Project #1 (2048 Game)

CSC 172 (Data Structures and Algorithms), Fall 2017,
University of Rochester
Due Date: 10/03/2017 (11:59 pm)

Introduction

In this project, you will have the opportunity to create a text version of the popular game 2048 in Java. You will build the project from scratch, i.e., no sample code will be provided. If you have never played the game before, you can play it online at http://2048game.com/.

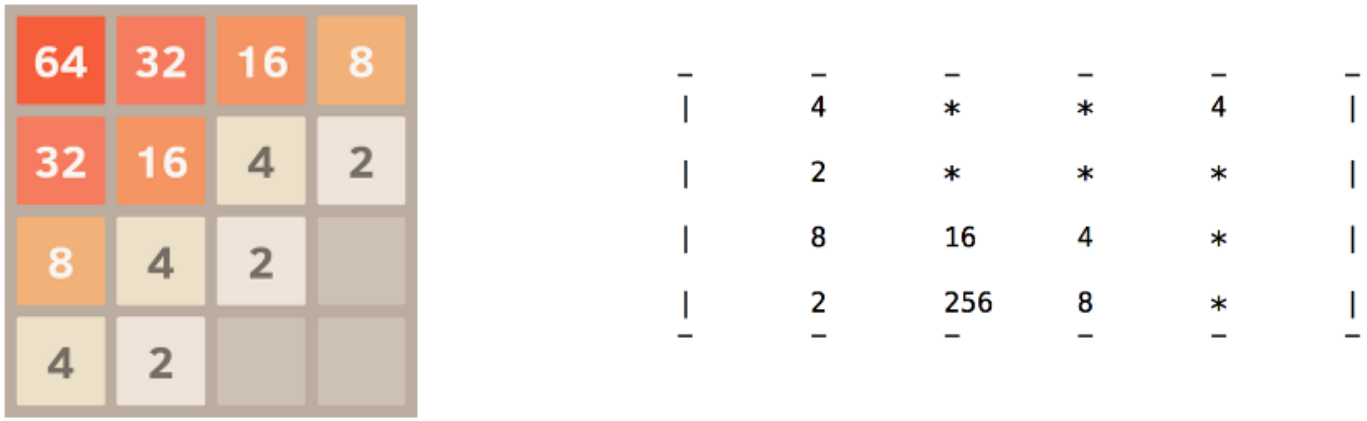

Figure 1: Examples of the 2048 game. The left one includes a colorful UI, the right one is a text-based version of the game which may look like this on the console. Feel free implement the colorful UI.

Background of Game

2048 is a single-player sliding block puzzle game by Italian web developer Gabriele Cirulli. The game's objective is to slide numbered tiles on a grid to combine them to create a tile with the number 2048. [1]

Specification

- You are going to implement a '2048' game which can be played by entering 'a', 's', 'd', and 'w' for left, down, right, up respectively.
- User should be able to quit or restart the game by entering 'q' and 'r' respectively. For quit and restart, the program must ask the user to confirm the action.
- Your program must be able to refresh the board after every key entered to reflect the updated board.
- You must write your code in Java. No other programming language is allowed throughout this course.

[1] cite from Wikipedia.

Your program must support the following functionality:

1. Move left, right, up, and down operations
2. Restart (New Game) or Quit the current Game options. Always ask user for confirmation
3. Except for the 6 keys (a,s,d,w,q, r), ignore any other key strokes.
4. Initial board should contain 2 random number (2 or 4) placed at any two random positions (If we restart the game, the position should vary)
5. Once the game begins, a new random number (2 or 4) need to be added for every valid move. If that can't be done, indicate the game is over
6. When generating a random number, make sure the probability of 2 and 4 is 0.8 and 0.2 respectively.
7. If a move does not change the position of any of the pieces, it's not a valid move. So, do not add any new piece
8. Keep a counter of how many valid moves are being made. At the end of the game, print the maximum number on the board, and the number of valid moves made.

