AMS 561 Computational Science

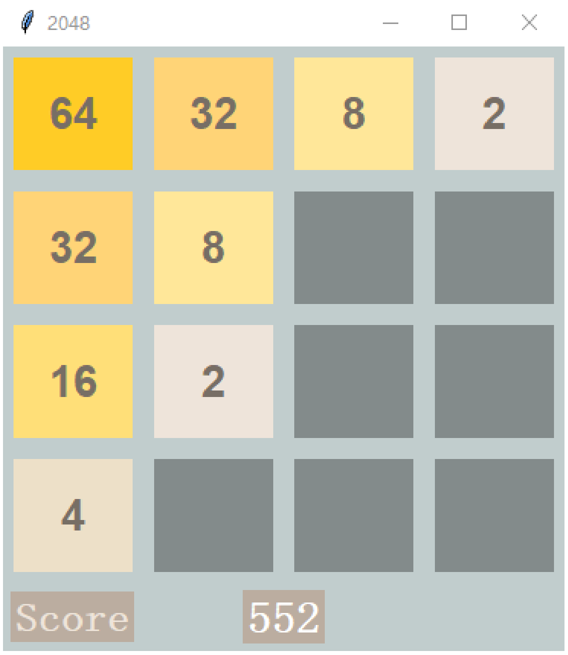
Group Project Report — 2048

April 30, 2022

Team Member: Yizhen Jia, Qiang Wang, Sikun Wang

The project objectives:

The creation of "2048" has brought a way to release stress and relax. The content of this mini game is presented by a 4*4 grid, and the game opens with two random 2 or 4 blocks. The rules of the game are simple: the player needs to press <Key> on keyboard to control all the blocks to move in the same direction in one operation, and two blocks with the same number will merge to become their sum after they collide, and a square with 2 or 4 will be generated



randomly after each operation. The player's victory goal is to finally get a "2048" square.

We decided to make this game on this project because we wanted to use the knowledge, we learned to enjoy life better. Our main goal is to implement the entire game in python, for which we would use some python modules, such as tkinter, pygame. We optimize the details of the game after the completion of the main procedure to let the players have fun. We color-coded specific digital modules so that the player could realize the progress of the game briefly in a rapid operation. Then by incorporating sound elements, players can get both visual and auditory feedback for each step of action, and the so-called audio-visual experience can be completed. We encountered bugs when we started to add features to the main program, because the main program was rather scattered, and the various iterations made us dizzy. Therefore, we used a new idea, that is we created two classes, Game and Board, where class Game was game operative logic and methods, and class Board was game board design and basic algorithms. This practice gives us a much cleaner code, and we will cover the information of this technique in detail on next modules.

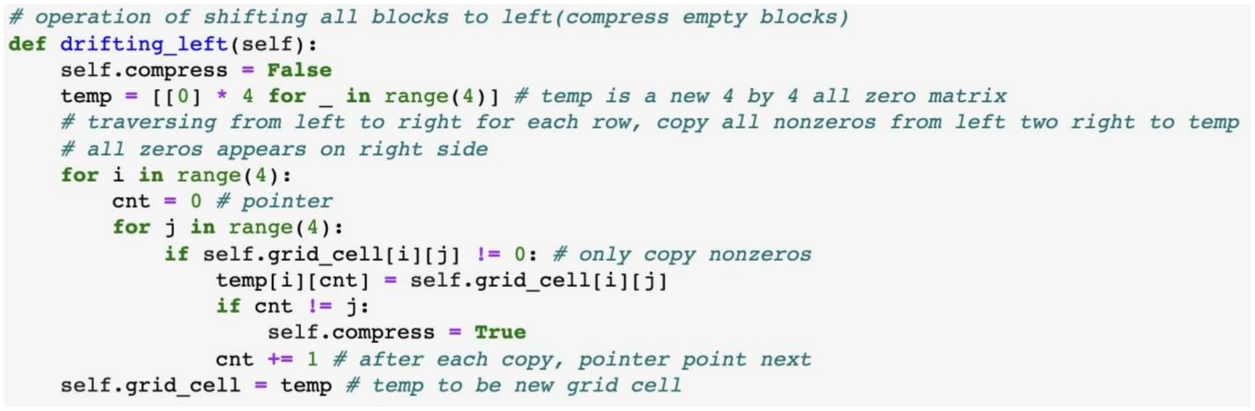
Techniques and tools used:

In this project, we use Python, including modules such as pygame (built for game development, includes graphics, sound) and tkinter (a Python binding to the TK GUI toolkit). In the process of working, we tried some of the techniques we learned from our assignments as well as the lecture, for example, constructing classes to realize certain methodologies.

- The whole project consists of two classes, “Game” and “Board”. The game class includes algorithms, which are responsible for how to implement functions, merge and move digital blocks. The Board class is the design of the game board, and It also acts as the connection from the first class. The second module provides functions for the first module to call to implement the operation.

We implemented the run of the player operation with the method of “drifting_left”. Choosing the other three directions would not make a significant difference to drifting left, we just choose a random direction since we only need one direction to satisfy all possible movement directions.

- Here, the function belongs to class Board, and it assists the merging of grids. By creating a temperate matrix, let each element in double for loops. For all nonzero elements on the same row, put them on the left and update the matrix. Before finishing this part, the symbol of compress is reset to True.



