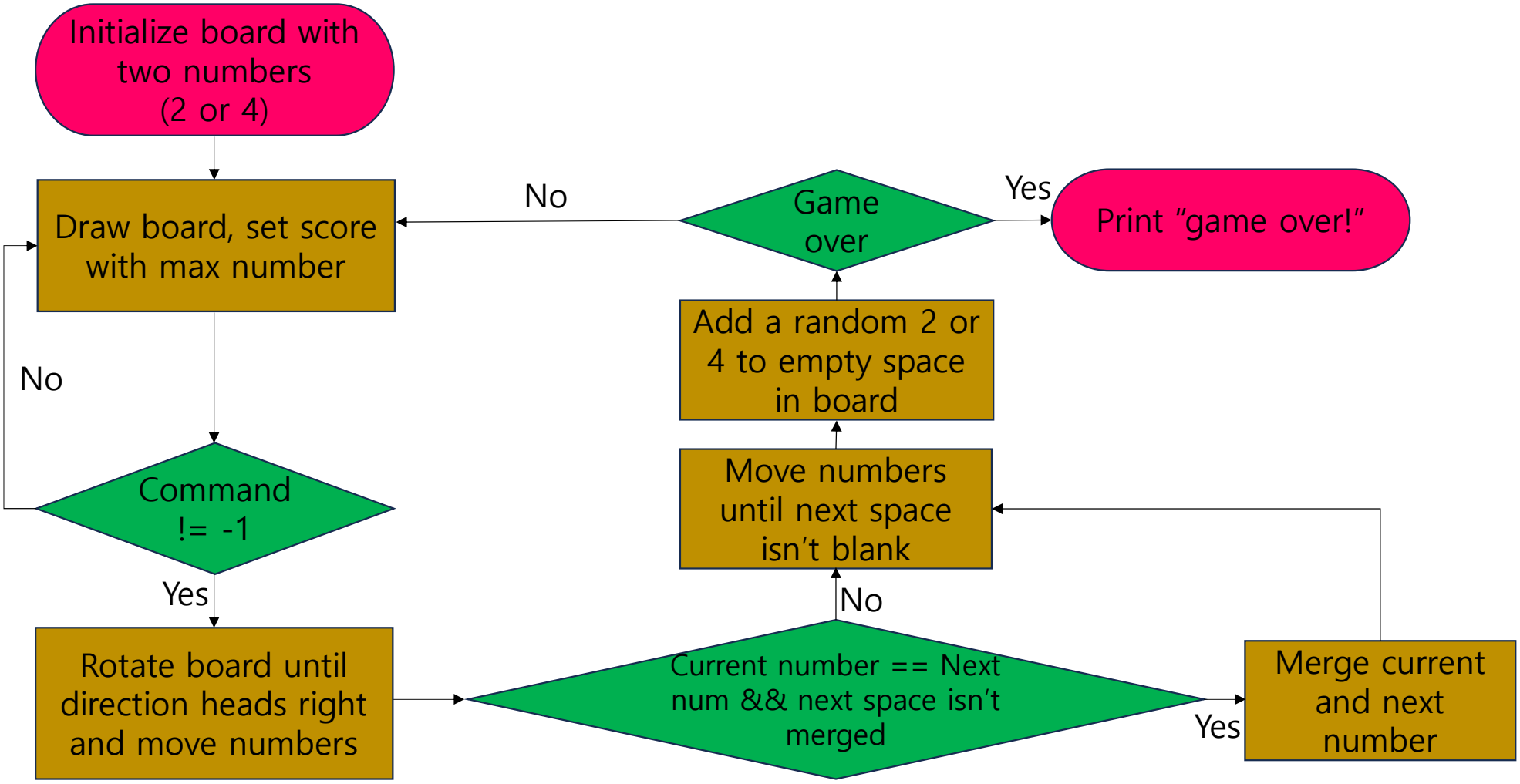
2048 Game Report

20221629

Flowchart of Program



Initialize board with two numbers
(2 or 4)

1. We start the program initializing the game board, in which we fill all the space on board with 0.

```
int main() {
    int command = -1;
    fp = fopen("output.txt", "w");
    init();
    srand(time(NULL));

    int tot = 0;

    /* make init board */
    tot += make_two_or_four();
    tot += make_two_or_four();
    draw_board(tot, -1);
```

2. To start the game with two random numbers (2 or 4), we call make_two_or_four function which creates a number in random space and add the created numbers to the tot variable.

```
/* game over flag & board state */
int game_over, b[4][4];
FILE* fp;

void init() {
    int i, j;
    game_over = 0;
    for (i = 0; i < 4; i++)
        for (j = 0; j < 4; j++)
            b[i][j] = 0;
}
```

Initialize board with two numbers
(2 or 4)

The make_two_or_four function creates 2 with probability of 2/3 and 4 with probability of 1/3. It first finds a random empty space in board and creates a number in it. If board is full, it will fail to create number and return 0.

