Final Project
The Game 2048 in Python 3.5

by
Bat Sukhbaatar

Programming in Computing 16
Dr. Puck Rombach
University of California, Los Angeles
March, 2017

PIC 16 Python with Applications

Outline:

1. Introduction
    1.1. Project Purpose
    1.2. Project Design
    1.3. Pseudocode

2. Implementation
    2.1. Defining the auxiliary function ‘handle_0s’
    2.2. Defining the conditions of game exit
    2.3. Defining the generation of random ‘2’ or ‘4’
    2.4. Defining the main function ‘game2048’
    2.5. Testing the game

3. Conclusion

4. References

1. Introduction

1.1 Project Purpose:

The ultimate purpose of the project is to simulate the game '2048' in Python. The game runs on the grid 4x4 and starts with two '2's randomly placed on the grid. At each turn, the game takes an input from a user and shifts the values on the grid to the right, left, down, or up. If there are any two same values on a row or column, separated by an empty cell (or cells), the values are added to the ceil and an empty ceil is generated (locations depend on a user input). At the end of each turn, random '2' or '4' is generated on one of the empty cells. The game ends when one of the ceils reach 2048 or there are no more empty cells to generate random '2' or '4.' Please visit http://2048game.com to get familiar with the game.

1.2 Project Design:

For the simplicity, the game will ask an input from a user and print out the game grid on the Python console (I will not use GUI). A user will provide one of the inputs, 'left,' 'right,' 'up,' or 'down' at each turn until the game ends. The 4x4 game grid will be represented by a 4x4 Numpy array. Zero's in a Numpy array will represent empty cells. The **key observation** is that any row or column in a 4x4 Numpy array is 1D array. Thus, we need a function that takes a 1D array to shift values to the left or right by ignoring 0's (empty cells). This function will be introduced soon, and its name is 'handle_0.' The **main operation** is that if the same 2 values are next to each other in a 1D array after shifting all elements to the left or right, the values have to added up and one '0' has to be generated in the direction of a user's input.

For example, a user supplied 'up' as an input. In this case, the algorithm has to access a 4x4 Numpy array column by column to shift values in each 1D array to the left by ignoring 0's. Then look for the same two nonzero elements in a 1D array from left to right. If a match is found, two values have to be added up and placed on the left of two elements, and '0' is placed on the right side of two elements. Finally, we have to shift all the elements in the 1D array to the left by ignoring '0's.

The exit conditions are either one of elements in a 4x4 Numpy array is 2048, or there is no more '0' (empty cell) in a 4x4 Numpy array to generate a random '2,' or '4' at the end of each turn. In other words, a 4x4 Numpy array will not be changed after trying to add a random '2,' or '4.'

1.3 Pseudocode:

```
Define a function handle_0s
Generate 4x4 np.array                          ### preparing layout
Generate two random 2's on the game layout

While True:
    Print(4x4 array)      ### display the game layout and ask the user's input
    Ask user's input
    Depending on users' input:
        For i in np.arange(4):
            Access the 4x4 array by row or column        ### depends on user's input
            Shift elements to the left or right through 'handle_0s'
            If two same values are next to each other:
                Add up the values to the left or right cell        ### depends on user's input
                Replace the other cell with 0
            Shift elements to the left or right by 'handle_0s'  ### insures no white spaces
                                                                ### between nonzero elements
    generate random '2' or '4' on one of the empty cells in the layout
    if player reaches 2048 or there is no empty cell to generate random '2' or '4':  ### Exit Conditions
        break
```

2. Implementation

2.1 Defining the auxiliary function 'handle_0s':

The 2048 involves shifting the non-zero elements to the left, right, down, or up, which boils down to shifting the non-zero elements of a 1D array to the right or left. For example, if a user input is **'up,'** we will access a 4x4 np.array (game layout) by column and shift all non-zero elements to the **left**.

