

# Solving Maze

Team 8

202235002 KangGiwon  
202235510 KimYooHyun  
202235033 KimHyeJeong  
202235116 JeonYoungmin

# TABLE OF CONTENTS



## INTRODUCTION

The operation method of our game



## BACKTRACKING

Where and how backtracking is used



## ANALYSIS

The code we made with C/C++ language



## CONCLUSION

A demonstration video for show the game

# 01

## Introduction

The operation method  
of our game





# GOAL



## Key

Need to get the key first

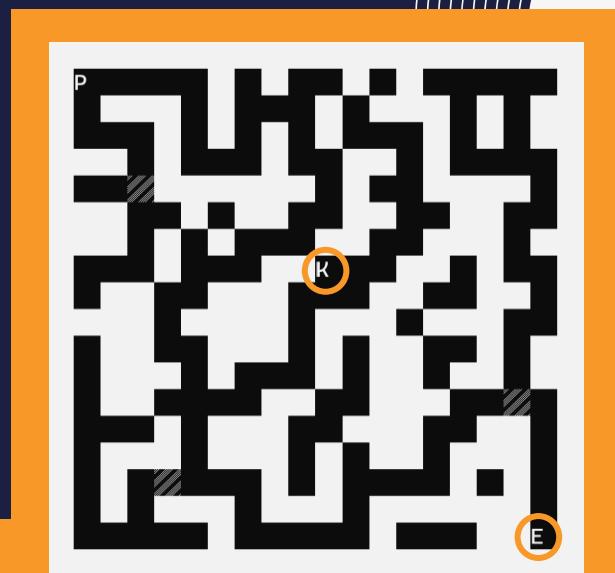
“K” in maze



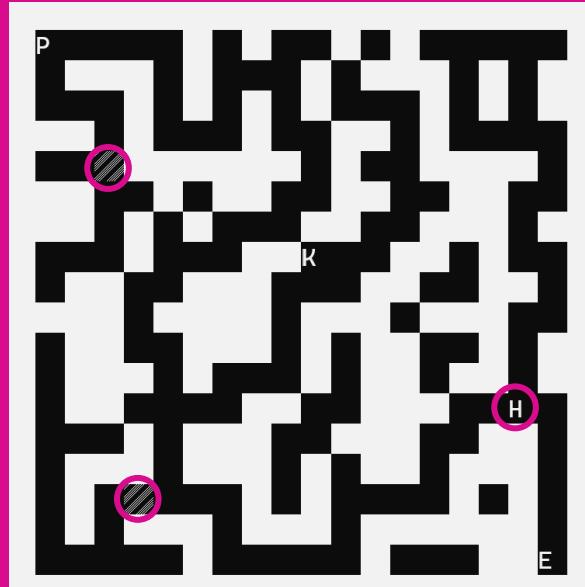
## Escape

User's destination

“E” in maze



# How to Solve?



## Breaking down a wall

It gives you chance to break setted block



## Show Hint

Which block should break to get to escape



Try to click space bar !

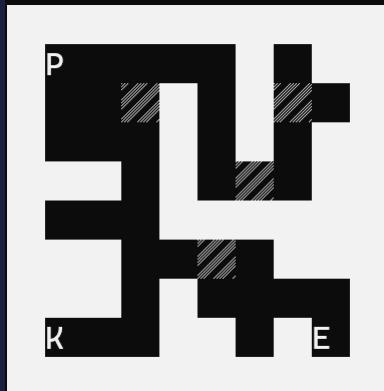


# BackTracking

How was the backtracking algorithm used  
in the game?

02

# BackTracking for HINT



**What the algorithm should think :**

which block is the one that corresponds to the hint?

