


luftaquila Update README.md

440ed02 29 minutes ago

1 contributor

264 lines (211 sloc) 9.7 KB

Raw Blame History   

전자공학프로그래밍 과제 1 보고서

전자공학과
201820908 오병준

1. 개발 요구 사항

보드 게임 프로그램 개발하기

1. 게임 규칙

- 0 ~ 7의 무작위 숫자가 적힌 5*5 사이즈 보드
- 보드의 좌상단 첫 번째 칸에서 시작.
- 동전이 위치한 칸의 숫자만큼 상하좌우 중 하나의 방향으로 이동.
- 동전이 보드 바깥으로 나가거나, 동전이 있는 칸의 숫자가 0일 경우 게임 종료.
- 동전의 최대 이동 가능 횟수 출력
 - 동전이 무한 번 움직일 수 있다면 -1 출력

2. 프로그램 구조

1. 매크로 변수

1. `Width` , `Depth` : 보드의 가로세로 폭 지정
2. `MAX_MOVEMENT` : 이동 가능 횟수의 최대치 지정

2. 전역 변수

1. `square_board` : 보드 데이터 저장 배열
2. `max_movement` : 최대 이동 가능 횟수 저장 변수

3. 함수

1. `setBoard(int max_width, int max_depth)`
보드의 크기에 맞게, 각 칸에 무작위 난수를 채워 넣어 보드를 생성한다.
 - 반환형 : `void`
 - `max_width` , `max_depth` : 생성할 보드의 가로세로 길이
2. `printBoard(int max_width, int max_depth)`
생성한 보드를 화면에 출력한다.
 - 반환형 : `void`
 - `max_width` , `max_depth` : 출력할 보드의 가로세로 길이
3. `playBoard(int width, int depth, intmovement_count)`
규칙에 따라 동전을 움직여 게임을 실행한다.
 - 반환형 : `void`

- `width`, `depth` : 동전의 현재 좌표
- `movement_count` : 동전이 현재까지 움직인 횟수

2. 코드 분석

1. `main()`

```
21  int main(void) {
22      setBoard(width, depth);
23      printBoard(width, depth);
24      playBoard(0, 0, 0);
25      printf("최대 이동 횟수: %d\n\n", max_movement_count);
26      return 0;
27  }
```

```
setBoard(width, depth);
```

`setBoard` 함수를 호출해 난수 보드를 생성한다.

```
printBoard(width, depth);
```

`printBoard` 함수를 호출해 생성한 보드를 화면에 출력한다.

```
playBoard(0, 0, 0);
```

`playBoard` 함수를 호출해 시작 위치를 `0, 0` 으로 지정하고, 게임을 시작한다.

```
printf("최대 이동 횟수: %d\n\n", max_movement_count);
```

`printf` 함수를 통해 최대 이동 횟수를 출력하고, 프로그램을 종료한다.

2. `setBoard()`

```
29  void setBoard(int max_width, int max_depth) {
30      srand(time(NULL));
31      for(int d = 0; d < max_depth; d++) {
32          for(int w = 0; w < max_width; w++)
33              square_board[w][d] = rand() % 8;
34      }
35  }
```

```
srand(time(NULL));
```

`rand` 함수의 시드값을 설정한다.

```
for(int d = 0; d < max_depth; d++) {
    for(int w = 0; w < max_width; w++)
```

```
square_board[w][d] = rand() % 8;
}
```

보드 배열 `square_board` 의 모든 요소에 대해 난수를 생성해 대입한다.

`rand() % 8` 은 생성한 난수를 8로 나눈 나머지가므로 0 ~ 7 사이의 무작위 값이 된다.

