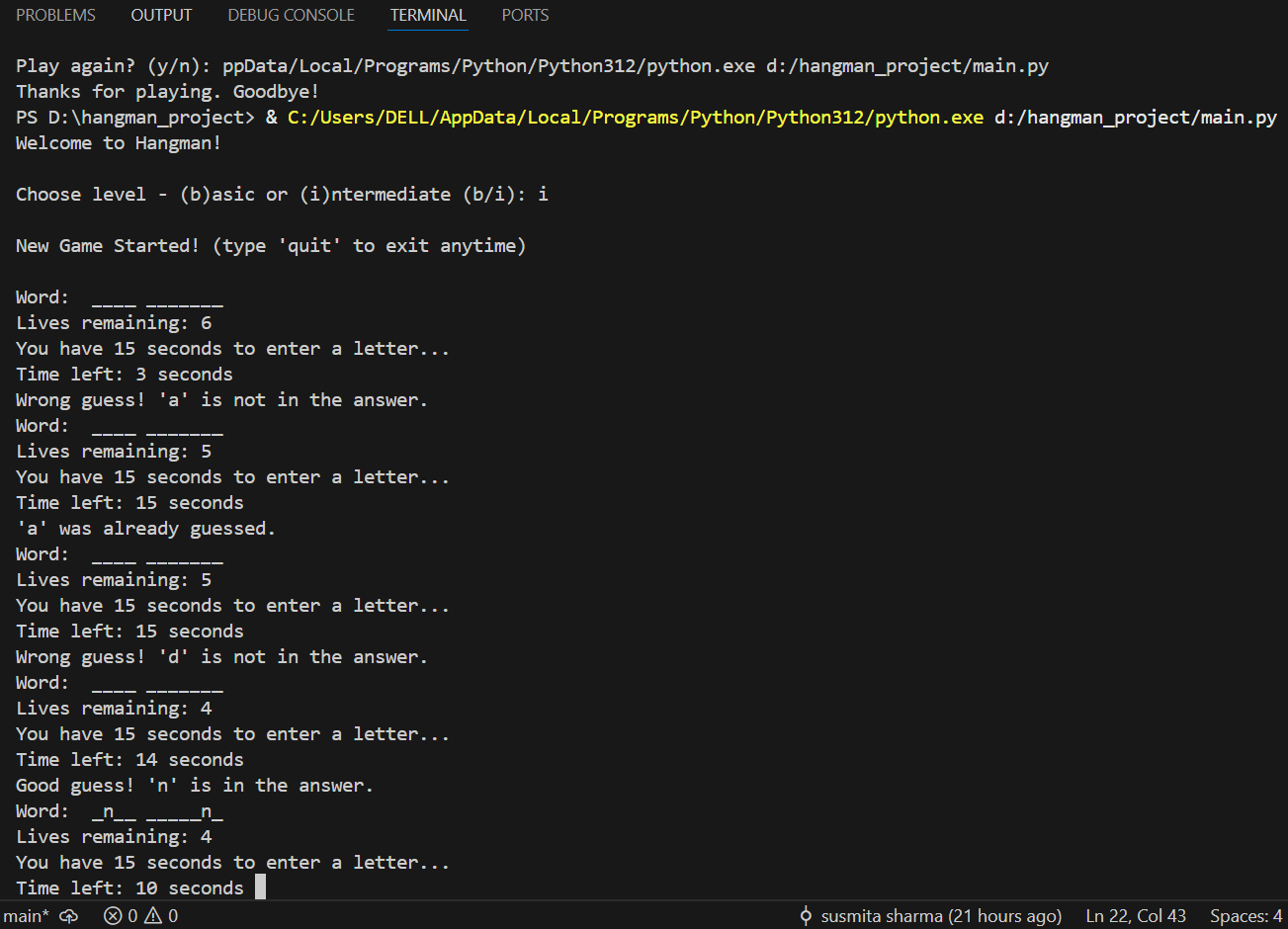
*Figure 5: Gameplay screenshot showing basic level*

Screenshot showing the basic level game ask the user to guess letter.



*Figure 6: Gameplay screenshot showing basic level and guessing the letters*

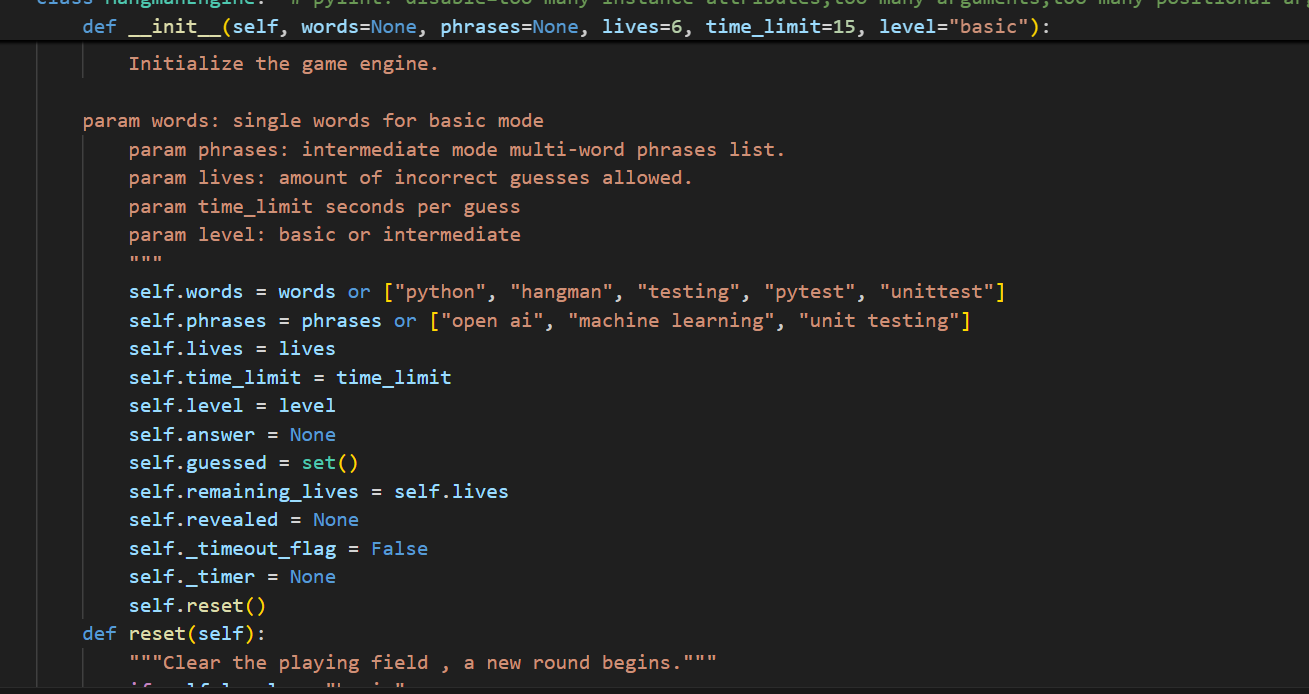
* 1. Intermediate: a phrase is generated randomly



*Figure 7: Gameplay screenshot showing intermediate* *player guessing phrase.*

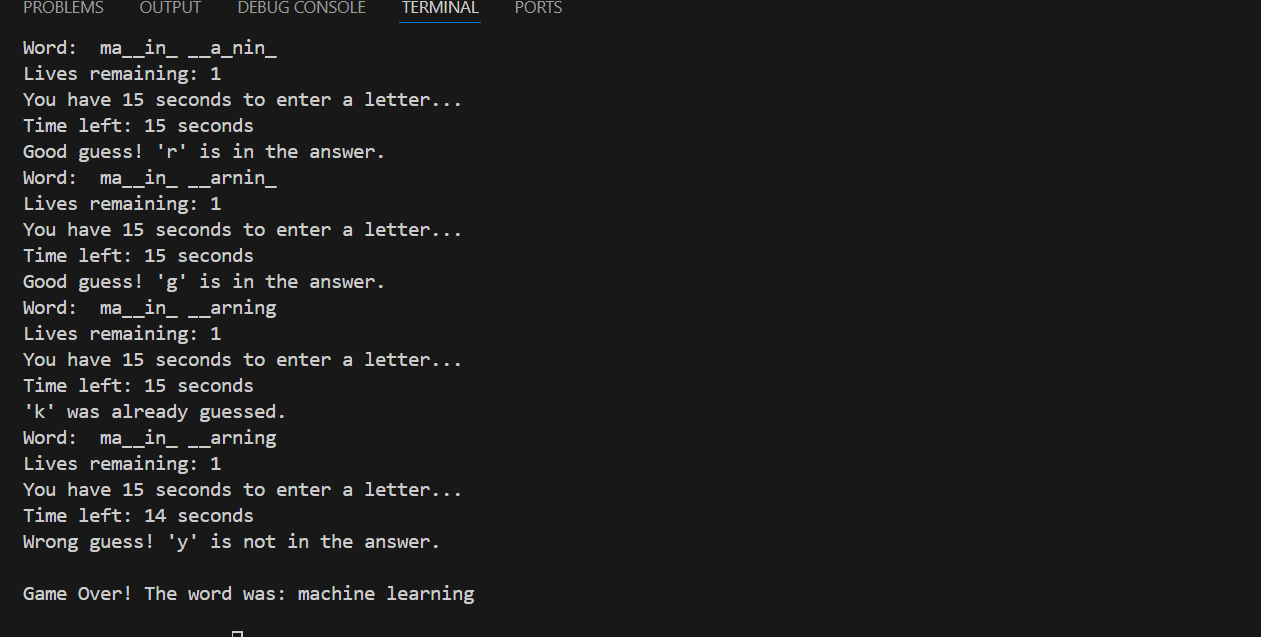
1. The word/phrase must be valid word(s) from a dictionary

Screenshot of the code where the words are defined for basic and intermediate level which are the valid words that will be displayed to user to guess from.



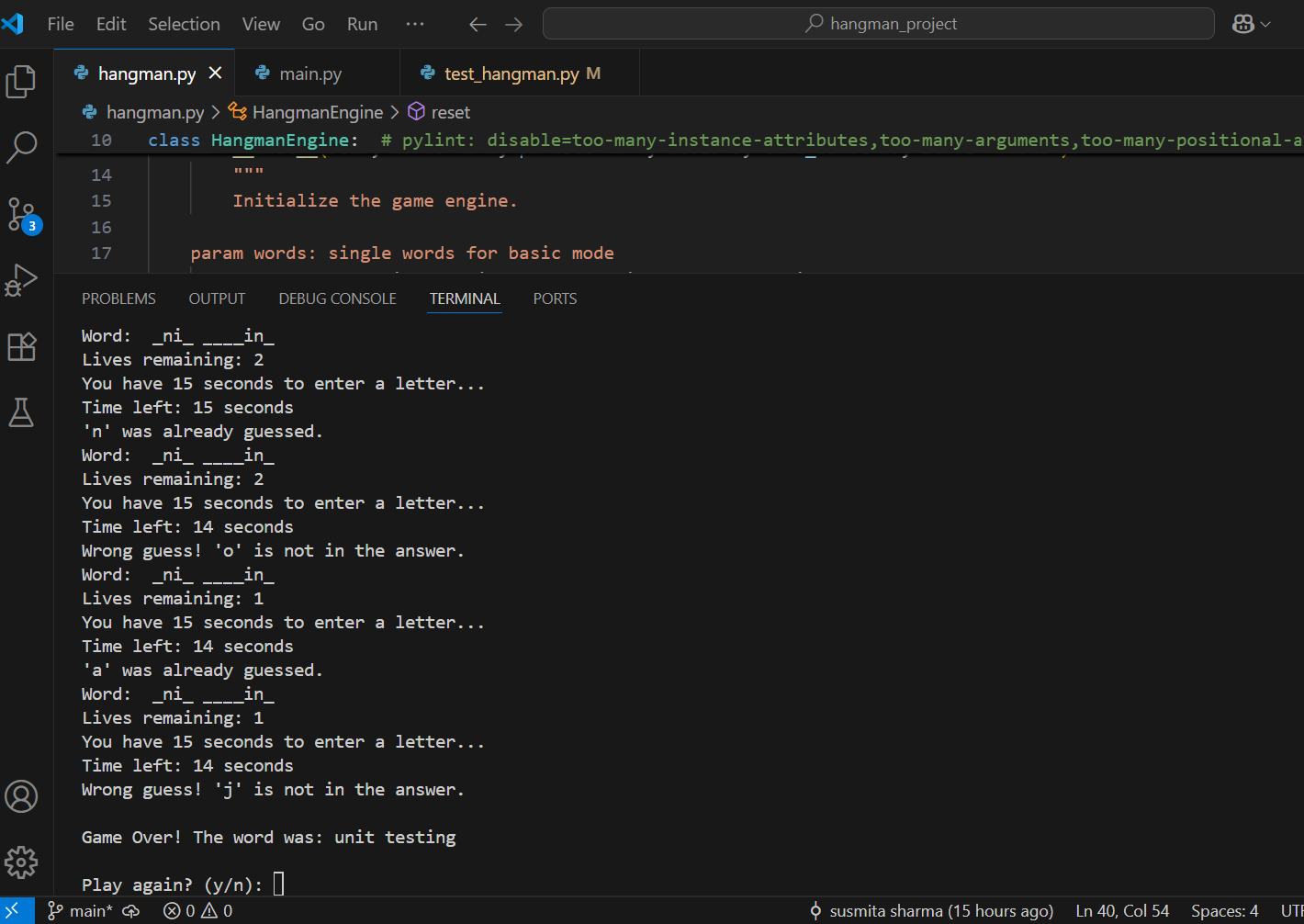
*Figure 8 : Code showing word/phrase valid word from a dictionary*

Screenshot showing the words are generates from the Dictionary



*Figure 9: Screenshot showing the words are generates from the Dictionary*

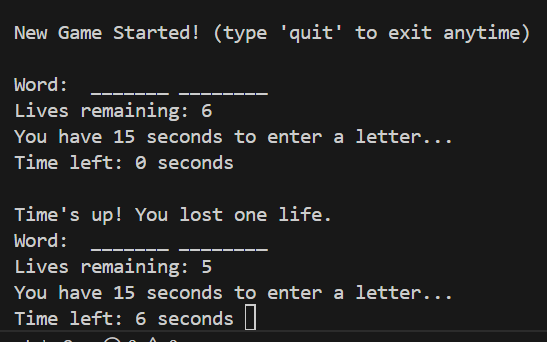
1. Screenshot showing Player will be presented with underscores representing the missing letters the player needs to find.



