● project requirement

Project Description

Tetris is a tile-matching puzzle game. Given an initially empty $m * n$ game matrix, a sequence of blocks falls down the playing matrix and stop when the bottom touches either the ground or other resting blocks. If these resting blocks form a solid horizontal line without gaps then the line disappears and any blocks above it fall down to fill the space. Any solid horizontal line should disappear after each block adjustment. The game terminates either if any block exceeds matrix vertical boundary after all solid horizontal lines are removed or the input sequence of blocks is exhausted.

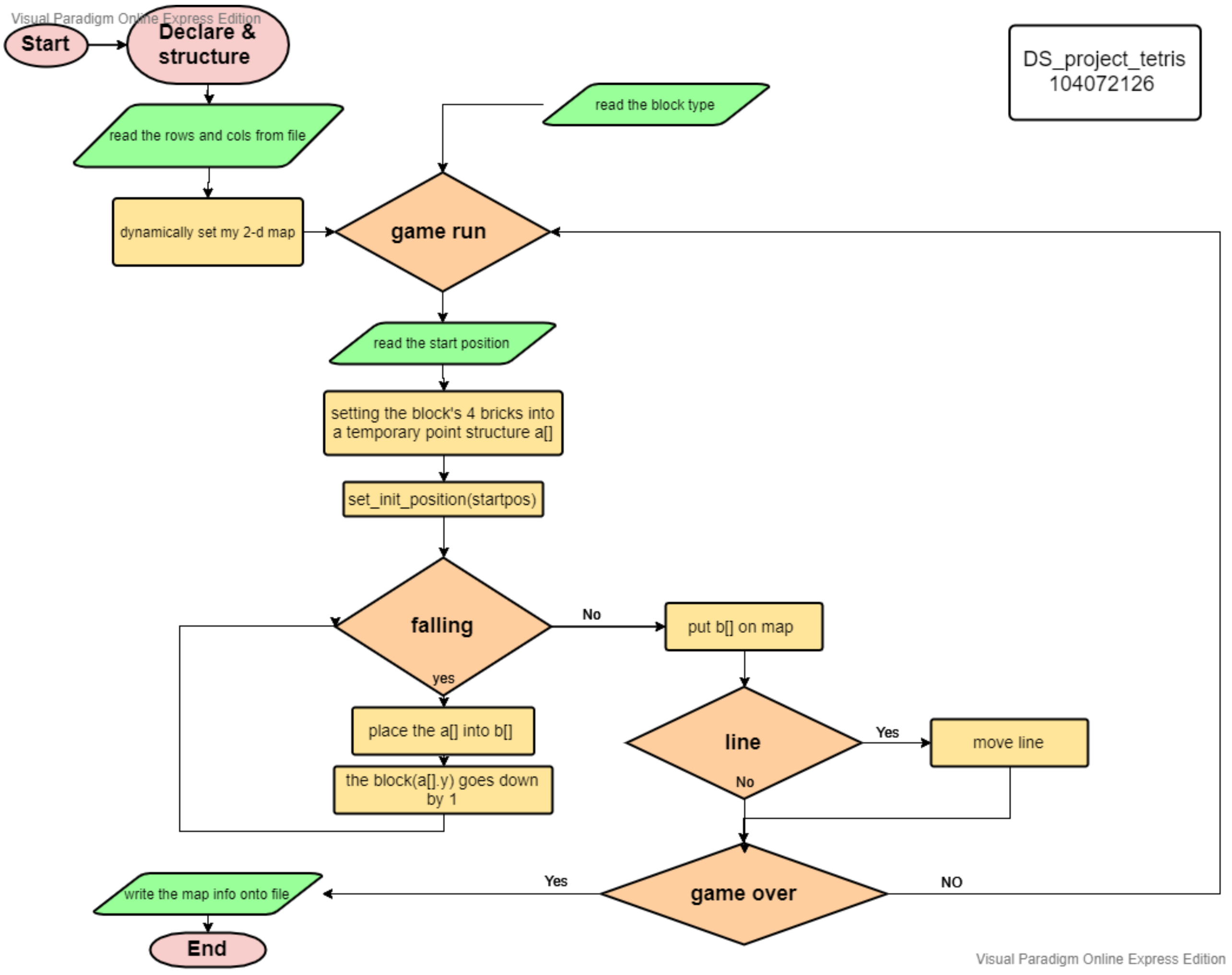
Input:

```
10  5
I1  5
I2  1
O   4
L3  3
End
```

output:

```
0 0 0 0 0
0 0 0 0 0
0 0 1 1 0
0 0 0 1 0
0 0 0 1 0
0 0 0 1 1
0 0 0 1 1
0 0 0 0 1
0 0 0 0 1
0 0 0 0 1
```

