Report

Shaikat Barua

September 2024

GitHub Link

1 Introduction

Scrabble Word Game tests players' ability to create words that make sense under a length restriction that is determined at random. Every word is scored using the standard Scrabble letter values, with extra points given for quick input. Gamers have 15 seconds to enter a word, and how quickly they respond will have an immediate impact on their score. There are ten rounds in the game, and the player's final score is based on the total points they have collected so far.

This game is amusing and intellectually stimulating, encouraging fast thinking, vocabulary growth, and time management abilities.

1.1 Objectives

- Create valid words within a randomly generated length.
- Score points based on the Scrabble letter values for each word.
- Submit words quickly to earn bonus points, as the score is higher if less time is used.
- Complete 10 rounds, with the total score displayed at the end of the game.

1.2 Requirements

- Word Length: Randomly generated between 3 and 7 letters.
- Time Limit: 15 seconds per word.
- Scoring System: Based on traditional Scrabble letter values (details provided).
- Valid Words: Must be from a predefined dictionary list.
- No Repeats: Players cannot submit the same word twice.

- Time Bonus: Score increases based on time saved under the 15-second limit (details on calculation to be added later).

2 Test-Driven Development (TDD) and Automated Unit Testing

2.1 Test-Driven Development (TDD) Process

In the development of the Scrabble Word Game, **Test-Driven Development (TDD)** was employed to ensure reliable and robust functionality. TDD is a software development methodology where tests are written before the actual code. The process follows three main steps:

1. **Write the Test First**: A test that describes the function's anticipated behavior is written for every feature. For example, a test was built to verify that a word such as "apple" would yield the correct score during the development of the score calculation.

2. **Run the Test and Watch It Fail**: Since the appropriate feature hasn't been implemented yet, the test will initially fail. This verifies the validity of the test and makes sure that no code that doesn't yet exist is being tested.

3. **Write the Code**:The code is developed to implement the feature once the test has been set up, making sure that the test is satisfied. Following this procedure, the game logic—including word validation, score computation, and timing—was gradually constructed.

4. **Run the Tests and Watch Them Pass**: The test is run again after the code is written to make sure it passes and the feature functions as intended. Until every test passes, this process is repeated.

5. **Refactor the Code**: If required, the code is refined or optimized without affecting its functionality, and the tests are repeated to make sure everything functions as intended.

   This TDD process ensures that each part of the game is tested thoroughly and works as intended before proceeding to the next feature, reducing bugs and improving overall stability.

2.2 Automated Unit Testing Tool: `unittest`

For the Scrabble Word Game, the `unittest` framework was used to automate the testing process. `unittest` is a standard Python testing framework that allows developers to create test cases, group them into test suites, and automate their execution.

The following areas were covered by automated unit tests:

- **Scrabble Score Calculation**: Test cases ensured that the Scrabble score for each word was calculated accurately based on the predefined letter values. For example, tests checked that "apple" returned 9 points and ensured case sensitivity was handled properly.
- **Word Length Enforcement**: Tests were written to check if the game prompted the user correctly when a word of the wrong length was entered. If a word didn't meet the randomly generated length, the game raised an error.
- **Valid and Invalid Word Handling**: Tests validated whether the game correctly recognized valid words from the dictionary and rejected invalid ones.
- **Preventing Repeated Words**: Tests were implemented to verify that the game did not allow users to enter the same word more than once during any round.
- **Time-Based Scoring Adjustment**: Tests verified the time taken by the user to input a word and ensured that if a word was entered faster, the score was adjusted accordingly.

The `unittest` framework automates the testing process by executing all test cases and providing immediate feedback on whether the code functions correctly. Any test failures indicate specific problems, allowing them to be resolved quickly. This integration of **TDD** and automated unit testing ensures the Scrabble Word Game is reliable, stable, and thoroughly tested.

3 Conclusion

The Scrabble Word Game was developed with an emphasis on correctness, robustness, and maintainability thanks to the application of **Test-Driven Development (TDD)** and the framework. Writing tests before coding allows for the desired game behavior at every stage, streamlining and reducing the likelihood of errors in the development process.The game's functionality could be continuously verified thanks to automated unit testing. The game was extensively tested to make sure features like time-based scoring adjustments, score computation, and word validation work as intended under a variety of scenarios. The final result was a stable and dependable product because TDD's modularity made it simple to find and fix bugs.

The TDD methodology not only helped prevent potential errors early on but also supported the iterative refinement of the code, leading to cleaner, more efficient implementations. Overall, this approach significantly contributed to the creation of a user-friendly, engaging, and well-tested Scrabble Word Game that meets the given requirements.

Future enhancements could include expanding the dictionary, improving time-based scoring algorithms, and adding new game modes to further enhance the player experience.

4 Lessons Learnt

Throughout the development of the Scrabble Word Game, several key lessons were learned. These insights reflect both the successes of the project and areas that could be improved in future iterations.

4.1 What Went Well

- **Efficiency of TDD:** Test-Driven Development (TDD) has proven to be an extremely effective methodology for ensuring that the code is accurate. Writing tests before implementing them made it possible to validate the game's key features at every development stage and to identify potential problems early on. As a result, the development process became more structured and error-free.
- **Automated Testing:** The use of the `unittest` framework enabled automated and repeatable tests, which allowed for quick identification of problems after changes to the code. This was especially beneficial when new features were added or code was refactored, ensuring that no previous functionality was broken.
- **Clear Objectives and Requirements:** Well-defined requirements and objectives guided the development process, ensuring that all essential features—such as time-based scoring, word validation, and preventing repetitive word inputs—were carefully integrated and tested.

4.2 Areas for Improvement

- **Handling of Time-Based Scoring:** Although the time-based scoring functioned as anticipated, there is room for improvement in the bonus multiplier. As things are, people that input data incredibly quickly are not completely rewarded by the score adjustment, which is linear. The game might be made more complex by introducing a more dynamic scoring system that changes according to word length or difficulty.
- **Dictionary Expansion:** The game's dictionary was limited to a small set of words for simplicity. For a more realistic and challenging game, a comprehensive dictionary, such as one based on official Scrabble dictionaries, should be integrated. This would enhance the player's experience by recognizing a broader range of valid words.

- **User Interface Improvements:** The current game runs in the console, which, while functional, limits user engagement. Future versions could benefit from the addition of a graphical user interface (GUI) to make the game more visually appealing and user-friendly.

4.3 How It Can Be Improved

- **Enhanced Time-Based Scoring:** To improve time-based scoring, a more advanced formula could be used that factors in word length, the complexity of the word, and user performance over multiple rounds. Implementing different difficulty levels could also offer varying time limits and scoring systems.
- **Integration of Comprehensive Dictionary:** By integrating an extensive word list, either from official sources or through an API, the game can become more challenging and realistic. This would allow for a wider range of valid words and encourage players to use more complex vocabulary.
- **Development of a GUI:** Adding a graphical user interface could significantly enhance the user experience. Libraries such as `Tkinter` or `PyQt` could be used to create a more interactive and visually engaging game environment. This would make the game more accessible to casual players who may not be comfortable with console-based interaction.

In conclusion, the project has been successful, and the core functionality has been implemented without any issues. There is room for improvement, nevertheless, especially in the areas of user engagement, dictionary size, and score complexity. By fixing these issues, future players will find the game to be more difficult and captivating.