# Pseudocode of Hint Function

```
Function Hint(Maze, Solve, canbreak, HstPlayer):
    havekey = HaveKey() // 열쇠 보유 여부 확인

    if not havekey:
        // 열쇠가 없으면 Hint1 함수 실행
        Call Hint1(Maze, Solve, HstPlayer, havekey)

    if havekey:
        // Solve 배열 초기화
        Call InitSolve(Solve)

        // 열쇠가 있으면 Hint2 함수 실행
        Hint2Result = Call Hint2(Maze, Solve, canbreak, HstPlayer, havekey)

    Return Hint2Result // Hint 2 실행 가능

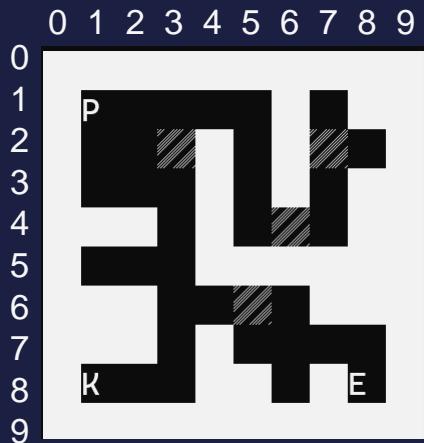
    Return 0 // 열쇠가 없는 경우 Hint1로 return
End Function
```



⋮ +

XX  
⋮ ⋮

# Process description



[2,7]
...
[4,6]
...
[1,5]
[1,4]
[1,3]
[1,2]
[1,1]

Candidate branch

Candidate

branch

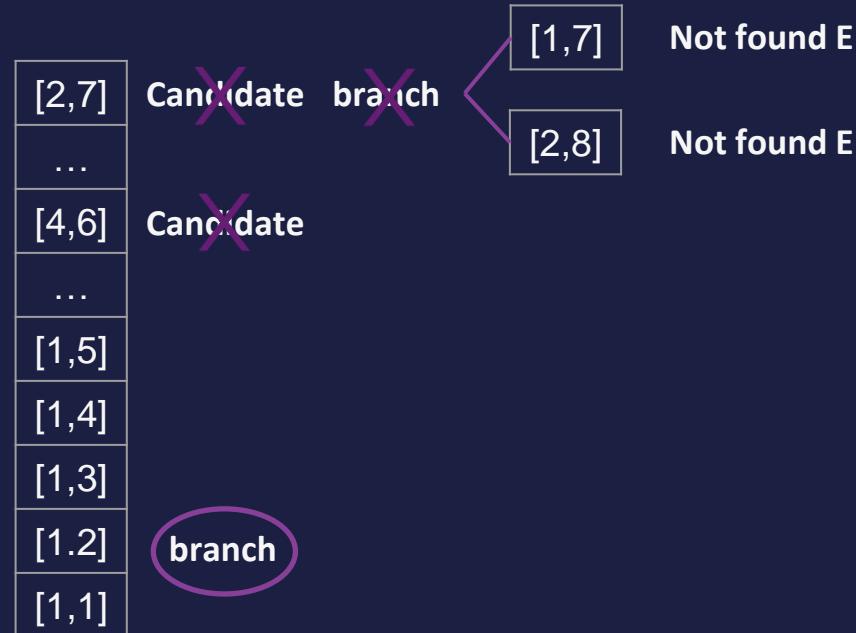
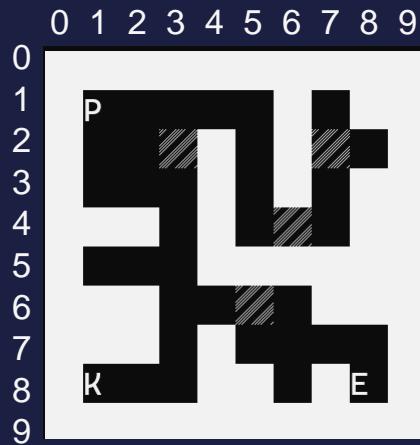


