॥ सिद्धिः भूषयते विद्याम् ॥

avantika
UNIVERSITY

CS3001 – ENGINEERING COMPUTING

Assignment 1

Nitya Kasturey AU19B1004

Bachelor of Technology

(2019-2020)

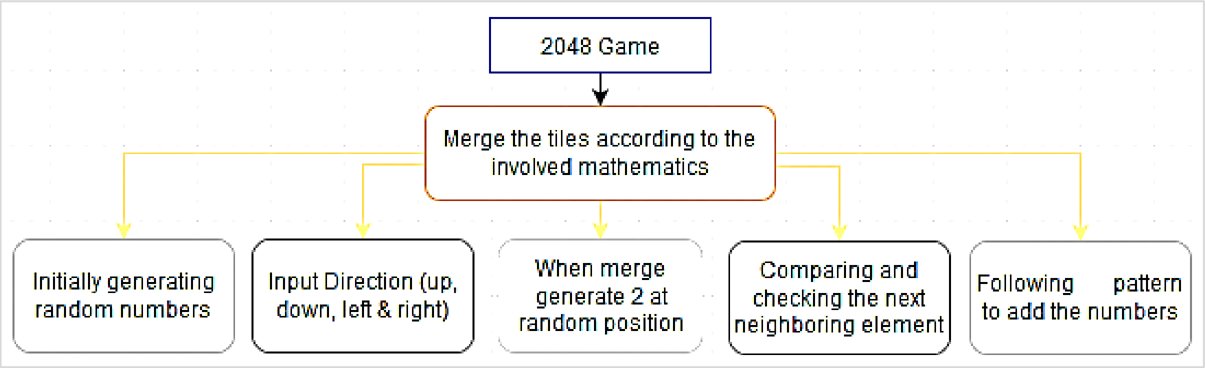
ABSTRACT

The **2048 puzzle game** deals with **merging, generating and adding of the tiles**. It involves **logical interactions** with different tiles following certain manner throughout the game. The exploration of mathematical ways to combine tiles makes it interesting and fun. The paper attempts to **analyse the mathematics and merging** involved in the game. A problem needs to be solved followed by **Computational thinking** consisting of Decomposition, Abstraction, Algorithm and Pattern recognition. Here the aim is to make a **python program** to **merge** the tiles according to the implemented mathematics. Using the programming language python to make the merging function to perform the user required tasks easily.

INTRODUCTION

2048 is sliding block puzzle game generally played on a 4x4 grid with tiles numbered $\mathbf{2^n}$ where **'n'** represents a **natural number**. The objective of the game is to combine tiles of the same number to eventually form the number 2048. The user can move in the four cardinal directions and after every move a new tile is generated randomly in the grid which is either numbered 2 or 4.

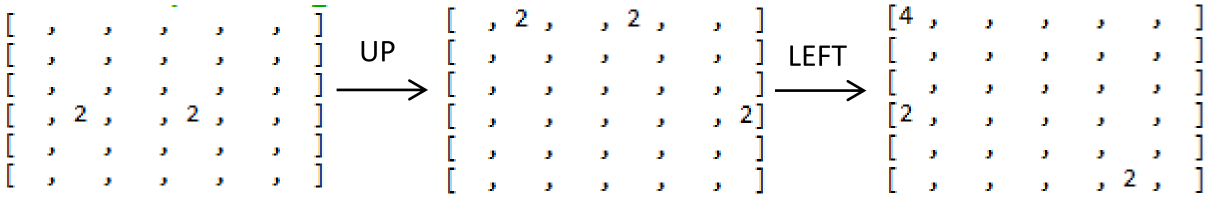
Here the grid will be **6 x 6** consisting the basic necessities of the game. The tiles in the game are the elements of the list which will be merged. The grid is 6 x 6 **matrix** in the form of **nested list**. The problem identified is basically to merge the tiles, this need to be solved according to the components of computational thinking and finally bringing the elements together to merge. **Decomposition**, breaking the problem into constituent sub-problems, Merging is decomposed in the following problems to easily program the solution.