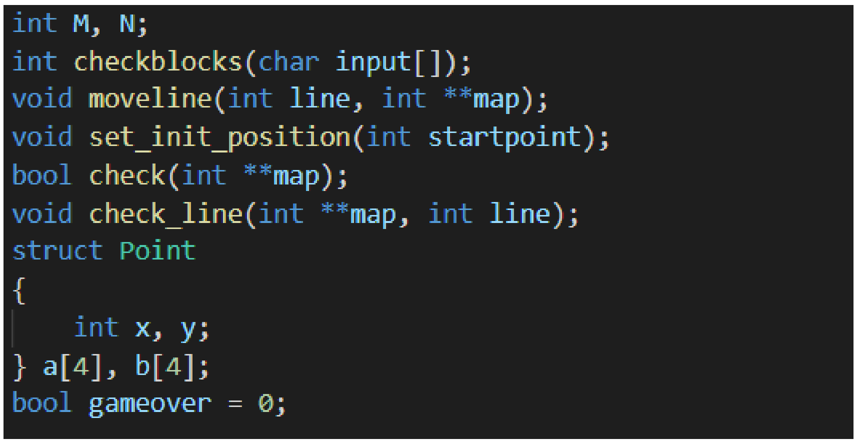
● my project flow chart


Visual Paradigm Online Express Edition
Start
Declare & structure
read the rows and cols from file
dynamically set my 2-d map
read the block type
DS_project_tetris 104072126
game run
read the start position
setting the block's 4 bricks into a temporary point structure a[]
set_init_position(startpos)
falling
No
yes
put b[] on map
place the a[] into b[]
the block(a[].y) goes down by 1
line
Yes
No
move line
game over
Yes
NO
write the map info onto file
End
Visual Paradigm Online Express Edition

