프로젝트 최종보고서

프로젝트 Puzzle

120210198 윤 동 성

목차

1. 프로젝트 개요

1.1 문제 제기 및 설계 목표

이 프로젝트에서, 요구하는 사항은 다음과 같다.

1) 각 함수를 완성하여 숫자 퍼즐을 구현한다.

2) 프로그램이 동작할수 있도록 main문을 구현. (Life를 고려한 프로그램 종료조건 등)

다만, 나는 이것만으로는 조금 부족하다고 여겨 숫자퍼즐의 해법 또한 출력될수 있도록 구현하고자 했다.

1.2 제한조건

각 함수의 제약조건이 주어져있다.

구현해야 할 함수와 제약조건은 다음과 같다.

- checkPuzzle(Puzzle):
  input : (list) 퍼즐게임판
  return: none

- find_0_loc(puzzle):
  input: (list) 퍼즐게임판
  return: (int)퍼즐의 빈칸의 위치

- move_to(puzzle,loc,command):
  input: (list) 퍼즐게임판
  (int) 퍼즐게임판에서 빈칸의 위치
  (int) 입력받은 key에 대한 command
  return: (int) command에 대한 동작 수행여부

2. 요구사항

2.1 설계 목표 설정

각 함수의 제약 조건을 지키면서 완성하는 것을 목표로 했으며,

추가로 구현하고자 하는 솔루션의 경우 숫자퍼즐이 섞이는 과정이 랜덤값을 이용해 퍼즐을 섞는 것이므로, 이를 역으로 저장하는 것을 구현한다.

2.2 제작

2.2.1 주요 함수 설명

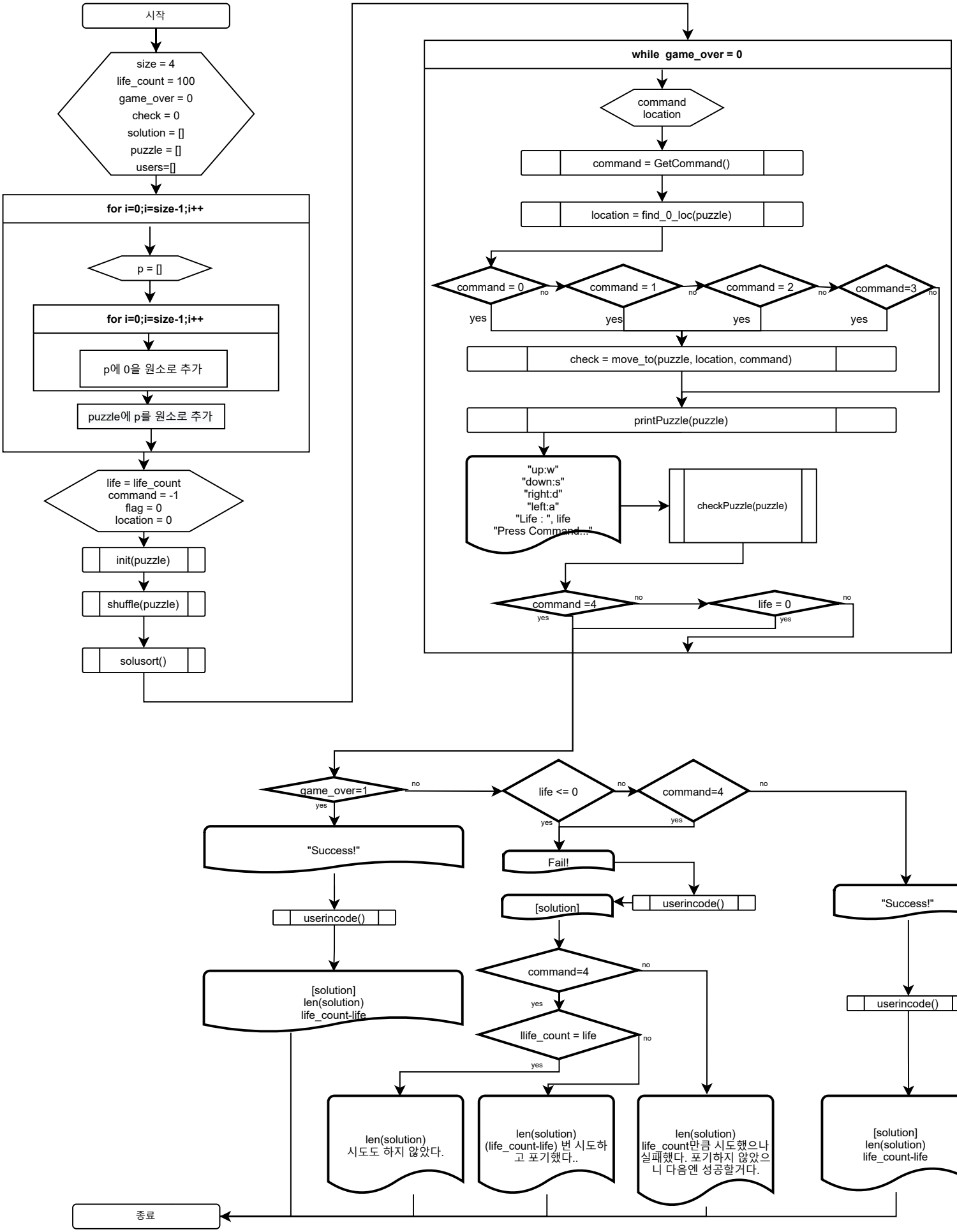
함수이름	설명
userincode	정상적으로 입력된 유저의 입력값을 리스트에 추가해준뒤 영문으로 매핑한다.
	input: none
	output: none
incode	solution 리스트를 역순으로 재배열해준뒤, 저장된 숫자값을 str로 매핑해준다.
	Input: none
	Output: solution(list)
solusort	solution 리스트에 저장된 값중 의미없는 값을 제거한다.
	input: none
	output: none
Test_one	퍼즐의 테스트를 위한 함수이다
	input: list
	output: list
Test_two	퍼즐의 테스트를 위한 함수이다
	input: list
	output: list
Test_three	퍼즐의 테스트를 위한 함수이다
	input: list
	output: list
init	퍼즐게임의 global변수와 게임판을 초기화한다 빈칸의 경우 0으로 초기화해준다
	input: list

	output: none
printPuzzle	퍼즐게임판을 출력해준다.
	input: (list) 퍼즐게임판
	ourput: none
checkPuzzle	퍼즐게임이 끝났는지를 체크한다.
	input: (list) 퍼즐게임판
	ourput: none
find_0_loc	퍼즐게임판에서 빈칸의 위치를 찾는다.
	input: (list) 퍼즐게임판
	ourput: (int)퍼즐의 빈칸의 위치
getch	콘솔창에 입력하는 key의 값을 받아온다.(Enter 필요없음)
	input: none
	ourput: (str) 입력받은 key
move_to	command에 대한 동작을 수행한다. 'w' 'W' - 빈칸을 아래로, 'a' 'A' - 빈칸을 오른쪽으로 's' 'S' - 빈칸을 위로, 'd' 'D' - 빈칸을 왼쪽으로 'q' 'Q' - 게임 종료
	input : (list) 퍼즐게임판 (int) 퍼즐게임판에서 빈칸의 위치 (int) 입력받은 key에 대한 command
	ourput: (int) command에 대한 동작 수행여부
shuffle	퍼즐게임판을 랜덤하게 섞는다
	input: (list) 퍼즐게임판
	ourput: none
GetCommand	getch()함수로 입력받은 key를 가져와 해당하는 key 값을 return 해준다. 'w', 'W', 'a', 'A', 's', 'S', 'd', 'D' key를 제외한 다른 key가 입력으로 들어올시 -1을 return 한다.
	input : none
	ourput: (int) getch()로 입력받은 key의 값
main	퍼즐게임 시작시 실행되며, 퍼즐게임판을 초기화하고, 입력받은 command에 대한 동작을 수행한다. life가 0이 되거나, 게임을 clear했을 시 프로그램을 종료한다.
	input : none
	return: 0

