FrontEnd-and-Dashboard-Developer

2048 Game

(Yet another) ***2048 game*** clone developed with ***Python*** and ***TKinter***.

Problem Statement

https://github.com/Prophet1999/FrontEnd-and-Dashboard-Developer/blob/14b43b6e600fd85ce3b766f6c39438e49098d345/Problem%20Statement.pdf

Overview

About the Game

All the playing methods are the same, all you have to do is move numbers on a grid to merge them to form the number 2048; after all, you can keep on playing the game by creating larger numbers. In each move, a new number of displays (2 or 4). The player has to move the numbers using arrow keys and try to collide the same two numbers which create a new number with the total sum of the two numbers. The gameplay design is simple that the user won't find it difficult to use and navigate.

Requirements

* ```Python 3.5```

* ```Tkinter```

Module for Coding a user-friendly interface as introduced in class.

It contains many function such as create_text(), create_rectangle(), label(), button(), bind_all() that makes the user interface interactive.

Design Principles

2048 is a simple mathematics puzzle game. It is a really addictive game and the main operation performed in this game is addition which makes it easy for all of us.

How to play 2048 :

1. There is a 4*4 grid which can be filled with any number. Initially two random cells are filled with 2 in it. Rest cells are empty.

2. we have to press any one of four keys to move up, down, left, or right. When we press any key, the elements of the cell move in that direction such that if any two identical numbers are contained in that particular row (in case of moving left or right) or column (in case of moving up and down) they get add up and extreme cell in that direction fill itself with that number and rest cells goes empty again.

3. After this grid compression any random empty cell gets itself filled with 2.

4. Following the above process we have to double the elements by adding up and make 2048 in any of the cell. If we are able to do that we wins.

5. But if during the game there is no empty cell left to be filled with a new 2, then the game goes over.

In above process you can see the snapshots from graphical user interface of 2048 game. But all the logic lies in the main code. So to solely understand the logic behind it we can assume the above grid to be a 4*4 matrix (a list with four rows and four columns). You can see below the way to take input and output without GUI for the above game.

Example :

Commands are as follows :

'↑' : Move Up

'↓' : Move Down

'←' : Move Left

'→' : Move Right

[0, 0, 0, 0]

[0, 0, 0, 0]

[0, 0, 0, 0]

[0, 0, 2, 0]

Press the command : ←

GAME NOT OVER

[0, 0, 0, 2]

[0, 0, 0, 0]

[0, 0, 0, 0]

[2, 0, 0, 0]

Press the command : ↓

GAME NOT OVER

[0, 0, 0, 0]

[0, 0, 0, 0]

[0, 0, 2, 0]

[2, 0, 0, 2]

Press the command : →

GAME NOT OVER

[0, 0, 0, 0]

[0, 0, 0, 0]

[2, 0, 0, 2]

[0, 0, 0, 4]

Press the command : ←

GAME NOT OVER

[0, 2, 0, 0]

[0, 0, 0, 0]

[4, 0, 0, 0]

[4, 0, 0, 0]

Press the command : ↓

GAME NOT OVER

[0, 0, 0, 0]

[0, 0, 0, 0]

[0, 0, 0, 0]

[8, 2, 0, 2]

.

.

.

And the series of input output will go on till we lose or win!

Programming Approach :

We will design each logic function such as we are performing a left swipe then we will use it for right swipe by reversing matrix and performing left swipe.

Moving up can be done by taking transpose then moving left.

Moving down can be done by taking transpose the moving right.

Code Walkthrough

1. Board:

Variables:

Bg_color: It is a dictionary that stores background color for every cell.

Color: It is a dictionary that stores foreground color for every cell.

Window: It is the main tkinter window.

gameArea: It is a tkinter frame widget.

gridCell: It is a 4×4 integer matrix which stores the actual integer value of all the cells.

Board: It is a 4×4 grid of tkinter label widget which displays the value of the cell on tkinter window. It is also used to configure the background and foreground of the cell according to its gridCell value.

Score: It stores the current score of the player.

Rest are just flag variables.

Functions:

__init__(self): It is the constructor function. It initializes all the variables with appropriate default values like '0' for gridCell, False for moved, merge and so on.

Reverse: It reverse the gridCell matrix.

Transpose: It uses zip function and takes transpose of the gridCell matrix.

CompressGrid: It moves all not empty cells to the left, so that merging can be done easily.

mergeGrid: It adds the gridCell value of two adjacent cells if they have same gridCell values.

Random_cell: It first stores all the empty cells in a list and then picks a random cell from the created list and make its gridCell value 2

Can_merge: It returns a boolean value denoting we can merge any two cells or not. We can merge two cells if and only if they hold the same gridCell value.

paintGrid: It assigns foreground and background color to each cell of the 4×4 grid corresponding to its gridCell value.

2. Game:

This class doesn't have many variables, it only has some Boolean variables indicating game status.

Functions:

__init__(self): It is the constructor function. It initializes all the variables with appropriate default values.

start: It calls random_cell twice to assign '2' to gridCell value of two random cells and then it paints the grid and after that, it calls link_keys to link up, down, left, and right keys.

Link_keys: First of all it checks if the game is already won or lost, and if it is, it executes a return statement without doing anything. Otherwise, it continues its execution.

Approach:

For left swipe, we will just compress and then merge the gridCell matrix and then if compress or merge is true (indicating the values of the matrix is affected by previous two functions), then we need to compress the grid again.

For moving up, we will take transpose then swipe left and again take transpose to return to the original order.

Moving down is same as moving up but we need to reverse the matrix.