Not found E

Not found E



# Process description



# Basic Background

0		Wall
1		Available Way
2		Escape Rout
3		Breakable Wall
4	K	
5	E	
6	H	Hint
10	P	Current Location



# Hint Function

```
int Hint1(int Maze[MAZE_SIZE][MAZE_SIZE], int Solve[MAZE_SIZE][MAZE_SIZE], position HstPlayer, bool& havekey) {
    GotoXY(0, 6);
    printf("havekey : %d", havekey);
    } how many maze cells it corresponds to in that position

    if Maze[HstPlayer.y - Y_AXIS][int(HstPlayer.x - X_AXIS) / 2] == 4 {
        havekey = true;
        return 1;
    } } the corresponding cell is 4(key),
        convert the havekey function to true and return 1

    if (H.isValid(Maze, HstPlayer, havekey)) {
        if (Solve[HstPlayer.y - Y_AXIS][int(HstPlayer.x - X_AXIS) / 2] == 2) {
            return 0;
        }

        Solve[HstPlayer.y - Y_AXIS][int(HstPlayer.x - X_AXIS) / 2] = 2;

        for (int i = 0; i < 4; i++) {
            int nx = HstPlayer.x + 2 * dx[i];
            int ny = HstPlayer.y + dy[i];
            if (Hint1(Maze, Solve, { nx, ny }, havekey) == 1) {
                return 1;
            }
        }
        Solve[HstPlayer.y - Y_AXIS][int(HstPlayer.x - X_AXIS) / 2] = 1;
    } } Move up, down, left, right, and call the Hint1 at each location,
        if the result is 1 (if the key was found), immediately to end the function

    return 0;
}
```

} If there's nowhere else to move, backtracking



```
int Hint2(int Maze[MAZE_SIZE][MAZE_SIZE], int Solve[MAZE_SIZE][MAZE_SIZE], int canbreak, position HstPlayer, bool& havekey) { // havekey 존재 여부 추가  
    GotoXY(0, 6);  
    printf("havekey : %d", havekey);
```

```
if (!havekey) { // havekey 없을때 hint 사용 불가  
    return 0;
```

```
if (Maze[HstPlayer.y - Y_AXIS][int(HstPlayer.x - X_AXIS) / 2] == 5) {  
    Solve[HstPlayer.y - Y_AXIS][int(HstPlayer.x - X_AXIS) / 2] = 2;  
    return 1;
```

```
if (canbreak && Maze[HstPlayer.y - Y_AXIS][int(HstPlayer.x - X_AXIS) / 2] == 3) {  
    Solve[HstPlayer.y - Y_AXIS][int(HstPlayer.x - X_AXIS) / 2] = 2;  
    Maze[HstPlayer.y - Y_AXIS][int(HstPlayer.x - X_AXIS) / 2] = 1;
```

```
for (int i = 0; i < 4; i++) {  
    int nx = HstPlayer.x + 2 * dx[i];  
    int ny = HstPlayer.y + dy[i];  
    if (Hint2(Maze, Solve, canbreak - 1, { nx,ny }, havekey) == 1) {  
        Maze[HstPlayer.y - Y_AXIS][int(HstPlayer.x - X_AXIS) / 2] = 6;  
        return 1;  
    }  
}  
// reset the wall for backtracking
```

```
Solve[HstPlayer.y - Y_AXIS][int(HstPlayer.x - X_AXIS) / 2] = 1;  
Maze[HstPlayer.y - Y_AXIS][int(HstPlayer.x - X_AXIS) / 2] = 3;
```

```
if (H.isValid(Maze, HstPlayer, havekey)) {  
    if (Solve[HstPlayer.y - Y_AXIS][int(HstPlayer.x - X_AXIS) / 2] == 2) {  
        return 0;  
    }
```

```
Solve[HstPlayer.y - Y_AXIS][int(HstPlayer.x - X_AXIS) / 2] = 2;  
  
for (int i = 0; i < 4; i++) {  
    int nx = HstPlayer.x + 2 * dx[i];  
    int ny = HstPlayer.y + dy[i];  
    if (Hint2(Maze, Solve, canbreak, { nx,ny }, havekey) == 1) {  
        return 1;  
    }  
}
```

```
Solve[HstPlayer.y - Y_AXIS][int(HstPlayer.x - X_AXIS) / 2] = 1;  
  
return 0;
```

} Found escapes within the path to go

} If you can break a wall, break it and move

} Recursively invokes the **Hint2** as it moves in four directions from the player's current position

