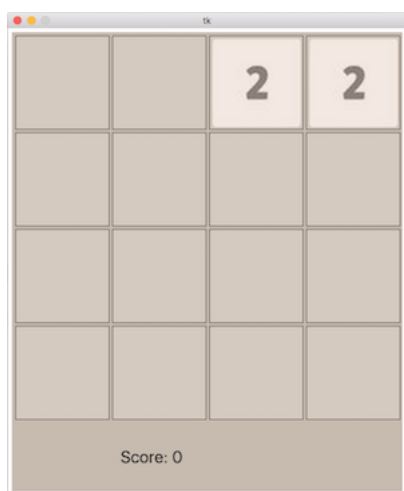


COMP1127 Project 2048

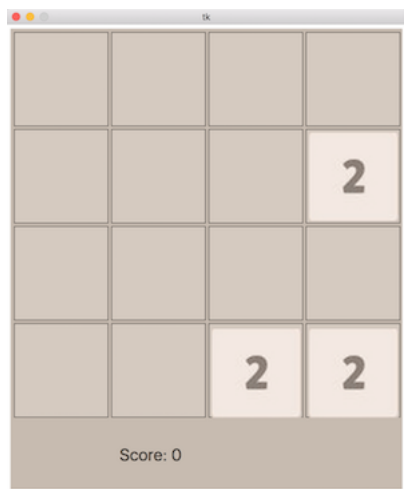
November 2, 2016

Game Description

2048 is a game with a 4x4 grid with numbered tiles, to play 2048 use the arrow keys to move all the tiles in the direction of the arrow key pressed. When two tiles with the same number touch they merge into one and the new tile is their sum. The goal of the game is generate a tile with the number 2048. 2048 starts with a board with just two numbers randomly placed on it as depicted



After each move after all the tiles have been moved in the specified direction, a new number (either a 2 or a 4) is randomly placed on the board. So if we look at the board shown above, if a user were to press the down arrow key both the twos on the board would shift all the way down and a new number would now randomly appear as depicted



Also remember when two tiles with the same number collide when moving then they combine to make one tile with a value of the sum of both of the tiles. So for example using the board above if the user were to now press the left arrow key the board would now look as depicted



Note the both twos on the last row combined to make a 4 and the two on the second row also moved all the way to the left. Also note a new two was randomly placed on the board on the last row.

One more thing to note is that pressing the left key again would do nothing to the state of the board, since nothing can move to left based on the current state of the board, but you can still move right, up or down.

Background

Shanice is a Computer Science student who really likes to play the game 2048 and also enjoys programming in Python. Shanice decided she wants to make her own 2048 game in Python and managed to build a graphical user interface(GUI) for the game but needs a little help getting the game to work properly. Help Shanice finish her game by completing the code she has so far.

Project Background

Shanice has given you what she has written so far, this includes an img folder, a gui.py file and a game.py file, she reminds you to ensure all these files in the same directory in order for the game to work properly and also tells you to ensure you are using Python 2.7.x, the code won't work so well with Python 3. First, you will need to set up your machine to run the gui, the first step is to install the Python Library 'Pillow' from PyPi. See below for the setup instructions based on your operating system:

If you need help please ask a Labtech to help you get it set up.

OS X or Linux

Open the terminal application and type the following command:

```
sudo pip install Pillow
```

Then you'll be good to go!

Windows

You will need to navigate to PyPa's website and install the latest version of pip for windows <https://pip.pypa.io/en/stable/installing/>. Then navigate to your command Prompt and execute the following command:

```
pip install Pillow
```

Then you'll be good to go!

Now once you've finished the setup you should be able to run the game.py file and see an empty game board. You can now go on to use your Python expertise to help Shanice finish up her game.