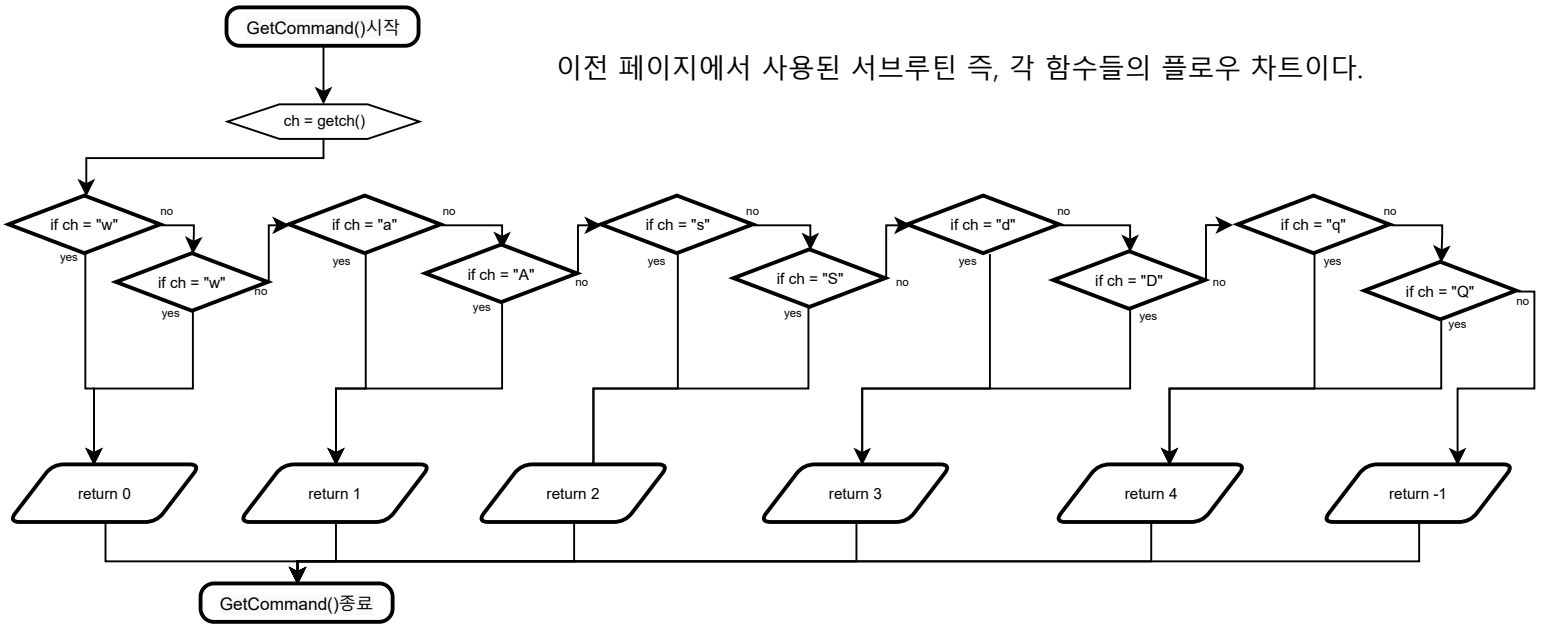
[표1] 주요함수

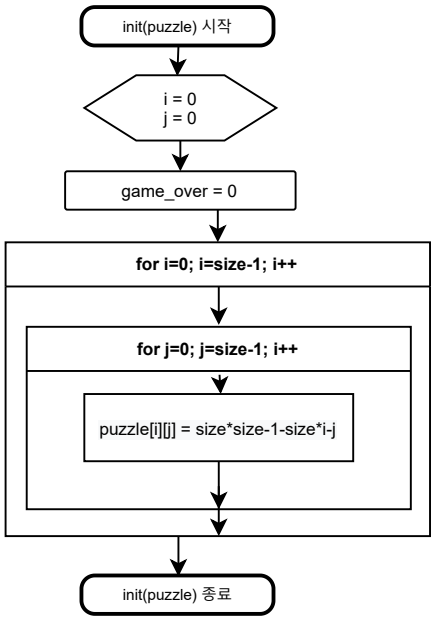
2.2.2 알고리즘 플로우 차트

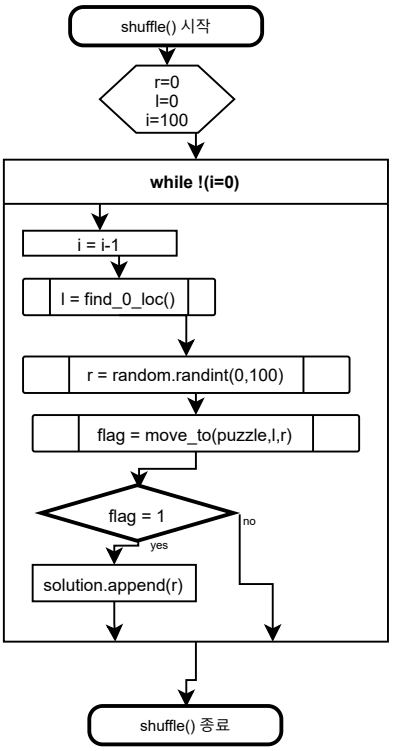
프로그램의 메인 흐름은 당 페이지의 순서도를 따른다.
흐름에 맞게 기호를 사용하여 작성했으며, 자세한 설명은 함수와 같이 설명했다.

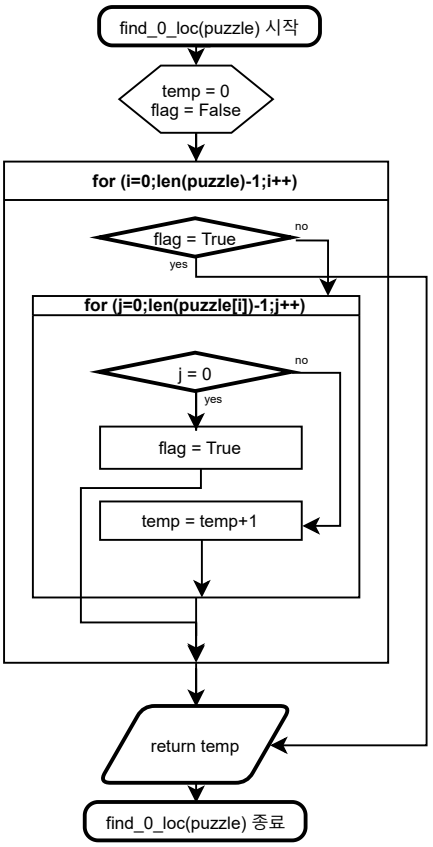


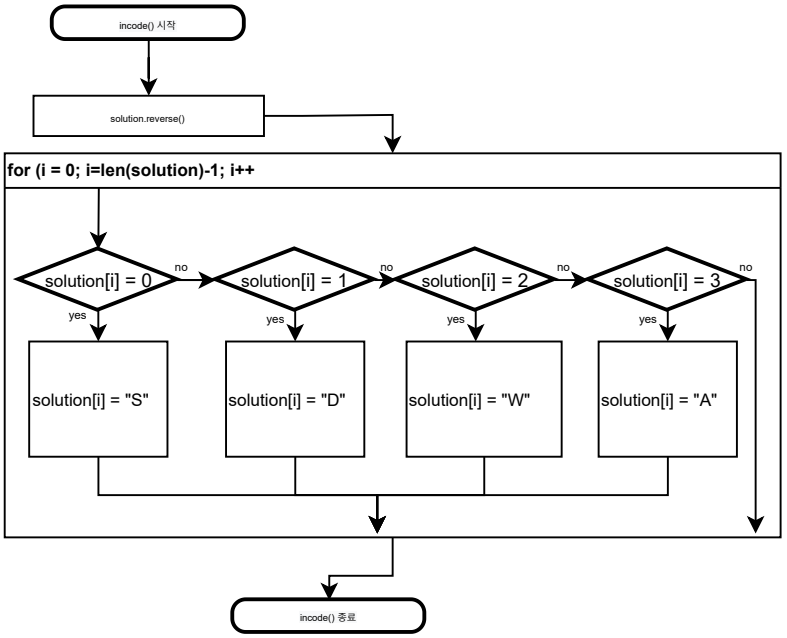
이전 페이지에서 사용된 서브루틴 즉, 각 함수들의 플로우 차트이다.