If the hint is correct, mark it as 6 i.e. Hint

} Same as hint1 function

} If there's nowhere else to move, backtracking

# Analysis

The code we made with  
C/C++ language

03



# Additional Function

```
//인게임
void GameMain(void) {
    int Maze[MAZE_SIZE][MAZE_SIZE] = {
        {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0},
        {0, 0, 1, 3, 0, 1, 0, 3, 1, 0, 0, 0},
        {0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0},
        {0, 0, 1, 1, 0, 1, 3, 1, 0, 0, 0, 0},
        {0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 1, 1, 3, 1, 0, 0, 0, 0, 0},
        {0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0},
        {0, 4, 1, 1, 0, 0, 1, 0, 5, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    };
    int Solve[MAZE_SIZE][MAZE_SIZE] = {};

    SetConsoleSize(140, X_AXIS);
    Clear();
    drawMaze(Maze);
    InitSolve(Solve);
    InitPlayerState();
    CanBreak = 1;
    HaveKey = false;
    while (1) {
        drawMaze(Maze);
        InputProcess(Maze, Solve);
        if (MazeEscape(Maze)) break;
    }
}
```

Maze Settings

Visualization maze



```
//키보드 입력
void InputProcess(int Maze[MAZE_SIZE][MAZE_SIZE], int Solve[MAZE_SIZE][MAZE_SIZE]) {
    stPlayer.x = stPlayer.x;
    stPlayer.y = stPlayer.y;

    if (GetAsyncKeyState(VK_SPACE) & 0x8000) { //한트 사용
        position HstPlayer;
        HstPlayer.x = stPlayer.x;
        HstPlayer.y = stPlayer.y;
        Hint(Maze, Solve, 1, HstPlayer);
        //PrintSolve(Solve);
    }

    if (GetAsyncKeyState(VK_LCONTROL) & 0x8000) { //벽 부수기
        BreakWall(Maze);
    }

    if (GetAsyncKeyState(VK_LSHIFT) & 0x8000) { //디버깅용 열쇠 획득키
        HaveKey = true;
    }

    if (GetAsyncKeyState(VK_LEFT) & 0x8000) { //왼쪽 키 입력
        stPlayer.x -= 2;
        if (!isValid(Maze)) stPlayer.x += 2;
    }

    if (GetAsyncKeyState(VK_RIGHT) & 0x8000) { //오른쪽 키 입력
        stPlayer.x += 2;
        if (!isValid(Maze)) stPlayer.x -= 2;
    }

    if (GetAsyncKeyState(VK_UP) & 0x8000) { //위쪽 키 입력
        stPlayer.y -= 1;
        if (!isValid(Maze)) stPlayer.y += 1;
    }

    if (GetAsyncKeyState(VK_DOWN) & 0x8000) { //아래쪽 키 입력
        stPlayer.y += 1;
        if (!isValid(Maze)) stPlayer.y -= 1;
    }

    GotoXY(0, 0);
    printf("현재 위치 : %d, %d", int(stPlayer.x - X_AXIS) / 2, stPlayer.y - Y_AXIS);
    GotoXY(0, 1);
    printf("벽을 부술 수 있는 횟수 : %d", CanBreak);

    if (Maze[stPlayer.y - Y_AXIS][int(stPlayer.x - X_AXIS) / 2] == 4) {
        HaveKey = true; //열쇠 획득한 경우
        GotoXY(0, 2);
        if (HaveKey) printf("현재 열쇠를 보유 중입니다.\n");
    }

    if (CanBreak <= 0) {
        GotoXY(0, 4);
        printf("벽을 더 이상 부술 수 없습니다!");
    }

    GotoXY(stPlayer.x, stPlayer.y);
    if (MazeEscape(Maze)) return;
    if (!isValid(Maze)) Player(Maze);
    Sleep(50);
}
```

