Name: Luong Do Ngoc Minh
Programming Practice class
Professor Eom, Hyeonsang

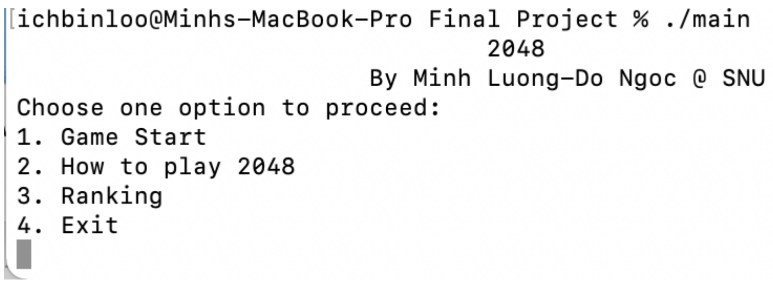
FINAL PROJECT REPORT
Developing the puzzle game 2048

2048 is a well-reputed game that has an easy game rule – moving the puzzles such that one title reaches 2048. For the final project, I remake this popular game in the language of C.

1. Building and running the game

To build the game, I list out all the necessary features that are listed below. Each feature consists of one to several supported functions, and some of them are dependent to other function.

To run the game, the user can download the source code from my GitHub and open the terminal of the saved destination to start playing the game.



```
ichbinloo@Minhs-MacBook-Pro Final Project % ./main
                                         2048
                           By Minh Luong-Do Ngoc @ SNU
Choose one option to proceed:
1. Game Start
2. How to play 2048
3. Ranking
4. Exit
```

2. Playing the game

The user can move the titles from the keyboard (‘a’ for moving to left, ‘d’ for moving to right, ‘s’ for moving down and ‘w’ for moving up). If there are no titles that can move or merge with other titles, or time is up, the game is over. After that, the user enters their name – and their data will be saved for the ranking features.

3. Features & Implementation

a. Main menu

The menu function will be called at the beginning of the game and after either option (1 - playing game, 2 - displaying the game instruction and 3 - ranking) has finished implementing. To do so, I make a while loop that will break if the given option is not from 1 to 4. If the option is 1, the menu() function will call the game() function; if the option is 2, it will display the instruction; if the option is 3, it will call the ranking() function; if the option is 4, it will break the loop and terminates the program. For other invalid inputs, the user will be asked to enter again. They will be asked until their input is correct.