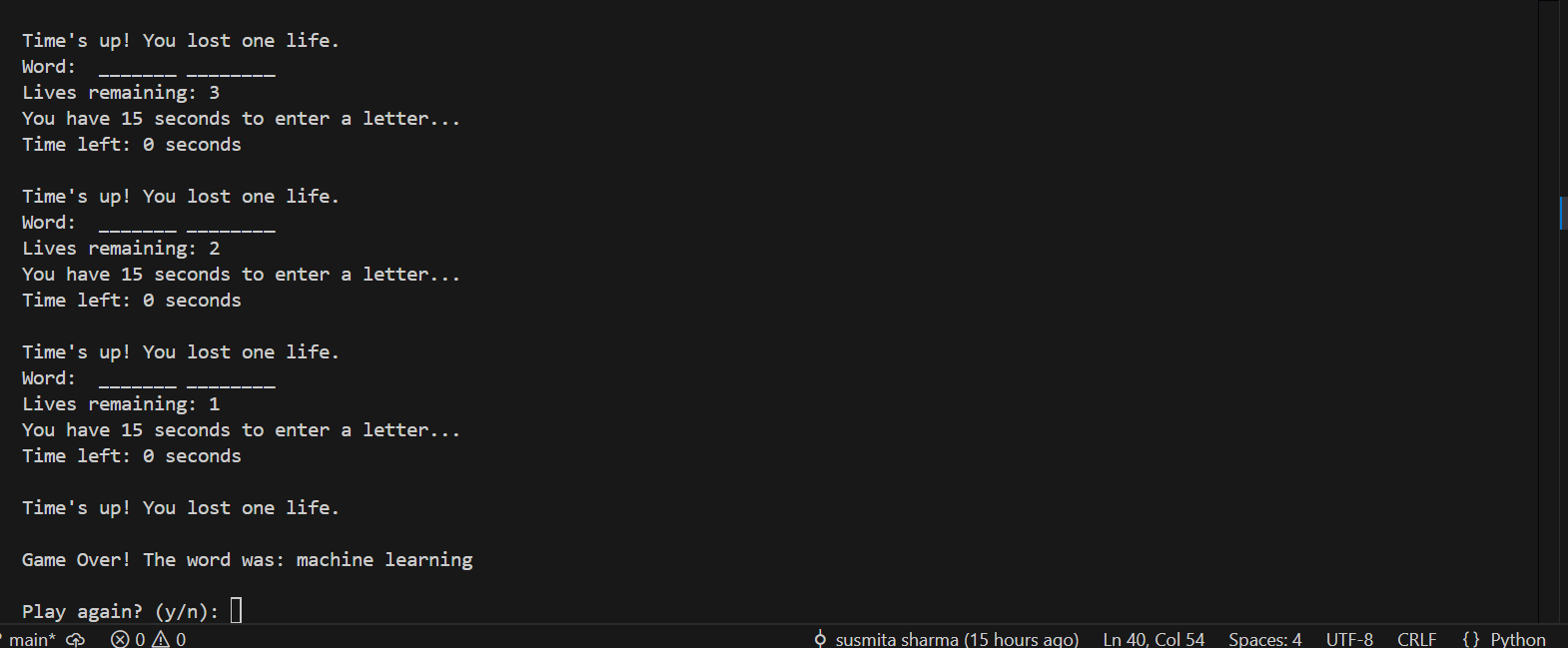
*Figure 10: Screenshot showing player presented with underscores for hidden letters.*

1. Player has 15 seconds to make a guess (Timer must be shown). If time is up, a life will

be deducted.

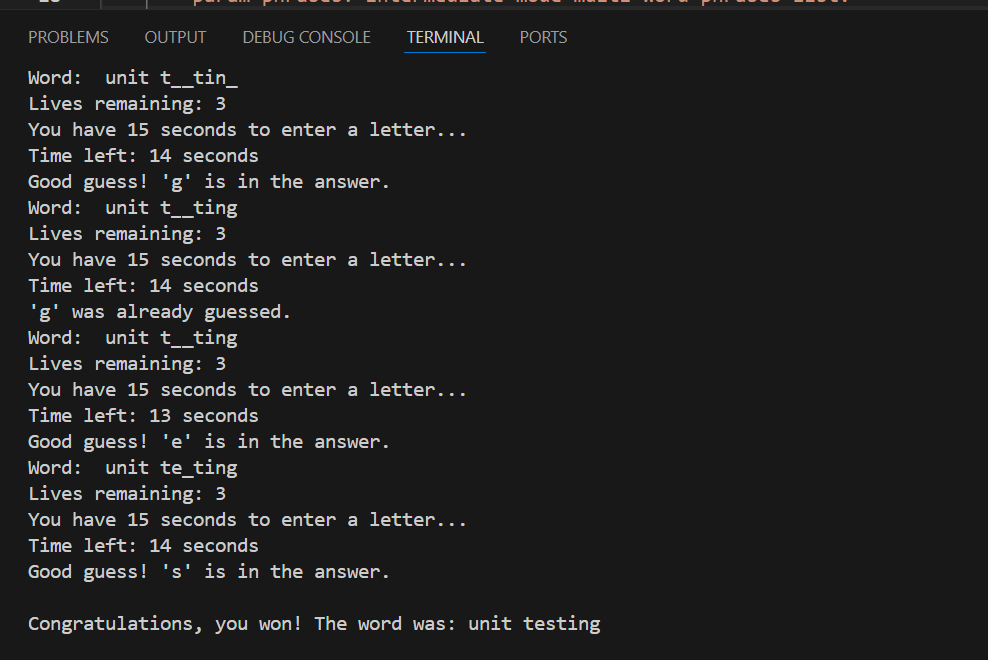


*Figure 11: Screenshot on timeout and losing life if guessed wrong.*



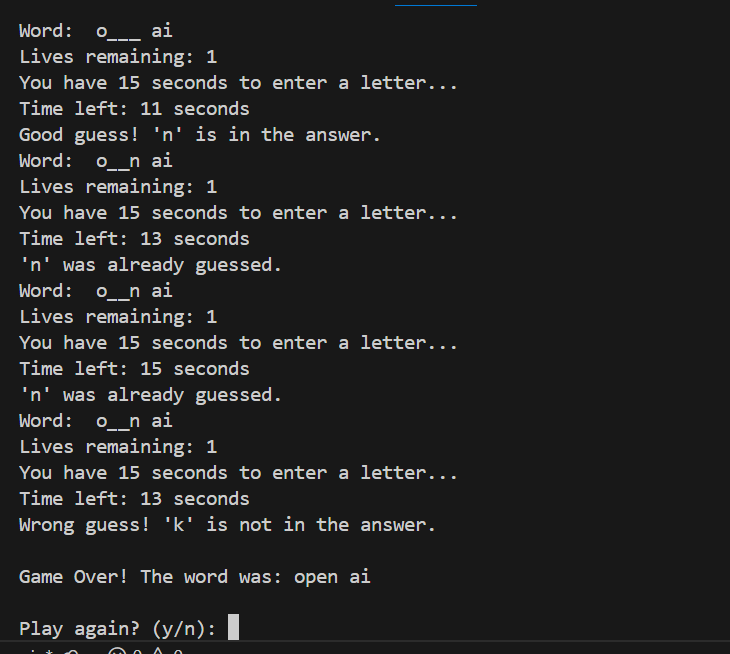
*Figure 11.1: Screenshot on timeout and losing life if guessed wrong.*

1. If the player’s chosen letter exists in the answer, then all places in the answer where that letter appear will be revealed.



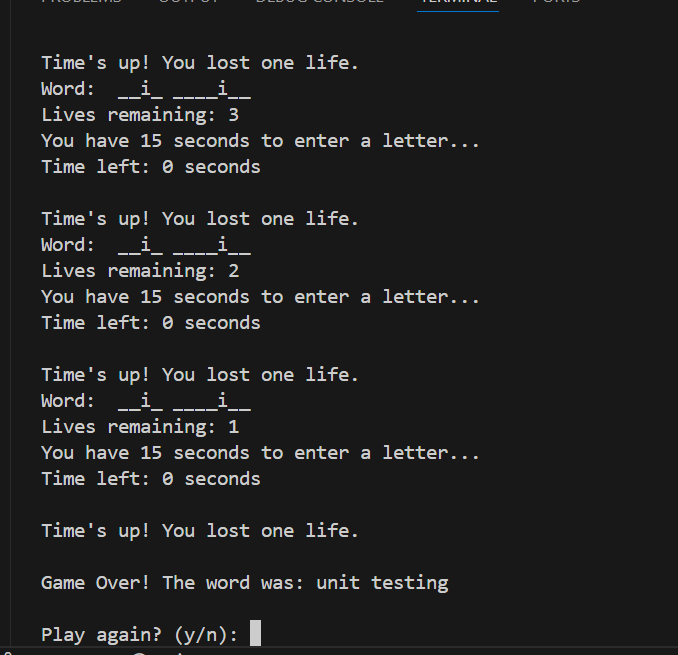
*Figure 12: Gameplay screenshot showing a correct guess revealing all matching letters.*

1. Every time the player guesses a letter wrong, the player’s life will be deducted.



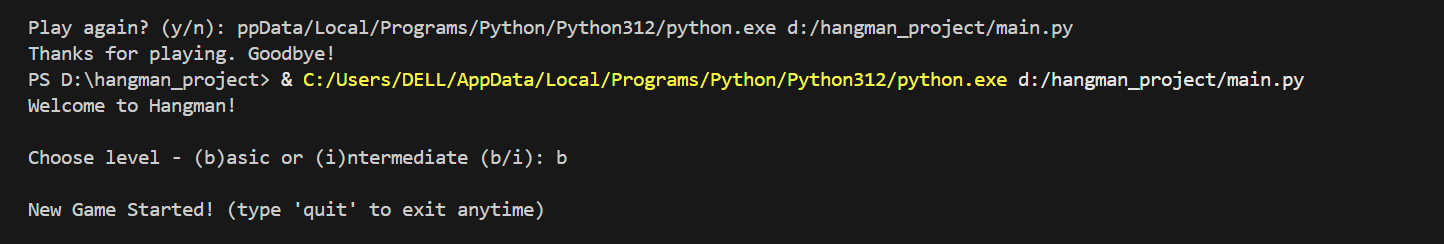
*Figure 13:* *Gameplay screenshot showing a wrong guess deducting a life.*

1. The player must find the missing word before the player’s life becomes zero.



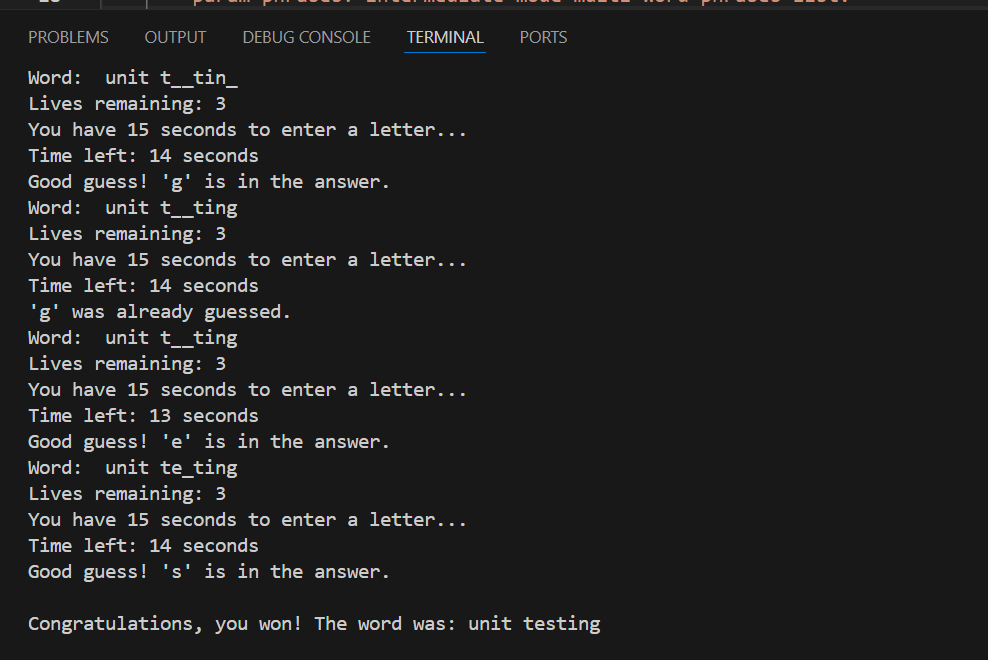
*Figure 14:* *Gameplay screenshot showing the game ending when lives reach zero.*

1. The game will keep going:
   1. Until the player quits the game or life becomes zero, and display the answer to the player.



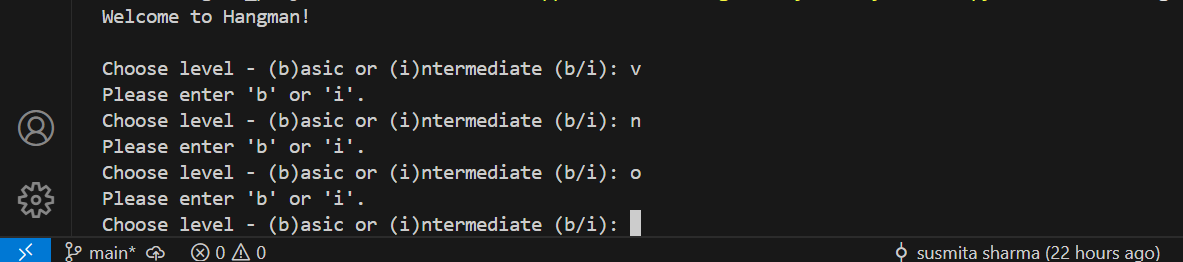
*Figure 15: Screenshot showing the game quit option*

* 1. After the player guesses the word(s) correctly.