2048 Game
Merge the tiles according to the involved mathematics
Initially generating random numbers
Input Direction (up, down, left & right)
When merge generate 2 at random position
Comparing and checking the next neighboring element
Following pattern to add the numbers

While merging, an important aspect is to add the numbers in the way of $\mathbf{2^n}$. Adding the numbers resulting into next power of 2 (n+1) is the **pattern** followed while merging of the tiles. The numbers are added with an increment in 'n' i.e. 2^{n+1}.

For eg: n = 1: $2^1 + 2^1 = 2^2 = 4,$
n = 2: $2^2 + 2^2 = 2^3 = 4,$

Python Program have a class named Grid that consist multiple functions named as random_two, merge etc. The merge function will take user input that will be the direction to merge the elements accordingly. In case of upward direction, merge function will add the equal elements in that column provided the number is in next of the current one, and will make the all elements to direct upward.



```
[  ,  ,  ,  ,  ,  ]
[  ,  ,  ,  ,  ,  ]
[  ,  ,  ,  ,  ,  ]
[  , 2,  , 2,  ,  ]
[  ,  ,  ,  ,  ,  ]
[  ,  ,  ,  ,  ,  ]
```

Original grid (nested list)

```
[  , 2,  , 2,  ,  ]
[  ,  ,  ,  ,  ,  ]
[  ,  ,  ,  ,  ,  ]
[  ,  ,  ,  ,  , 2]
[  ,  ,  ,  ,  ,  ]
[  ,  ,  ,  ,  ,  ]
```

All the elements are directed upwards, generating 2 at random position i.e. here at right

```
[4 ,  ,  ,  ,  ,  ]
[  ,  ,  ,  ,  ,  ]
[  ,  ,  ,  ,  ,  ]
[2 ,  ,  ,  ,  ,  ]
[  ,  ,  ,  ,  ,  ]
[  ,  ,  ,  , 2,  ]
```

All the elements are directed leftwards and adding the numbers (2 & 2 = 4)

DESIGN AND IMPLEMENTATION

- **Functions and Modules**

  According to 2048 game structure and its merging process, building of multiple functions inside a class named Grid for each section will lead into specification of the working process. In the python program we have created following function and imported some modules for performing specific task:

Functions	Specifications
`__init__( )`	Constructor, object is created, initializes the attributes of the class. An empty list is initialized here named as grid_row
`Random_two( )`	Appending the empty list inside the main list, Appending 2 at random position of the list as well as ' ' space inside the list. This will create a nested list of as matrix of 6 x 6
`Print_grid( )`	Will print the current grid (nested list) – grid_row whenever the functions is called
`merge( )`	Take input direction form user, using conditions based on directions and neighbouring elements will compare and merge the elements according to 2^n. When tiles are merged, generate 2 at random position and print resulting grid

Modules	Usage
Random `import random`	`index = random.randint(0,5)` :- generating random 2 at random positions
Math `import math`	`math.pow(2,n+1)` :- comparing with multiple of 2 in the list

- **Architecture**

  The functions inside the class are interrelated to each for performing their task. They are dependent on output of each and every function of the class. We access the class attributes and methods using self keyword which binds the attribute with the given arguments. The architecture followed to connect above mentioned functions is:

```
__init__(self)    self.grid_row = []
```

random_two(self)

Input: *self*.grid_row = []