} Click on the space bar

} Click on Left Ctrl Key

} Orientation key input setting

} Describe the current maze search situation

```
//벽 부수기
void BreakWall(int Maze[MAZE_SIZE][MAZE_SIZE]) {
    GotoXY(0, 4);
    bool flag = false;
    if [CanBreak > 0] {
        for (int i = 0; i < 4; i++) {
            int nx = stPlayer.x + 2 * dx[i];
            int ny = stPlayer.y + dy[i];
            if (Maze[ny - Y_AXIS][int(nx - X_AXIS) / 2] == 3 || Maze[ny - Y_AXIS][int(nx - X_AXIS) / 2] == 6) {
                Maze[ny - Y_AXIS][int(nx - X_AXIS) / 2] = 1;
                flag = true;
            }
        }
        if (flag) CanBreak--;
    }
    //벽이 아닌 곳을 부수라고 했을 경우 -
    if (CanBreak > 0 && flag == false) {
        printf("부수기 실패");
        GotoXY(0, 5);
        printf("부술 수 없는 벽입니다!");
        Sleep(50);
        GotoXY(0, 5);
        printf("부수기 실패");
    }
    return;
}
```

**CanBreak** represents the number of walls that can be broken.

```
//미로 탈출 여부
bool MazeEscape(int Maze[MAZE_SIZE][MAZE_SIZE]) {
    if (Maze[stPlayer.y - Y_AXIS][int(stPlayer.x - X_AXIS) / 2] == 5 && HaveKey) {
        return true; //열쇠를 가지고 탈출한 경우
    }
    return false;
```

If you come to escape without eating the key, you cannot escape

Decrease the **CanBreak** value by 1 when the **flag** variable is set to true. once you break the wall, the **CanBreak** value decreases.