- **Description in detail (7 steps)**

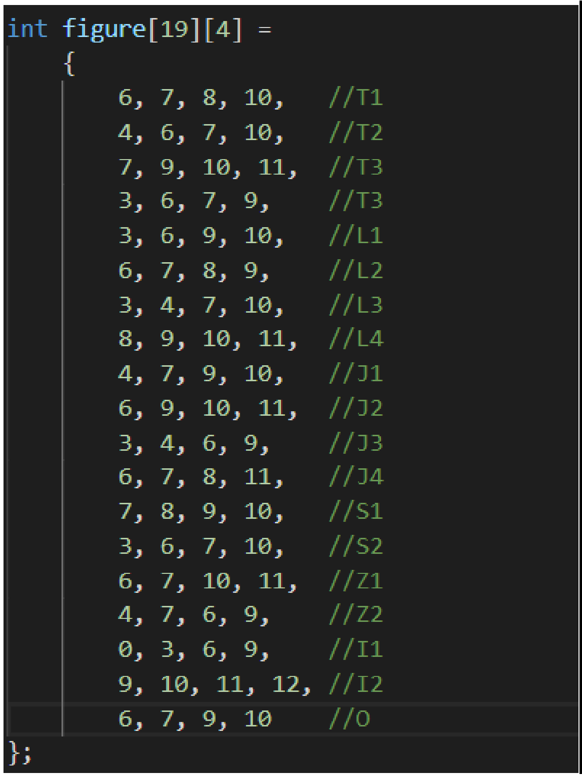
1. Declaration and the blocks structure

/* Declare a “Point” structure for storing blocks*/

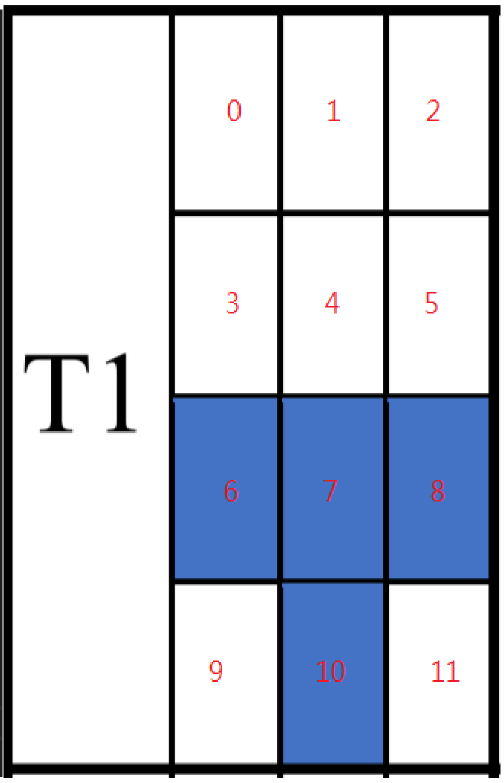


```
int M, N;
int checkblocks(char input[]);
void moveline(int line, int **map);
void set_init_position(int startpoint);
bool check(int **map);
void check_line(int **map, int line);
struct Point
{
    int x, y;
} a[4], b[4];
bool gameover = 0;
```

/*declare a 2-dimentional array to store 19 type blocks’ 4 bricks*/
/*using 4*3 blocks matrix to point the bricks’ position*/

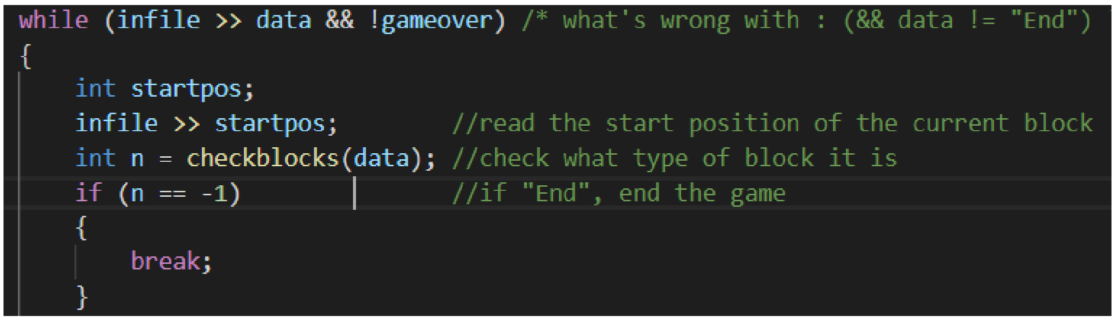


```
int figure[19][4] =
    {
        6, 7, 8, 10,    //T1
        4, 6, 7, 10,    //T2
        7, 9, 10, 11,   //T3
        3, 6, 7, 9,     //T3
        3, 6, 9, 10,    //L1
        6, 7, 8, 9,     //L2
        3, 4, 7, 10,    //L3
        8, 9, 10, 11,   //L4
        4, 7, 9, 10,    //J1
        6, 9, 10, 11,   //J2
        3, 4, 6, 9,     //J3
        6, 7, 8, 11,    //J4
        7, 8, 9, 10,    //S1
        3, 6, 7, 10,    //S2
        6, 7, 10, 11,   //Z1
        4, 7, 6, 9,     //Z2
        0, 3, 6, 9,     //I1
        9, 10, 11, 12,  //I2
        6, 7, 9, 10     //O
};
```