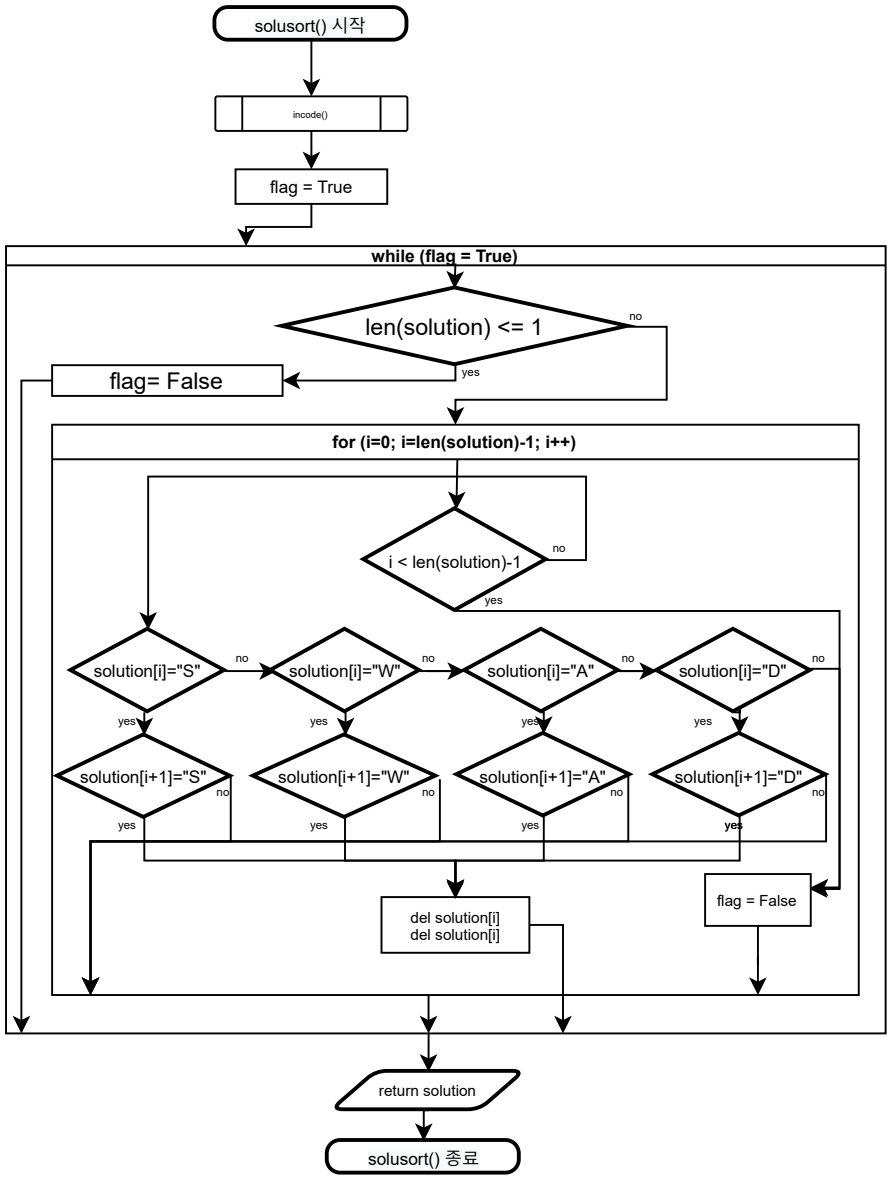


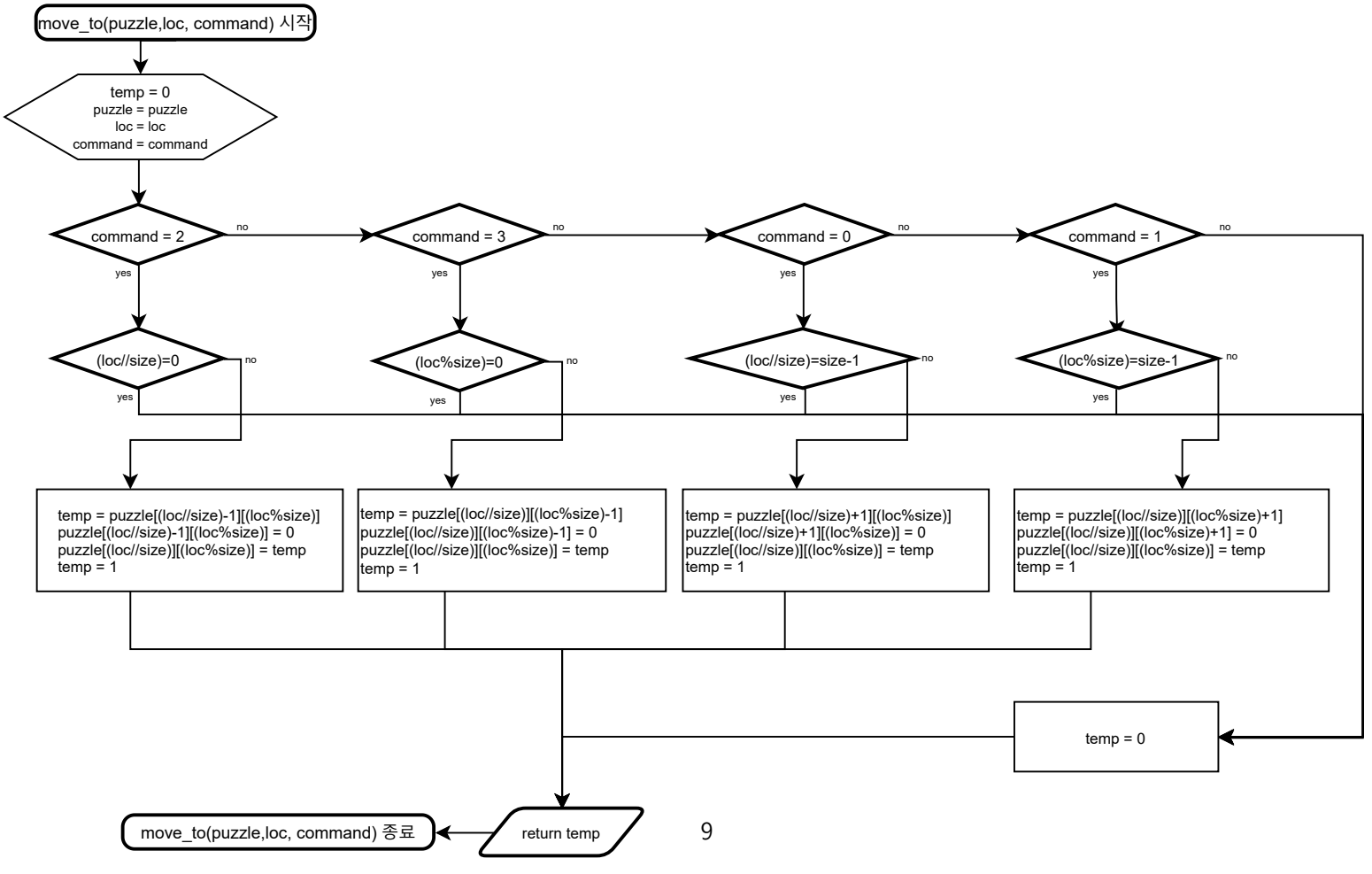


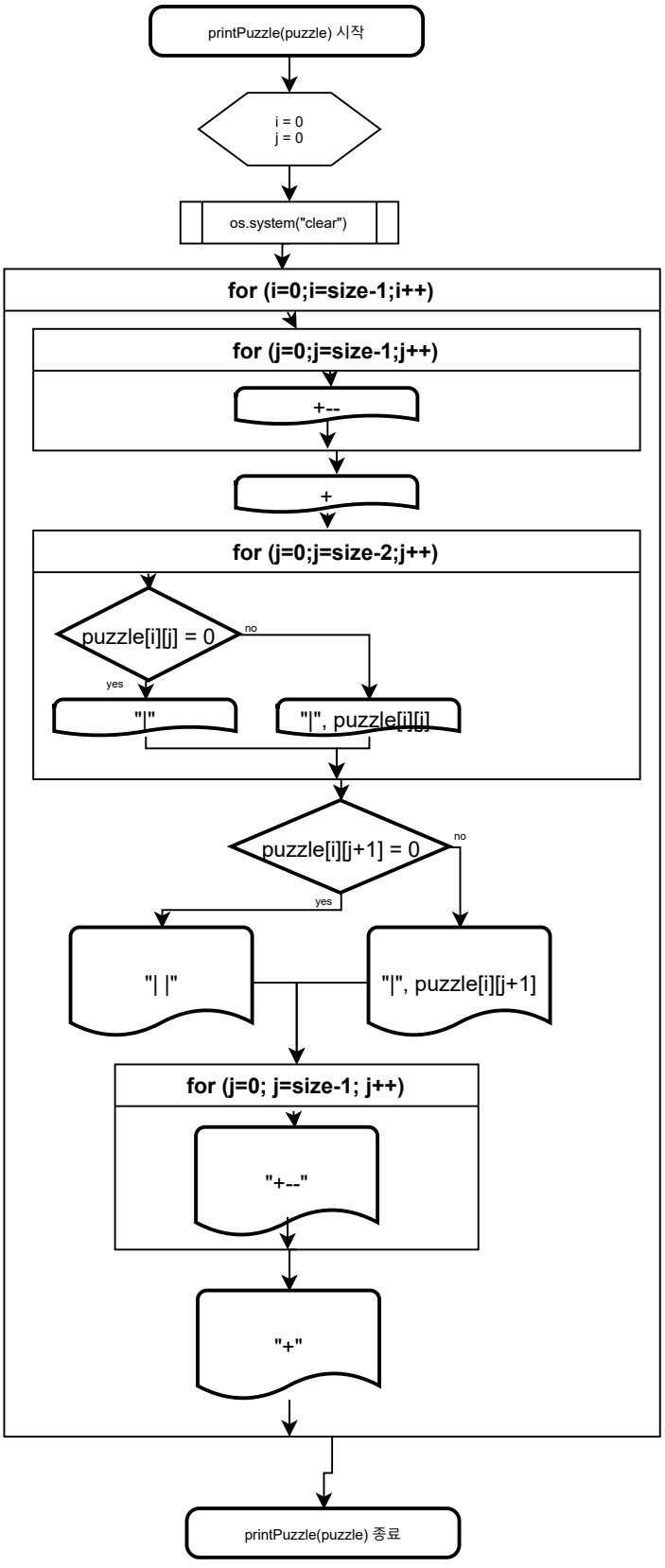


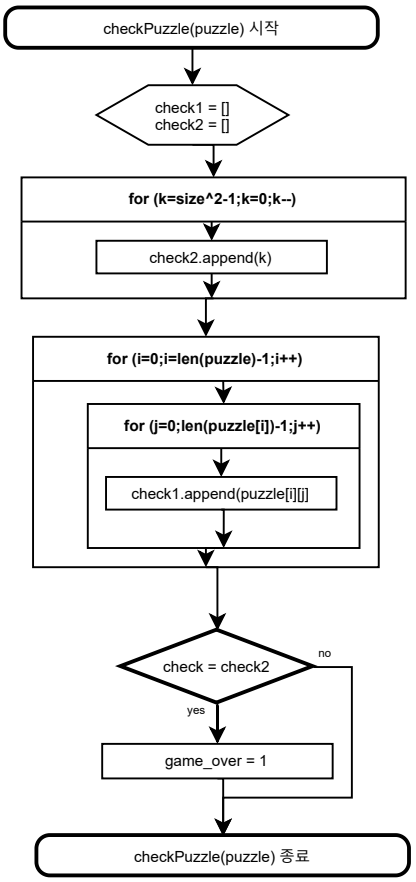


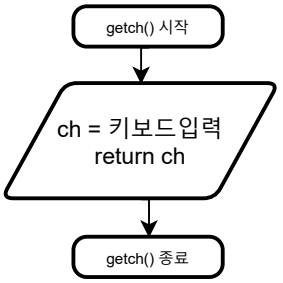


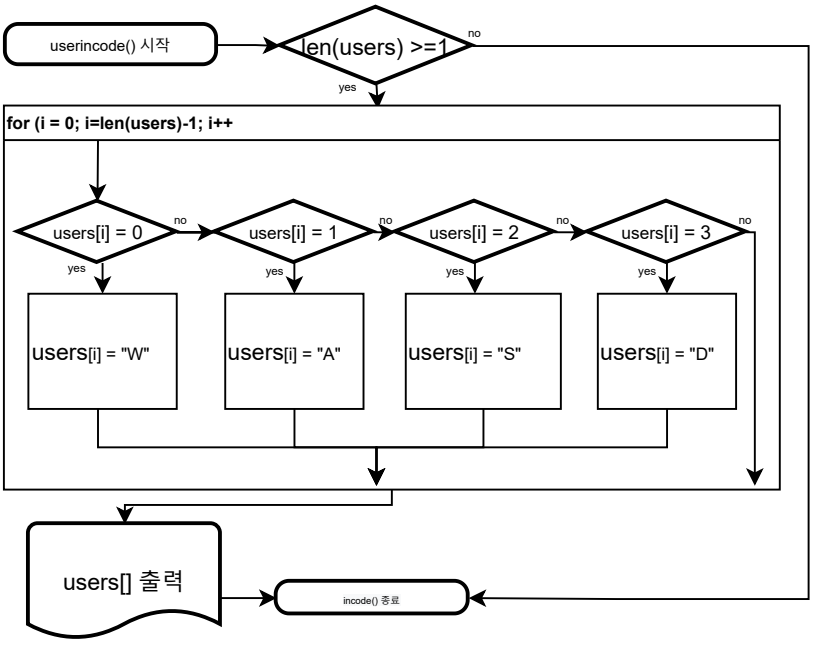

solusort() 시작
incode()
flag = True
while (flag = True)
len(solution) <= 1
no
yes
flag= False
for (i=0; i=len(solution)-1; i++)
i < len(solution)-1
no
yes
solution[i]="S"
solution[i]="W"
solution[i]="A"
solution[i]="D"
solution[i+1]="S"
solution[i+1]="W"
solution[i+1]="A"
solution[i+1]="D"
del solution[i]
del solution[i]
flag = False
return solution
solusort() 종료


move_to(puzzle,loc, command) 시작
temp = 0
puzzle = puzzle
loc = loc
command = command
command = 2
command = 3
command = 0
command = 1
(loc//size)=0
(loc%size)=0
(loc//size)=size-1
(loc%size)=size-1
temp = puzzle[(loc//size)-1][(loc%size)]
puzzle[(loc//size)-1][(loc%size)] = 0
puzzle[(loc//size)][(loc%size)] = temp
temp = 1
temp = puzzle[(loc//size)][(loc%size)-1]
puzzle[(loc//size)][(loc%size)-1] = 0
puzzle[(loc//size)][(loc%size)] = temp
temp = 1
temp = puzzle[(loc//size)+1][(loc%size)]
puzzle[(loc//size)+1][(loc%size)] = 0
puzzle[(loc//size)][(loc%size)] = temp