3. `playBoard()`

```
37 void playBoard(int width, int depth, int movement_count) {
38     char flag[5] = { 0, };
39     if(!square_board[width][depth]) return;
40     else if(max_movement_count < 0) return;
41     else movement_count++;
42
43     if(movement_count > max_movement_count) max_movement_count = movement_count;
44     if(movement_count > MAX_MOVEMENT) max_movement_count = -1;
45
46     if(width + square_board[width][depth] < Width) flag[0] = flag[1] = 1;
47     if(width - square_board[width][depth] >= 0) flag[0] = flag[2] = 1;
48     if(depth + square_board[width][depth] < Depth) flag[0] = flag[3] = 1;
49     if(depth - square_board[width][depth] >= 0) flag[0] = flag[4] = 1;
50
51     if(flag[0]) {
52         if(flag[1]) playBoard(width + square_board[width][depth], depth, movement_count);
53         if(flag[2]) playBoard(width - square_board[width][depth], depth, movement_count);
54         if(flag[3]) playBoard(width, depth + square_board[width][depth], movement_count);
55         if(flag[4]) playBoard(width, depth - square_board[width][depth], movement_count);
56     }
57 }
```

```
char flag[5] = { 0, };
```

상하좌우 각 방향에 대해 이동 가능한지 여부를 저장하는 배열 `flag` 를 선언하고 모든 요소를 0으로 초기화한다.

- `flag[0]` : 해당 칸에서 이동 가능한 방향이 하나라도 존재하는지 여부를 저장한다.
0이면 이동 가능한 방향이 존재하지 않는다는 의미이다.
- `flag[1] ~ flag[4]` : 순서대로 오른쪽, 왼쪽, 아래, 위 방향으로 이동할 수 있는지 여부를 저장한다.

```
if(!square_board[width][depth]) return;
else if(max_movement_count < 0) return;
else movement_count++;
```

현재 동전이 위치한 칸의 숫자가 0 이거나, 최대 이동 가능 횟수를 초과했다면 함수를 종료한다.

해당사항이 없다면, 1회 이동할 수 있다는 의미이므로 이동 횟수를 1회 증가시킨다.

```
if(movement_count > max_movement_count) max_movement_count = movement_count;
if(movement_count > MAX_MOVEMENT) max_movement_count = -1;
```

현재 이동 횟수가 저장된 최대 이동 횟수보다 크다면, 최대 이동 횟수를 갱신한다.

만약 현재 이동 횟수가 최대 이동 가능 횟수를 초과했다면, 최대 이동 횟수에 -1 을 저장한다.

```
if(width + square_board[width][depth] < Width) flag[0] = flag[1] = 1;
if(width - square_board[width][depth] >= 0) flag[0] = flag[2] = 1;
```

```

if(depth + square_board[width][depth] < Depth)  flag[0] = flag[3] = 1;
if(depth - square_board[width][depth] >= 0)      flag[0] = flag[4] = 1;

```

상하좌우 각 방향으로 이동할 수 있는지 여부를 판단하고,
각 방향에 대한 이동 가능 여부를 `flag` 배열의 해당하는 요소에 저장한다.

```

if(flag[0]) {
    if(flag[1]) playBoard(width + square_board[width][depth], depth, movement_count);
    if(flag[2]) playBoard(width - square_board[width][depth], depth, movement_count);
    if(flag[3]) playBoard(width, depth + square_board[width][depth], movement_count);
    if(flag[4]) playBoard(width, depth - square_board[width][depth], movement_count);
}

```

`flag[0]` 이 참인지 판단해 이동 가능한 방향이 존재하는지 확인한다.
이동 가능한 방향이 하나라도 존재한다면, 각 방향에 대한 이동 가능 여부를 확인한다.
이후, 이동 가능한 모든 방향으로 동전의 위치를 갱신해 `playBoard` 함수를 재귀 호출한다.

재귀 호출된 함수에서 다시 이동 가능 여부를 조사하므로, 더 이상 이동이 불가능할 때까지 재귀 호출이 이루어진다.

4. `printBoard()`

```

59 void printBoard(int max_width, int max_depth) {
60     for (int d = 0; d < max_depth; d++) {
61         for (int w = 0; w < max_width; w++)
62             printf("%2d", square_board[w][d]);
63         printf("\n");
64     }
65     printf("\n");
66 }

```

```

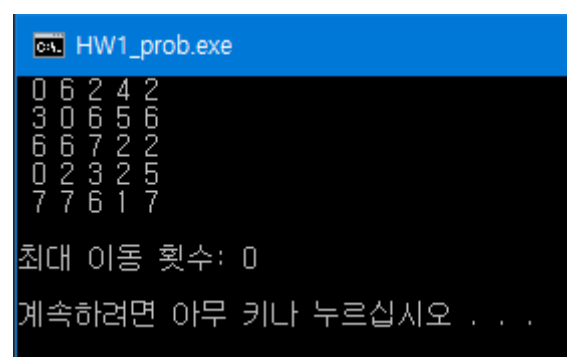
for (int d = 0; d < max_depth; d++) {
    for (int w = 0; w < max_width; w++)
        printf("%2d", square_board[w][d]);
    printf("\n");
}
printf("\n");

```

`square_board` 2차원 배열의 구조를 그대로 화면에 출력한다.
한 행의 출력이 끝나면 한 줄을 개행한다.

