**2048**

Due Monday, July 19th at 10:30 AM

You know it… you’ve played it… the goal of this lab will be to recreate the iconic game, [2048](https://play2048.co/)! (If you haven’t played it before or just want a refresher, follow that hyperlink.) You’ll get plenty of practice working with loops, conditionals, and 2-D arrays as you work on this lab.

**Starter Code**

The code for this project consists of two files:

1. main.py contains the code that drives the game (i.e. parses user input)

* This file is already written for you and your main job will be to understand what is being done in it. It uses the functions defined in game.py by importing it at the top of the file.

1. game.py will contain all the functions that perform the logic of the game (i.e. moving the numbers to the left, checking if the user has won)

* This file has a handful of empty functions which are called in 2048.py that you will need to fill in, with contracts that will help you understand what they should do and return. These functions will be explained further in the following sections.

Try running the starter code! Not much will happen, but it’s good to know where you’re starting from.

*FYI: If you are looking for a challenge, feel free to ignore any/all starter code and take on the project from scratch!*

**Your Job**

Start by understanding the flow of the control loop in main.py. Take a look at the contracts of the functions that are called in main.py and defined in game.py to get a feel for what they do and why they are being used. If anything doesn’t make sense, any of the CAs will be happy to answer your questions or walk you through it!

Once you feel (at least somewhat) confident about the existing code, start working on the functions in

game.py. Use the details below as well as the provided function contracts to understand what each function is meant to do. Remember to keep your functions *modular* – each function should perform one concise task. You can always create more helper functions if you want!

**Functions**

* start
  + This function is provided to you. It initializes the game by creating the data structure which acts as the game board, providing initial instructions to the user, and populating the game board with the first two numbers. It prints this board before returning it.
* add\_new\_2
  + This function takes the game board data structure as a parameter and returns nothing. It needs to generate a row and column value that does not already contain a value and put a 2 in that spot.
* move\_left
  + This function takes the game board as a parameter and returns an updated game board. It should move all the values on the board as far to the left as possible as well as merge any neighboring and equal values in the same row. Something to consider when merging:
    - If there are three or four of the same value in a row, which pair(s) do you want to merge? Or do you want to merge them all into one? Feel free to go with whatever makes the most sense to you, or if it’s all the same, default to what the original game does.
* move\_right
  + Same as move\_left, except numbers should be moved and merged as far to the right as possible.
* move\_up
  + This function takes the game board as a parameter and returns an updated game board. It should move all the values on the board as far upwards as possible as well as merge any neighboring and equal values in the same row.
* move\_down
  + Same as move\_up, except numbers should be moved and merged as far downwards as possible.
* board\_full
  + This function takes the game board as a parameter and returns True if the board is full of non-zero values, and False otherwise.
* no\_more\_moves
  + This function takes the game board as a parameter and returns True if no more moves are possible and the game should end as either a win or a loss, and False if moves are still possible. There are a few cases in which moves are still possible, more than if there are still spaces left on the board – make sure you get them all.
* board\_full
  + This function is provided to you. It takes the game board as a parameter and formats its values into a 4 x 4 grid, with dashes (“-“) representing empty spaces, and prints the grid. It does not return anything.

**Hints and Tips**

It can be difficult, if not overwhelming, to envision how to implement this game at first. Even with some of the functions broken down and explained, actually defining something like move\_left involves more steps than you may realize. For example, a move to the left involves both shifting values all the way to the left *and* merging neighboring numbers that are the same. Consider doing these two actions in two separate helper functions, one for merging and one for shifting.

Something else to consider is that moving right is a lot like moving left… the numbers are just going in the opposite direction. Moving up is then a lot like moving right if you change the rows into columns, and moving down is a lot like moving up, again in the opposite direction. Before you write functions for shifting up, merging up, shifting right, merging right, and so on, you may want to see if you can leverage the way these transformations are related to limit the times you re-write very similar code.

If you have functionality questions you’re not sure about (e.g. what to do if three of the same number are neighboring), feel free to take creative liberty and do what you think makes sense. Otherwise, go into the [game](https://play2048.co/) and see what it does in that situation and replicate it!

When debugging your program, print print *print*. It’s unbelievably helpful to see what your code is doing, rather than just believing that it works how you think it should. For this project specifically, you can use the print\_board function to see the board at any given time.

**Looking for a challenge? (*very* optional)**

Right now, with the provided control loop in 2048.py, the game ends automatically when the user gets a 2048 (or whatever specified target value) on their board. Make the game so that the user gets a congratulatory message when they reach the target and then can choose to continue.

What if the user wants to play on a board of a different size? Make the game use a 5 x 5 board rather than 4 x 4. Or, better yet, allow the user to choose their board size and then actually use that size for their game.

Make an undo option for the game that allows user to go back one move or more… if you’re brave.

And anything else you want!