temp = 1
temp = puzzle[(loc//size)][(loc%size)+1]
puzzle[(loc//size)][(loc%size)+1] = 0
puzzle[(loc//size)][(loc%size)] = temp
temp = 1
temp = 0
return temp
move_to(puzzle,loc, command) 종료


printPuzzle(puzzle) 시작
i = 0
j = 0
os.system("clear")
for (i=0;i=size-1;i++)
for (j=0;j=size-1;j++)
+--
+
for (j=0;j=size-2;j++)
puzzle[i][j] = 0
no
yes
"|"
"|", puzzle[i][j]
puzzle[i][j+1] = 0
no
yes
"| |"
"|", puzzle[i][j+1]
for (j=0; j=size-1; j++)
"+--"
"+"
printPuzzle(puzzle) 종료


checkPuzzle(puzzle) 시작
check1 = []
check2 = []
for (k=size^2-1;k=0;k--)
check2.append(k)
for (i=0;i=len(puzzle)-1;i++)
for (j=0;len(puzzle[i])-1;j++)
check1.append(puzzle[i][j]
check = check2
no
yes
game_over = 1
checkPuzzle(puzzle) 종료


getch() 시작
ch = 키보드입력
return ch
getch() 종료


userincode() 시작
len(users) >=1
no
yes
for (i = 0; i=len(users)-1; i++
users[i] = 0
users[i] = 1
users[i] = 2
users[i] = 3
yes
no
users[i] = "W"
users[i] = "A"
users[i] = "S"
users[i] = "D"
users[] 출력
incode() 종료

코드 설명

전역변수

먼저 초기에 전역 변수로 설정된 변수는

```
size=4
life_count=100
game_over=0
check=0
```

[코드1] 전역변수

이며,

여기서 size는 게임판의 크기, lifecount는 맞추기 위한 제한 횟수,

Game_over=0은 퍼즐이 다 맞춰졌는지를 체크하는 flag이다.