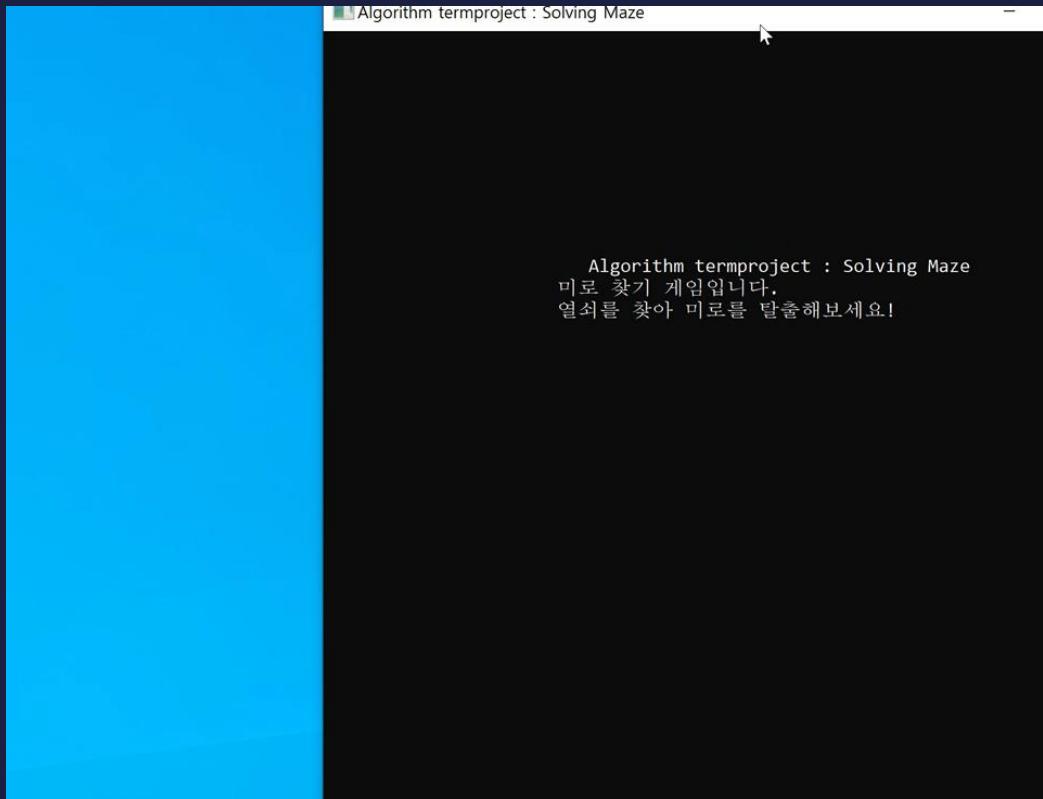
# Conclusion

Play a demonstration video

04

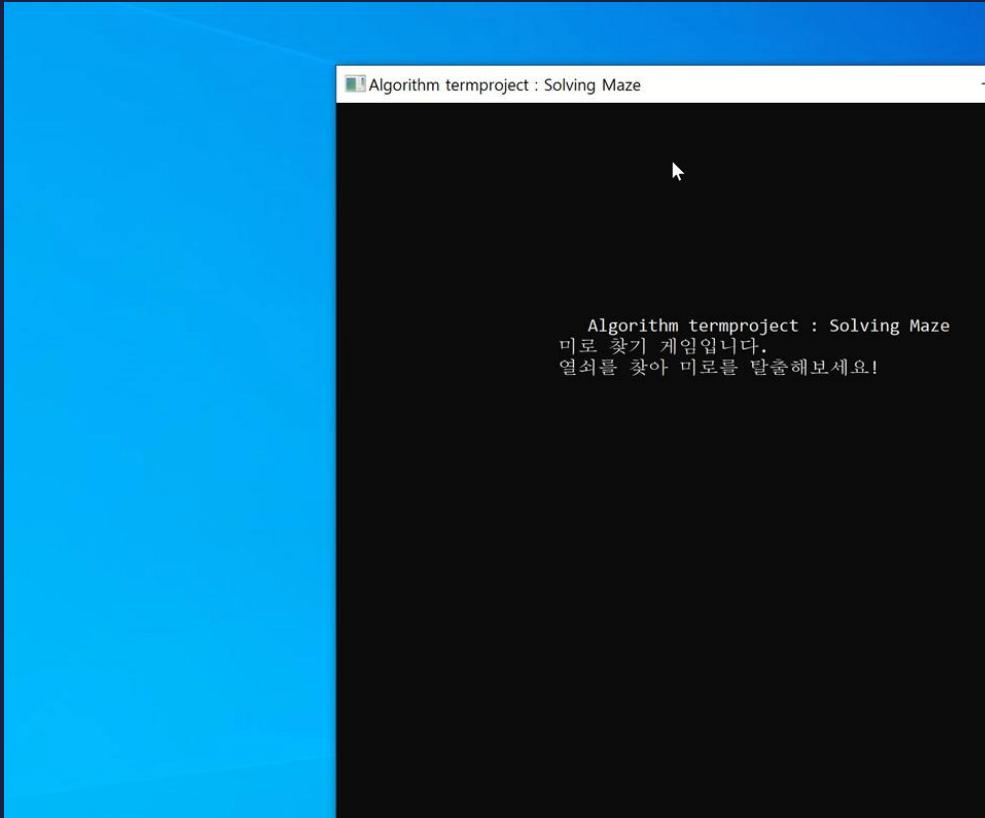


# Game Debugging Video





# Game Execution Video



# Realization of TermProject



It was a meaningful time for me to learn more about how algorithms work in the process of coding because I was happy to use the knowledge gained through active learning to create results

**KangGiwon**

I think it was a good opportunity to learn more about backtracking by continuously advancing Maze solve, which started with Active Learning, to this Term project.



**KimYoohyun**

# Realization of TermProject

It seems that we have learned more about this topic while thinking about what algorithms are needed for the project we want to proceed with.

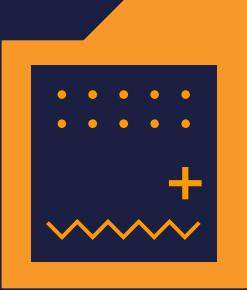


**KimHyejeong**



**JeonYoungming**

Following Active Learning, it was an opportunity to better understand backtracking through the process of deepening the Mazesolve problem by adding constraints such as breaking down walls or allowing escape only after obtaining keys.



# Thank You

From Team 8