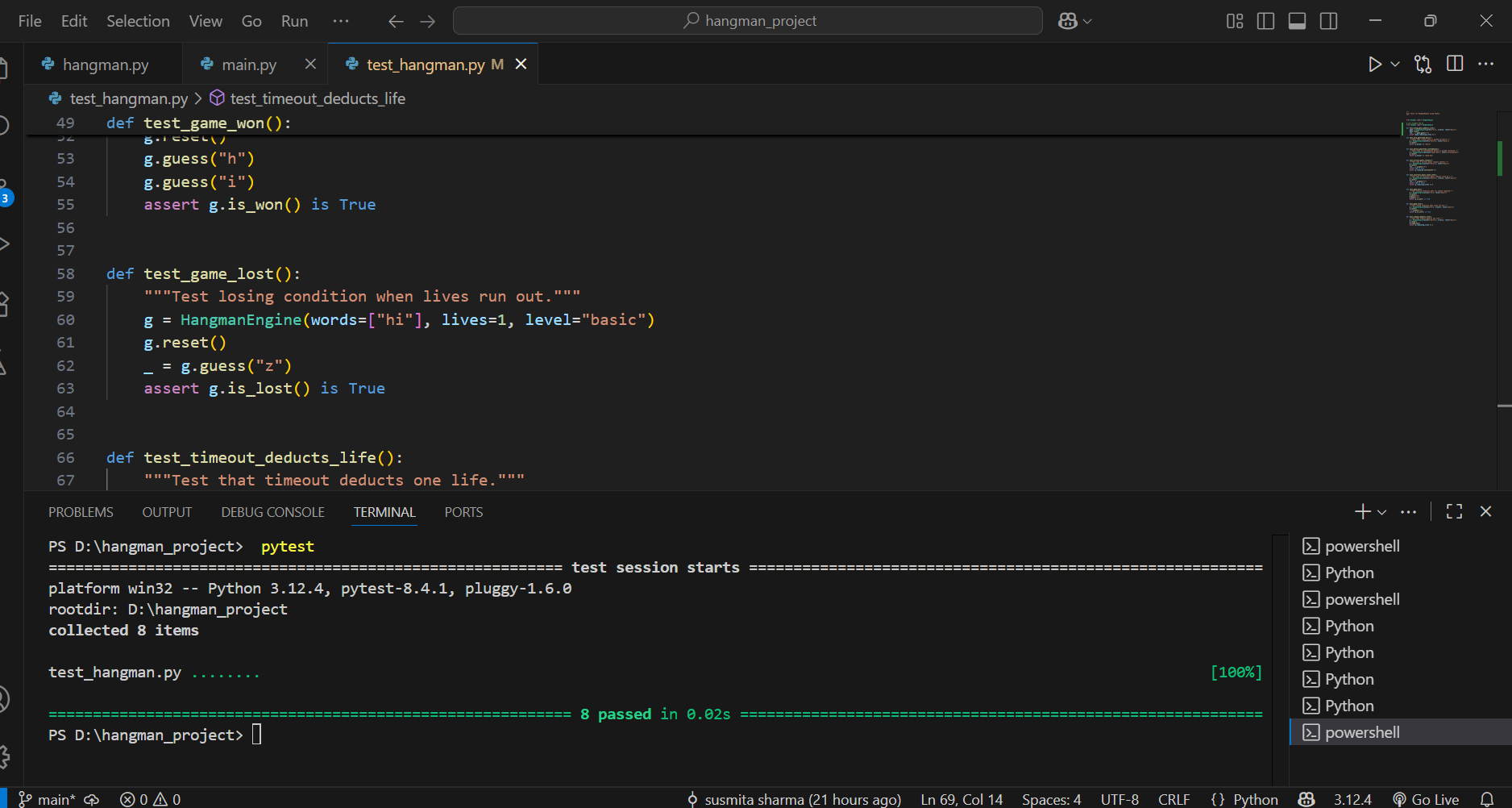
*Figure 16:Screenshot showing the guessed letters are all correct.*

Screenshot showing if user types any other level other than basic or intermediate the loops continues and asks the user to enter either I or b



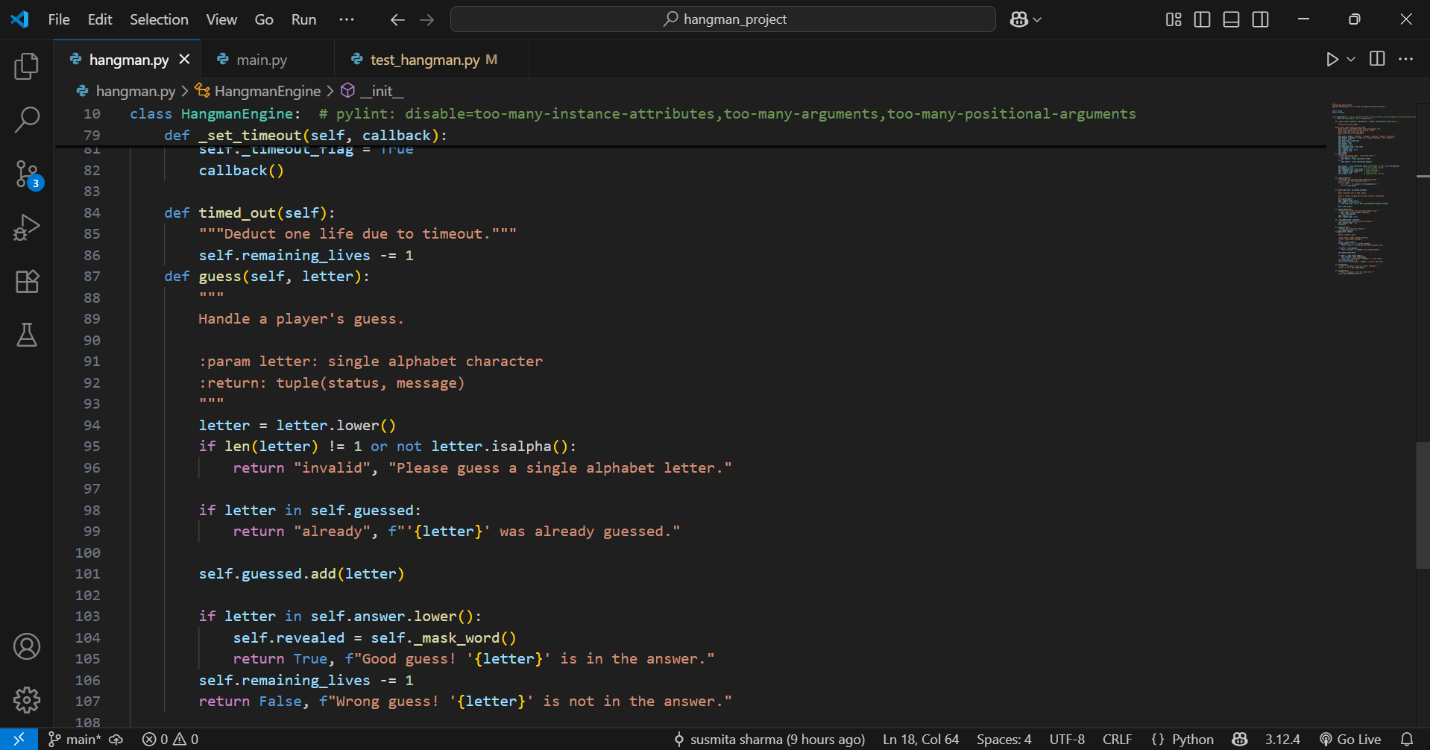
*Figure 17: Screenshot showing if user doesnot enter the desired basic or intermediate level it keeps on asking for correct one to enter.*

## Refactoring of code



*Figure 18:Screenshot showing refactoring of code.*

Screenshot showing the logic for timer



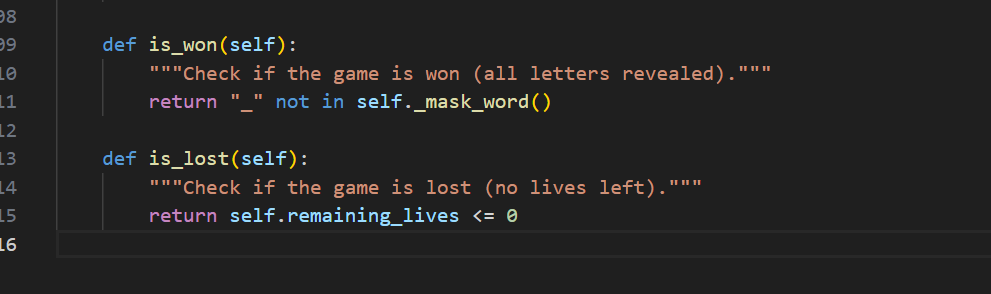
*Figure 19:Screenshot showing the logic of timeout code.*

Sreenshot showing life deduction logic from player in case of wrong letter entered each time.



*Figure 20:Screenshot showing life deduction logic in code.*

Screenshot for the logic if guessed the right word revel and if wrong lose and quit game



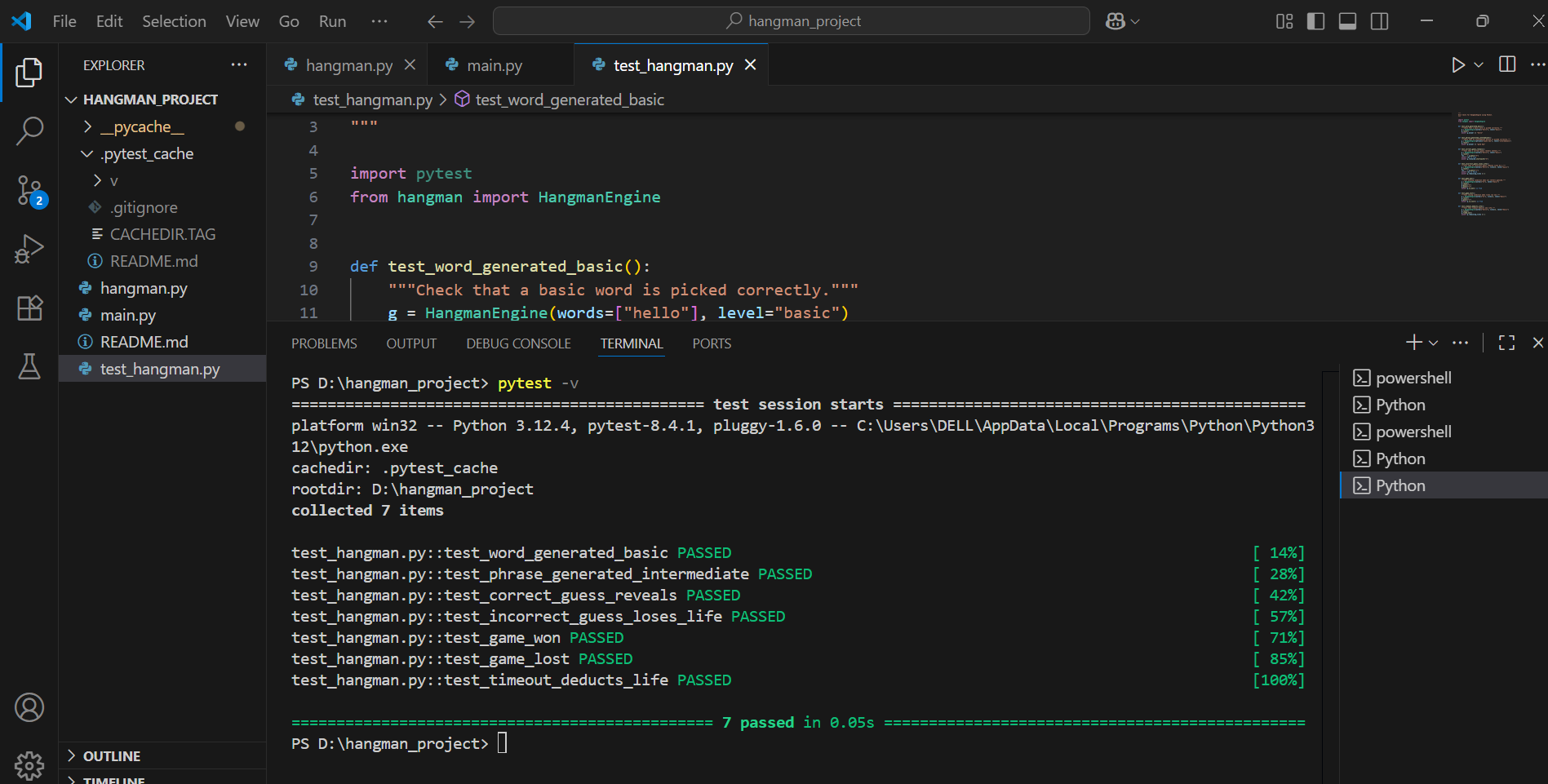
*Figure 21: Screenshot showing logic if right word revel and wrong lose or quit game logic.*

## Automate Unit Testing

Pytest was used as automated testing tool. All game features were tested using unit test

* test\_word\_generated\_basic() and test\_phrase\_generated\_intermediate() to verify word/phrase selection.
* test\_correct\_guess\_reveals() to ensure letters are revealed correctly.
* test\_incorrect\_guess\_loses\_life() and test\_timeout\_deducts\_life() to verify life deduction.
* test\_game\_won() and test\_game\_lost() to confirm win/loss conditions.

