Yahtzee

Help Center

Overview

Yahtzee is a dice game played with 5 dice where you try to score the most points by matching certain combinations. You can play the game here. In Yahtzee, you get to roll the dice three times on each turn. After the first roll, you may hold as many dice as you would like and roll the remaining free dice. After this second roll, you may again hold as many dice as you would like and roll the rest. Once you stop (either because you have exhausted your three rolls or you are satisfied with the dice you have), you score the dice in one box on the score card.

For this mini-project, we will implement a strategy function designed to help you choose which dice to hold after your second roll during the first turn of a game of Yahtzee. This function will consider all possible choices of dice to hold and recommend the choice that maximizes the expected value of your score after the final roll.

To simplify the mini-project, we will only consider scores corresponding to the "upper" section of the scorecard. Boxes in the upper section correspond to numbers on the dice. After each turn, you may choose one empty box and enter the sum of the dice you have with the corresponding number. For example, if you rolled $(2, 3, 3, 3, 4)$, you could score 2 in the Twos box, 9 in the Threes box, or 4 in the Fours box. (Restricting scoring to the upper section will also allow you to debug/test your strategy function on smaller numbers of dice.)

Testing your mini-project

The provided template includes the function `gen_all_sequences` from lecture (which you should not modify) as well as a `run_example` function that you may modify as you see fit while developing, debugging, and testing your strategy function. The template includes the stubs for the four functions that you will need to implement for this mini-project. Remember you should be testing each function as you write it. Don't try to implement all of the functions. If you do, you will have errors that will interact in inexplicable ways making your program hard to debug.

- As you implement your strategy function, we suggest that you build your own collection of tests using the `poc_simpletest` module that we have provided. Please review this page for an overview of the capabilities of this module. These tests can be organized into a separate test suite that you can import and run in your program as we demonstrated for Solitaire Mancala.
- If you try to test your code with five dice, it will be more difficult to understand what is going on. Instead, we recommend that you should test first with two dice. In particular, you may want to work out some examples by hand with two dice and create a small test suite that checks these examples. If you are having trouble with the case of two dice, you should take a look at this week's practice activity.
- Finally, submit your code (with the calls to `run_example` commented out) to this Owltest page. This page will automatically test your mini-project. It will run faster if you comment out the call to `run_example` before submitting. Note that trying to debug your mini-project using the tests in OwlTest can be very tedious since they are slow and give limited feedback. Instead, we *strongly* suggest that you first test your program using your own test suite. Programs that pass these tests are much more likely to pass the OwlTest tests.

Remember that OwlTest uses Pylint to check that you have followed the coding style guidelines for

this class. Deviations from these style guidelines will result in deductions from your final score. Please read the feedback from Pylint closely. If you have questions, feel free to consult this page and the class forums.

When you are ready to submit your code to be graded formally, submit your code to the CourseraTest page for this mini-project that is linked on the main assignment page.

Implementation

Your task is to implement the following four functions: `score`, `expected_value`, `gen_all_holds`, and `strategy`. These four functions should do the following:

- `score(hand):` This function takes as input a tuple `hand` that represents the die values in the given Yahtzee hand. Since ordering of the die values in Yahtzee hands is unimportant, tuples corresponding to Yahtzee hands will always be stored in sorted order to guarantee that each tuple corresponds to a unique Yahtzee hand. The function `score(hand)` computes a score for `hand` as the maximum of the possible values for each choice of box in the upper section of the Yahtzee scorecard.
- `expected_value(held_dice, num_die_sides, num_free_dice):` This function computes the expected value of the scores for the possible Yahtzee hands that result from holding some dice and rolling the remaining free dice. The dice being held are specified by the sorted tuple `held_dice`. The number of sides and the number of dice that are free to be rolled are specified by `num_die_sides` and `num_free_dice`, respectively. You should use `gen_all_sequences` to compute all possible rolls for the dice being rolled. As an example, in a standard Yahtzee game using five dice, the length of `held_dice` plus `num_free_dice` should always be five.
- `gen_all_holds(hand):` This function takes a sorted tuple `hand` and returns the set of all possible sorted tuples formed by discarding a subset of the entries in `hand`. The entries in each of these tuples correspond to the dice that will be held. If the tuple `hand` has the entries $(h_0, h_1, \ldots, h_{m-1})$, the returned tuples should have the form $(h_{i_0}, h_{i_1}, \ldots, h_{i_{k-1}})$ where $0 \leq k \leq m$ and the integer indices satisfy $0 \leq i_0 < i_1 < \ldots < i_{k-1} < m$. In the case where values in the tuple `hand` happen to be distinct, the set of tuples returned by `gen_all_holds` will correspond to all possible subsets of `hand`.
- `strategy(hand, num_die_sides):` This function takes a sorted tuple `hand` and computes which dice to hold to maximize the expected value of the score of the possible hands that result from rolling the remaining free dice (with the specified number of sides). The function should return a tuple consisting of this maximal expected value and the choice of dice (specified as a sorted tuple) that should be held to achieve this value. If there are several holds that generate the maximal expected value, you may return any of these holds.

As you implement these four functions, make sure that you think about the best order to write and test them in. You may add extra helper functions if so desired. However, the signature of the four functions above must match the provided description as they will be tested by the machine grader.

Coding `gen_all_holds`

Implementing `gen_all_holds` is one of the main challenges of this mini-project. While its implementation is short, the actual code requires thought. Since tuples are immutable, your algorithm for computing the required set of tuples cannot directly delete from the tuple `hand`. Instead, `gen_all_holds` should compute the set of all possible holds in a manner very similar to that of `gen_all_sequences`. In particular, your implementation should iterate over the entries of `hand` and compute all possible holds for the first k entries in `hand` using all possible holds for the first

$k-1$ entries of `hand`.

To help you test your implementation of `gen_all_holds`, you may make use of this short test suite:

```
#import poc_holds_testsuite
#poc_holds_testsuite.run_suite(gen_all_holds)
```

Once you have working code, you may wish to extend the `score` function to include scores from the lower section of the Yahtzee scorecard. (This is optional and isn't graded.) With this extension, the `strategy` function gives quite accurate recommendations. Impress your friends by consistently beating them at Yahtzee!

Created Wed 4 Jun 2014 6:35 PM PDT
Last Modified Fri 15 Jan 2016 8:48 PM PST