3. 실행 결과 분석

I. 0회 이동



```

0 6 2 4 2
3 0 6 5 6
6 6 7 2 2
0 2 3 2 5
7 7 6 1 7

최대 이동 횟수: 0
계속하려면 아무 키나 누르십시오 . . .

```

첫 시작 칸인 `square_board[0][0]` 의 값이 0이므로, 단 1회도 이동할 수 없다.
`playBoard()` 의 첫 번째 호출에서

```

if(!square_board[width][depth]) return;

```

에 걸려 바로 함수가 종료되므로 `max_movement_count` 는 선언 후 초기화된 값인 0이다.
따라서, 최대 이동 횟수는 0회가 된다.

II. 1회 이동

```
C:\ HW1_prob.exe
6 2 2 1 6
0 4 6 6 5
0 5 0 2 6
1 5 2 2 1
7 5 1 3 5

최대 이동 횟수: 1
계속하려면 아무 키나 누르십시오 . . .
```

첫 시작 칸인 `square_board[0][0]` 의 값이 6으로, 보드의 폭과 길이를 초과한다.
따라서 동전은 우측 또는 아래쪽으로 단 1회 이동할 수 있으며, 이동한 후 보드를 이탈해 게임이 종료된다.

코드상에서는

```
if(!square_board[width][depth]) return;
else if(max_movement_count < 0) return;
else movement_count++;
```

에 따라 해당되는 조건이 없으므로 이동 횟수가 1회 증가한다.
이후, 이동 가능한 방향 조사를 통과하지 못하므로 함수가 종료된다.

III. 3회 이동

```
C:\ HW1_prob.exe
1 3 7 0 6
0 0 1 4 3
0 1 6 2 7
5 0 6 0 6
7 7 2 1 5

최대 이동 횟수: 3
계속하려면 아무 키나 누르십시오 . . .
```