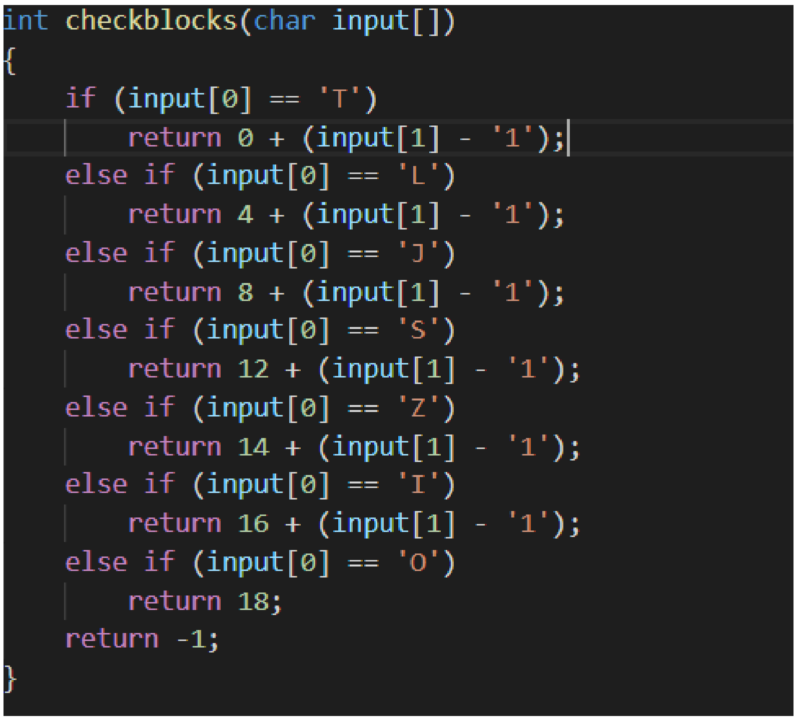
2. Game run until game over

/*read a string from file and store into “data”*/
/*put the string into checkblock() to identify which type of blocks it is*/



```
while (infile >> data && !gameover) /* what's wrong with : (&& data != "End")
{
    int startpos;
    infile >> startpos;         //read the start position of the current block
    int n = checkblocks(data); //check what type of block it is
    if (n == -1)                //if "End", end the game
    {
        break;
    }
```

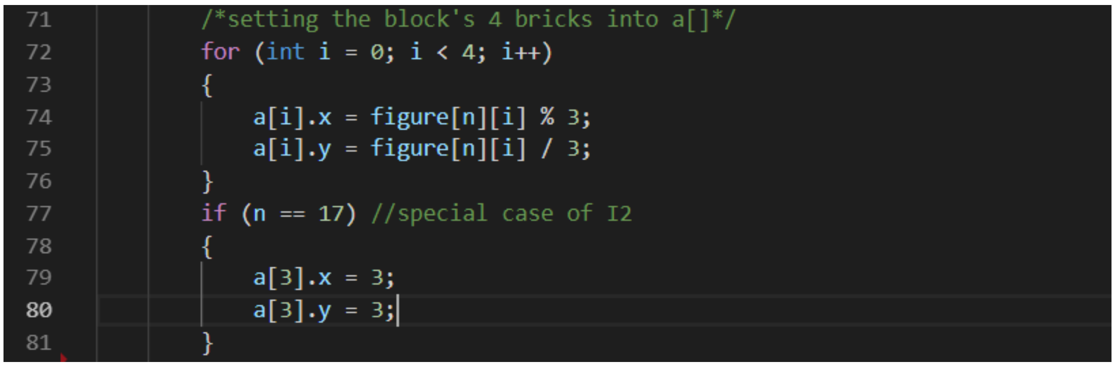
/*based on my initial 2-d array, we can judge the right row of the type blocks and return the row number*/



```
int checkblocks(char input[])
{
    if (input[0] == 'T')
        return 0 + (input[1] - '1');
    else if (input[0] == 'L')
        return 4 + (input[1] - '1');
    else if (input[0] == 'J')
        return 8 + (input[1] - '1');
    else if (input[0] == 'S')
        return 12 + (input[1] - '1');
    else if (input[0] == 'Z')
        return 14 + (input[1] - '1');
    else if (input[0] == 'I')
        return 16 + (input[1] - '1');
    else if (input[0] == 'O')
        return 18;
    return -1;
}
```

3. Setting the block's 4 bricks into a temporary point structure a[]

/*by previous n=checkblocks(), we can set the relative x y to temporary blocks :a[] */