Check는 누른 키가 올바른 키인지 체크하는 변수

```
solution = []
users = []
```

[코드2] 추가 전역변수

또한, 나는 솔루션을 비교하기위해 솔루션, users이라는 리스트를 추가했다.

이후의 문장은 함수 정의와 메인에 해당하는 함수로 구성되어있으며,

메인 함수를 따라가며 설명하도록 하겠다.

라이프는 초기 선언한 life_count로 결정되며,

Command는 -1으로 초기화되어있다.

메인

```
puzzle = []
for i in range(size):                # 크기가 size * size인 퍼즐게임판
    p = [0 for i in range(size)]
    puzzle.append(p)

life = life_count                    # 게임 생명 카운트
command = -1
flag=0
```

```
location=0
random.seed(time.time())            # 난수를 생성하기 위한 설정
init(puzzle)                        #  퍼즐게임판 초기화
shuffle(puzzle)                     # 퍼즐게임판 섞기
solusort()
```

[코드3] 메인함수

puzzle이라는 퍼즐 리스르를 먼저 생성한 뒤, size에 맞게 0을 채운다.

그리고 초기에 설정한 life_count를 life로 받고, command값을 -1로 초기화한뒤

flag, location또한 0으로 초기화, random.seed라는 난수를 생성하기 위한 코드를 추가해준뒤

init 함수로 넘어간다.

init함수

init함수는 다음과 같다.

```
def init(puzzle):
    i=0
    j=0
    game_over = 0
    for i in range(size):
        for j in range(size):
            puzzle[i][j] = size*size-1-size*i-j;
```

[코드4] init 함수

살펴보자면, 메인에서 0으로 채워진 퍼즐 리스트의 길이에 맞춰 숫자를 size^2-1~0까지 채워주는 함수이다.

shuffle함수

다음으로 shuffle 함수를 먼저 살펴보면

```
def shuffle(puzzle):
    r=0
    l=0
    i = 100
    while i:
        i=i-1
        l = find_0_loc(puzzle)
        r = random.randint(0,100)%4
        #TODO
        flag = move_to(puzzle,l,r)
        if flag ==1:
```

```
            solution.append(r)
```

*여기서 TODO 이하는, 솔루션을 만들기 위해 추가한 것으로 아래에서 추가설명하도록 하겠다.

find_0_loc함수

l변수는 find_0_loc(puzzle)을 통해 값을 받아야 하기에 find_0_loc함수를 먼저 작성했다.

초기에는, 리턴값이 없다고 제약조건을 받았으나, 방법이 없어서 추후 공지가 나오기 전까지, 임시로 loc라는 리스트를 생성후 이 리스트를 수정하는 방식을 택했다.

```
def find_0_loc(puzzle):

#TODO
    loc = []
    temp1 = 0
    for i in puzzle:

        if 0 in i:
            loc.append(temp1)
            loc.append(i.index(0))

            break;
        temp1 = temp1 + 1
```

이후, int로 리턴받는다는 제약조건이 추가된 뒤에, 이게 몇 번째에 있는지를 리턴받는 방식으로 수정했다.

```
    temp = 0
    flag = False
    for i in puzzle:
        if flag == True:
            break;
        for j in i:
            if j == 0:
                flag = True
                break;
            else: temp = temp + 1


    return temp
```

이제, 다시 셔플 함수로 돌아와보면,

```
def shuffle(puzzle):
    r=0
    l=0
    i = 100
    while i:
        i=i-1
        l = find_0_loc(puzzle)
        r = random.randint(0,100)%4
        #TODO
        flag = move_to(puzzle,l,r)
        if flag ==1:
            solution.append(r)
```

move_to 함수

다음 실행 순서는 move_to 라는 함수이다.

이 move_to 라는 함수는 말그대로 입력을 받고 위치를 옮겨주는 함수로써, 셔플 함수 내에서는 100번의 랜덤값을 입력으로 받는다.

퍼즐, 0의 위치, command를 받아 0의 위치를 옮겨주는 함수인데,

여기서 위, 아래의 움직임을 리스트를 통해 행/열로 받았으면 좋았겠지만, int값으로 받았기에,

인덱스를 size로 나눈뒤 몫과 나머지를 이용하면, 예를들어 4x4 퍼즐일경우 16번째에 0이 있을 때, 인덱스는 0이 되므로, 몫은 3, 나머지는 3으로 puzzle[0][3]을 정확하게 가리키는 것을 이용했다.

그리고, 예외처리도 추가해야 했는데, 더 이상 왼쪽으로 옮길수 없을 때, 또는 잘못된 값을 입력받았을때에 라이프가 줄어들면 안되기에, 이 경우 if-else를 통해 예외처리를 했다.

```
def move_to(puzzle,loc, command):#////////////HERE

#TODO
    temp = 0

    global size
    if command == 2:
        if (loc//size) == 0:
            temp = 0
        else:
            temp = puzzle[(loc//size)-1][(loc%size)]
            puzzle[(loc//size)-1][(loc%size)] = 0
            puzzle[(loc//size)][(loc%size)] = temp
            temp = 1

    elif command == 3:
        if (loc%size) == 0:
```

```
            temp = 0
        else:
            temp = puzzle[(loc//size)][(loc%size)-1]
            puzzle[(loc//size)][(loc%size)-1] = 0
            puzzle[(loc//size)][(loc%size)] = temp
            temp = 1
    elif command == 0:
        if (loc//size) == size-1:
            temp = 0
        else:
            temp = puzzle[(loc//size)+1][(loc%size)]
            puzzle[(loc//size)+1][(loc%size)] = 0
            puzzle[(loc//size)][(loc%size)] = temp
            temp = 1
    elif command == 1:
        if (loc%size) == size-1:
            temp = 0
        else:
            temp = puzzle[(loc//size)][(loc%size)+1]
            puzzle[(loc//size)][(loc%size)+1] = 0
            puzzle[(loc//size)][(loc%size)] = temp
            temp = 1
    else:
        temp = 0
    return temp
```

이제 셔플 함수까지가 완성이 되었으며, 다시 메인으로 돌아오면

solusort 함수

```
solusort()
```

라는 함수가 있다.

이 함수의 경우 내가 솔루션을 만들어보고자 추가한 함수이다.

먼저 이 함수에 쓰이는 리스트 solution은 셔플 함수에 의해서만 추가되며,

이때, 셔플 함수도 마찬가지로, 왼쪽으로 갈수 없지만 왼쪽으로 가는 등의 입력을 받을수 있기에, 실제로 퍼즐에서 0의 이동이 일어났을때만 그 순서대로 이동한 값을 solution리스트에 추가해줬다.

incode() 함수

또한 solusort()에서 쓰이는 incode() 함수의 경우 셔플에서 일어난 상황이 int로 기록되기에, 이를

실질적으로 WASD로 바꾸기 위한 함수이며, 이때, 셔플에서 0을 W-A-S-D로 움직였다면, 솔루션은 그 역순으로 반대로 움직여주면 되므로, A-W-D-S일 것이다. 따라서, 먼저 순서를 역순으로 바꿔준뒤, 다시 이를 반대로 움직이기 위해서는 어떤 키가 입력되야 하는가를 볼 수 있도록 값을 수정해 주었다.

```
def incode():
    solution.reverse()
    for i in range(len(solution)):
        if solution[i] == 0 :
            solution[i] = "S"
        elif solution[i] == 1:
            solution[i] = "D"
        elif solution[i] == 2:
            solution[i] = "W"
        elif solution[i] == 3:
            solution[i] = "A"
    return solution
```

그리고 solusort에서 일어나는 작업은 필요없는 입력 부분을 제거하는 것인데,

예를들어 S-W라면 0은 아래로 갔다가 위로 올라간다. 즉 제자리라는 것이다.