Running pytest –v confirmed that all test passed successfully. Screenshot of all the test cases are below

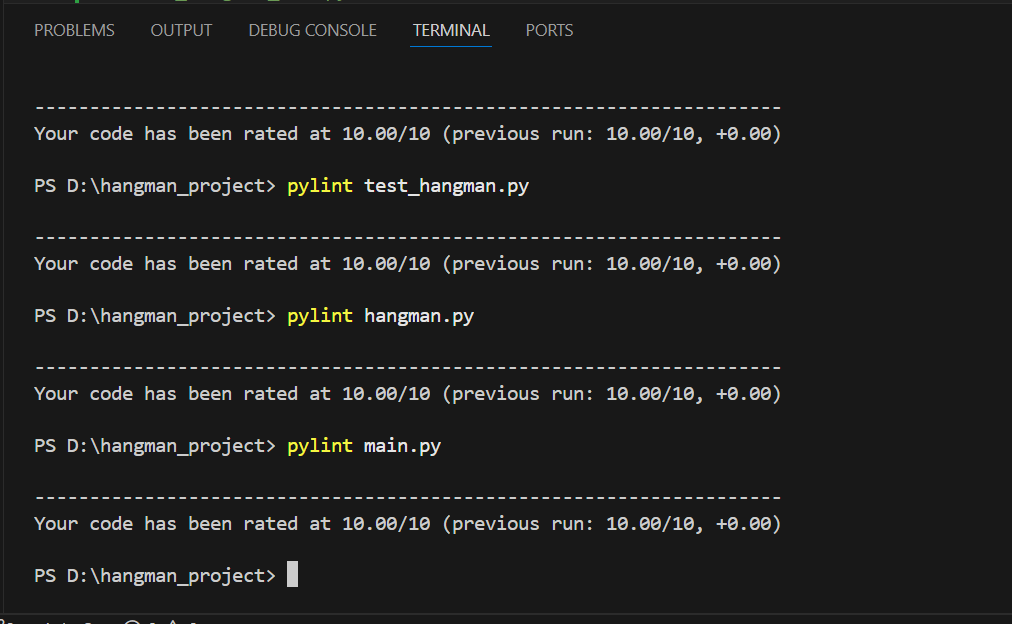


*Figure 22:Screenshot showing all the test cases being passed indicated by color green.*

## Code Quality and Static Analysis

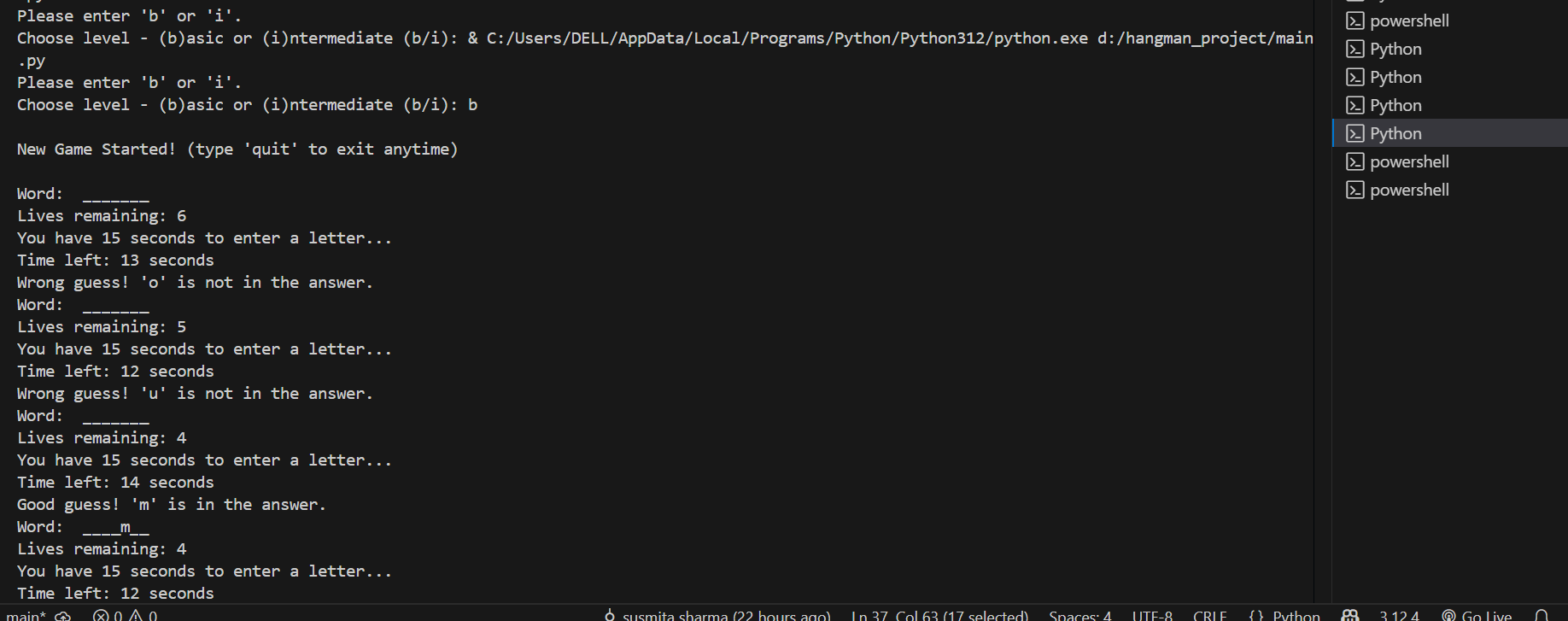
To maintain high quality code standards, pylint and flake8 were used to check for style, formatting and potential errors. The following steps were performed pylint hangman.py test\_hangman.py flake8 which reported all issues including missing docstring, line lengths, unused imports and blank line inconsistencies were corrected.

Screenshot showing after all these modifications the code achieved a pylint rating of 10/10 and passes all flake 8 checks

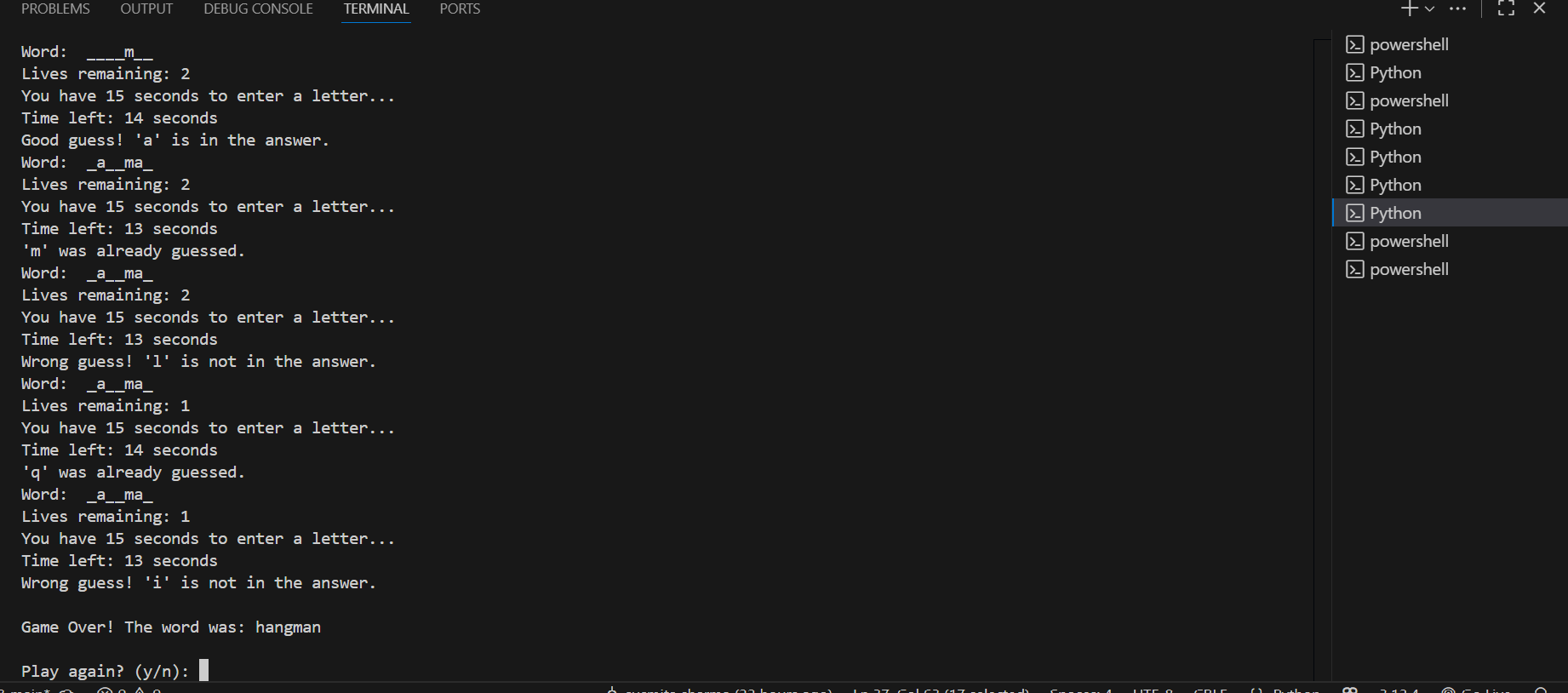


*Figure 23:Screenshot showing pylint rating for all the code.*

Game play screenshot for basic level



*Figure 24:Screenshot showing gameplay of basic level*

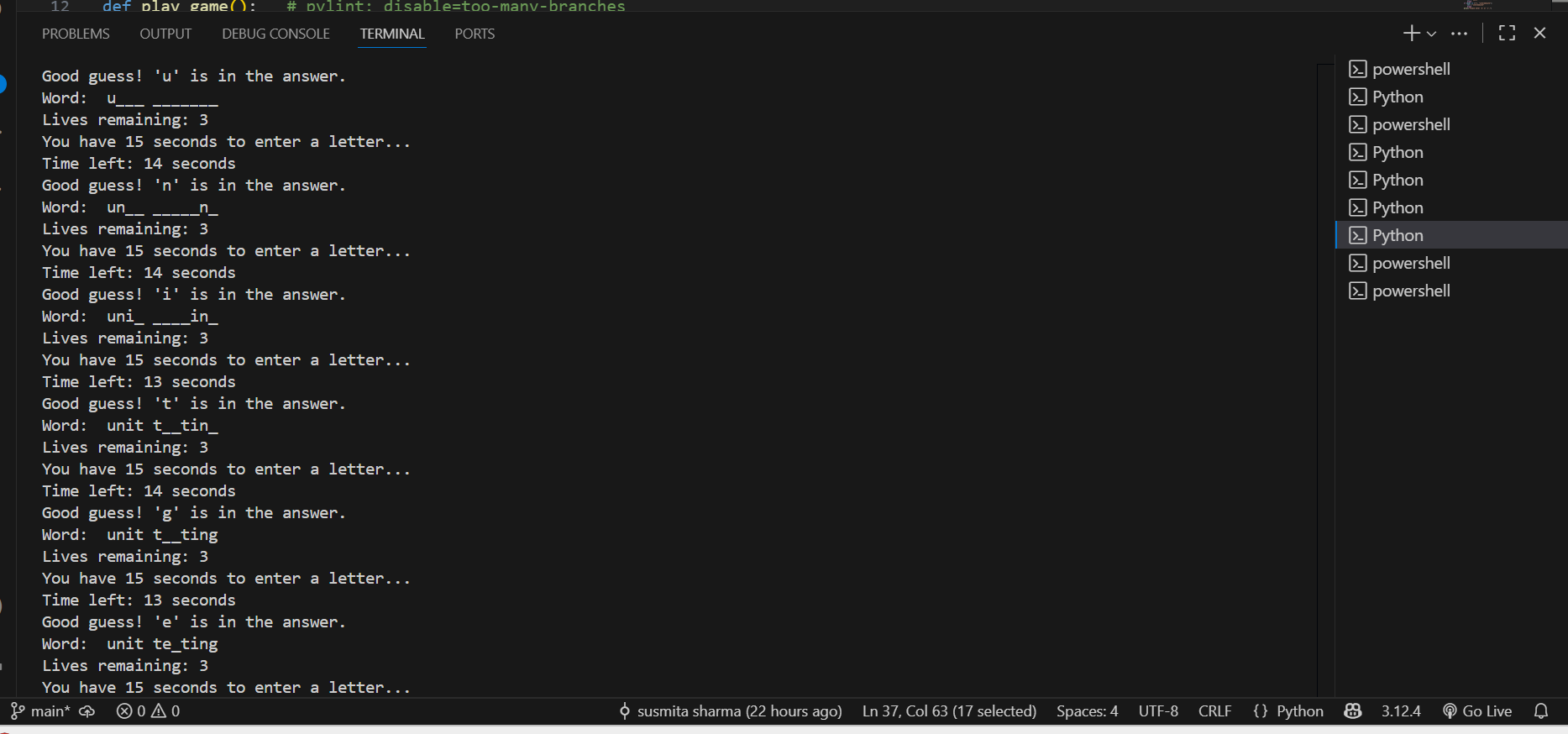


*Figure 24.1 :Screenshot showing gameplay of basic level*

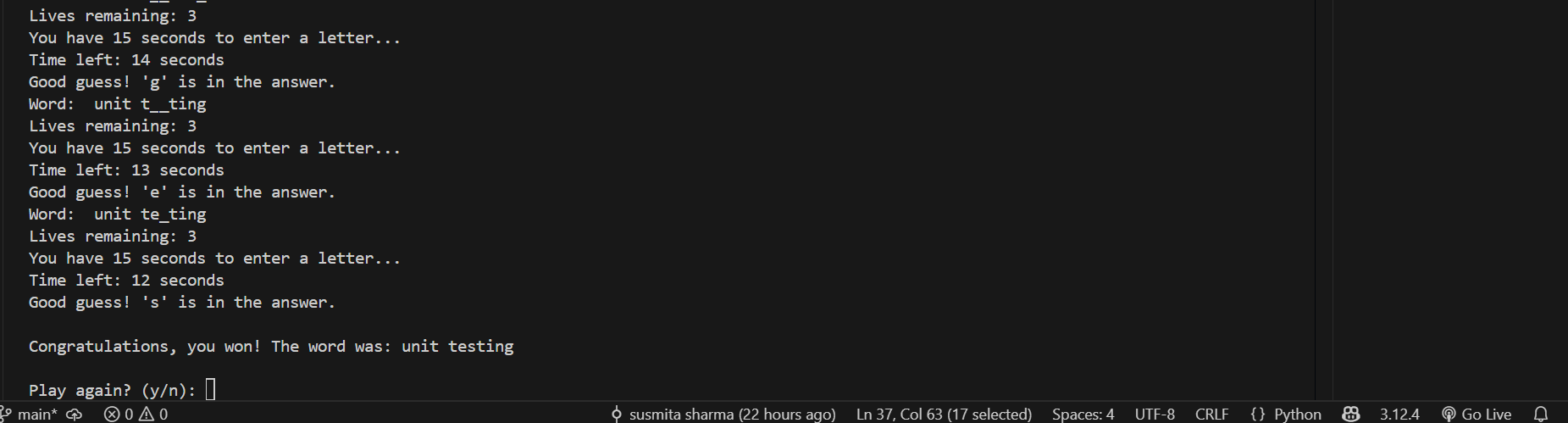
Game play screenshot for intermediate level



*Figure 25 :Screenshot showing gameplay of intermediate level.*



*Figure 25.1: Screenshot showing gameplay of intermediate level.*



*Figure 25.2: Screenshot showing gameplay of intermediate level.*

# **Conclusion**

Working on the Hangman Project provided a clear understanding of how Test Driven Development(TDD) and automated testing improve software quality. TDD encourages modular and organized code making it easier to maintain and expand in the future. Automated tests with Pytest quickly identified errors a d confirmed that changes do not break existing functionality. Using tools like Pylint and Flake8 ensures the code is clean, readable and follows Python best practices. The game met all the required features including two levels that is basic and intermediate levels, hidden letters, 15- second timer and life tracking. All the unit tests successfully verified all critical aspects of gameplay.