Abstraction

Now, having you just "Build 2048" by yourself may be both too confusing and too big of a problem for you to struggle through as budding Computer Scientists, so this entire project is going to rely on the idea of abstraction. And what is abstraction? At the simplest level, abstraction helps us simplify complex programs. Using abstraction, we can work with our program using the appropriate details at any given step to help simplify our jobs while we ensure the program is working the way it needs to.

As you'll see, in practice, abstraction can help us divide and conquer by breaking large and complex problems which we may not be comfortable solving into bunches of smaller problems that we are comfortable solving. You will use the same idea through Labs and Workshops where you will focus on particular aspects of the project. With these smaller problems, the details we need to keep track of are often simplified less is going on and the problem at hand is much more manageable or requires less thought about the overall problem and more about a specific one. By chaining together our solutions to all of these smaller problems, we can then solve larger and larger problems, abstracting away the details of how we previously solved the small problems until we finally complete our overarching problem. Here is a way of splitting the problem:

Print a 4x4 2D array as the board

Write a simple function that can traverse a 2D array of Integer and prints its content. As the maximum number we are printing is 2048, make sure the printed numbers are aligned.

Randomly filling the Array

You can randomly populate the array with only 2 numbers at the beginning. The value of the numbers should be either 2 or 4 (2 with high probability and 4 with less probability).

Place a random number

Write a function that would add a new number to the current array. If there is no such position where a number can be written, your function should return False to indicate so (which usually indicates the game is over)

Move in a particular direction

This is the key function of the project. In this step, we will handle how to make a move, i.e moving to the right, left, up, or down. Once you read a moving command from the keyboard, your program should go into the corresponding 'if' statement and execute the corresponding moving logics. You will work on during Lab and Workshop sessions.

Let's take moving right as an example for the moving logics. Once a 'd' is read, you need to initiate moving right logic. Your code should go though all rows and perform the following two things:

- If two values are the same and there is no the number between them, add them up and place in the right most available place. (be careful about the calculating order if there are three or more same numbers in a row).
- If values are not the same and there is available space on the right, move them accordingly.

Ending a move

Next step is to see how to combine them and make an actual move. Once this is done, your game can move and know when to end, which means it is functioning. Once all your tests pass, you'll most likely have an awesome, bug-free version of 2048 that you can play on your computer!

In this step, you need to figure out how you should end a move using some of the abstractions that you have built.

First off, what is a move? Well, moves occur when you swipe in a direction, right? However, that's not the entire story, because every moves places another random piece on the board. So here's a list of the things you'll need to do for any move:

- Read commend from keyboard and calculate moving logic
- Add a number randomly on the board (if space is available; otherwise, end game)
- Clear the screen
- Print out the board again

In addition, every time you print out the board, you must clear the screen right before you print (see useful links) or your game will not look very good. The provided tests will hopefully ensure you don't make that mistake, but to reiterate: you must clear and print the board together. Every time. Do not forget this part!

How to Submit

Zip (archive) all the project source files and a READ ME file and save it as Proj1.zip. Upload the file to Blackboard.

Your README file should include any specific instruction which is useful for the project. It should also include all the features (including additional features) that you have implemented.

Make sure all your source files are properly commented so that user can browse your code without getting lost.

Grading Policy

- 0 pts if your code does not compile/plagiarism
- 5 pts if your code compiles correctly (Even an empty project that compiles without error)
- 15 pts if your code displays the initial board and quits and restarts without error/exception
- 80 pts for other functionalities

Useful Links

Intro of the game: https://en.wikipedia.org/wiki/2048_(video_game)
Python implementation: http://nifty.stanford.edu/2017/mishra-2048/#overview

Acknowledgement

Part of the text is taken verbatim from '2048 in Python' designed by Kunal Mishra'. The project is one of the nifty assignments designed for distributing great assignment ideas and their materials. Please refer to http://nifty.stanford.edu/ for further details. You are most welcome to go through materials posted there for suggestions and recommendations. But make sure your code does not violate the honesty policy of U of R.