그렇기에, S-W, A-D, W-S, D-S와 같은 의미없는 입력을 제거하는 반복문을 추가했다.

이때 W-W-W-S-S-S와 같은 입력이 있을 수도 있으므로, while문 안에서 의미없는 입력이 없을때까지 반복해서 제거하도록 작성했다.

```
def solusort():
    incode()
    flag = True
    while flag:
        if len(solution) <= 1:
            break;
        for i in range(len(solution)):
            if i < len(solution)-1 and solution[i] == "S":
                if solution[i+1] == "W":
                    del solution[i]
                    del solution[i]
                    break;
            elif i < len(solution)-1 and solution[i] == "W":
                if solution[i+1] == "S":
                    del solution[i]
                    del solution[i]
                    break;
            elif i < len(solution)-1 and solution[i] == "A":
```

```
                if solution[i+1] == "D":
                    del solution[i]
                    del solution[i]
                    break;
            elif i < len(solution)-1 and solution[i] == "D":
                if solution[i+1] == "A":
                    del solution[i]
                    del solution[i]
                    break;
            else:
                flag = False
```

main(while)

이제, 다음 메인문은 게임이 진행되는 while문이다.

여기서 while문은 game_over이라는 전역변수가 1으로 바뀔때까지 반복되며,

loc함수로 먼저 0의 위치를 가져온뒤, 0123이라는 입력이 들어왔을 시에, 위에서 작성한 move_to 함수를 통해 0의 위치와 입력받은 값의 방향으로 인근한 숫자와의 위치를 바꿔준다.

그리고, 0의 이동이 일어났는지 체크 후, 이동이 일어났다면 life를 빼준다.

그리고 입력시마다 퍼즐을 그려주게 되며, life를 출력하게 된다.

이때, 유저의 입력값 또한도 추후 비교를 위해 저장할수 있도록 했다.

```
while game_over != 1:
    command = GetCommand()
#TODO
#퍼즐게임판에서 빈칸을 찾는다(find_0_loc(_))
    location = find_0_loc(puzzle)
#key 에 대한 동작을 수행한다.
    if command==0 or command==1 or command==2 or command==3:
        check = move_to(puzzle, location, command)
        if check == 1:
            users.append(command)

        life = life - check
    else :
        pass;
    printPuzzle(puzzle) # 퍼즐게임판 그리기
    # print(solution) # test용 솔루션 출력
#게임 진행에 필요한 정보 출력
    print('\033[32m'+"up    : w\nleft  : a\ndown  : s\nright : d\n"+'\033[0m')
    print('\033[33mLife : %d\n\033[0m'%life)
    print('\033[31m'+"Press Command...\n"+'\033[0m')
```

```
#TODO
#퍼즐게임 종료조건 체크한다.
    checkPuzzle(puzzle)
    if command == 4:
        break;
    if life == 0:
        break;
```

GetCommand함수

먼저 GetCommand()를 살펴보면

```
def GetCommand():
    ch = getch()

    if ch =='w' or ch=='W':
        return 0
    elif ch=='a' or ch=='A':
        return 1
    elif ch=='s' or ch=='S':
        return 2
    elif ch=='d' or ch=='D':
        return 3
    elif ch=='q' or ch=='Q':
        return 4
    else:
        return -1
    return -1
```

wasd 또는 WASD를 입력받고 이 값을 숫자로 매핑시켜주는 함수이다.

printPuzzle함수

다음으로 printPuzzle은 다음과 같다.

```
def printPuzzle(puzzle):
    i=0
    j=0
    os.system("clear");
    for i in range(size):
        for j in range(size):
            print("+--",end="")
        print("+")
        for j in range(size-1):
            if puzzle[i][j] == 0:
                print("|   ",end="")
            else:
```

```
                print("|"+"\033[33m"+"%2d"%puzzle[i][j]+"\033[0m",end='')
        if(puzzle[i][j+1] == 0):
            print("|  |")
        else:
            print("|"+"\033[33m%2d\033[0m|\n"%puzzle[i][j+1],end='')
    for j in range(size) :
        print("+--",end="");
    print("+");
```

반복문을 통해 퍼즐을 양식에 맞게 출력하는 함수이다.

checkPuzzle함수

다음 함수인 checkPuzzle 함수의 경우

퍼즐이 모두 맞춰졌는가를 출력하는 함수인데,

나는 여기서 지금의 퍼즐상태와, 퍼즐이 맞춰졌을 때의 상태를 리스트로 비교하는 방법을 썼다.

먼저 맞춰졌을 때의 상태를 위해 size에 맞게, size가 4라면 15~0이 들어있는 리스트를 만들어준 뒤.

퍼즐의 경우 [[0,1,2,3],[4,5,6,7],[8,9,10,11],[12,13,14,15]]와 같은 형태로 되어있기에, 이를 리스트 안에 리스트가 없도록 모두 분리하는 작업이 필요했다.

따라서, for문을 이용해 퍼즐안에서 원소를 하나씩 추출해서 추가해줬다.

이후 맞춰졌을때의 퍼즐, 지금의 퍼즐을 비교해서 퍼즐이 모두 맞춰졌다면 전역변수 game_over을 1로 바꿔주도록 했다.

```
def checkPuzzle(puzzle) :

#TODO
    check1 = []
    check2 = []
    for k in range(size**2)-1,-1,-1):
        check2.append(k)

    for i in puzzle:
        for j in i:
            check1.append(j)
    if check1 == check2:
        global game_over
        game_over = 1
```

다시 메인으로 돌아오면, 퍼즐이 모두 맞춰졌을 경우 while문의 조건에 의해 while문을 나가게 되고, q가 눌렸을 때, 또는 life가 모두 소진되었을때도 break문으로 while문을 나가게 된다.

```
#퍼즐게임 종료조건 체크한다.
    checkPuzzle(puzzle)
    if command == 4:
        break;
    if life == 0:
        break;
```

출력

다음으로 결과 출력에 대한 부분인데,

이 부분의 경우 나는 좀 복잡하게 추가한 것이 있는 편이다.

먼저, game_over값이 True일 경우, 즉, 1일경우에 Success가 출력되게 되는데,

나는 여기에 앞서 만든 solution을 이용해서 해답을 출력할 수 있도록 만들었다.

userincode함수

userincode 또한 내가 추가해준 함수인데, 추후 솔루션과의 비교를 위해 사용자가 입력한 값을 다시 영문자로 바꿔서 저장, 출력해주는 함수이다.

```
def userincode():
    if len(users) >= 1:
        for i in range(len(users)):
            if users[i] == 0 :
                users[i] = "W"
            elif users[i] == 1:
                users[i] = "A"
            elif users[i] == 2:
                users[i] = "S"
            elif users[i] == 3:
                users[i] = "D"
        print("Your input is..")
        print("[",end="")
        for i in range(len(users)):
            if i == (len(users)-1):
                print(users[i], end="")
                if len(users)%5 == 0:
                    print("]")
```

```
                else : print("--"*(5-(len(users)%5))+"]")
            elif (i+1)%5 == 0:
                print(users[i]+"]\n[", end="")
            else :
                print(users[i], end="-")
    else: pass
```

그리고, 해답에서의 try 횟수와 내가 시도한 횟수를 비교해 줄 수 있도록 작성하였다.

```
"""퍼즐게임 결과 출력"""
print("\n\n\n\n\n")
if game_over:
    print("\033[31m"+"      Success!\n\n\n"+"\033[0m")
    userincode()
    print("Standard Solution is:\n[",end="")
    for i in range(len(solution)):
        if i == (len(solution)-1):
            print(solution[i], end="")
            if len(solution)%5 == 0:
                print("]")
            else : print("--"*(5-(len(solution)%5))+"]")
        elif (i+1)%5 == 0:
            print(solution[i]+"]\n[", end="")
        else :
            print(solution[i], end="-")
    print("The standard solution is possible with '%d' attempts, but you have
tried '%d' times." %(len(solution), (life_count-life)))
```

그리고 퍼즐이 모두 맞춰지지 않았을 경우, 분기를 두번 하도록 했다.