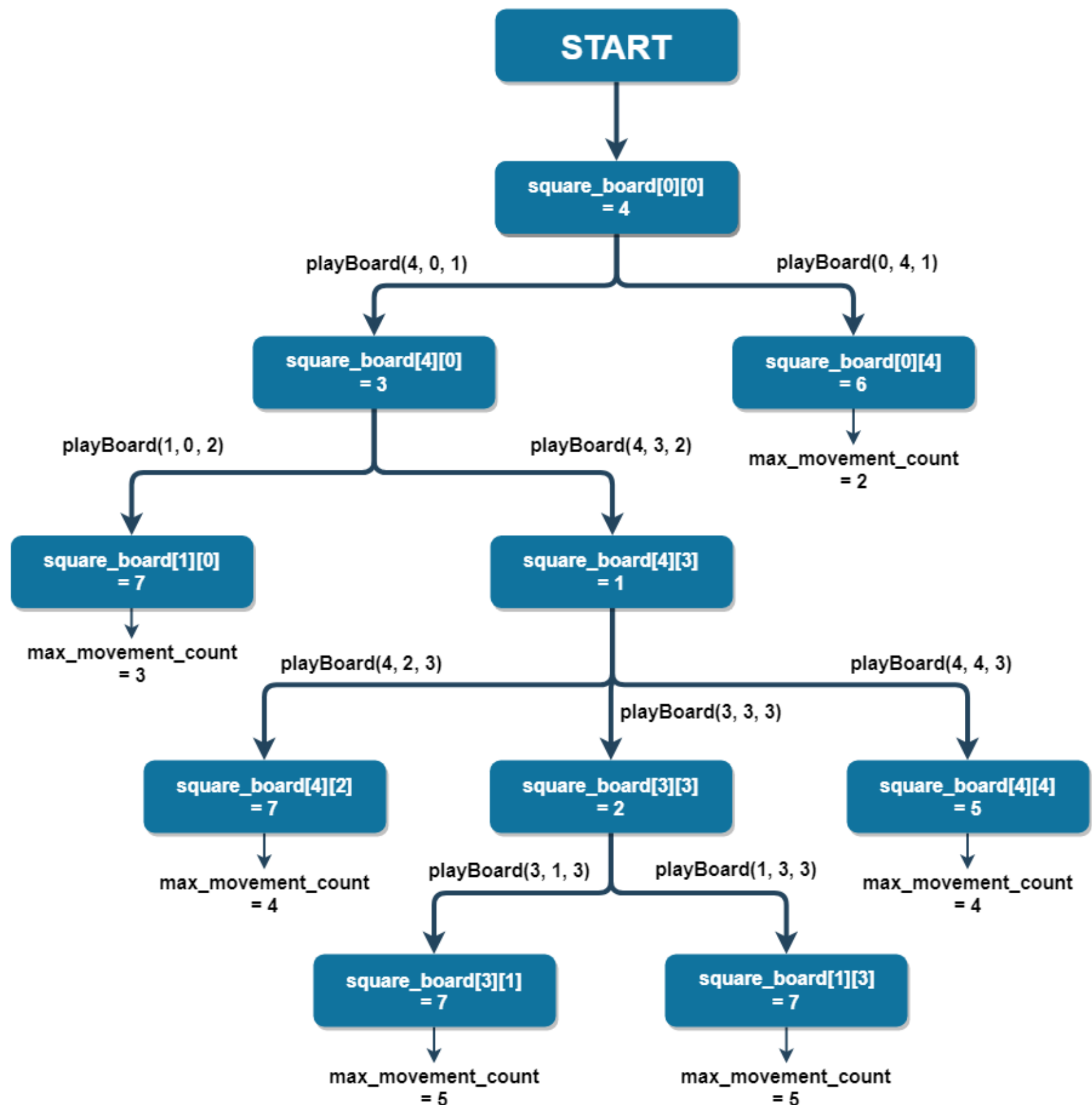
1. 처음 시작 칸인 `square_board[0][0]` 의 값이 1로, 우측 및 아래로 이동 가능하다.
2. `max_movement_count` 가 1 이 된다.
3. 따라서 이동 가능 방향 조사를 거친 `flag` 배열은 `[1, 1, 0, 1, 0]` 이 된다.
4. `flag[1]` 과 `flag[3]` 이 참이므로, `playBoard(1, 0, 1)` 과 `playBoard(0, 1, 1)` 이 재귀 호출된다.
 - i. `playBoard(1, 0, 1)`
`max_movement_count` 가 2 가 된다.
`square_board[1][0]` 이 3 이므로, 우측 및 아래쪽으로 이동 가능하다.
앞선 과정을 거쳐 `playBoard(4, 0, 2)` 와 `playBoard(1, 3, 2)` 가 재귀 호출된다.
 - a. `playBoard(4, 0, 2)`
`max_movement_count` 가 3 이 된다.
`square_board[4][0]` 이 6 이므로, 모든 방향에서 보드를 초과하여 게임이 종료된다.
 - b. `playBoard(1, 3, 2)`
`square_board[1][3]` 이 0 이므로, 게임이 종료된다.
 - ii. `playBoard(0, 1, 1)`
`square_board[0][1]` 이 0 이므로, 게임이 종료된다.
5. 따라서, 최대 이동 횟수는 3 이 된다.

IV. 5회 이동

```
C:\ HW1_prob.exe
4 7 0 2 3
4 3 5 7 1
6 0 0 3 7
2 7 3 2 1
6 2 5 2 5

최대 이동 횟수: 5
계속하려면 아무 키나 누르십시오 . . .
```

앞선 예시들과 동일하게, 더이상 이동할 수 없을 때까지 재귀 호출을 반복한다.
이를 도식화하면 다음과 같다.



V. 무한 번 이동

```

C:\ HW1_prob.exe
3 6 7 7 0
1 2 5 5 0
7 6 7 7 1
2 3 4 2 5
4 5 5 4 7

최대 이동 횟수: -1
계속하려면 아무 키나 누르십시오 . . .
  