The probability's mechanism is embodied by creating a random number and if it's divided by 3, function will create 4, and if not, function will create 2. Since the probability of a random number being a multiple of 3 is 1/3, this mechanism shouldn't be a problem.

```
int make_two_or_four() {
    int num, empty = 0, emptypos[16][2];
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            if (b[i][j] == 0)
            {
                emptypos[empty][0] = i;
                emptypos[empty][1] = j;
                empty++;
            }
        }
    }
    if (empty != 0)
    {
        num = rand() % empty;
        if (num % 3 != 0)
            b[emptypos[num][0]][emptypos[num][1]] = 2;
        else
            b[emptypos[num][0]][emptypos[num][1]] = 4;
        return b[emptypos[num][0]][emptypos[num][1]];
    }
    /* return 2 or 4 that makes in this times */
    /* if can not make two or four, then return 0 */
    return 0;
}
```

Draw board, set score with max number

The draw_board function looks complicated but it's actually simple.

First, we initialize score with the maximum value and checks whether tot really is the sum of all spaces

Then we print out score and visualize the board to the users by printing out different colors to a number.

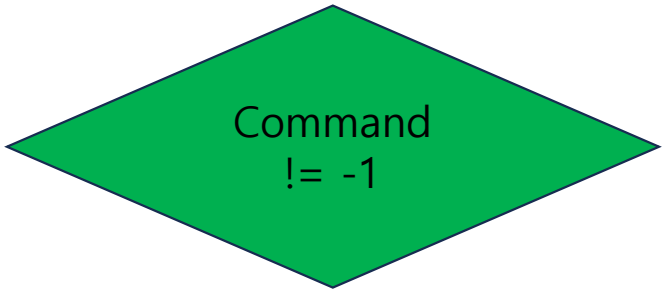
```
void draw_board(int tot, int command) {
    int i, j, k, c[8][8], score;
    /* console clear */
    system("clear");

    score = 0;
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            if (b[i][j] > score)
                score = b[i][j];
            tot -= b[i][j];
        }
    }
    if (tot != 0)
        exit(-1);
    /* calculate score & check sum of all block equals variable tot */

    printf("    Score : %d\n", score);
    fprintf(fp, "%d %d\n", score, command);

    for (i = 0; i < 4; i++) {
        for (j = 0; j < 4; j++) {
            c[i][j] = 32;
            for (k = 0; k < 50; k++) {
                if (b[i][j] == (1 << k)) {
                    c[i][j] = 32 + (k % 6);
                    break;
                }
            }
        }
    }
    for (i = 0; i < 4; i++) {
        for (j = 0; j < 4; j++)
            fprintf(fp, "%d ", b[i][j]);
        fprintf(fp, "\n");
    }

    for (i = 0; i < 4; i++) {
        puts("");
        puts("");
        for (j = 0; j < 4; j++)
            printf("\033[%dm%5d\033[0m", c[i][j], b[i][j]);
        puts("");
        puts("");
    }
}
```



We receive the command and internally set the value of command as an integer.

If the command is invalid, it will be initialized as -1 which will skip all the process and loop until the command is valid.

```c
int GetCommand() {
    int ch = getch();

    switch (ch)
    {
    case 'd':
    case 'D': return 0;
    case 'w':
    case 'W': return 1;
    case 'a':
    case 'A': return 2;
    case 's':
    case 'S': return 3;
    default: return -1;
    }

    return -1;
}
```

```c
do {
    command = GetCommand();

    if (command != -1) {
        if (set_board(command, b)) {
            tot += make_two_or_four();
            if (is_game_over() == 0)
                game_over = 1;
            draw_board(tot, command);
        }
    }

} while (!game_over);
```

Rotate board until direction heads right and move numbers

Since there are 4 directions, we only embody the code for the right direction and use rotate function to handle with other directions.