Output: Appending elements, resulting 2 at random position

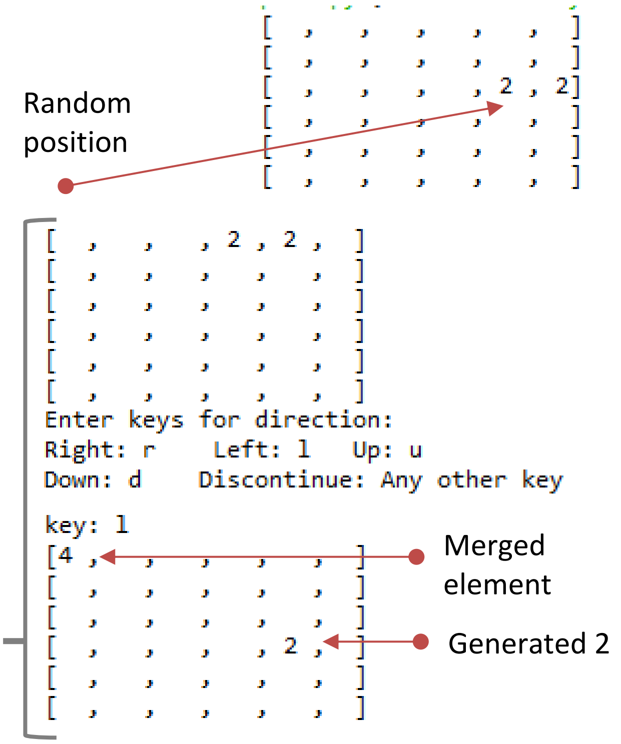
print_grid(self)

Input: *self*.grid_row

Output: Print the grid (nested list)

merge(self)

Input: *self*.grid_row Output: Merged elements and random numbers inside the list



```
[ ,  ,  ,  ,  ,  ]
[ ,  ,  ,  ,  ,  ]
[ ,  ,  ,  , 2 , 2]
[ ,  ,  ,  ,  ,  ]
[ ,  ,  ,  ,  ,  ]
[ ,  ,  ,  ,  ,  ]
```

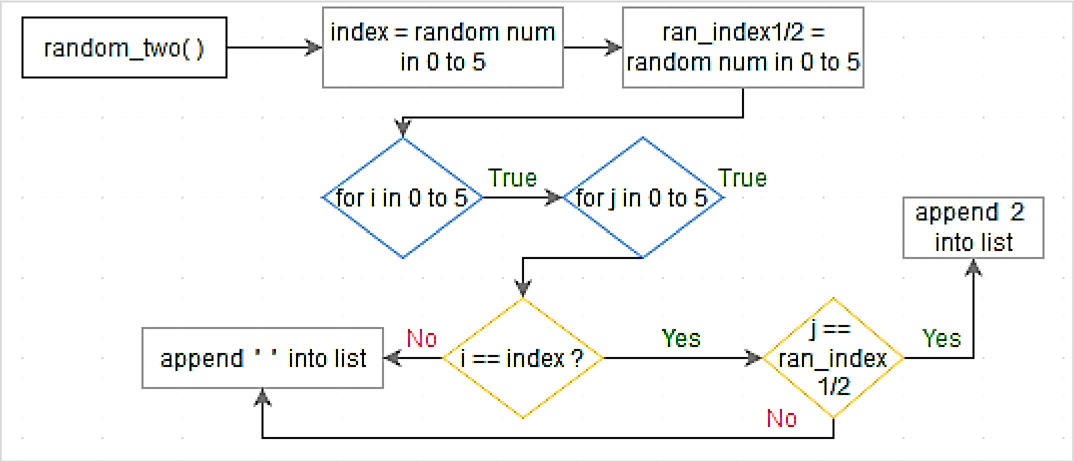
```
[ ,  ,  , 2 , 2 ,  ]
[ ,  ,  ,  ,  ,  ]
[ ,  ,  ,  ,  ,  ]
[ ,  ,  ,  ,  ,  ]
[ ,  ,  ,  ,  ,  ]
[ ,  ,  ,  ,  ,  ]
Enter keys for direction:
Right: r    Left: l   Up: u
Down: d    Discontinue: Any other key

key: l
[4 ,  ,  ,  ,  ,  ]
[ ,  ,  ,  ,  ,  ]
[ ,  ,  ,  ,  ,  ]
[ ,  ,  ,  , 2 ,  ]
[ ,  ,  ,  ,  ,  ]
[ ,  ,  ,  ,  ,  ]
```