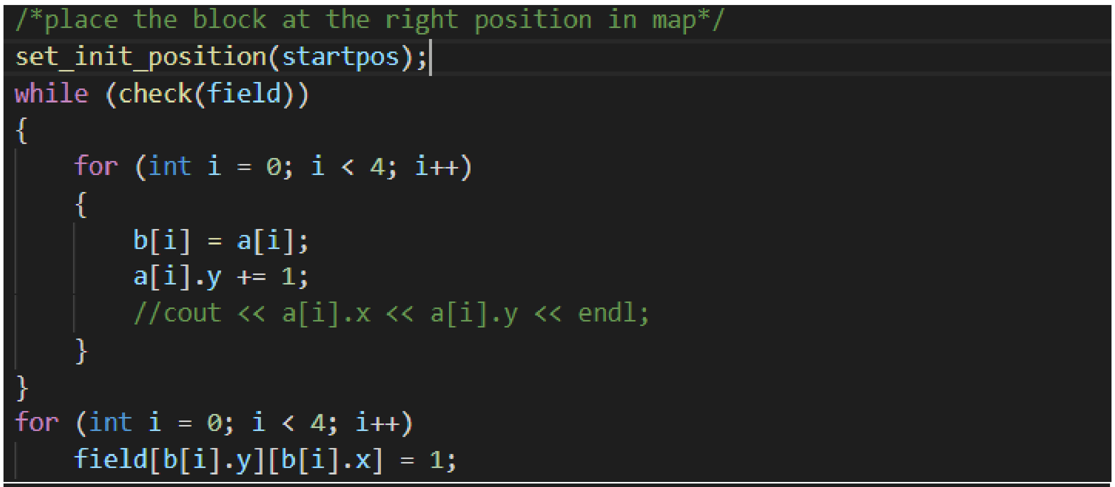


```
        /*setting the block's 4 bricks into a[]*/
        for (int i = 0; i < 4; i++)
        {
            a[i].x = figure[n][i] % 3;
            a[i].y = figure[n][i] / 3;
        }
        if (n == 17) //special case of I2
        {
            a[3].x = 3;
            a[3].y = 3;
        }
```

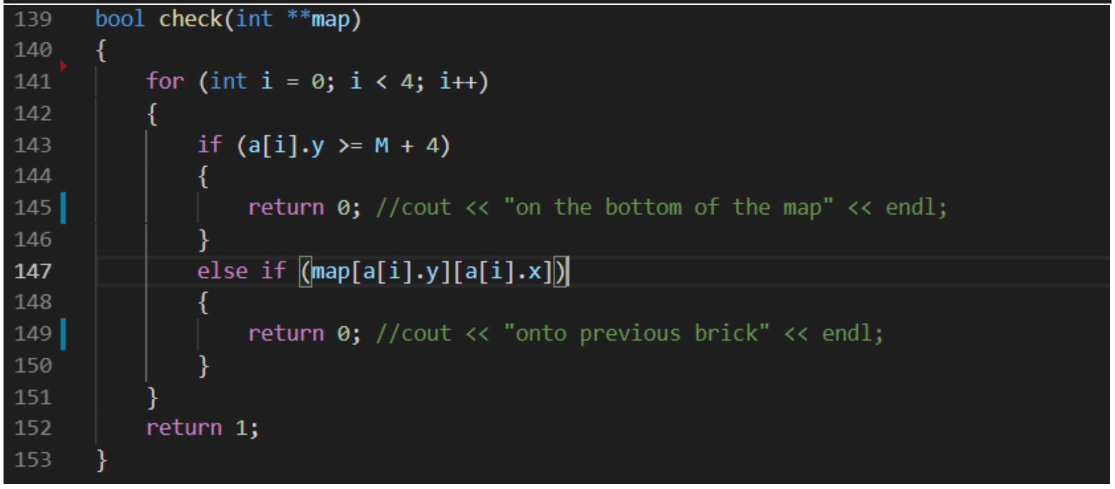
4. Falling the block onto the right position

/*while the tmp block a[] is legal(check(field)==1), store a[] to certain block b[], and move down a[] by 1*/