```

1. 처음 시작 칸인 `square_board[0][0]` 의 값이 3이므로, 우측 및 아래로 이동 가능하다.
2. `max_movement_count` 가 1 이 된다.
3. `playBoard(3, 0, 1)` 과 `playBoard(0, 3, 1)` 을 재귀 호출한다.
 - i. `playBoard(3, 0, 1)`
`square_board[3][0]` 이 7 이므로, 모든 방향에서 보드를 초과하여 게임이 종료된다.
 - ii. `playBoard(0, 3, 1)`
`square_board[0][3]` 이 2 이므로, 우측 및 위로 이동 가능하다.
`playBoard(0, 1, 2)` 와 `playBoard(2, 3, 2)` 을 재귀 호출한다.

이 때, `playBoard(0, 1, 2)` 의 값이 `1` 이므로, 동전은 처음 시작점인 `square_board[0][0]` 로 다시 돌아올 수 있다. 따라서 동전이 위의 코스를 순환하며 끊임없이 움직일 수 있게 된다.

위 과정을 반복하면서 `movement_count` 가 1씩 증가하다가, 25를 초과하는 순간 아래 코드

```
if(movement_count > MAX_MOVEMENT) max_movement_count = -1;
```

에 의해 `max_movement_count` 에 `-1` 이 저장되고, 다음 재귀 호출 시

```
else if(max_movement_count < 0) return;
```

에 의해 종료되며 최대 이동 횟수로 `-1`을 출력한다.

4. 전체 코드

HW1_prob.c

README.md

```
1 //학과 : 전자공학과
2 //학번 : 201820908
3 //이름 : 오병준
4 //개발환경(visualstudio버전) : gcc 6.3.0
5
6 #include <stdio.h>
7 #include <stdlib.h>
8 #include <time.h>
9
10 #define Width 5
11 #define Depth 5
12 #define MAX_MOVEMENT 25
13
14 int square_board[Width][Depth];
15 int max_movement_count = 0;
16
17 void setBoard(int max_width, int max_depth);
18 void playBoard(int width, int depth, int movement_count);
19 void printBoard(int max_width, int max_depth);
20
21 int main(void) {
22     setBoard(Width, Depth);
23     printBoard(Width, Depth);
24     playBoard(0, 0, 0);
25     printf("최대 이동 횟수: %d\n\n", max_movement_count);
26     return 0;
27 }
28
29 void setBoard(int max_width, int max_depth) {
30     srand(time(NULL));
31     for(int d = 0; d < max_depth; d++) {
32         for(int w = 0; w < max_width; w++)
33             square_board[w][d] = rand() % 8;
34     }
35 }
36
37 void playBoard(int width, int depth, int movement_count) {
38     char flag[5] = { 0, };
39     if(!square_board[width][depth]) return;
40     else if(max_movement_count < 0) return;
41     else movement_count++;
42
43     if(movement_count > max_movement_count) max_movement_count = movement_count;
44     if(movement_count > MAX_MOVEMENT) max_movement_count = -1;
45
46     if(width + square_board[width][depth] < Width) flag[0] = flag[1] = 1;
47     if(width - square_board[width][depth] >= 0) flag[0] = flag[2] = 1;
48     if(depth + square_board[width][depth] < Depth) flag[0] = flag[3] = 1;
49     if(depth - square_board[width][depth] >= 0) flag[0] = flag[4] = 1;
50
51     if(flag[0]) {
52         if(flag[1]) playBoard(width + square_board[width][depth], depth, movement_count);
53         if(flag[2]) playBoard(width - square_board[width][depth], depth, movement_count);
54         if(flag[3]) playBoard(width, depth + square_board[width][depth], movement_count);
55         if(flag[4]) playBoard(width, depth - square_board[width][depth], movement_count);
56     }
57 }
58
59 void printBoard(int max_width, int max_depth) {
60     for (int d = 0; d < max_depth; d++) {
61         for (int w = 0; w < max_width; w++)
62             printf("%2d", square_board[w][d]);
63         printf("\n");
64     }
65     printf("\n");
66 }
```

Combo
Max 949