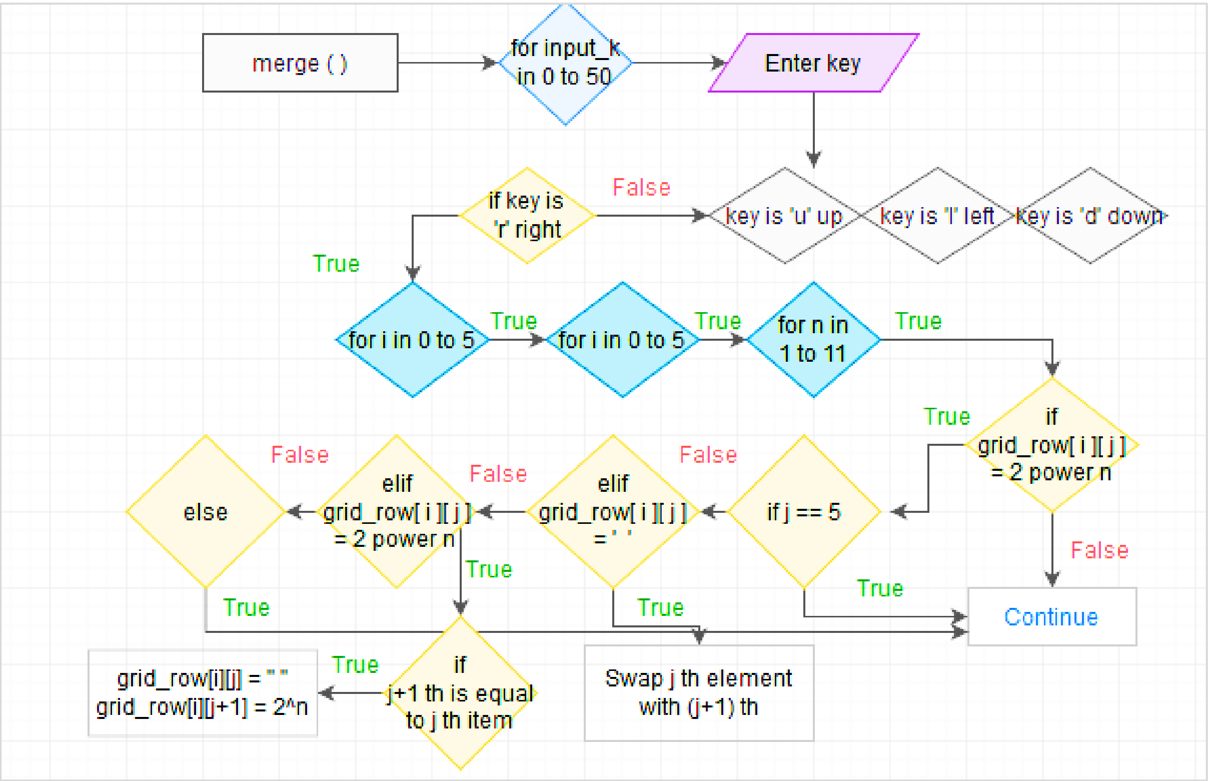
- **Algorithm**

  To merge the elements in the list we followed the following algorithm to reach the solution in an organised way with the help of decomposition and pattern recognition.