/*finally draw b[] onto map*/

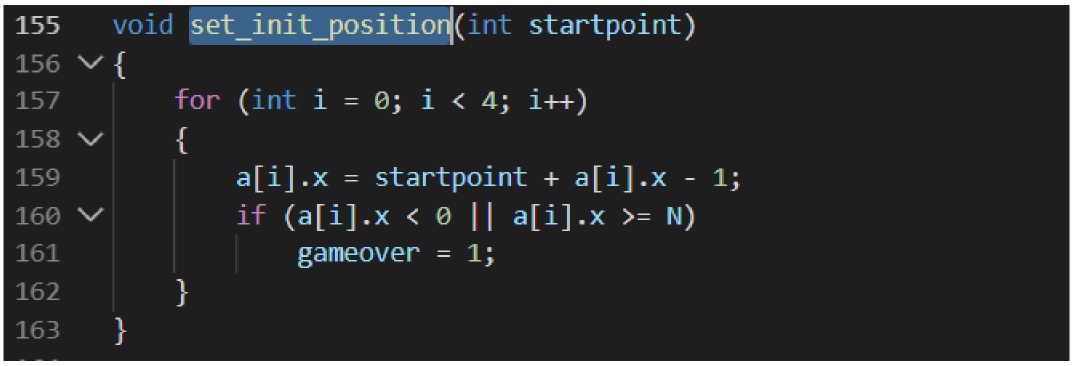


```
/*place the block at the right position in map*/
set_init_position(startpos);
while (check(field))
{
    for (int i = 0; i < 4; i++)
    {
        b[i] = a[i];
        a[i].y += 1;
        //cout << a[i].x << a[i].y << endl;
    }
}
for (int i = 0; i < 4; i++)
    field[b[i].y][b[i].x] = 1;
```



```
bool check(int **map)
{
    for (int i = 0; i < 4; i++)
    {
        if (a[i].y >= M + 4)
        {
            return 0; //cout << "on the bottom of the map" << endl;
        }
        else if (map[a[i].y][a[i].x])
        {
            return 0; //cout << "onto previous brick" << endl;
        }
    }
    return 1;
}
```

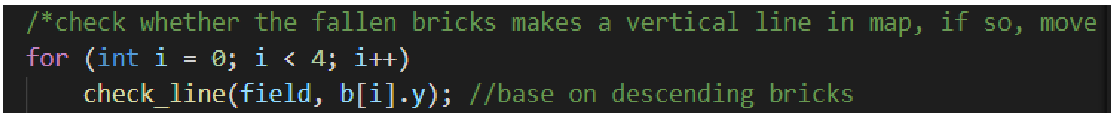
/*set the tmp block a[].x to right place after considering start point*/



```
void set_init_position(int startpoint)
{
    for (int i = 0; i < 4; i++)
    {
        a[i].x = startpoint + a[i].x - 1;
        if (a[i].x < 0 || a[i].x >= N)
            gameover = 1;
    }
}
```

5. Check line

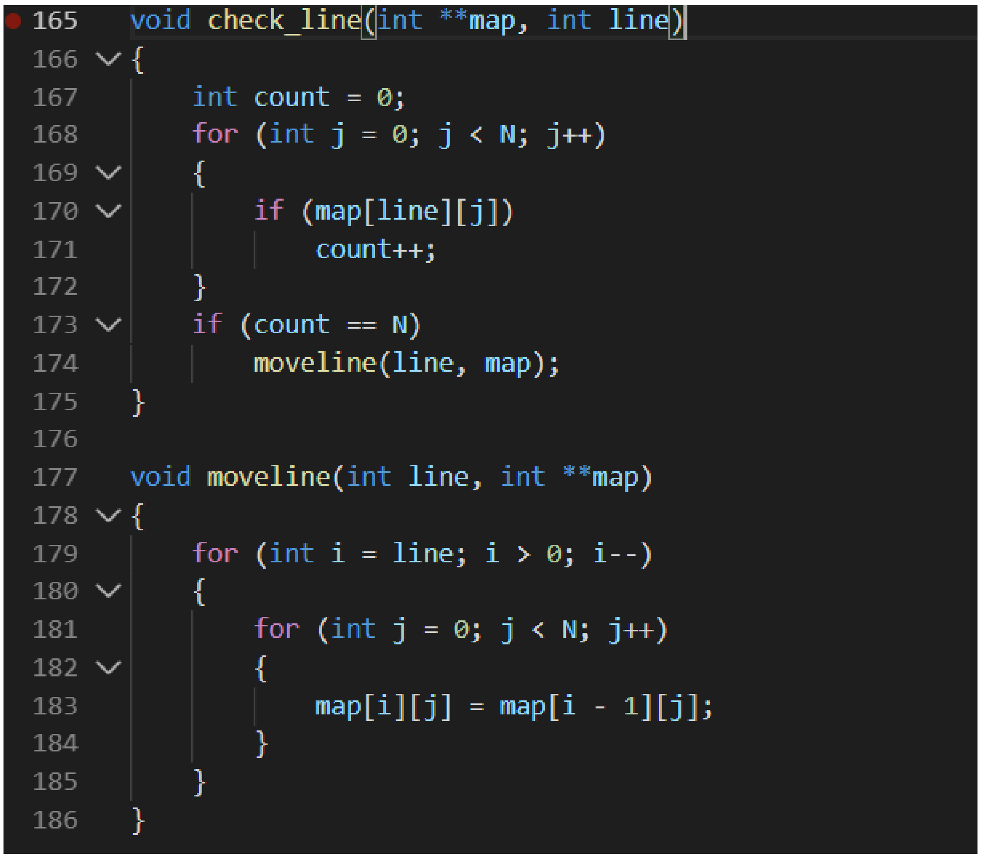
/*we only need to check the renew part of the map, which is the y of b[]*/



```
/*check whether the fallen bricks makes a vertical line in map, if so, move
for (int i = 0; i < 4; i++)
    check_line(field, b[i].y); //base on descending bricks
```