Shanice was kind enough to give us a description and some examples on what she has written. **Please Do Not Modify Shanice's Code**

Table 1: Shanice's Documentation

| Code | Description | Usage example |
|-------------------|--|---|
| setup() | This function sets up the window frame and the game board that will be drawn for you. No GUI operations can happen before this function is called. This function returns the frame ADT and the board ADT as a tuple. Throughout the code this frame or the board will be required as parameters in the functions. Wherever there is a reference to the Window Frame ADT and the Game Board ADT it is referring to the two values returned from this function call. Game Board has a special attribute called score that stores the value of the score, if updating the score update this value before trying to update the score on the GUI. | <pre>>>>import gui >>>frame, board = gui.setup() >>>board.score 436 >>></pre> |
| start(game_frame) | This function starts the game. It takes a Game Frame as it's only parameter and then starts the game by displaying the board. All code related to setting up the board must happen before starting the game. | <pre>>>>import gui >>>frame, board = gui.setup() >>>gui.start(frame)</pre> |
| random_number() | This function returns a random NumberTile, a NumberTile will be used to represent a tile on the board and has two attributes a value and an image. To access the image or value of a number you need to store it in a variable and then use the '.' notation. | <pre>>>>import gui >>>x = gui.random_number() >>>x.value 2</pre> |

| | | |
|---|--|--|
| put(game_board, key, number, row, column) | This function takes 5 parameters as input: the first is the Game Board, the second is a unique key as a String to identify the NumberTile being placed on the board, the third is NumberTile to be placed on the board and then the final two parameters are integers row and column that specify the row and column where the tile should go on the board. The function also returns True if it successfully placed the tile on the board. The Game Board also has a special attribute called numbers which is a dictionary mapping the unique identifier to a tuple containing the row and column of that tile. This function adds to that dictionary mapping and maps the unique String identifier passed to it to a tuple containing the row and column the number was placed at on the board. | >>>import gui >>>frame,board = gui.setup() >>>gui.put(board, "1", gui.random_number(), 0, 2) True >>>board.numbers { "Tile number 1": (0,2) } >>> |
| find_position(game_board, key) | This function takes two parameters as input: the first is the Game Board and the second is the unique String identifier for the NumberTile you want to find the position of. | >>>import gui >>>frame,board = gui.setup() >>>gui.put(board, "1", gui.random_number(), 0, 2) True >>>gui.find_position(board, "1") (0,2) >>> |
| remove_number(game_board, key) | This function takes two parameters as input: the first is the Game Board and the second is the unique String identifier for the NumberTile you want to remove from the board. | >>>import gui >>>frame,board = gui.setup() >>>gui.remove_number(board, "id1") |
| find_number(number) | This function takes one parameter, an Integer and returns it's NumberTile representation. The possible numbers are 2,4,8,16,32,64,128,256,512,1024 and of course 2048. | >>>import gui >>>x = gui.find_number(8) >>>x.value 8 |

| | | |
|---|---|--|
| <code>move_number(game_board, key, direction, update_grid, move_by_distance)</code> | <p>This function takes 5 parameters as input and moves a tile by one slot in the specified direction: the first is the Game Board, the second is a unique key as a String to identify the NumberTile being placed on the board, the third is an Integer representing the direction to move the tile and then the final two parameters are two functions: the first to update the state of the grid, this should be a function that takes 3 parameters and the other should be a function that takes 5 parameters and actually animates the movement of a tile across the board, these will be explained in more detail later. The function returns True if it was able to successfully move the Tile, False otherwise</p> | <pre>>>>import gui >>>frame,board = gui.setup() >>>gui.move_number(board, "1",1, some_function, some_function) False >>></pre> |
| <code>update_score(game_board)</code> | <p>This function updates the score displayed on the GUI at the bottom of the board. It takes one parameter which is the Game Board</p> | <pre>>>>import gui >>>frame, board = gui.setup() >>>board.score = 1000 >>>gui.update_score(board)</pre> |
| <code>move_tile(game_board, key, horizontal, vertical)</code> | <p>This function updates the NumberTile on the GUI by moving it horizontally and vertically by the distance specified. The function should take 4 parameters: a Game Board, the unique identifier as a String, the distance to move the tile horizontally as an integer and the distance to move the tile vertically as an integer. Note: a negative value for the horizontal parameter will move the tile to the left, while a positive will move it to the right and a negative value for the vertical parameter will move the tile upwards, while a positive will move it downwards.</p> | <pre>>>>import gui >>>frame, board = gui.setup() >>>gui.move_tile(board, "1", -20, 0) >>></pre> <p>The example above would move the Tile identified by "1" to the left by 20 pixels on the GUI</p> |
| <code>game_over(game_board, winner)</code> | <p>This function takes two parameters a Game Board and a boolean value. The boolean value should be True if game is over and the user reached 2048 and should be False otherwise. The function updates the GUI to display an appropriate message to the user.</p> | <pre>>>>import gui >>>frame, board = gui.setup() >>>gui.game_over(game_board, True)</pre> |

Problem Set

Please look through the code given in the main.py file, Shanice tried her best to get you started, use the comments to better understand what the purpose of the code in the file is.