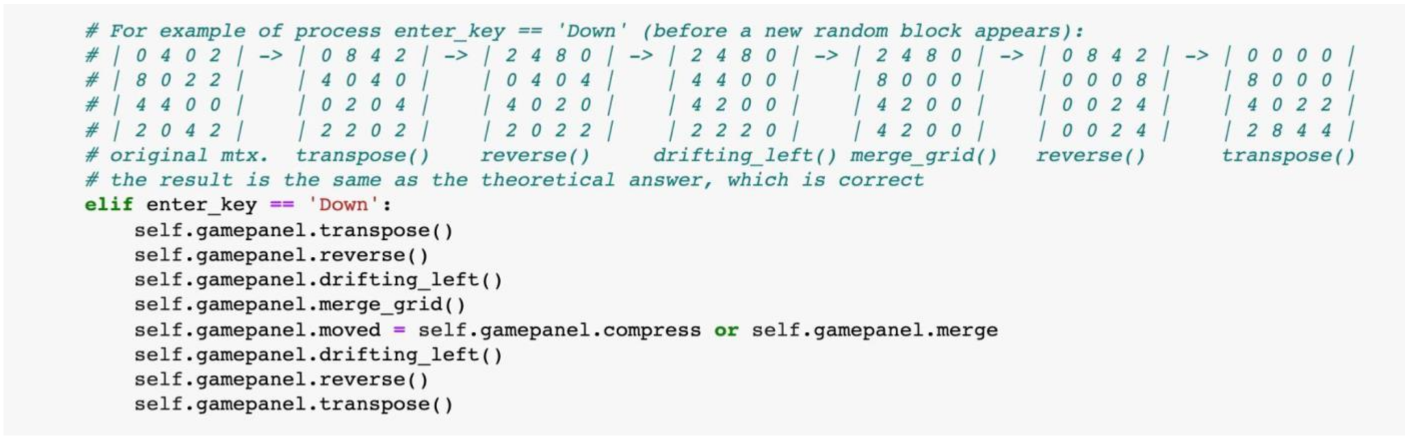
```python
# operation of shifting all blocks to left(compress empty blocks)
def drifting_left(self):
    self.compress = False
    temp = [[0] * 4 for _ in range(4)] # temp is a new 4 by 4 all zero matrix
    # traversing from left to right for each row, copy all nonzeros from left two right to temp
    # all zeros appears on right side
    for i in range(4):
        cnt = 0 # pointer
        for j in range(4):
            if self.grid_cell[i][j] != 0: # only copy nonzeros
                temp[i][cnt] = self.grid_cell[i][j]
                if cnt != j:
                    self.compress = True
                cnt += 1 # after each copy, pointer point next
    self.grid_cell = temp # temp to be new grid cell
```

(Figure of def of one direction)

Then there is the most important game algorithm to realize the movement operation: After defining “drifting_left”, we did not simply define the other three directions of operation in the same way. Instead, we transferred all other three operations (UP, DOWN, and Right) with a sequence of reverse(), transpose(), and drifting_left() functions:

- We set the change in one direction, let’s say drifting_left, which we just did, that makes each grid slide to left and then change the direction of the matrix so that we can let the squares move as required. Here we use the more specific down-operation as an example to explain step by step.
- (1) Get the transpose version of the original matrix.
- (2) Get the reverse matrix of the last matrix that changes the left and right direction.

- (3) Start the combination and movement of the number blocks. The steps have two functions to complete that are gamepanel.merge_grid and the update for some symbols.
- (4) Finalized the other girds to leave after the first change. The operation of right and up are like DOWN. During this transformation, the symbol of compress and gamepanel.merge are labeled to determine whether a new random cell is needed.



```
# For example of process enter_key == 'Down' (before a new random block appears):
# | 0 4 0 2 | -> | 0 8 4 2 | -> | 2 4 8 0 | -> | 2 4 8 0 | -> | 2 4 8 0 | -> | 0 8 4 2 | -> | 0 0 0 0 |
# | 8 0 2 2 |    | 4 0 4 0 |    | 0 4 0 4 |    | 4 4 0 0 |    | 8 0 0 0 |    | 0 0 0 8 |    | 8 0 0 0 |
# | 4 4 0 0 |    | 0 2 0 4 |    | 4 0 2 0 |    | 4 2 0 0 |    | 4 2 0 0 |    | 0 0 2 4 |    | 4 0 2 2 |
# | 2 0 4 2 |    | 2 2 0 2 |    | 2 0 2 2 |    | 2 2 2 0 |    | 4 2 0 0 |    | 0 0 2 4 |    | 2 8 4 4 |
# original mtx.  transpose()    reverse()     drifting_left() merge_grid()  reverse()      transpose()
# the result is the same as the theoretical answer, which is correct
elif enter_key == 'Down':
    self.gamepanel.transpose()
    self.gamepanel.reverse()
    self.gamepanel.drifting_left()
    self.gamepanel.merge_grid()
    self.gamepanel.moved = self.gamepanel.compress or self.gamepanel.merge
    self.gamepanel.drifting_left()
    self.gamepanel.reverse()
    self.gamepanel.transpose()
```

(Figure of trans process of DOWN-operation

Challenges and Learning:

In summary, the first challenge we encountered was when we had the main ideas and thoughts, and the next step we needed to build the main code from scratch. It took us a long time to complete the first version of the code body, but the process went smoothly, and we were able to confirm that it worked without too many changes. One of the very cool operations is our implementation of some of the game rules in code. For example, the movement between digital modules, we used merge and compress in certain sequences. Such an approach saves the computing power of the computer to a certain extent.

Later, we encountered a new challenge. It was also the most mentally draining. As mentioned at the beginning, we always encountered bugs when we tried to add new features such as score board and operation sound to the main code, and as we fixed them at the beginning, they got worse and worse. The reason for this was that the code was too messy for us, and the loops were nested in each other, making us often overwhelmed with adjustments. To solve this problem, we modularized the existing body code so that each module had its own role and the calls between functions became clearer, then we were able to successfully add new features.