먼저, for문의 경우 한줄에 5개씩 해답을 출력하는 문장이며,

해답을 출력 후에도 몇가지로 나눴는데,

1. Q 버튼을 이용해 종료한 경우

    1-1 한번도 시도하지 않은 경우

    1-2 한번이상 시도한 경우

2. 라이프를 모두 소진했기에, 게임이 종료된 경우

로 나눴으며,

모두 솔루션을 먼저 출력 뒤에 각각에 맞는 메시지가 출력되도록 했다.

이때, 마지막 부분에 있는 else: Success 부분의 경우, 없어도 잘 동작하는 것을 확인했으나, 혹시모를 예외처리를 위해 유지하기로 결정했다.

```
else:
    if life <= 0 or command == 4:
        print("\033[31m"+"      Fail!\n\n\n"+"\033[0m")
        userincode()
        print("Standard Solution is:\n[",end="")
        for i in range(len(solution)):
            if i == (len(solution)-1):
                print(solution[i], end="")
                if len(solution)%5 == 0:
                    print("]")
                else : print("--"*(5-(len(solution)%5))+"]")
            elif (i+1)%5 == 0:
                print(solution[i]+"]\n[", end="")
            else :
                print(solution[i], end="-")
        if command==4:
            if life_count == life:
                print("The standard solution is possible with '%d' attempts, b
ut you haven't even tried." %len(solution))

            else:
                print("The standard solution is possible with '%d' attempts, b
ut you gave up on '%d' attempts." %(len(solution), (life_count-life)))

        else :
            print("The standard solution is possible with '%d' attempts." %len
(solution))
            print("You've failed '%d' attempts, but you haven't given up, so n
ext time it'll be possible."  %life_count )
    else:
        print("\033[31m"+"      Success!\n\n\n"+"\033[0m")
        userincode()
        for i in range(len(solution)):
            if i == (len(solution)-1):
                print(solution[i], end="")
                if len(solution)%5 == 0:
                    print("]")
                else : print("--"*(5-(len(solution)%5))+"]")
            elif (i+1)%5 == 0:
                print(solution[i]+"]\n[", end="")
```

```
            else :
                print(solution[i], end="-")
        print("The standard solution is possible with '%d' attempts, but you h
ave tried '%d' times." %(len(solution), (life_count-life)))
```

test함수

이후 추가적인 작업으로 테스트 함수에서도 솔루션이 맞게 수정될 수 있도록 추가 작성했다.

```
def Test_one(puzzle): #ANS : W W A A A
    puzzle[0][0] = 15;  puzzle[0][1] = 14;  puzzle[0][2] = 13;  puzzle[0][3] =
 12;
    puzzle[1][0] = 0;   puzzle[1][1] = 10;  puzzle[1][2] = 9;   puzzle[1][3] =
 8;
    puzzle[2][0] = 11;  puzzle[2][1] = 6;   puzzle[2][2] = 5;   puzzle[2][3] =
 4;
    puzzle[3][0] = 7;   puzzle[3][1] = 3;   puzzle[3][2] = 2;   puzzle[3][3] =
 1;
    global solution
    solution.clear()
    solution = ["W","W","A","A","A"]
    return puzzle

def Test_two(puzzle):  #ANS : W A W D D W A A
    puzzle[0][0] = 15;  puzzle[0][1] = 14;  puzzle[0][2] = 0;   puzzle[0][3] =
 12;
    puzzle[1][0] = 11;  puzzle[1][1] = 10;  puzzle[1][2] = 13;  puzzle[1][3] =
 9;
    puzzle[2][0] = 7;   puzzle[2][1] = 5;   puzzle[2][2] = 4;   puzzle[2][3] =
 8;
    puzzle[3][0] = 3;   puzzle[3][1] = 6;   puzzle[3][2] = 2;   puzzle[3][3] =
 1;
    global solution
    solution.clear()
    solution = ["W", "A", "W", "D", "D", "W", "A", "A"]
    return puzzle

def Test_three(puzzle): #ANS : A S S D W A S D W A A W W A
    puzzle[0][0] = 11;  puzzle[0][1] = 15;  puzzle[0][2] = 13;  puzzle[0][3] =
 12;
    puzzle[1][0] = 14;  puzzle[1][1] = 6;   puzzle[1][2] = 10;  puzzle[1][3] =
 8;
    puzzle[2][0] = 0;   puzzle[2][1] = 7;   puzzle[2][2] = 9;   puzzle[2][3] =
 4;
```

```
    puzzle[3][0] = 3;   puzzle[3][1] = 2;   puzzle[3][2] = 5;   puzzle[3][3] =
 1;
    global solution
    solution.clear()
    solution = ["A", "S", "S", "D", "W", "A", "S", "D", "W", "A", "A", "W", "W
", "A"]
    return puzzle
```

2.3 시험

먼저 솔루션이 정상적으로 동작하는지 알아보기 위해 메인문에

```
print(solution) # test용 솔루션 출력
userincode() # test용
```

을 추가해서 테스트해본 결과

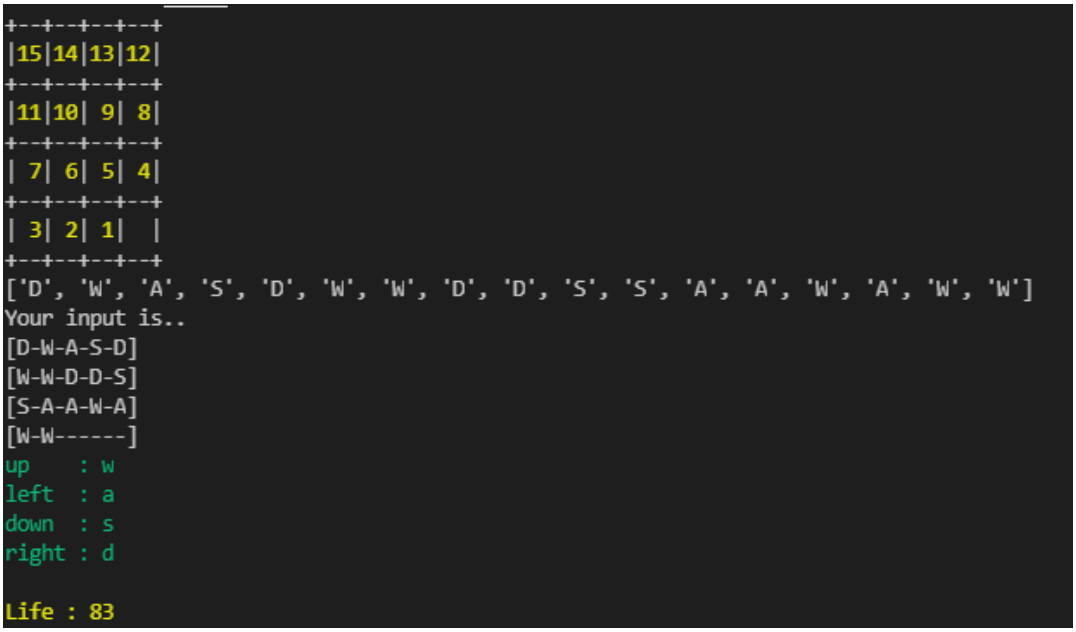


Figure 1 솔루션 테스트화면

Figure 2 success 출력

정상적으로 동작하는 것을 확인할 수 있었다.

또한 내가 경우의 수를 나눈 경우도 잘 동작하는 것을 확인했다.