Problem 1

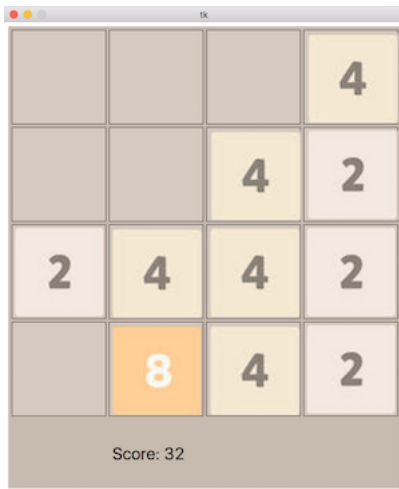
Shanice couldn't decide how to represent the game board's 4x4 grid as code in python. You will represent the board as a multidimensional list, where 0 would represent an empty slot in the grid. Initialize the grid variable to be a 4-dimensional list where all the slots are empty (i.e. all the slots are 0).

[1 mark]

Problem 2

Throughout the game you're going to need to know which slots on the board are empty. Write a function called ***empty_slots*** that returns all the empty slots on the grid. The function should take no parameters and return a list of tuples where each tuple represents the row and column of an empty slot. **Note: Rows and columns start from counting 0**

For example if the grid represented a board that looked as depicted



| | | | |
|---|---|---|---|
| | | | 4 |
| | | 4 | 2 |
| 2 | 4 | 4 | 2 |
| | 8 | 4 | 2 |

Score: 32

The following would be the expected result of your function

```
>>>empty_slots()
[(0,0), (0,1), (0,2), (1,0), (1,1), (3,0)]
>>>
```

[2 marks]

Problem 3

Remember in 2048 after each move a new number is placed at a randomly chosen empty slot. Write a function called ***random_position*** that returns a random empty slot as a tuple containing the row and column. **Hint: The python random library might be helpful**

Using the above board as an example the following result might be expected

```
>>>random_slot()
(0,2)
>>>random_slot()
(3,0)
```

[1 mark]

Problem 4

You now have enough code to start putting random numbers on the game board. Write a function called ***add_random_number*** that takes a game board as it's parameter and places a random NumberTile on a random slot on the board and also updates the grid variable to match the state of the board. This function should return the row and the column of the newly added tile as a Tuple. Make use of the functions you have already written and those available to you through the gui library Shanice provided. **Since you will need to ensure the identifier for each number is unique Shanice provided you with a dictionary called helper that can be used to ensure each key is unique, assign each Tile a unique key with the value "Tile number x" where x is a number unique to that tile. Use the "count" entry to help ensure the key is unique for each number by incrementing it each time a new number is added onto the board. Also remember the identifier has to be a String**

[3 marks]

Go ahead and test out your code now. Inside of the main section at the bottom of the file after the board is set up add two random numbers onto the board. The main should look something like this

```
if __name__ == '__main__':
    frame, board = gui.setup()

    # Finishing setting up your GameBoard here
    add_random_number(board)
    add_random_number(board)

    gui.start(frame)
```

Problem 5

When moving the tiles in the various directions across the board we're going to need to be able to reference each tile by it's unique identifier. Write a function called ***find_identifier*** that takes the Gameboard, the row as an integer and the column as an integer as parameters and returns the unique key for that tile.

Remember the GameBoard has a special dictionary attribute called numbers that maps the unique identifier to the row and column as a tuple

```
>>>find.identifier(board, 0, 2)
"Tile number 1"
>>>
```

[2 marks]

Problem 6

You'll also need to update the grid variable based on the move that takes place. Write a function called ***update_grid*** that takes 3 parameters: the row as an integer, the column as an integer and the direction as an integer. Shanice was nice enough to define the directions as integers for us at the top of the file as well. Based on the direction update the grid variable by taking the value at the row and column passed in

and move it by one space in the direction that was passed in, if the move is possible update the grid and return the new row and column as a tuple, if the move is not possible return the row and column that was passed to the function as a tuple.