The area of improvement is that the game could include additional features like hints or scoring for a more engaging experience. Expanding the word and phrases dictionary would make the game more varied and challenging.

**GitHub Repository:** <https://github.com/susmitasharmaa/S225-PRT582-SOFTWARE-ENGINEERING-PROCESS-AND-TOOLS.git>

# **References**

Howard, M., LeBlanc, D., & Viega, J. (2010). *24 deadly sins of software security: programming flaws and how to fix them*. McGraw-Hill.<https://cdu.primo.exlibrisgroup.com/permalink/61CDU_INST/j6pesm/alma991001929281103446>

Nabilah, N. (2021). Using Hangman game application for the EFL classroom: Its efficacy for learners to master vocabulary. Proceeding of International Conference on Language Pedagogy, 1(1), 136–143. [https://doi.org/10.16920/jeet/2024/v38i2/24186](https://doi.org/10.16920/jeet/2024/v38i2/24186" \t "_new)

S., G. J., Doshi, S., Alex, A. K., U., Y., & J. L., F. D. (2025). A game-based approach to teach basic Python programming. *Journal of Engineering Education Transformations, 38*(2), 7–16. [https://doi.org/10.16920/jeet/2024/v38i2/24186](https://doi.org/10.16920/jeet/2024/v38i2/24186" \t "_new)