Pseudo code

```
• __init__( )
    1. List grid_row initialized as empty
• Random_two( )
    1. Index is random no. (0 to 5)
    2. Row_Index is random no. (0 to 5)
    3. For i in 0 to 5 and For j in 0 to 5
           If I is equal to index
               If I is equal to Row_index
                      Do: add 2 in the list
               Else not equal Do: add " " in list
           Else not equal to index Do: add " "
• Print_grid( )
    1. Print current grid (list)
• Merge( )
    1. Key = Get user input direction
    2. If key is "r" (right)
         For i in 0 to 5 and For j in 0 to 5
             For n in 0 to 11
               If grid_row [i][j] is equal to $2^n$
                   If j is equal to 5
                     Do: continue
                   If grid_row[i][j] is ' '
                    Do: Swap with (j+1)th item
                   If grid_row[i][j] is $2^n$
                       If [i][j] is equal to [i][j+1]
                           Do: [j + 1] is $2^n$
                                [ j ] is ' '
                   Else continue
               Else continue
    3. if key is "l" (left)
         For I in 0 to 5 and for j in 5 to 0
             Following the same pattern as of
             key right with swapping and
             merging with j – 1 th element
    4. if key is "u" (up)
         For I in 5 to 0 and for j in 0 to 5
             Following the same pattern as of
             key right with swapping and
             merging with i – 1 th row element
    5. if key is "l" (up)
         For I in 0 to 6 and for j in 0 to 6
             Following the same pattern as of
             key right with swapping and
             merging with i + 1 th row element
```





Data types included in the program are:

Data types	Usage
Sequence, list	`self.grid_row = [ ]`
String	`self.grid_row[i].append(" ")`
Integer - int	`self.grid_row[i].append(2)`

The program results into providing merged elements in the list in a user input direction. While working on the coding the first thing was to merge elements into a single right like in right. As all other cases follow the procedures to merge as well, creating a single directional merging at initial point provided way to merge. Comparing the elements of the list with the neighbouring elements by using math power function of math module rather than creating a list etc. gave a broad way to take the 2^n where n can be up to the desired end.

CONCLUSION

The 2048 puzzle game has a main objective of combining tiles of the same number to eventually form the number 2048. The user can move in the four cardinal directions and after every move a new tile of 2 is generated randomly. The identified problem here is to merge the elements which can be done following computing principles i.e. Decomposition, Pattern recognition, Algorithm and Abstraction. As the basic aim of the game is to add numbers according to the 2^n, addition resulting into 2^{n+1}. The document highlights a python program including multiple functions for specific purpose, algorithm followed and output of the program. The program is made to merge the elements by adding them into user input direction and generating 2 at random positions.