A move is only possible if there is an empty slot in the direction of the new slot based on the direction or if that slot has the same number. If the slot has the same number the resulting slot should contain the sum of the values

See below for an example

```
>>>LEFT = 4
>>>grid = [[0,2,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]]
>>>update_grid(0,1, LEFT)
(0,0)
>>>grid
[[2,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]]
>>>grid = [[2,2,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]]
>>>update_grid(0,1, LEFT)
(0,0)
>>>grid
[[4,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]]
>>>grid = [[2,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]]
>>>update_grid(0,0, LEFT)
(0,0)
>>>
[7 marks]
```

Problem 7

You need a function to animate the movement of the tile on the GUI. You will animate the Tile by moving it in the specified direction in small increments until it has moved the full distance it needs to go. Write a function called ***animate_movement*** that will take 5 parameters: the GameBoard, the unique identifier, the full distance to move horizontally as an Integer, the full distance to move vertically as an Integer and the direction which is also an Integer. Shanice defined one more variable for us called `transition_value`. Your function will recursively move the tile by this value until the distance left to be moved is less than the value of `transition_value`, when this is the case it will move the number by that remaining distance and return True, your function should return False if it was unable to move the Tile.

[12 marks]

Problem 8

Remember after each move, tiles will need to be moved in the direction specified until it either reaches the end of the board or the number beside it wasn't identical. Write a recursive function called ***move***, this function takes the Game Board, the identifier for the tile and the direction it should move the tile. This function will move the tile by one slot in the direction specified until it is no longer able to do so.

[6 marks]

You've shown Shanice what you have so far and she's impressed! She asks you to test your code by modifying the main section at the bottom of the file to look something like this.


```

if __name__ == '__main__':
    frame, board = gui.setup()

    # Finishing setting up your GameBoard here
    row, column = add_random_number(board)
    mkey = find_key(board, row, column)
    frame.bind("<Down>", lambda event: move(board, mkey, DOWN))

    gui.start(frame)

```

You should now be able to run the game and if the random tile generated wasn't at the bottom of the board already then by pressing the down arrow key the tile should move all the way to the bottom. If this is not the result you should go back through the functions you have written so far to ensure you've done them correctly. Experiment by replacing "<Down>" with other values from the controls array at the top of the file, "<Down>" will respond to the down arrow key, "<Up>" will respond to the up arrow key, "<Right>" will respond to the right arrow key and "<Left>" will respond to the left arrow key. Shanice will tell you a little more about how this works soon.

Now that you have a function that can move a single Tile in a direction but remember after each move all the tiles on the board should move in that direction. You now need to write a function that can move all the tiles on a board in a direction.

Problem 9

Write a function called *move_all_down* that moves all the tiles on the board downwards. This function should take one parameter, the GameBoard. The function should start moving tiles from the first column and move each tile in that column downwards starting from the lowermost tile and then follow the same procedure for the rest of the columns.

[5 marks]

Problem 10

Write a function called *move_all_up* that moves all the tiles on the board upwards. This function should take one parameter, the GameBoard. The function should start moving tiles from the first column and move each tile in that column upwards starting from the uppermost tile and then follow the same procedure for the rest of the columns.

[5 marks]

Problem 11

Write a function called *move_all_right* that moves all the tiles on the board to the right. This function should take one parameter, the GameBoard. The function should start moving tiles from the first row and move each tile in that row to the right starting from the rightmost tile and then follow the same procedure

for the rest of the rows.

[5 marks]

Problem 12

Write a function called *move_all_left* that moves all the tiles on the board to the left. This function should take one parameter, the GameBoard. The function should start moving tiles from the first row and move each tile in that row to the left starting from the leftmost tile and then follow the same procedure for the rest of the rows.

[5 marks]

Problem 13

Now you have code that can move all the tiles in any direction now you need a way to decide when to use which function. Write a function called *move_all* that takes 2 parameters the GameBoard and an event variable. The function should then based on the direction specified by the event variable move all the tiles on the board.