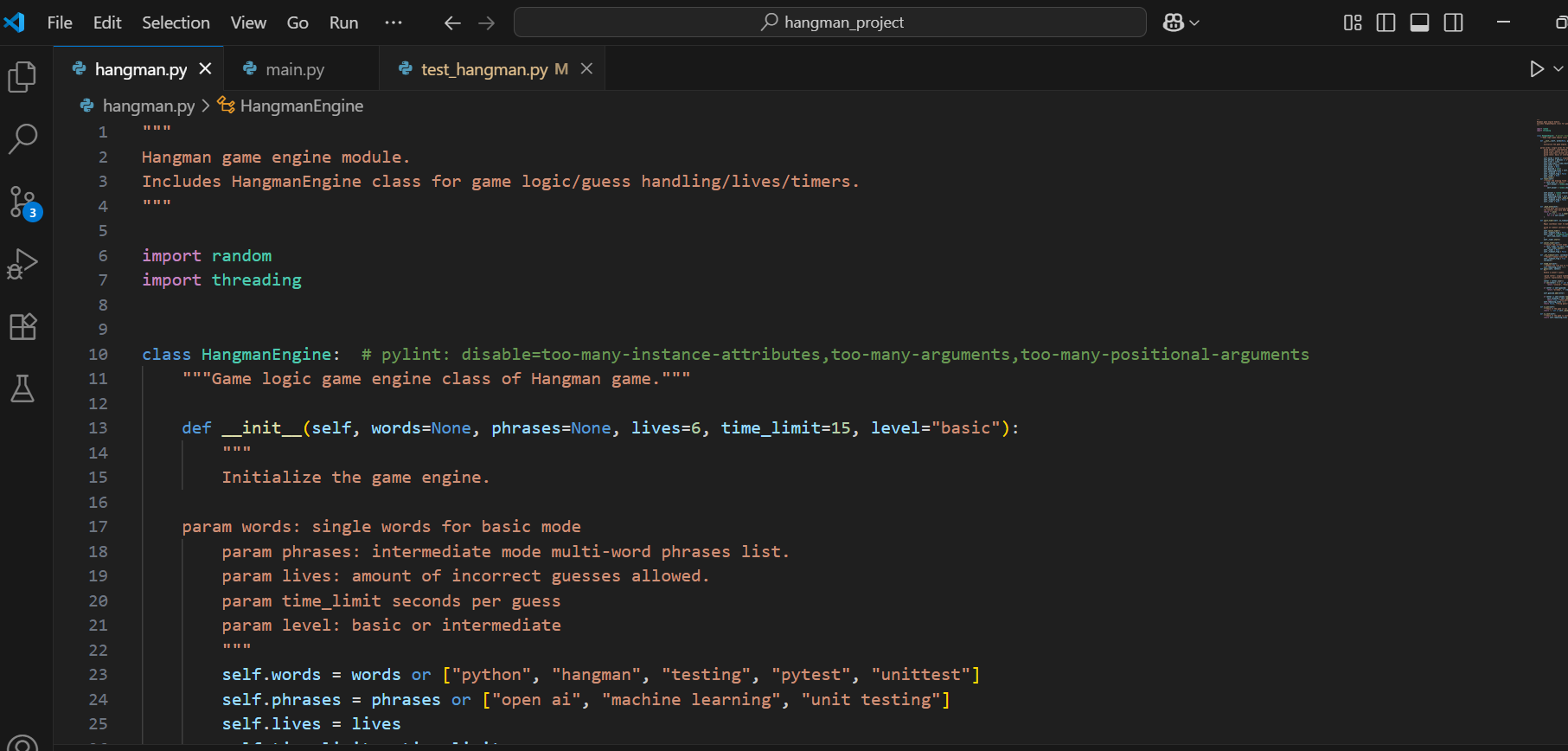
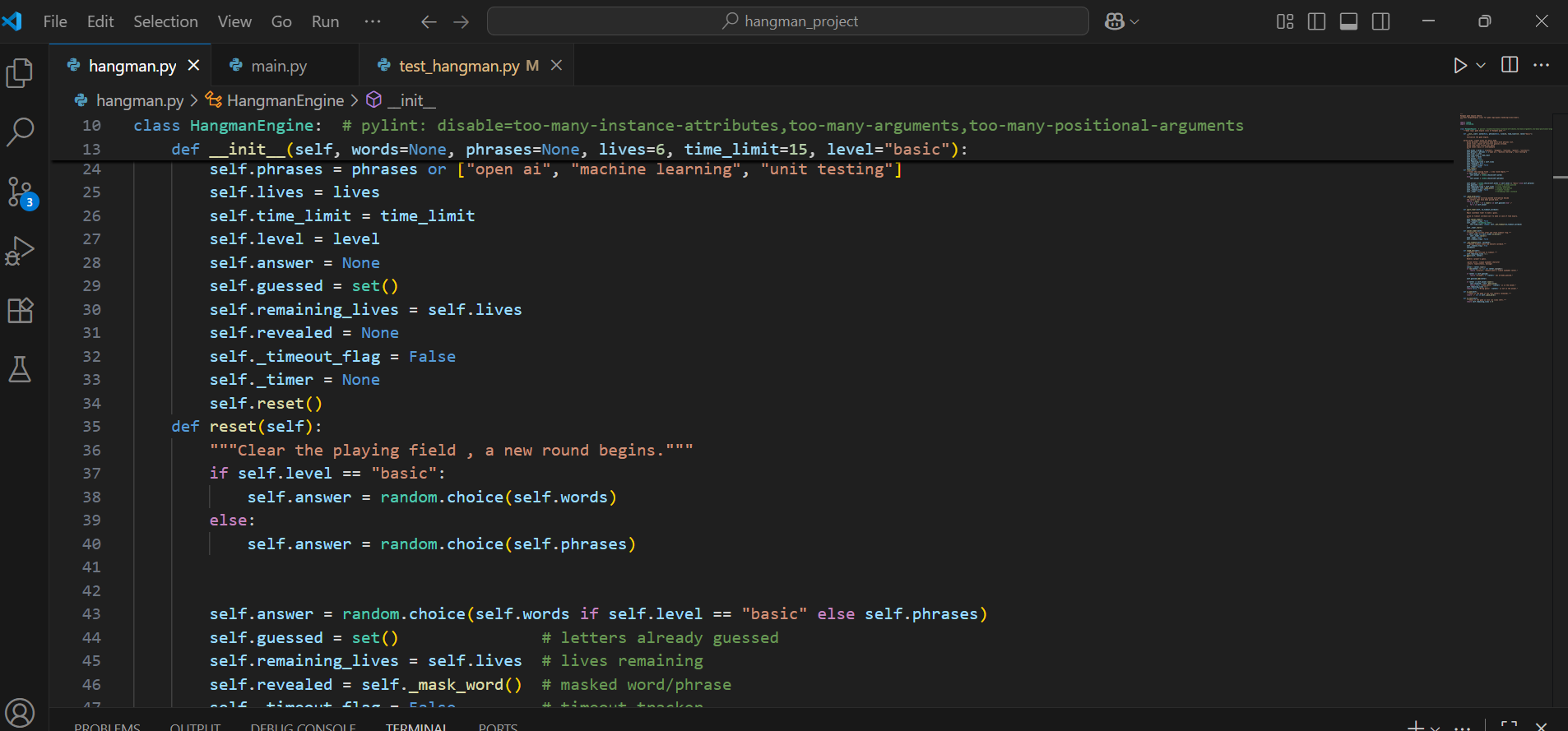
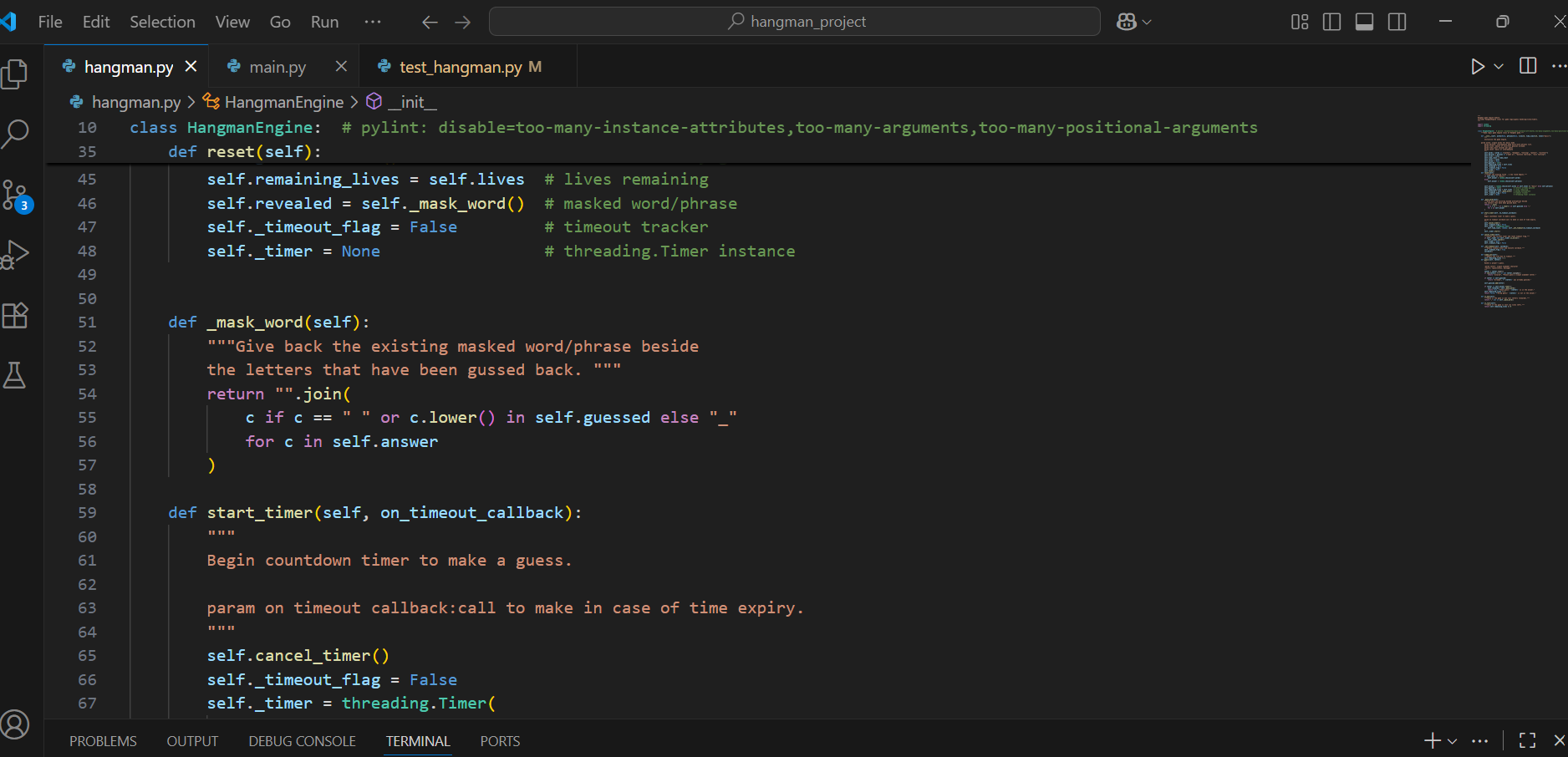
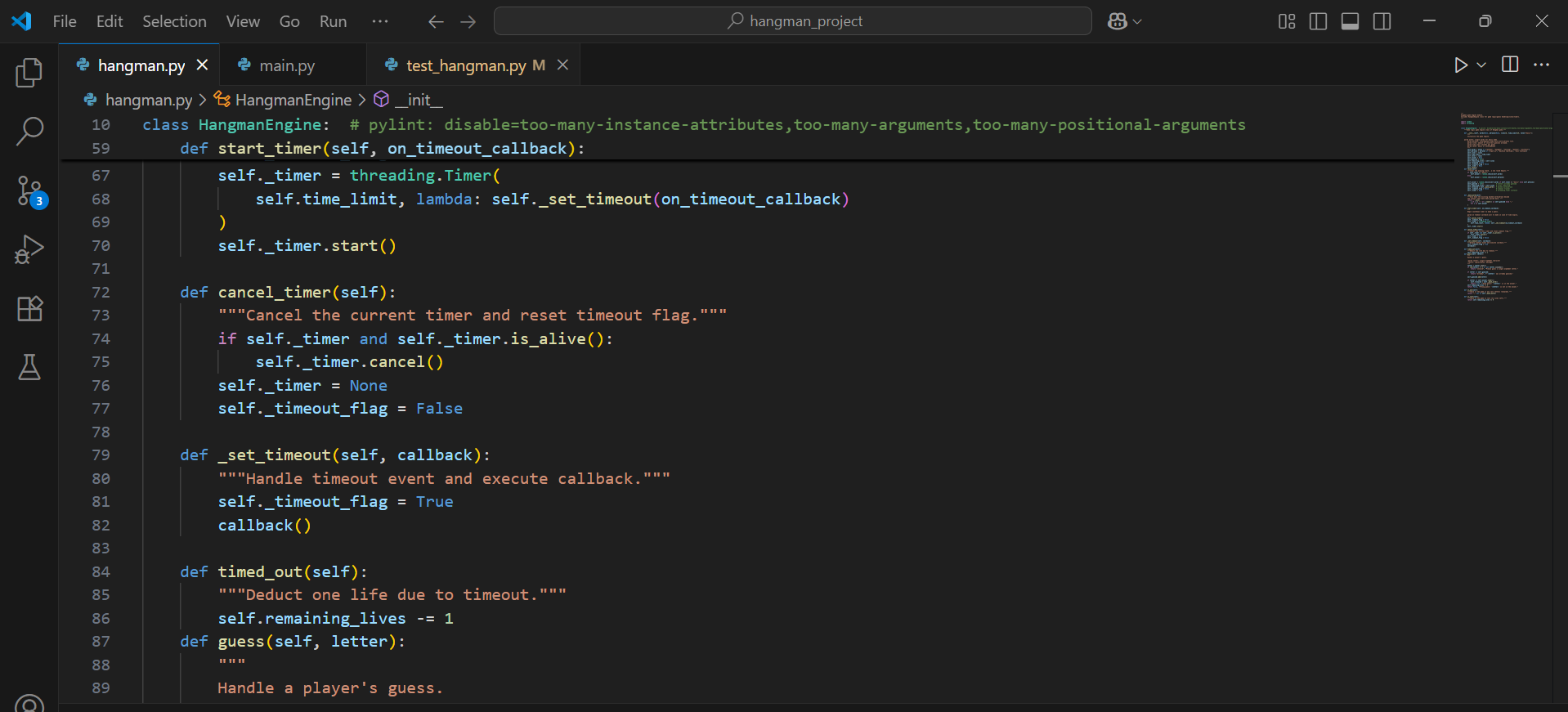
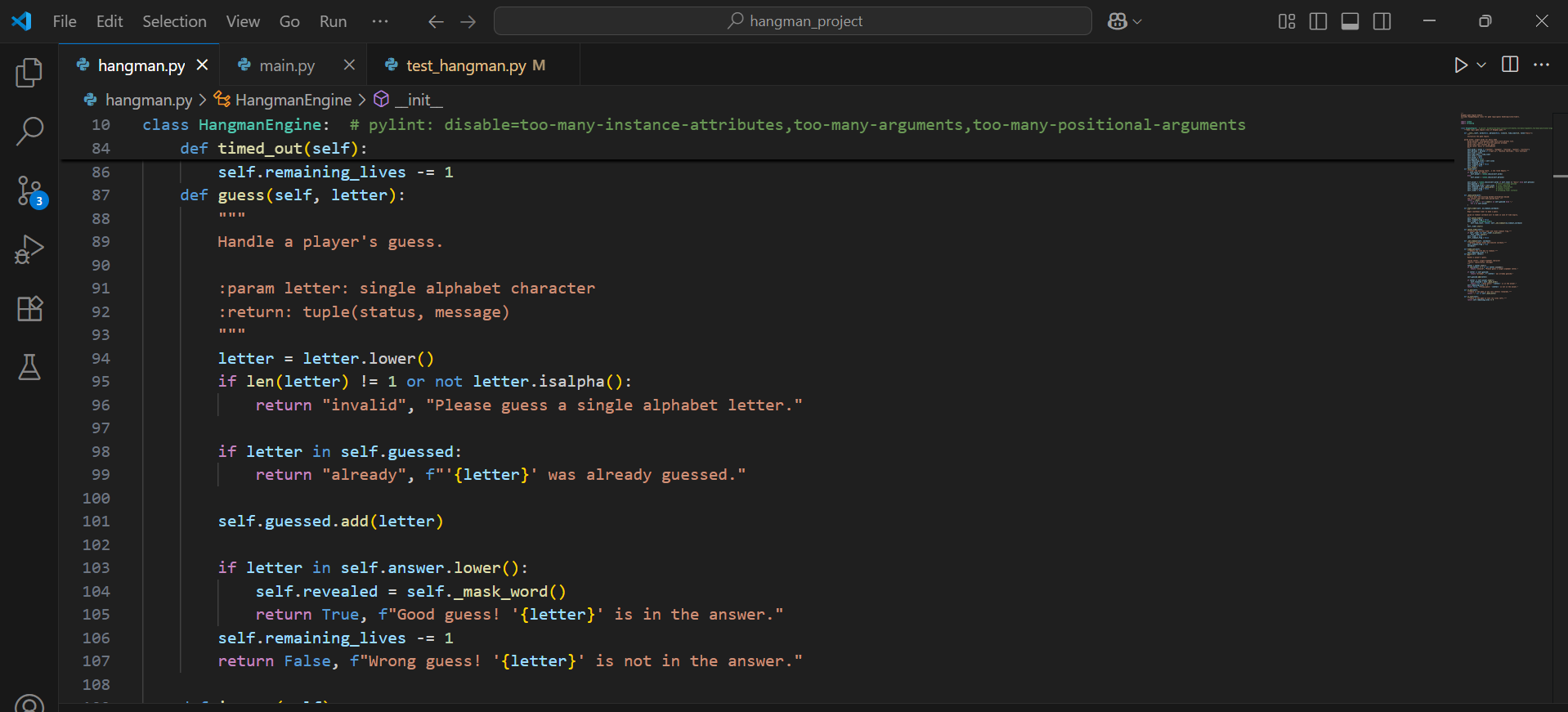
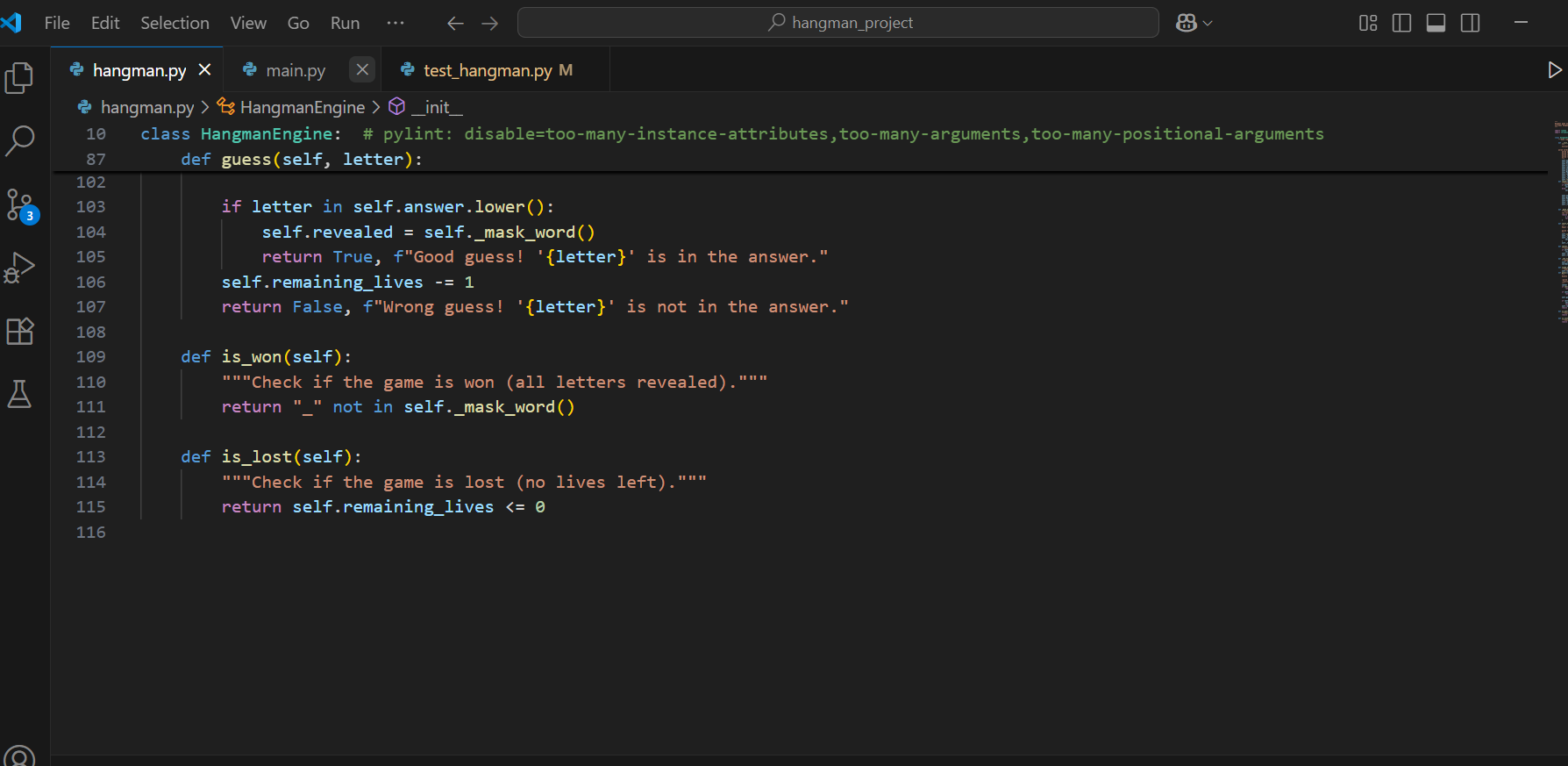
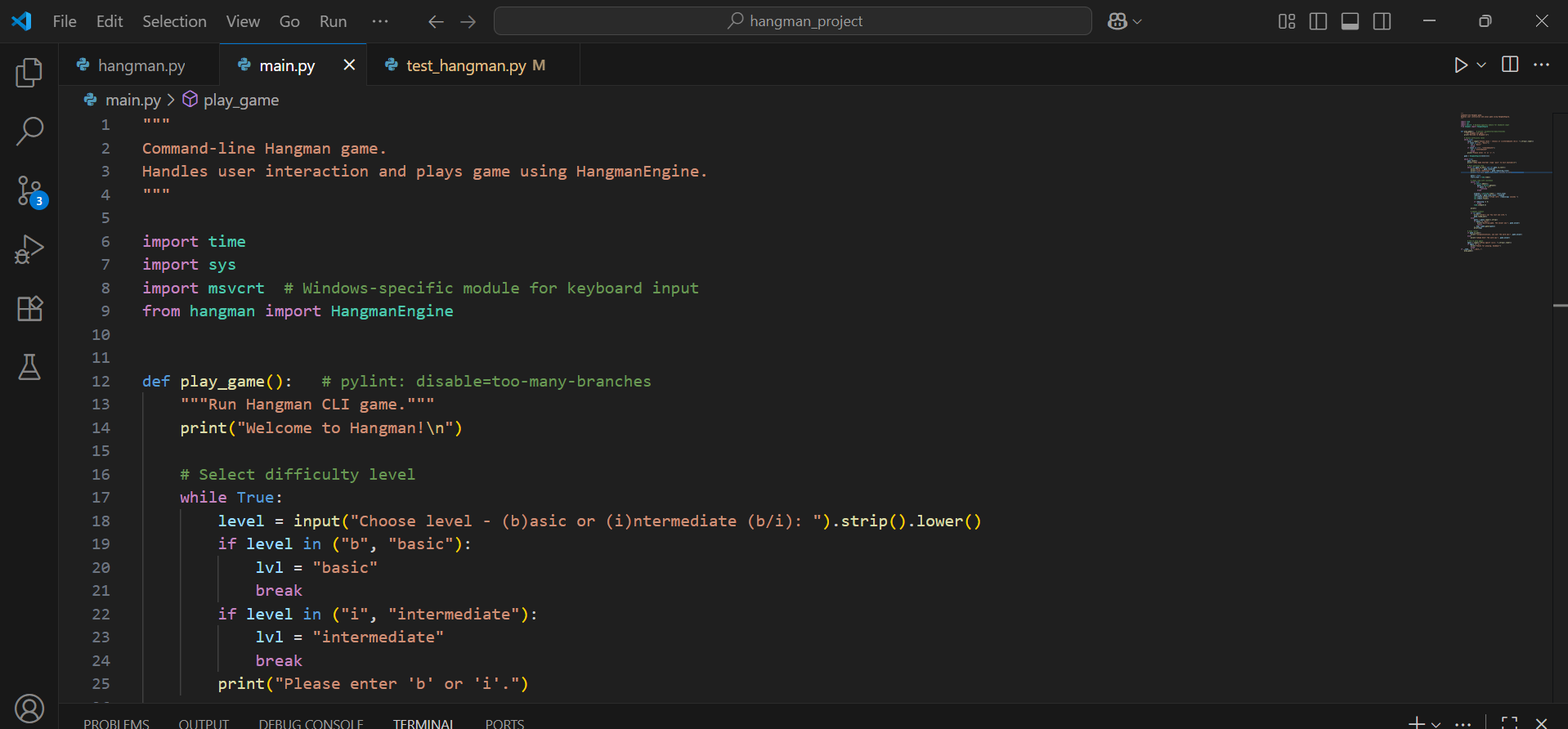
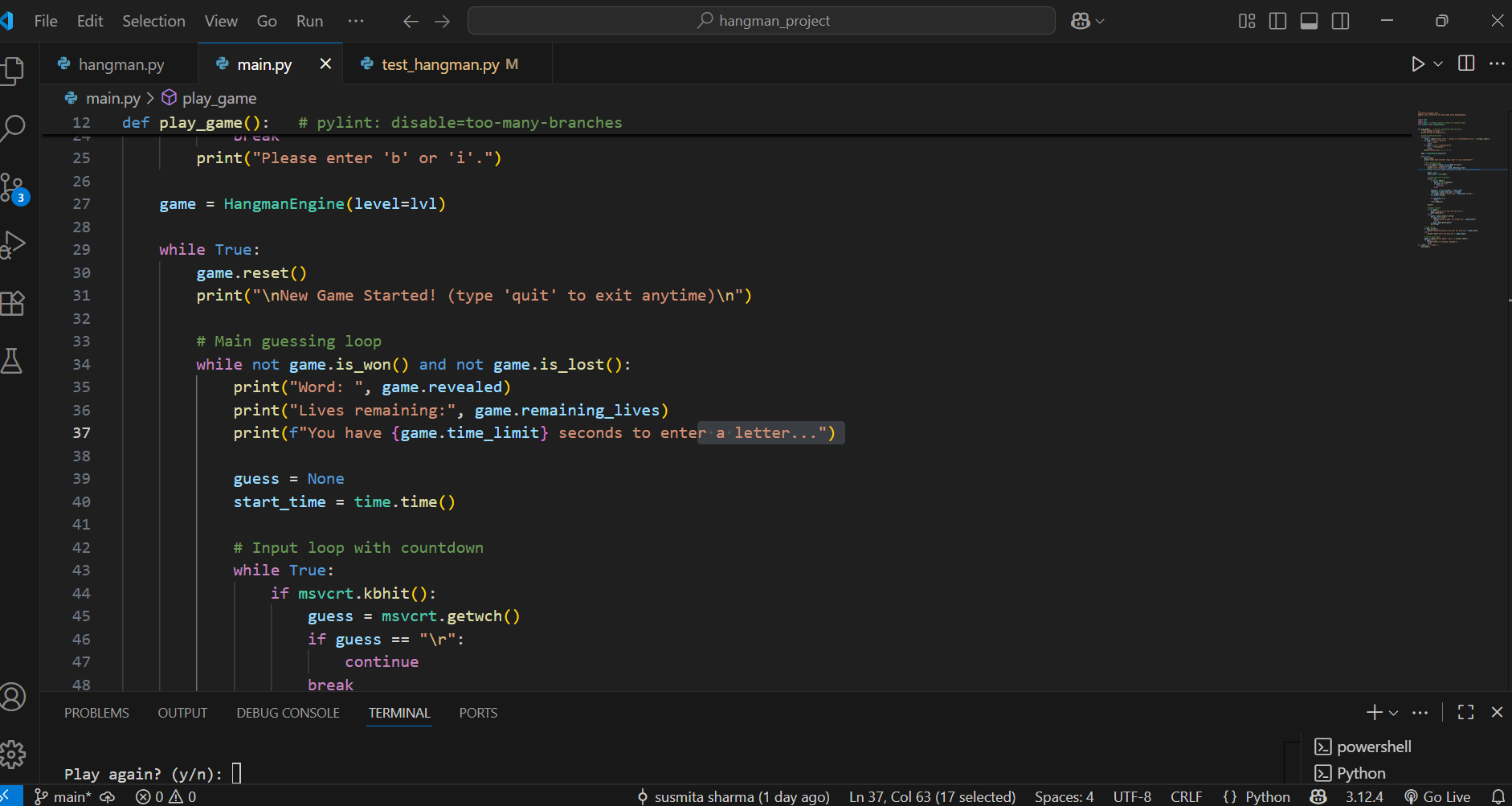
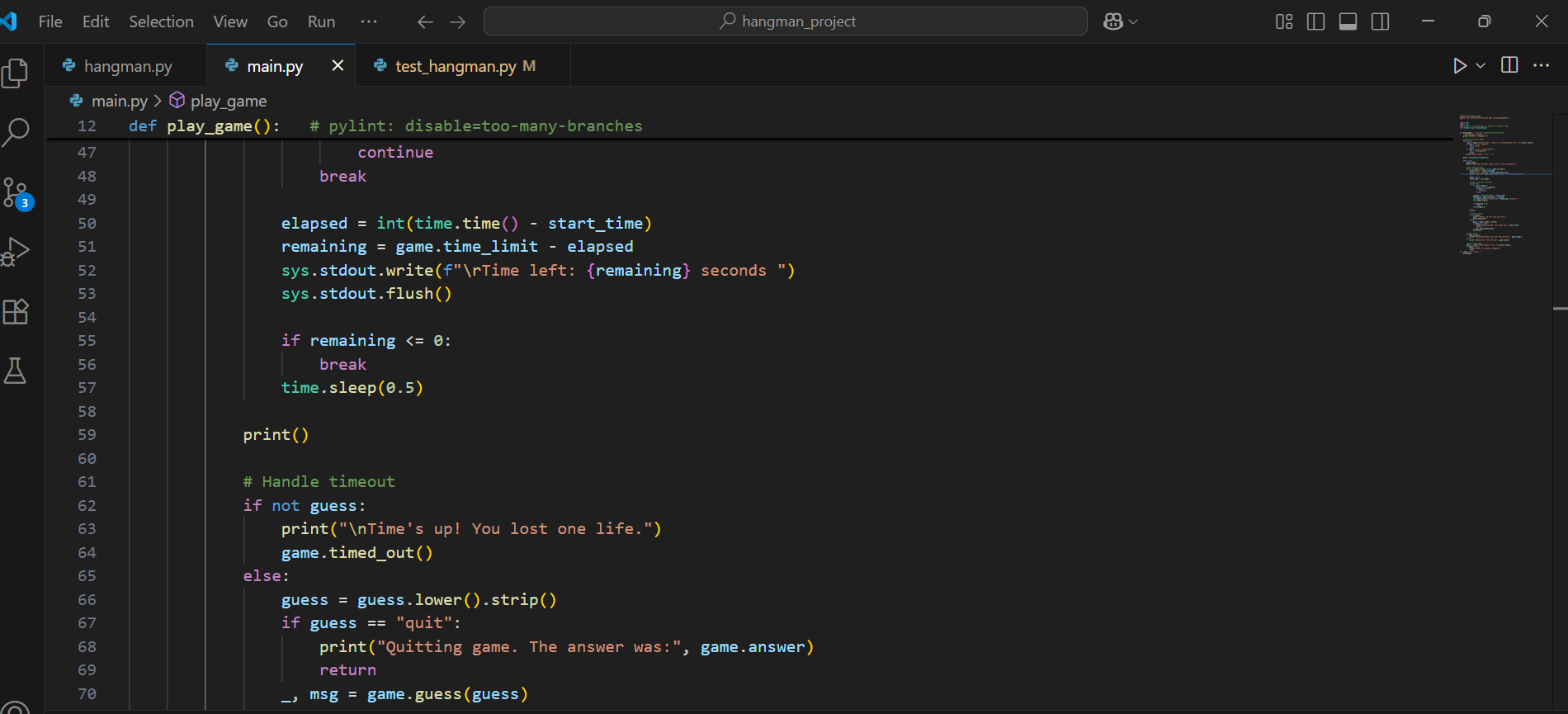
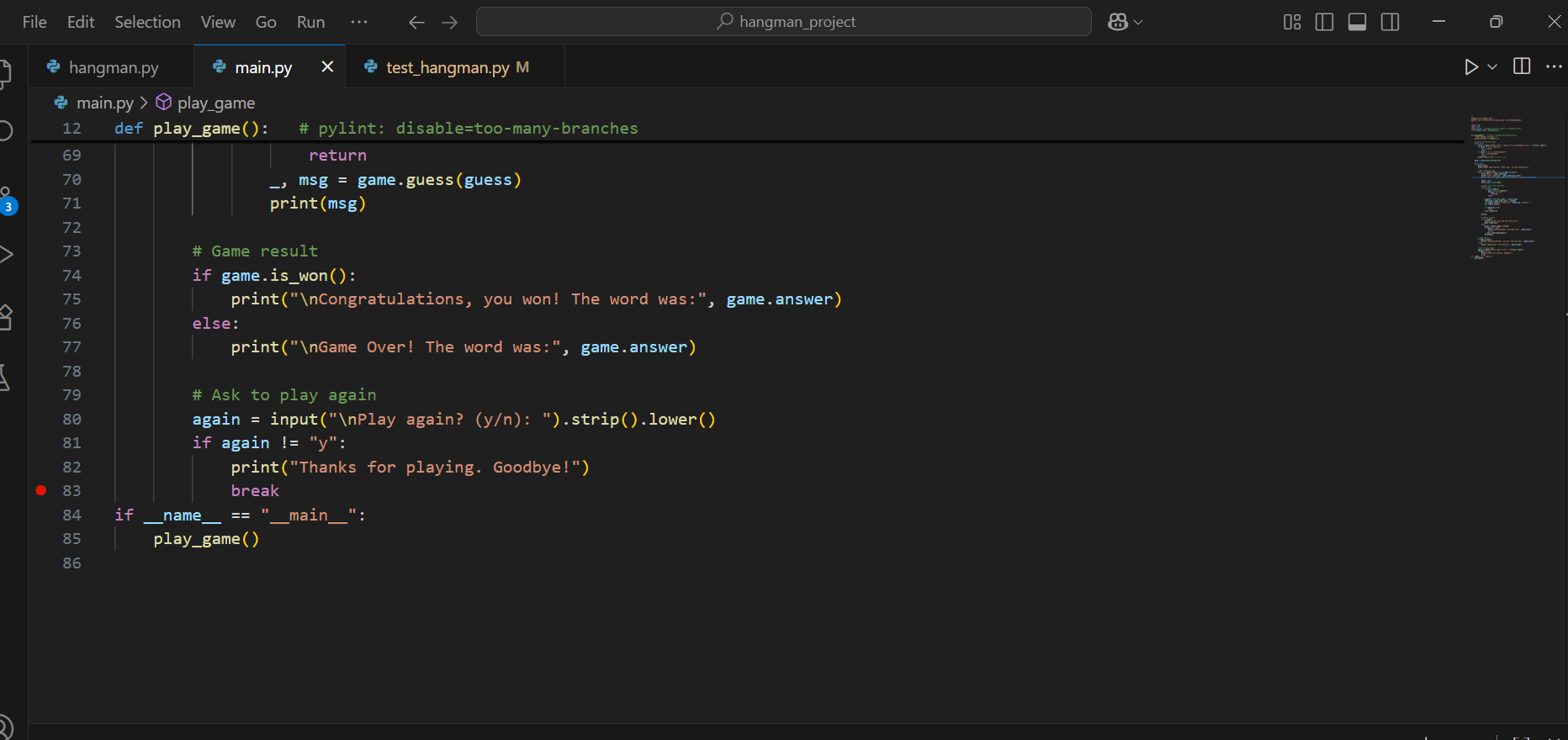
Vrdoljak, S., & Velki, T. (2016). Using Hangman game as an educational tool. In EDULEARN16 Proceedings (pp. 1622–1629). IATED. <https://doi.org/10.21125/edulearn.2016.1323>