PYTHON PROGRAM

```python
import math
import random

class Grid:
    """Class Grid make the grid in form of matrix (nested list) and merge the
    elements in the list"""

    def __init__(self):
        """Creating and Initializes the nested list variable grid_row"""
        self.grid_row = []

    def random_two(self):
        """Generate 2 at random position in the self.grid_row (matrix)"""
        #random position for inserting 2
        index = random.randint(0,5)
        row_index1 = random.randint(0,5)
        row_index2 = random.randint(0,5)

        for i in range(6):
            # Append an empty sublist inside the list
            self.grid_row.append([])
            # Append 2 at random position inside the sublist through comparison
            for j in range(6):
                if i == index:
                    if ((j == row_index1) or (j == row_index2)):
                        self.grid_row[i].append(2)
                    else:
                        self.grid_row[i].append(" ")
                else:
                    self.grid_row[i].append(" ")
    def print_grid(self):
        """Print the grid - matrix """
        for i in range(0,6):
            print('[%s]' % ' , '.join(map(str,self.grid_row[i])))


    def merge(self):
        """Merge the elements of the list (tiles) depending on factors like
        direction, number and next element"""

        print("""Enter keys for direction:
Right: r    Left: l   Up: u    Down: d    Discontinue: Any other key""")
        # Taking the input direction from user
        for z in range(0,100):
                key = input("key: ")

                if key == "u":
                    for i in range(5,-1,-1):
                        for j in range(0,6):
                            for n in range(1,11):
                                if self.grid_row[i][j]==math.pow(2, n):
                                    if i==0:
                                        continue
                                    #Swap the elements if empty (" ")
                                    elif self.grid_row[i-1][j] == " ":
                                        temp = self.grid_row[i][j]
                                        self.grid_row[i][j] = self.grid_row[i-1][j]
                                        self.grid_row[i-1][j] = temp
                                    #inserting 2 at i-1 and " " (empty) at i th
                                    elif self.grid_row[i-1][j] == math.pow(2, n):
                                        if self.grid_row[i-1][j] == self.grid_row[i][j]:
                                            self.grid_row[i][j] = " "
                                            self.grid_row[i-1][j] = int(math.pow(2,n+1))

                                        else:
                                            self.grid_row[i-1][j] = self.grid_row[i-1][j]
                                            self.grid_row[i][j] = self.grid_row[i][j]
                                    else:
                                        continue
                                else:
                                    continue

                elif key == "r":
                    for i in range(0,6):
                        for j in range(0,6):
                            for n in range(1,10):
                                if self.grid_row[i][j]==math.pow(2, n):
                                    if j==5:
                                        continue
                                    elif self.grid_row[i][j+1] == " ":
                                        temp = self.grid_row[i][j]
                                        self.grid_row[i][j] = self.grid_row[i][j+1]
                                        self.grid_row[i][j+1] = temp
                                    elif self.grid_row[i][j+1] == math.pow(2, n):
                                        if self.grid_row[i][j] == self.grid_row[i][j+1]:
                                            self.grid_row[i][j+1] = int(math.pow(2,n+1))
                                            self.grid_row[i][j] = " "
                                        else:
                                            self.grid_row[i][j] = self.grid_row[i][j]
                                            self.grid_row[i][j+1] = self.grid_row[i][j+1]
                                    else:
                                        continue
                                else:
                                    continue

                elif key == "l":
                    for i in range(0,6):
                        for j in range(5,-1,-1):
                            for n in range(1,11):
                                if self.grid_row[i][j]==math.pow(2, n):
                                    if j==0:
                                        continue
                                    elif self.grid_row[i][j-1] == " ":
                                        temp = self.grid_row[i][j]
                                        self.grid_row[i][j] = self.grid_row[i][j-1]
                                        self.grid_row[i][j-1] = temp
                                    elif self.grid_row[i][j-1] == math.pow(2, n):
                                        if self.grid_row[i][j] == self.grid_row[i][j-1]:
                                            self.grid_row[i][j-1] = int(math.pow(2,n+1))
                                            self.grid_row[i][j] = " "
                                        else:
                                            self.grid_row[i][j] = self.grid_row[i][j]
                                            self.grid_row[i][j-1] = self.grid_row[i][j-1]
                                    else:
                                        continue
                                else:
                                    continue


                elif key == "d":
                    for i in range(0,6):
                        for j in range(0,6):
                            for n in range(1,10):
                                if self.grid_row[i][j]==math.pow(2, n):
                                    if i==5:
                                        continue
                                    elif self.grid_row[i+1][j] == " ":
                                        temp = self.grid_row[i][j]
                                        self.grid_row[i][j] = self.grid_row[i+1][j]
                                        self.grid_row[i+1][j] = temp
                                    elif self.grid_row[i+1][j] == math.pow(2, n):
                                        if self.grid_row[i+1][j] == self.grid_row[i][j]:
```

```
                                            self.grid_row[i+1][j] = int(math.pow(2,n+1))
                                            self.grid_row[i][j] = " "
                                        else:
                                            self.grid_row[i+1][j] = self.grid_row[i+1][j]
                                            self.grid_row[i][j] = self.grid_row[i][j]
                                    else:
                                        continue
                                else:
                                    continue

                else:
                    print("Game is discontinued")
                    break

                # Generate 2 at random position after merging of the elements
                g_index = random.randint(0,5)
                r_index = random.randint(0,5)

                for i in range(0,6):
                    for j in range(0,6):
                        if self.grid_row[g_index][r_index]==" ":
                            self.grid_row[g_index][r_index]=2

               #print the matrix(list) with merged elements and a new element
                for i in range(0,6):
                            print('[%s]' % ' , '.join(map(str,self.grid_row[i])))

# calling the class methods using instance variables
G1 = Grid()
G1.random_two()
G1.print_grid()
G1.merge()
```