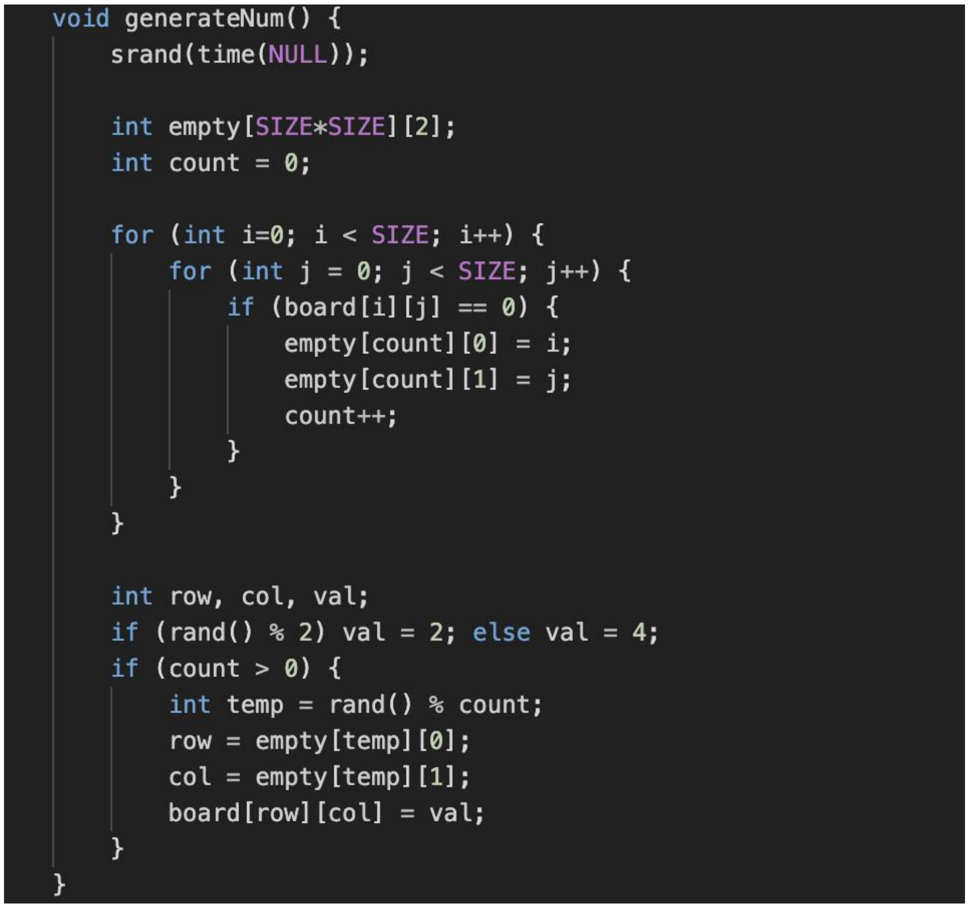
b. Moving and merging numbers

For moving and merging, we assume that the program will move right on default. The program will check each component for each row of the board. If it has not yet reached the end of the board and the next component is 0, the component that we are considering will be moved to the location that the next one is a non-zero component. For merging, the program will continue checking each component for each row of the board. If the next one has the same value, we will merge two numbers and set the value of that considered component to 0. If moving right is successful, the function returns 1, otherwise 0. For other directional moving features, if moving is possible (meaning the function returns 1), we will generate a new number in a random location on the board.

To support the directional moving features, I make another function called "rotate()". It will turn the board by 90 degrees in anti-clockwise direction without using extra space. Following that notion, to move left, we call rotate() function twice, move right (if possible), then rotate() twice. To move up, we call rotate() function three times, check whether moving right is possible, then call rotate() function once more. Finally, to move down, we rotate the board, check whether moving right is possible, then rotate the board three times.

c. Generate a new number

We can generate a new number with the help of rand() function from the C standard library function. We need a random seeding srand(time(NULL)) to make the random function work properly and run at the running time. We also make sure that at the generated location, there does not exist any non-zero components.



```c
void generateNum() {
    srand(time(NULL));

    int empty[SIZE*SIZE][2];
    int count = 0;

    for (int i=0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            if (board[i][j] == 0) {
                empty[count][0] = i;
                empty[count][1] = j;
                count++;
            }
        }
    }

    int row, col, val;
    if (rand() % 2) val = 2; else val = 4;
    if (count > 0) {
        int temp = rand() % count;
        row = empty[temp][0];
        col = empty[temp][1];
        board[row][col] = val;
    }
}
```

d. **Check the winning and defeating status**

The winning status is asserted when the value of one title in the board reaches 2048. The defeating status is asserted when either the time is running out, or there does not exist any values that can merge or can move. After the winning and defeating status is declared, the data of the user (their name, their scores, their number of moves and combos) will be saved in a text file called “scores.txt”.

e. **Calculate and update the scores**

The scores will be updated after the user moves the title. Once one combo (4, 8, 16, etc.) is made, it increases the scores by the value of the made combo.

f. **Clear the screen and re-draw the board for every key press**

For every key press, the screen will be cleared, and the board will be drawn again. It can do so by printing `"\033[?25l\033[2J"`.

g. **Ranking system**

The ranking of one user is based on: whether they succeed making a 2048 or not, their scores, their moves (less is better) and the number of the combos they made. It can do so by reading the text file “scores.txt”, saving the data in a temporary array, sorting with the given condition and return the result. When there is no data to rank, the program returns a message and goes back to the menu.

h. **Time attack**

The user must make the 2048 title in five minutes. Otherwise, the user loses the game. The left time is calculated by saving the end time, start time and the current time. The time is updated after one key press. If the time is up, the game ends and the board components reset to 0.

i. **Combo count**

After one combo is made, the combo count increments by one.

4. Troubleshooting points

a. **Receiving inputs continuously without pressing Enter**

To receiving the input continuously like the native game, I make two functions called setBufferedInput() and signal_callback_handler(). The first function receives the Boolean condition – if it is false, then the program starts receiving the input continuously; if it is true, it stops. The second function helps the compiler register a signal handler for when a button is pressed.

b. **Display format**

To make the display universal regardless of any increasing number and different username length, I set the output width for each number and string. Thanks to this, the board and data always look neat.