Figure 3 바로 종료시 화면

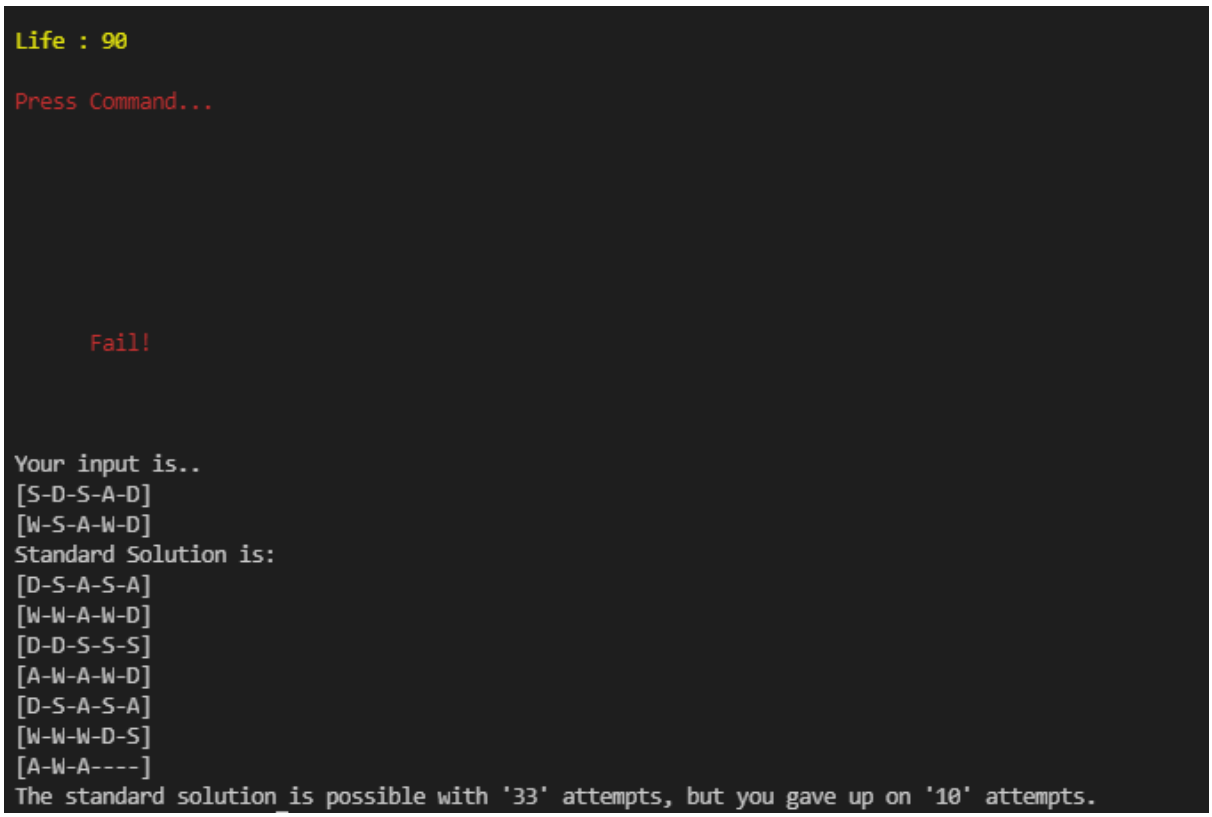


Figure 4 1번이상 시도후 종료화면

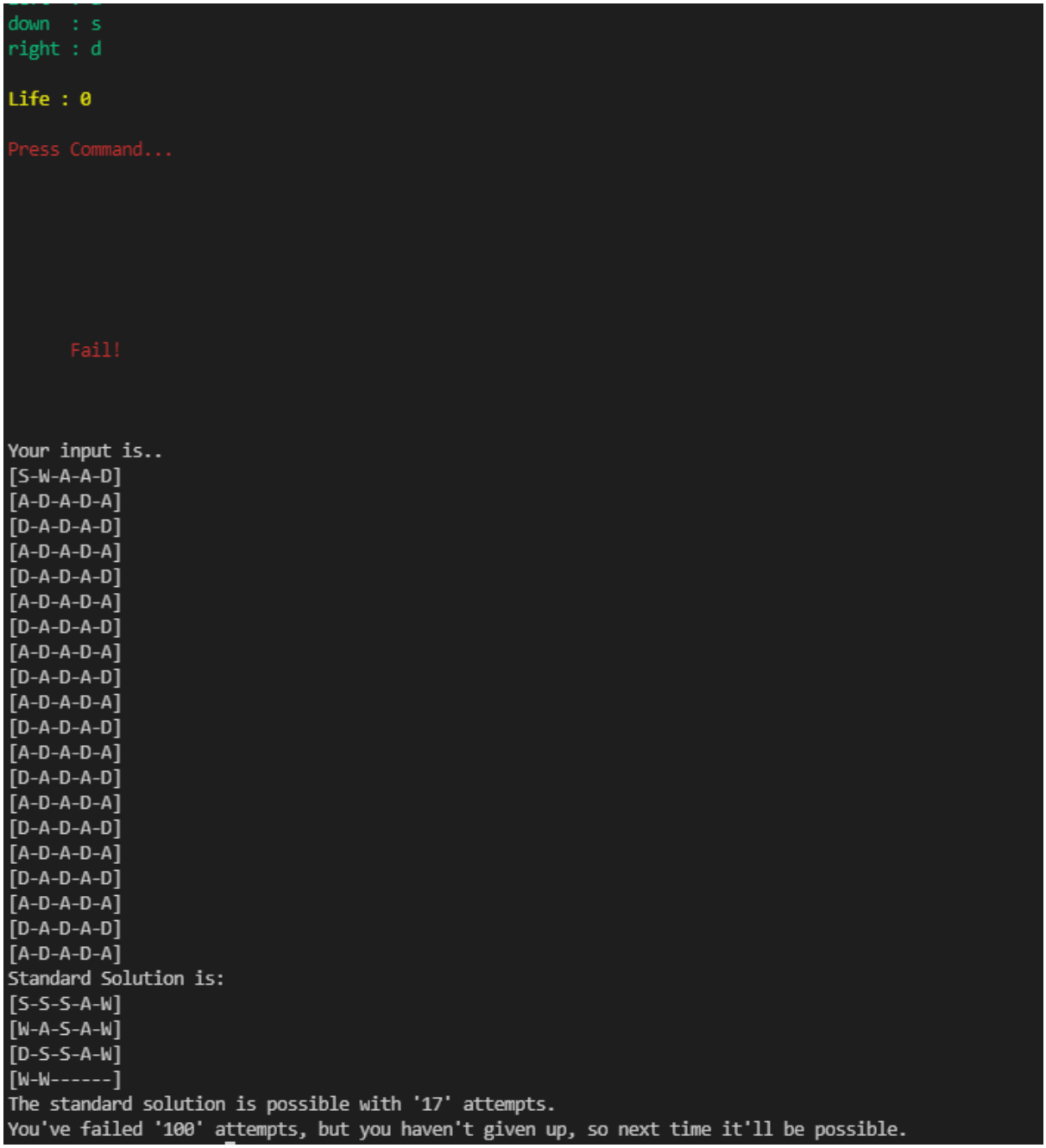


Figure 5 Life 소진으로 인한 종료화면

2.4 평가

일차원적인 솔루션을 제시하긴 했으나, 솔루션을 제대로 제시할수 있는 알고리즘을 알았다면 더 좋은 프로그램이 되었을거라는 아쉬움이 남았으며, 코드가 지저분한 감이 없지 않아, 좀더 최적화되었다면 좋지 않았을까라는 아쉬움이 남았다.

또한, 결과출력에서

```
"""퍼즐게임 결과 출력"""
print("\n\n\n\n\n")
if game_over:
    print("\033[31m"+"      Success!\n\n\n"+"\033[0m")
else:
    if life <=0 or command == 4:
        print("\033[31m"+"      Fail!\n\n\n"+"\033[0m")
    else:
        print("\033[31m"+"      Success!\n\n\n"+"\033[0m")
```

마지막 else 부분의 경우, 없어도 잘 동작하는데, 예외처리라 추정하여 유지하긴 했으나, 정확히 어떤 경우에대한 예외인지 찾질 못한 부분도 아쉬운 부분이다.

2.5 환경

cspro에 SSH로 접속, 구현

3. INDEX