The function will accept a 1D array and direction (‘left’ or ‘right’) and spits a 1D array where nonzero elements are shifted to the left or right, ignoring 0’s (0’s represent empty cells in the game layout).

```
import numpy as np
def handle_0s(x, direction = 'left'):
        if direction == 'left':
                (dum, ) = np.where(x == 0)
                x = np.delete(x, dum)
                x = np.append(x, np.array([0 for _ in range(4-len(x))]))
                return x
        elif direction == 'right':
                (dum, ) = np.where(x == 0)
                x = np.delete(x, dum)
                x = np.append(np.array([0 for _ in range(4-len(x))]), x)
                return x
        else:
                print('check input/n')

handle_0s(np.array([1,0,2,0]))
Out[2]: array([1, 2, 0, 0])
handle_0s(np.array([1,0,2,0]),  'right')
Out[3]: array([0, 0, 1, 2])
```

2.2 Defining the conditions of game exit:

If one of the cells of a 4x4 np.array reaches 2048, the game ends. We can use np.max(4x4 array) to get the maximum value of the cells and check if it equals to 2048. Also, the game ends when there is no 0’s or empty cells in a 4x4 array to generate random ‘2’ or ‘4,’ which means a 4x4 array at the start of the loop and the 4x4 array at the end of the loop are exactly the same; because a new random element isn’t added and any of nonzero elements aren’t cancelled out. Thus, we have to keep track of a 4x4 array at the start and end of the loop to compare them.

2.3 Defining the generation of random '2' or '4':

I observed that '2's are produced randomly on one of the empty ceils most of the time. Thus, I will use `np.random.choice([2, 4], size = 1, p = [0.9, 0.1])`. Also, it is crucial to ensure that a random '2' or '4' pops up in an empty cell (a cell containing 0). Thus, my code will make sure that we pick a cell containing '0' and use np.random.choice at the end of each turn.

2.4 Defining the main function 'game2048':

```
3 def game2048():
4
5     x = np.zeros((4,4), dtype = int)    #### Preparing data
6     dum = np.random.choice(range(16), 2, replace = False)
7
8     for i in dum:
9         row = int(i / 4)
10         col = i % 4
11         x[row, col] = 2
12         x[row, col] = 2
13
14     while True:     ####  Game begins here
15
16         print(x)
17         print('\n')
18
19         direction = input('left or right or up or down? (Dont use quote):
   ')
20
21         if direction == 'left':
22
23             dum1 = x.copy()
24
25             for i in np.arange(4):
26
27                 dum = handle_0s(x[i, :].copy(), 'left')
28
29                 for j in np.arange(3):
30
31                     if dum[j] == dum[j + 1] and dum[j] > 0:
32                         dum[j] = dum[j].copy() + dum[j + 1].copy()
33                         dum[j + 1] = 0
34                         dum = handle_0s(dum.copy(), 'left')
35
```

```
36
37
38 x[i, :] = dum.copy()
39
40             while True:
41                 new_pop = np.random.choice(range(16))
42
43                 if x[int(new_pop / 4), new_pop % 4] == 0:
44
45                     x[int(new_pop / 4), new_pop % 4] = np.random.choice([2
   , 4], size = 1, p = [0.9, 0.1])
46                     break
47
48
49         elif direction == 'right':
50
51             dum1 = x.copy()
52
53             for i in np.arange(4):
54
55                 dum = handle_0s(x[i, :].copy(), 'right')
56
57                 for j in np.arange(3,0,-1):
58
59                     if dum[j] == dum[j-1] and dum[j] > 0:
60                         dum[j] = dum[j].copy() + dum[j-1].copy()
61                         dum[j-1] = 0
62                         dum = handle_0s(dum.copy(), 'right')
63
64                 x[i, :] = dum.copy()
65
66             while True:
67                 new_pop = np.random.choice(range(16))
68
69                 if x[int(new_pop / 4), new_pop % 4] == 0:
70
71                     x[int(new_pop / 4), new_pop % 4] = np.random.choice([2
   , 4], size = 1, p = [0.9, 0.1])
72                     break
73
74         elif direction == 'up':
75
76             dum1 = x.copy()
77
78             for i in np.arange(4):
79
80                 dum = handle_0s(x[:, i].copy(), 'left')
81
82                 for j in np.arange(3):
```

```
83
84
85                  if dum[j] == dum[j + 1] and dum[j] > 0:
86                        dum[j] = dum[j].copy() + dum[j + 1].copy()
87                        dum[j + 1] = 0
88                        dum = handle_0s(dum.copy(), 'left')
89
90                x[:, i] = dum.copy()
91
92             while True:
93                new_pop = np.random.choice(range(16))
94
95                if x[int(new_pop / 4), new_pop % 4] == 0:
96
97                    x[int(new_pop / 4), new_pop % 4] = np.random.choice([2
   , 4], size = 1, p = [0.9, 0.1])
98                    break
99
100           elif direction == 'down':
101
102               dum1 = x.copy()
103
104               for i in np.arange(4):
105
106                   dum = handle_0s(x[:, i].copy(), 'right')
107
108                   for j in np.arange(3, 0, -1):
109
110                       if dum[j] == dum[j-1] and dum[j] > 0:
111                           dum[j] = dum[j].copy() + dum[j-1].copy()
112                           dum[j-1] = 0
113                           dum = handle_0s(dum.copy(), 'right')
114
115                   x[:, i] = dum.copy()
116
117               while True:
118                   new_pop = np.random.choice(range(16))
119
120                   if x[int(new_pop / 4), new_pop % 4] == 0:
121
122                       x[int(new_pop / 4), new_pop % 4] = np.random.choice(
   [2, 4], size = 1, p = [0.9, 0.1])
123                       break
124
125           elif direction == 'break':
126
127               print('User exited the game')
128               break
129
```

```
130
131
132             else:
133                 dum1 = x.copy() + 1
134                 print('Insert correct direction\n')
135
136
137             if np.all(dum1 == x):
138                 print('You lost, please try again/n')
139                 break
140
141             elif np.max(x) == 2048:
142                 print('Congragulations, you won the game/n')
143                 break
144             else:
145                 pass
146
    return None
```