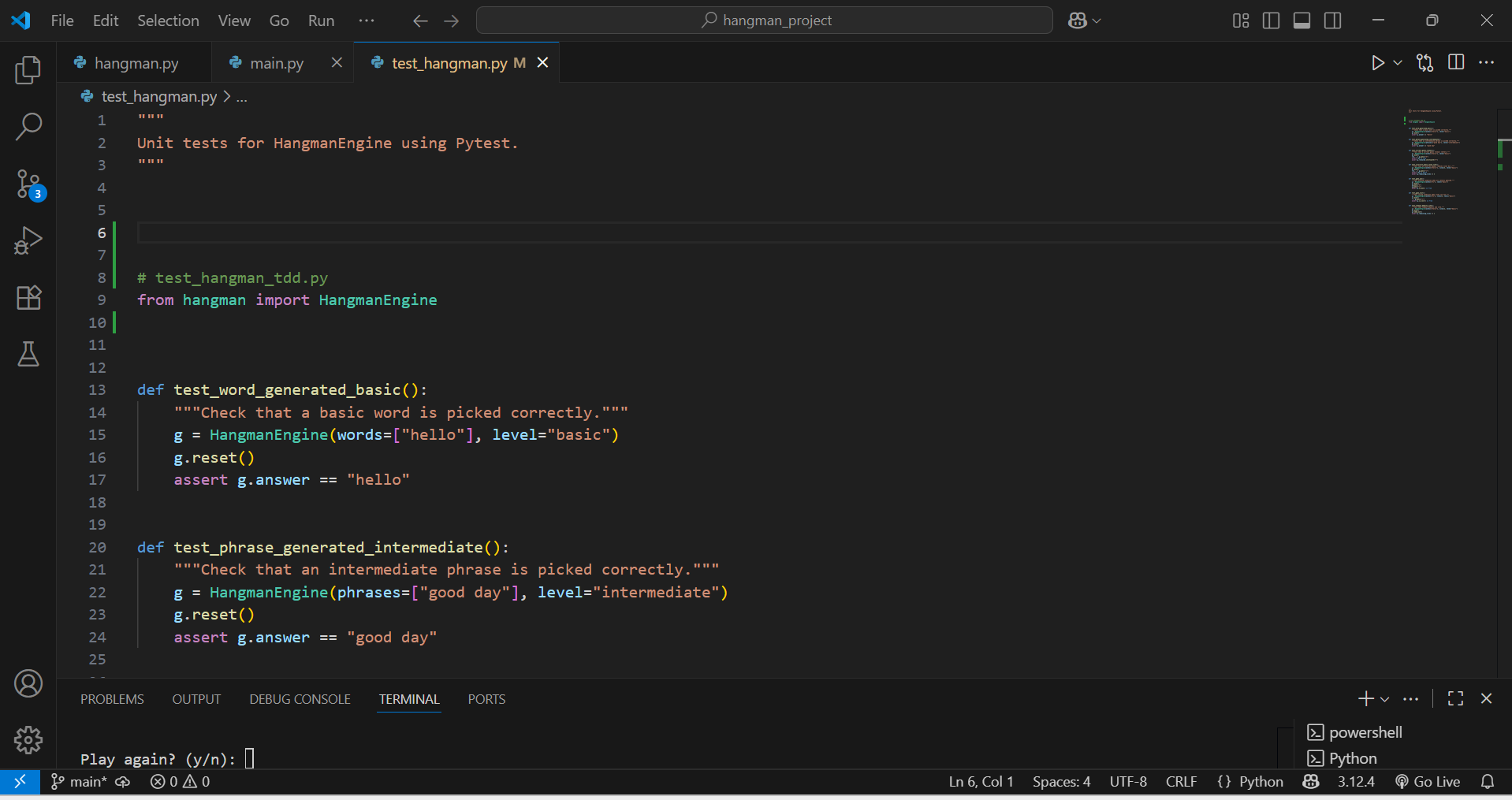
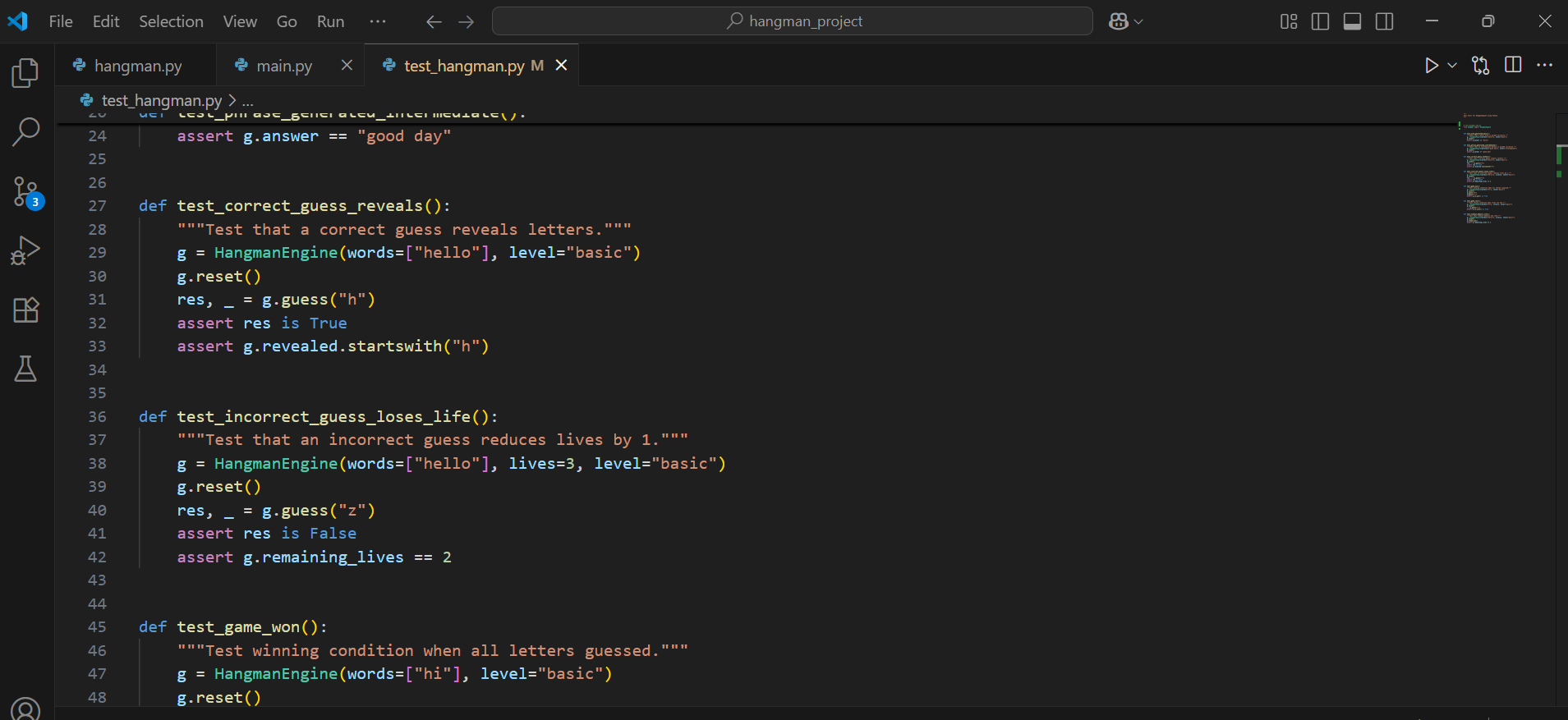
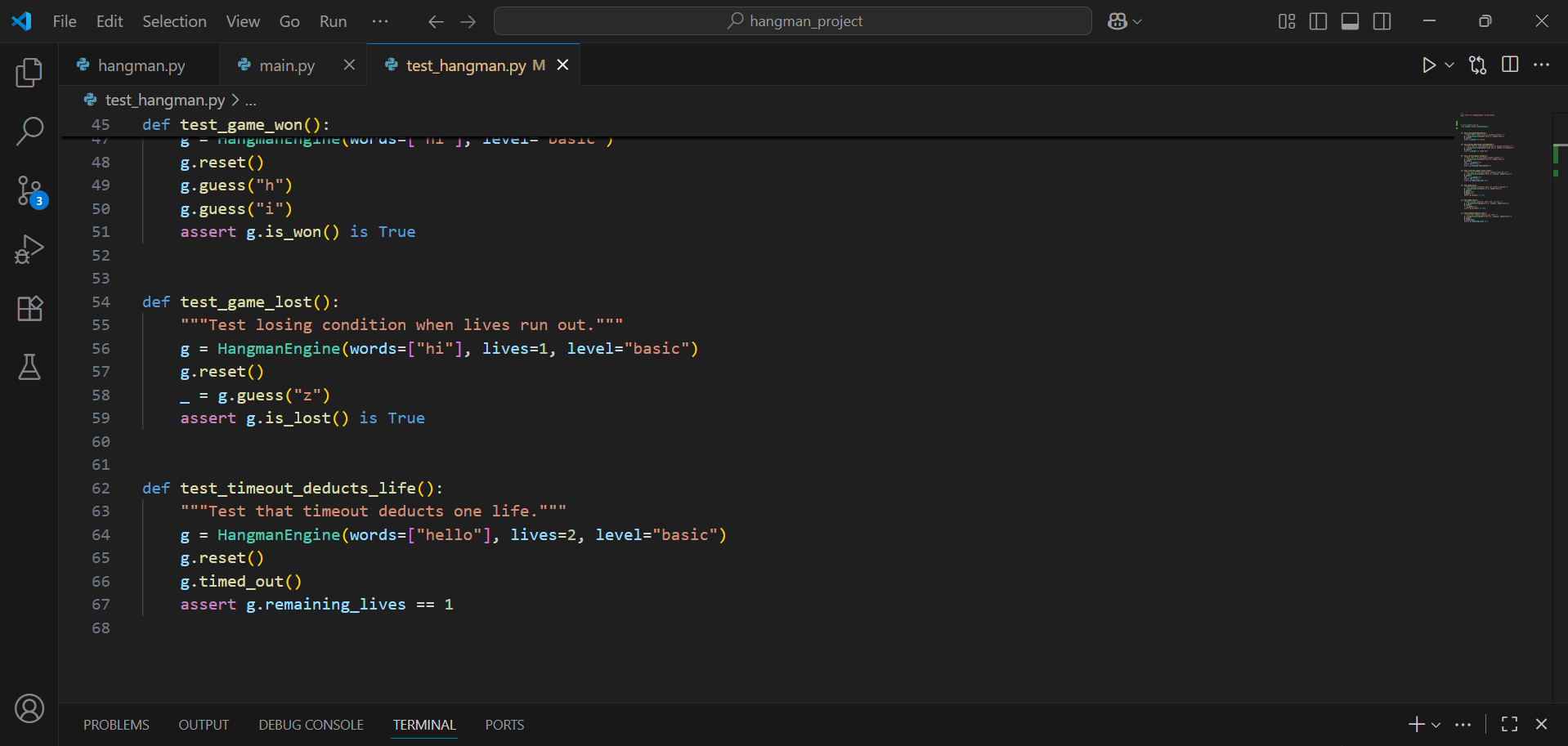
Whittaker, J. (2002). *How to Break Software: A Practical Guide to Testing*. P&C ECS. <https://cdu.primo.exlibrisgroup.com/permalink/61CDU_INST/j6pesm/alma991002261165203446>