For example, if we receive the upward command, we rotate the board 90 degrees clockwise once, execute as if we received the rightward command, and rotate it 3 more times after everything is set. This should set the board's direction back to the original state, with an effect as if we executed and upward command.

```
int set_board(int dir, int b[4][4]) {
    int x, y, move = 0, merge[4][4] = { 0, };
    for (int i = 0; i < dir; i++)
        rotate(b);
    for (int i = 0; i < 4; i++){
        for (int j = 3; j >= 0; j--){
            x = i, y = j;
            if (b[x][y] != 0){
                while (y < 3 && b[x][y + 1] == 0){
                    b[x][y + 1] = b[x][y];
                    b[x][y] = 0;
                    y++;
                    move = 1;
                }
            }
        }
    }
    for (int i = 0; i < 4; i++){
        for (int j = 3; j >= 0; j--){
            x = i, y = j;
            if (b[x][y] != 0 && y < 3 && b[x][y] == b[x][y + 1] && !merge[x][y + 1]){
                b[x][y + 1] += b[x][y];
                b[x][y] = 0;
                merge[x][y + 1] = 1;
                move = 1;
            }
        }
    }
    for (int i = 0; i < 4; i++){
        for (int j = 3; j >= 0; j--){
            x = i, y = j;
            if (b[x][y] != 0){
                while (y < 3 && b[x][y + 1] == 0){
                    b[x][y + 1] = b[x][y];
                    b[x][y] = 0;
                    y++;
                    move = 1;
                }
            }
        }
    }
    for (int i = 0; i < 4 - dir; i++)
        rotate(b);
    if (move == 0)
        return 0;
    return 1;
}
```

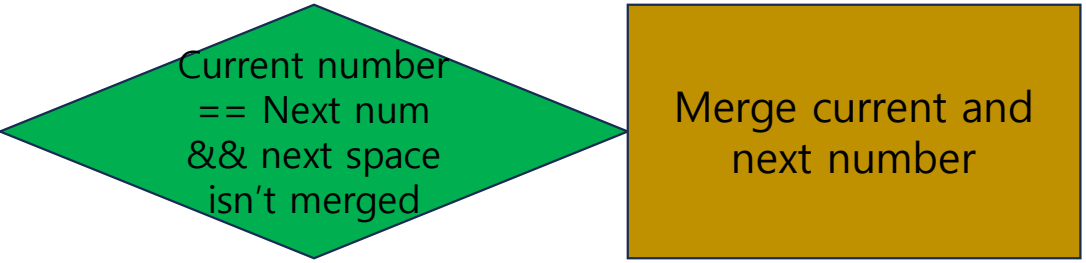
Rotate board until direction heads right and move numbers

<First for loop>

After the first rotation, we move numbers by examining where they should end. Since you can't merge twice in a single move, we create a merge array to check whether current space was merged or not.

If it's possible to move to next space, we move on to next space. We do this for every elements on board. We don't think about merging in here.

```
int set_board(int dir, int b[4][4]) {
    int x, y, move = 0, merge[4][4] = { 0, };
    for (int i = 0; i < dir; i++)
        rotate(b);
    for (int i = 0; i < 4; i++){
        for (int j = 3; j >= 0; j--){
            x = i, y = j;
            if (b[x][y] != 0){
                while (y < 3 && b[x][y + 1] == 0){
                    b[x][y + 1] = b[x][y];
                    b[x][y] = 0;
                    y++;
                    move = 1;
                }
            }
        }
    }
    for (int i = 0; i < 4; i++){
        for (int j = 3; j >= 0; j--){
            x = i, y = j;
            if (b[x][y] != 0 && y < 3 && b[x][y] == b[x][y + 1] && !merge[x][y + 1]){
                b[x][y + 1] += b[x][y];
                b[x][y] = 0;
                merge[x][y + 1] = 1;
                move = 1;
            }
        }
    }
    for (int i = 0; i < 4; i++){
        for (int j = 3; j >= 0; j--){
            x = i, y = j;
            if (b[x][y] != 0){
                while (y < 3 && b[x][y + 1] == 0){
                    b[x][y + 1] = b[x][y];
                    b[x][y] = 0;
                    y++;
                    move = 1;
                }
            }
        }
    }
    for (int i = 0; i < 4 - dir; i++)
        rotate(b);
    if (move == 0)
        return 0;
    return 1;
}
```



<Second for loop>

We start to merge elements that needs merging. We check whether the next value equals current value and whether the next space has been already merged.