/*if all component of a given row is 1, then we move line*/

/*moving a give line by shifting all the upper line down 1 unit */

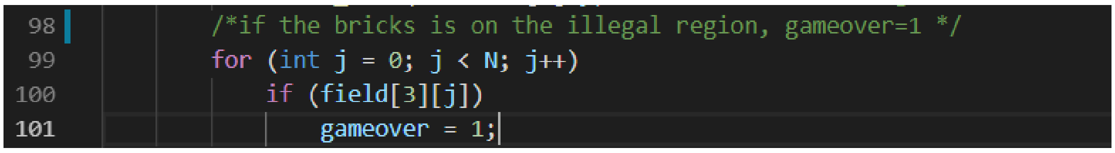


```
void check_line(int **map, int line)
{
    int count = 0;
    for (int j = 0; j < N; j++)
    {
        if (map[line][j])
            count++;
    }
    if (count == N)
        moveline(line, map);
}

void moveline(int line, int **map)
{
    for (int i = line; i > 0; i--)
    {
        for (int j = 0; j < N; j++)
        {
            map[i][j] = map[i - 1][j];
        }
    }
}
```

6. Check game over

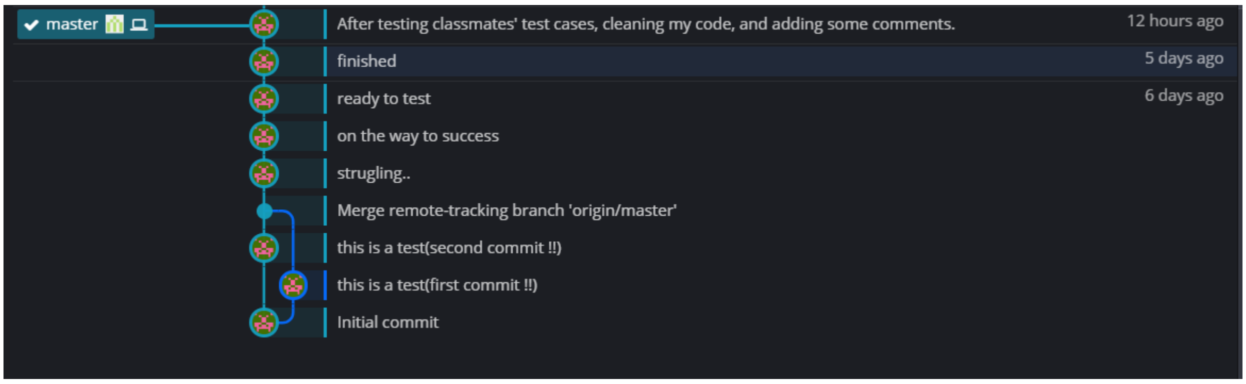
/*check the most top line+1 to identify whether the placed blocks exceed the map*/



```
        /*if the bricks is on the illegal region, gameover=1 */
        for (int j = 0; j < N; j++)
            if (field[3][j])
                gameover = 1;
```

7. “Back to while” or “break while & write file”

● git_record



```
After testing classmates' test cases, cleaning my code, and adding some comments.    12 hours ago
finished                                                                              5 days ago
ready to test                                                                         6 days ago
on the way to success
strugling..
Merge remote-tracking branch 'origin/master'
this is a test(second commit !!)
this is a test(first commit !!)
Initial commit
```