Yusoff, Y., Nazmi, A., Izzat, M., Irwan, M. S., Zulfahmi, M., & Sallehuddin, R. (2021). Hangman–Hangaroo game design using automata theory. *International Journal of Innovative Computing, 11*(1), 7–11. [https://doi.org/10.11113/ijic.v11n1.275](https://doi.org/10.11113/ijic.v11n1.275" \t "_new)

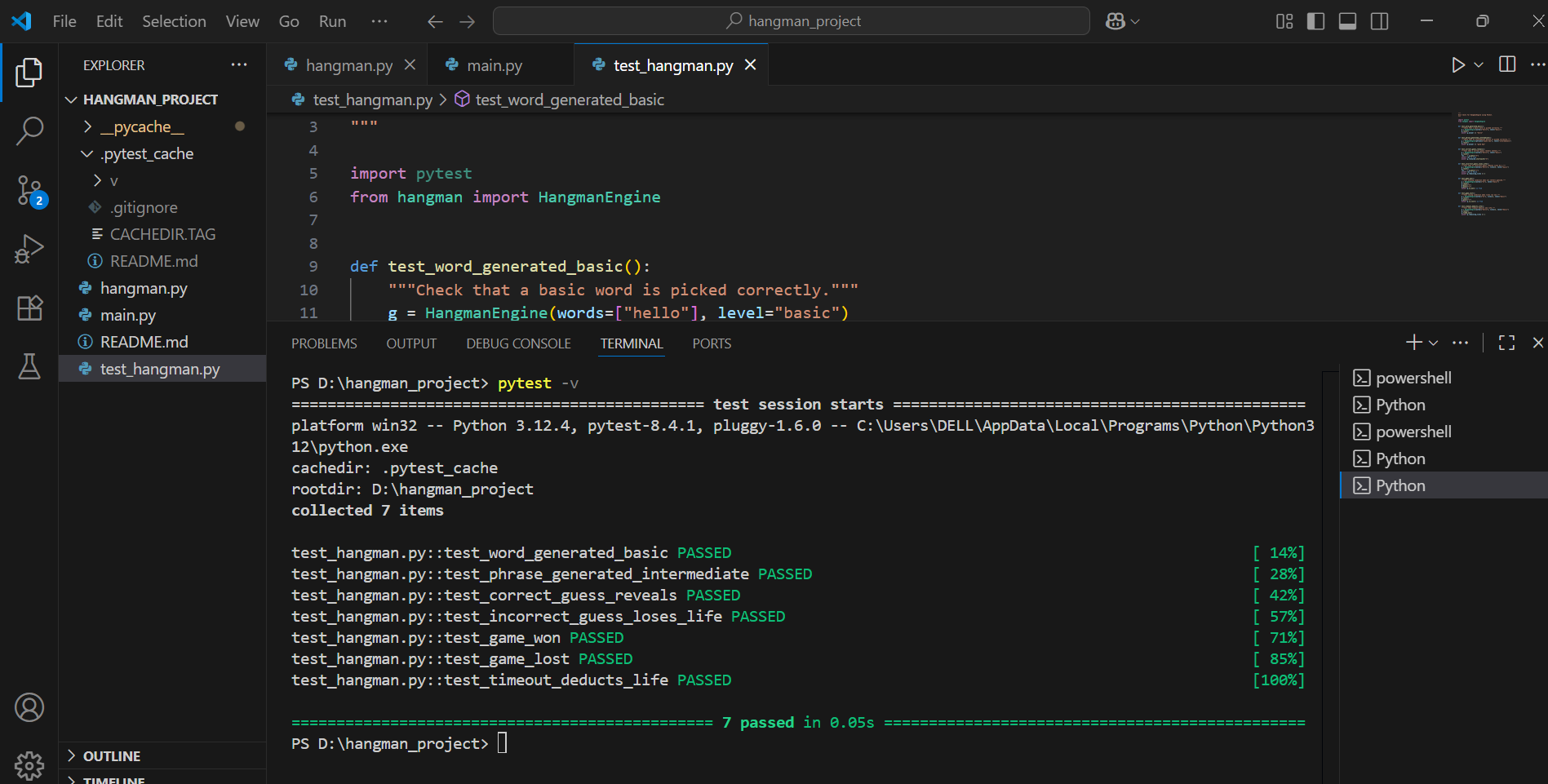
# **Appendices**

## Appendix 1: Code snippet

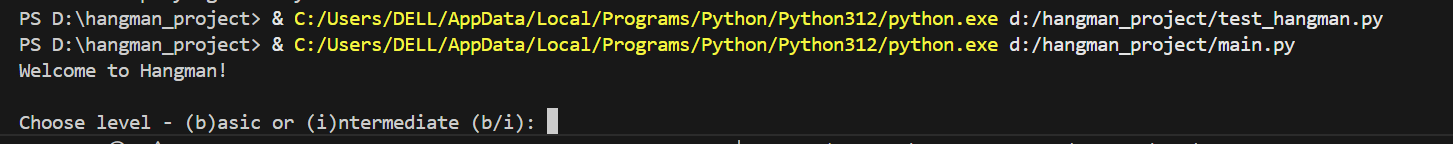
  

## Appendix 2: Unit test result

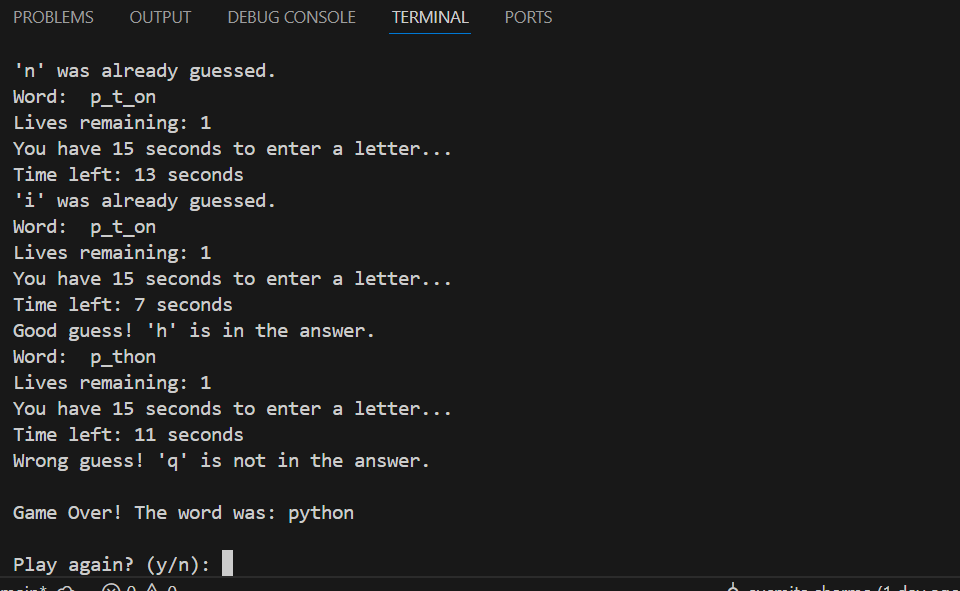


## Appendix 3: Screenshot

1.Main menu of Hangman game



2.In game interface showing masked word and lives



3.Gave over screen