2.5 Testing the game:

```
1 game2048()
2 [[0 0 0 0]
3  [0 0 0 0]
4  [2 0 2 0]
5  [0 0 0 0]]
6
7
8 left or right or up or down? (Dont use quote): up
9     [[2 0 2 0]
10     [0 0 0 0]
11     [0 0 0 0]
12     [0 0 0 2]]
13
14
15    left or right or up or down? (Dont use quote): left
16    [[4 0 0 0]
17     [0 2 0 0]
18     [0 0 0 0]
19     [2 0 0 0]]
20
21
22    left or right or up or down? (Dont use quote): right
23    [[0 0 0 4]
24     [0 0 0 2]
25     [0 0 0 0]
26     [0 2 0 2]]
```

```
27
28
29    left or right or up or down? (Dont use quote): down
30    [[0 0 0 2]
31     [0 0 0 0]
32     [0 0 0 4]
33     [0 2 0 4]]
34
35
36    left or right or up or down? (Dont use quote): down
37    [[0 0 0 0]
38     [0 0 0 0]
39     [2 0 0 2]
40     [0 2 0 8]]
41
42
43    left or right or up or down? (Dont use quote): left
44    [[0 0 0 0]
45     [0 0 0 0]
46     [4 0 0 0]
47     [2 8 4 0]]
48
```

The program continues until one of the cells reaches 2048 or there is no more empty cell (‘0’) to generate a new random number.

3. Conclusion

Defining the function ‘handle_0s’ made my algorithm way easier. Although the main function looks long, those are almost the same codes inside 4 else-if statements (4 directions, 4 possible user inputs). Thus, understanding the codes in only one chunk (one direction) is sufficient to understand the whole codes in the main function. The major downside in this code is the code to produce random ‘2’ or ‘4’ on one of the empty cells because it is computationally expensive. It randomly selects one cell from the 16 cells of the game layout, and if it is an empty cell, it will generate a random number. If a selected cell isn’t empty (‘0’), it will keep randomly selecting from the 16 cells until it finds one. If I create a set to keep track of empty and nonempty cells, it will resolve the issue. However, the entire code will be a bit complicated. Overall, the code is simple, straightforward, and gets the job done.

4. References

Lecture Notes Dr. Rombach, PIC 16, University of California, Los Angeles, March 2017

http://2048game.com