

Assignment 03

Create a testing strategy for the Yahtzee scorer code that was generated and document your journey.



You

modify the code to calculate all possible categories after every round and add maximum possible



ChatGPT

To modify the code so that it calculates the maximum possible score for all categories after every round, we can adjust the logic to present the player with the scores they would receive for each category if they choose to score in that category at that moment. Then, the player can make an informed decision based on the potential scores. This modification helps players understand their possible scores before making their choice.

Here's how you can implement this:

This version of the code calculates the possible scores for each category based on the current roll of the dice and displays those scores to the player. It helps players make strategic decisions about which category to score in, considering the maximum possible points they could earn with their current roll.



You

above code does not give option to reroll. Incorporate reroll option for 3 times in each turn



ChatGPT

To incorporate the reroll option for three times in each turn, we need to modify the main game loop to allow the player to reroll the dice up to three times. After each reroll, the player should be given the option to either select a category to score in or continue rerolling the dice. Here's the modified code with the reroll option:

In this version of the code, the player is given the option to reroll the dice up to three times in each turn. After each reroll, the player is presented with the possible scores for the current roll. The player can then choose a category to score in or continue rerolling the dice.



You

give a testing strategy to test the above code



ChatGPT

Testing the code involves ensuring that it functions correctly across various scenarios, covering different aspects such as input validation, dice rolling, scoring calculations, and overall game flow. Here's a testing strategy to thoroughly test the provided code:

1. Input Validation Testing:

- Test entering non-integer values for the number of players.
- Test entering invalid options for rerolling (e.g., "y" instead of "yes" or "no" instead of "yes/no").
- Test entering invalid category names during scoring selection.

2. Dice Rolling Testing:

- Verify that the dice are rolled correctly.
- Test that the dice values are within the range of 1 to 6.

3. Reroll Testing:

- Test rerolling all dice.
- Test rerolling specific dice by providing valid indices.
- Test rerolling with different combinations of rerolls.

4. Scoring Calculation Testing:

- For each scoring category, test with different combinations of dice to ensure correct scoring.
- Test each scoring category with a variety of dice combinations that satisfy and don't satisfy the scoring condition.
- Ensure that scoring is calculated accurately after each turn and at the end of the game.

5. Game Flow Testing:

- Test playing the game with various numbers of players (including edge cases like 0 players and large numbers of players).
- Ensure the game progresses through all rounds correctly.
- Test that the game terminates after completing all rounds.
- Verify that the final scores are displayed correctly.

6. **Boundary and Edge Case Testing:**

- Test with the maximum number of players.
- Test with the minimum number of players.
- Test with the maximum number of rerolls.
- Test with the minimum number of rerolls (no rerolls).

7. **Integration Testing:**

- Test the integration of rerolling, scoring, and game flow to ensure they work together seamlessly.
- Verify that the player's choices and game state are maintained correctly between turns.

8. **Robustness Testing:**

- Test the code's behavior when unexpected errors occur (e.g., unexpected user inputs, dice rolling failures).
- Ensure the code handles exceptions gracefully and provides clear error messages.

9. **Performance Testing:**

- Test the code's performance with large numbers of players and rounds to ensure it remains responsive and doesn't have significant slowdowns.

10. **Manual Playthrough Testing:**

- Manually play through the game with various scenarios to ensure everything behaves as expected and matches the intended gameplay.

By following this testing strategy and covering these aspects, you can ensure that the code is thoroughly tested and robust, providing a reliable and enjoyable Yahtzee game experience.

PA

You

Please give test cases code

These test cases cover the following scenarios:

1. **Test Die Rolling:** Verifies that the `roll_die` function produces values within the range of 1 to 6.
2. **Test Dice Rolling:** Ensures that the `roll_dice` function produces lists of 5 dice, each with values within the range of 1 to 6.
3. **Test Possible Scores Calculation:** Checks that the `calculate_possible_scores` function accurately calculates possible scores for a given set of dice.
4. **Test Main Game Flow:** Simulates a full game with two players, ensuring that the game proceeds as expected and provides final scores.

These test cases cover critical aspects of the Yahtzee game implementation, including dice rolling, scoring calculation, and the overall game flow. You can run these test cases to ensure that your implementation behaves as expected and passes all the defined scenarios.



You

use pytest to generate test cases for above code



ChatGPT

Here's how you can use Pytest to generate test cases for the Yahtzee game code:

First, make sure you have Pytest installed. You can install it using pip:

```
bash
```

Copy code

```
pip install pytest
```

Then, create a Python file named `test_yahtzee.py` (or any other name you prefer) and write your test cases using Pytest. Below is an example of how you can structure your test file:

In this example:

- We import the functions to be tested from the `yahtzee` module.
- We define individual test functions using the Pytest framework.
- The `@patch` decorator is used to mock the input and output streams for the `main` function.
- Each test function checks a specific aspect of the code's functionality.

To run these tests, simply navigate to the directory containing your test file and run `pytest`. Pytest will automatically discover and execute your test functions, providing output indicating whether each test passed or failed.

PA

You

give unit tests for each function like `score_one` `score_two` and so on using `assert`



ChatGPT

Creating unit tests for each scoring function in the Yahtzee game can help ensure each scoring rule is implemented correctly. Here's how you can write unit tests using the `assert` statement for the individual scoring functions like `score_ones`, `score_twos`, and so on. This example assumes each scoring function is defined in a file named `yahtzee.py`.

First, let's structure the test file. For illustration, I'll show you how to test the `score_ones`, `score_twos`, and a couple of other scoring functions like `score_three_of_a_kind` and `score_full_house`. You can follow this pattern to write tests for the remaining functions.

This script manually tests each scoring function by calling them with predefined dice rolls and comparing the output against the expected results using assertions. If any assertion fails, it will raise an `AssertionError`, indicating the function does not behave as expected for the given input. If all tests pass, it will print "All tests passed!"

To run these tests, simply execute the script with your Python interpreter. Remember, this example assumes you have all the scoring functions (`score_ones`, `score_twos`, etc.) implemented in a file named `yahtzee.py` and that they are correctly functioning according to the Yahtzee game rules.



You

give tests for four of a kind, chance and all other functions as well



ChatGPT

Certainly! Here are additional test cases for the remaining scoring functions including `score_four_of_a_kind`, `score_small_straight`, `score_large_straight`, and `score_yahtzee`.

These additional tests cover the remaining scoring functions, ensuring they behave correctly for different combinations of dice rolls. As before, running this script with your Python interpreter will execute the tests and indicate whether they pass or fail. If all tests pass, it will print "All tests passed!" indicating that the scoring functions are working as expected.