Similarly, right is same as moving left+reverse.

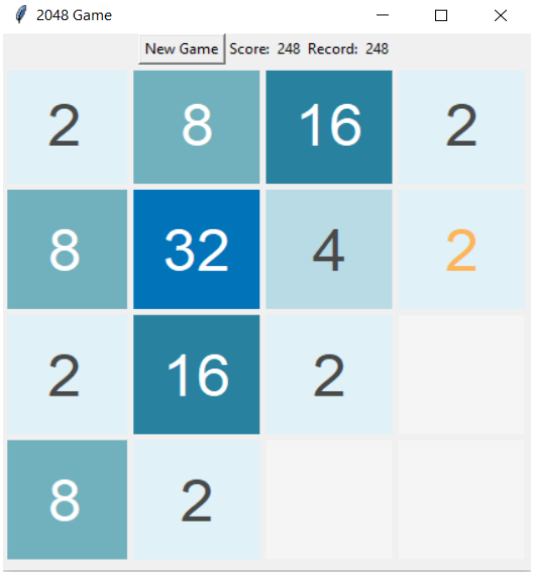
After every operation, we need to check the game status, if all cells are occupied and we cannot even merge any two cells i.e. the state where no movement can change the matrix, then the game is over.

If any cell value has reached **2048**, then the player is won and a message box is flashed on the screen announcing the winner.

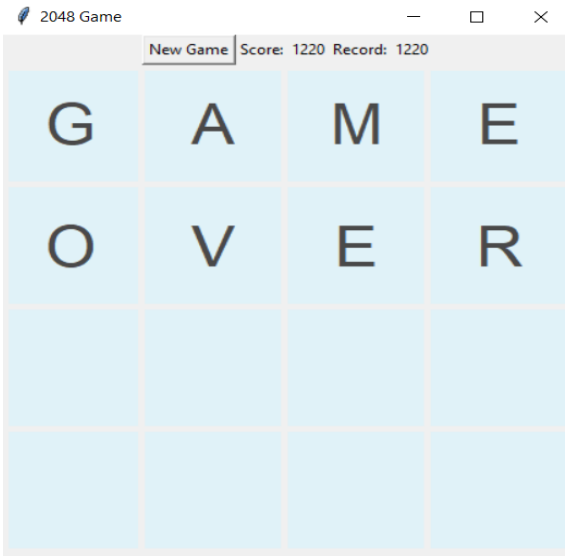
Screenshots

Screenshot 1

Screenshot 2



Screenshot 3



Screenshot 4

Change from 2048 to 4096 as end number

In the original game, sometimes the new tile that pops up is a 4. Let's ignore that possibility for the sake of simplicity.

We've 16 squares in total and considering the new tile popped up at the position we chose, to make it to 4096, we need minimum these tiles:

2048, 1024, 512, 256, 128, 64, 32, 16, 8, 4, 2, 2

These can be combined from right to left to get to 4096. That's a total of 12 tiles. Since that's less than 16 (max number of tiles that can fit), 4096 is possible. Following along:

8192: 4096, 2048, 1024, ..., 4, 2, 2 Total 13

16384: 8192, 4096, 2048, ..., 4, 2, 2 Total 14

32768: 16384, 8192, 4096, ..., 4, 2, 2 Total 15

65536: 32768, 16384, 8192, 4096, ..., 4, 2, 2 Total 16

Next is 131072 which will require at least 17 tiles which can't fit on the board.

So, the largest possible tile is 65536.

EDIT: Since the game pops up some 4's at times, 131072 is still theoretically possible if the last tile that pops up is a 4 making the board look like this:

131072: 65536, 32768, 16384, 8192, 4096, 2048, 1024, 512, 256, 128, 64, 32, 16, 8, 4, 4

that's 16 tiles in total.

So now we know that 65536 is reachable in 4x4 grid so we just have to modify our result value from 2048 to 4096.

Whenever we encounter 4096 as a value on any one grid cell after each operation we just exit the program and flash "You Win".

GitHub Repo Link

https://github.com/Prophet1999/FrontEnd-and-Dashboard-Developer

How To Run The Project?

To run this project, you must have installed Python on your PC. After downloading the project, you have to follow the steps below:

Step 1: Extract/Unzip the file if the file is zip or rar.

Step 2: Install and update all Python Modules and Libraries specially TKinter UI as it is needed for the project.

Step 3: Open latest version of Python IDEL then open "Source Code.py".

Step 4: After file is opened just run the module to see the magic happen!!!

Scope of making this an 8x8 from 4x4

2048 is a highly popular game even though it's quite old and really simple as compared to other games today.

2048 looks very easy when you start playing it, but it gets tougher and tougher with time.

But this is not the case with 2084 8x8. The game is really easy as you have a bigger board of 8x8 blocks to achieve the target of 2048.

It is similar to 4x4 grid except the fact that here we have to use 8x8 grid.

Difference Between 4x4 And 8x8 Grid

The classic 2048 game has a 4x4 board. And the user has to make a single tile display 2048 as the final number in order to win the game. 4x4 tiles make the game very hard and you have to think twice before taking the next step. But this is not the case with 2048 8x8. 2048 tile can easily be obtained even if you randomly move the tiles. The reason being, the tiles are four times as compared to what we get in 2048.

Thoughts / Probable solutions / Problems faced while designing the solution

1. The Matrix manipulation and handling was tricky and difficult.

2. There is no way to optimize the operations on the matrix.

3. Lots of research needed for the algorithm.

4. Minimum Understanding while playing the game.

5. Adding the scores and leaderboards was a time taking process.