If both conditions are clear, we merge the current value into the next value and note it into the merge array. We do this for every space in board and we don't think about moving the merged elements here.

```
int set_board(int dir, int b[4][4]) {
    int x, y, move = 0, merge[4][4] = { 0, };
    for (int i = 0; i < dir; i++)
        rotate(b);
    for (int i = 0; i < 4; i++){
        for (int j = 3; j >= 0; j--){
            x = i, y = j;
            if (b[x][y] != 0){
                while (y < 3 && b[x][y + 1] == 0){
                    b[x][y + 1] = b[x][y];
                    b[x][y] = 0;
                    y++;
                    move = 1;
                }
            }
        }
    }
    for (int i = 0; i < 4; i++){
        for (int j = 3; j >= 0; j--){
            x = i, y = j;
            if (b[x][y] != 0 && y < 3 && b[x][y] == b[x][y + 1] && !merge[x][y + 1]){
                b[x][y + 1] += b[x][y];
                b[x][y] = 0;
                merge[x][y + 1] = 1;
                move = 1;
            }
        }
    }
    for (int i = 0; i < 4; i++){
        for (int j = 3; j >= 0; j--){
            x = i, y = j;
            if (b[x][y] != 0){
                while (y < 3 && b[x][y + 1] == 0){
                    b[x][y + 1] = b[x][y];
                    b[x][y] = 0;
                    y++;
                    move = 1;
                }
            }
        }
    }
    for (int i = 0; i < 4 - dir; i++)
        rotate(b);
    if (move == 0)
        return 0;
    return 1;
}
```

Move numbers until next space isn't blank

<Third for loop and below>

Merged elements might have to keep moving so we repeat the first for loop to make sure every merged elements have reached its end. After every move is done, we must rotate the board back to its original direction.

Finally, we check whether there was actually a movement on board. If there wasn't, it means we have received an invalid command meaning we shouldn't move on to the next step.

```
int set_board(int dir, int b[4][4]) {
    int x, y, move = 0, merge[4][4] = { 0, };
    for (int i = 0; i < dir; i++)
        rotate(b);
    for (int i = 0; i < 4; i++){
        for (int j = 3; j >= 0; j--){
            x = i, y = j;
            if (b[x][y] != 0){
                while (y < 3 && b[x][y + 1] == 0){
                    b[x][y + 1] = b[x][y];
                    b[x][y] = 0;
                    y++;
                    move = 1;
                }
            }
        }
    }
    for (int i = 0; i < 4; i++){
        for (int j = 3; j >= 0; j--){
            x = i, y = j;
            if (b[x][y] != 0 && y < 3 && b[x][y] == b[x][y + 1] && !merge[x][y + 1]){
                b[x][y + 1] += b[x][y];
                b[x][y] = 0;
                merge[x][y + 1] = 1;
                move = 1;
            }
        }
    }
    for (int i = 0; i < 4; i++){
        for (int j = 3; j >= 0; j--){
            x = i, y = j;
            if (b[x][y] != 0){
                while (y < 3 && b[x][y + 1] == 0){
                    b[x][y + 1] = b[x][y];
                    b[x][y] = 0;
                    y++;
                    move = 1;
                }
            }
        }
    }
    for (int i = 0; i < 4 - dir; i++)
        rotate(b);
    if (move == 0)
        return 0;
    return 1;
}
```

Add a random 2 or 4 to empty space in board

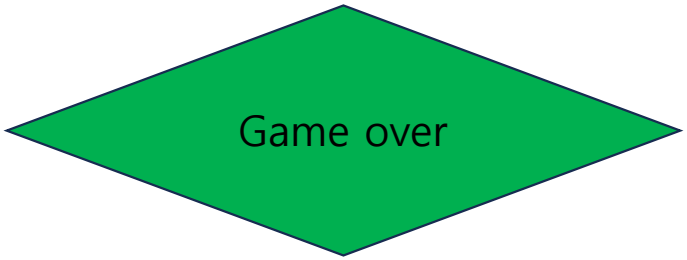
After we finished our command, we now have to add a new number at a random empty space on board. We add this by using the make_two_or_four function again and apply the changes to the tot variable.

If the function cannot create a new number, or if the board is full after creating a random number, we will have to check whether the game is over or still playable.

```
do {
    command = GetCommand();

    if (command != -1) {
        if (set_board(command, b)) {
            tot += make_two_or_four();
            if (is_game_over() == 0)
                game_over = 1;
            draw_board(tot, command);
        }
    }

} while (!game_over);
```



We check the board by is_game_over function. First for loop checks whether the board has an empty space. If an empty space exists, we no longer need to check the board. Function returns 1 and we repeat the whole playing sequence again.

Second for loop checks whether there are some elements that can be merged. If there is, then the game is still playable.

If all for loops are passed, the function will return 0 meaning making moves is impossible leading to a game over.

```
int is_game_over() {
    for (int i = 0; i < 4; i++)
        for (int j = 0; j < 4; j++)
            if (b[i][j] == 0)
                return 1;
    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
            if (b[i][j] == b[i + 1][j] || b[i][j] == b[i][j + 1])
                return 1;
    /* if game over return 0, else then return 1 */
    return 0;
}
```

Print “game over!”

The game has ended so we tell the user that the game is over. We print “game over!” and exit the program.