An event variable represents which key on the keyboard the user pressed. For example if you named the event variable event, then to access the String value of the key the user pressed you just use type event.keysym.

Here is what the event.keysym would return if the arrow keys were pressed.

Right Arrow Key - event.keysym would return "Right"

Up Arrow Key - event.keysym would return "Up"

Left Arrow Key - event.keysym would return "Left"

Down Arrow Key - event.keysym would return "Down"

Shanice provided you with a dictionary called helper at the top of the file that might be helpful however, you are not required to use it.

[3 marks]

Shanice sees how well you're progressing and now decides to inform you that the Window Frame returned from gui.setup() can be used to assign and unassign keyboard presses to a function using the *bind* and *unbind* functions. See below for an example

```
if __name__ == '__main__':  
    frame, board = gui.setup()  
    frame.bind("<Down>", lambda event: print event.keysym)  
    frame.unbind("<Down>")
```

She lets you know that bind takes two parameters, the first parameter to bind is a string representing the keyboard key and the second is function that should take one parameter, which is the event variable. Shanice also gave us a list called controls that has the strings representing each arrow key which can be found at the top of the file.

unbind takes one parameter which is the string representing the keyboard key.

Problem 14

Write a function called **keyboard_callback** that takes an event variable, Window Frame and the Game-Board. This function should move all the tiles on the board and then add a random number to the board. However the function should only add a new number to the board if after trying to move all the pieces on the board, it was able to move at least one. For example look at the following board



If a user pressed the up arrow key, the board would not change since all the tiles are already at the top, therefore a new number would not be added to the board.

Hint: use the grid variable to help

[7 marks]

Problem 15

Write a function called **bind** that takes 2 parameters, the Window frame and the Game Board and binds all the strings in the controls list to the **keyboard_callback** function. Use the map operation and lambda functions to accomplish this.

[6 marks]

Problem 16

Write a function called **unbind** that takes 1 parameter, the Window frame unbinds all the strings in the controls list from keyboard_callback function. Use the map operation and a lambda function to accomplish this.

[6 marks]

Problem 17

Awesome the game is almost finished, finish setting up your Game Board and modify the main section at the bottom of the file so that it adds 2 numbers onto the board and then binds the arrow keys to the

appropriate function.

[2 marks]

You should now be able to run the program and interact with the board using the arrow keys but as expected the game still isn't functioning properly. We need to add code so that when 2 tiles of the same number combine the new tile shown is their sum. Also the game needs to keep track of the score.

Each time a tile is merged the score is increased by the value of the new tile.

Problem 18

At the moment when two tiles combine one is just being placed above the other in the GameBoard, both tiles will have the same value in the Game Board numbers dictionary. Write a function called ***merge*** that takes two parameters the Game Board and a unique identifier for the tile that might have just combined with another. In this function you will need to get the row and column for the given identifier and then using the GameBoard numbers dictionary verify if there is another tile that in the same position, if there is then you will need to remove it from the dictionary, remove both tiles from the board, update the score and put a new tile with the new value in the same position and then return True. If there is no other tile in the same position return False.

[9 marks]

Problem 19

You now need to use or merge function when we're animating. Modify ***animate_movement*** so that when the distance left becomes less than the transition value it not only moves the tile by that distance but checks if the tile was merged. If it was then return False so that the game know to stop moving, otherwise return True.

[5 marks]

At this point your game is now ready to be played! You should be able to play the game and watch the numbers add up as you or until you run out of moves. Shanice thanks you but asks you to do a couple more things.

Problem 20

At the moment the game doesn't let you know when you've won or when you have no more moves left. Write a function ***is_game_over*** that takes the Game Board as a parameter and uses the gui library to show if the user won or not and returns True if the game is over, False otherwise.

[10 marks]

Problem 21

To improve gameplay, modify ***keyboard_callback*** to unbind the keyboard before beginning to move all the pieces and then rebind them back to the function if the game is not over.

[4 marks]

You're finished, Shanice is very grateful for your help!

Challenge

There still might be a few Kinks in the game that you might like to fix or additions you think might make it better. Save this file as a separate file called bonus when submitting. The main.py file should contain no bonus code.