2048 (Merge)

Help Center

Overview

2048 is a simple grid-based numbers game. The rules of the game are described here.

In the first two assignments, we will implement a version of the 2048 game. Although the original game is played on a 4×4 grid, your version should be able to have an arbitrary height and width. In this first assignment, we will focus on only one aspect of the game: merging tiles.

We have provided the following template that contains the basic code that you will need to implement the merge function. The signature (name and parameters) of the functions in this file must remain unchanged, but you may add any additional functions or other code that you need to.

Coding Style

In this class, you will be asked to strictly follow a set of coding style guidelines. Good programmers not only get their code to work, but they also write it in a way that enables others to easily read and understand their code. Please read the style guidelines carefully and get into the habit of following them right from the start.

Testing Your Code

As always, testing is a critical part of the process of building your mini-project. Remember you should be testing your code as you write it. Don't try to implement everything and then test, You will have lots of errors that all interact in strange ways that make your program very hard to debug.

Throughout this course, we will be using a machine grader (OwlTest) to help you assess your code. You can submit your code to this Owltest page. The OwlTest page has a pale yellow background and does not submit your project to Coursera. OwlTest is just meant to allow you to test your mini-project automatically. 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 tests. To get you started, there are some tests for this project given at the end of the description.

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. Note that the CourseraTest page looks similar to the OwlTest page, but they are *not* the same! The CourseraTest page has a white background and does submit your grade to Coursera.

Merge

For this assignment, you will implement a function `merge(line)` that models the process of merging all of the tile values in a single row or column. This function takes the list `line` as a parameter and returns a new list with the tile values from `line` slid towards the front of the list and merged. Note that you should return a *new* list and you should *not* modify the input list. This is one of the more challenging parts of implementing the game.

In this function, you are *always* sliding the values in the list towards the front of the list (the list position with index 0). You will make sure you order the entries in `line` appropriately when you call this function later in the next assignment. Empty grid squares with no displayed value will be assigned a value of zero in our representation.

For example, if a row of the board started as follows:

And you slide the tiles left, the row would become:

Note that the two leftmost tiles merged to become a 4 and the third 2 just slides over next to the 4.

A given tile can only merge once on any given turn, although many pairs of tiles could merge on a single turn.

For the above example, the input to the `merge` function would be the list `[2, 0, 2, 2]`. The function should then produce the output `[4, 2, 0, 0]`. We suggest you begin to implement this function as follows:

1. Start with a result list that contains the same number of 0's as the length of the `line` argument.
2. Iterate over the `line` input looking for non-zero entries. For each non-zero entry, put the value into the next available entry of the result list (starting at position 0).

Notice if you only follow this process, you would end up with the result `[2, 2, 2, 0]`.

Now you should think through what you should add to your function in order to merge tiles. Keep in mind, however, that any tile should only be merged once and that these merges should happen in order from lowest index to highest index. For instance, on the input `[2, 0, 2, 4]`, the result should be `[4, 4, 0, 0]`, not `[8, 0, 0, 0]`.

Note that there are many ways to implement the `merge` function. The objective of this project is for you to use what you've learned in our previous classes to implement a complex function. You have already learned all of the Python required to implement `merge`, so the challenge is to think carefully about what the function does and how to best accomplish that.

Here is one basic strategy to implement the merge function:

1. Iterate over the input and create an output list that has all of the non-zero tiles slid over to the beginning of the list with the appropriate number of zeroes at the end of the list.
2. Iterate over the list created in the previous step and create another new list in which pairs of tiles in the first list are replaced with a tile of twice the value and a zero tile.
3. Repeat step one using the list created in step two to slide the tiles to the beginning of the list again.

This is *not* the most efficient way of implementing `merge`. While it is fine to implement it in this way, we challenge you to think of other ways of doing it that do not require creating so many lists. Ultimately, how you approach the problem is up to you.

As you work on your `merge` function, here are some simple tests you should try:

- `[2, 0, 2, 4]` should return `[4, 4, 0, 0]`

- `[0, 0, 2, 2]` should return `[4, 0, 0, 0]`
- `[2, 2, 0, 0]` should return `[4, 0, 0, 0]`
- `[2, 2, 2, 2, 2]` should return `[4, 4, 2, 0, 0]`
- `[8, 16, 16, 8]` should return `[8, 32, 8, 0]`

These tests are by no means exhaustive and are just meant to get you started.

Created Tue 27 Jan 2015 9:57 AM PST

Last Modified Wed 6 Jan 2016 12:07 PM PST