Report of Number Puzzle Development

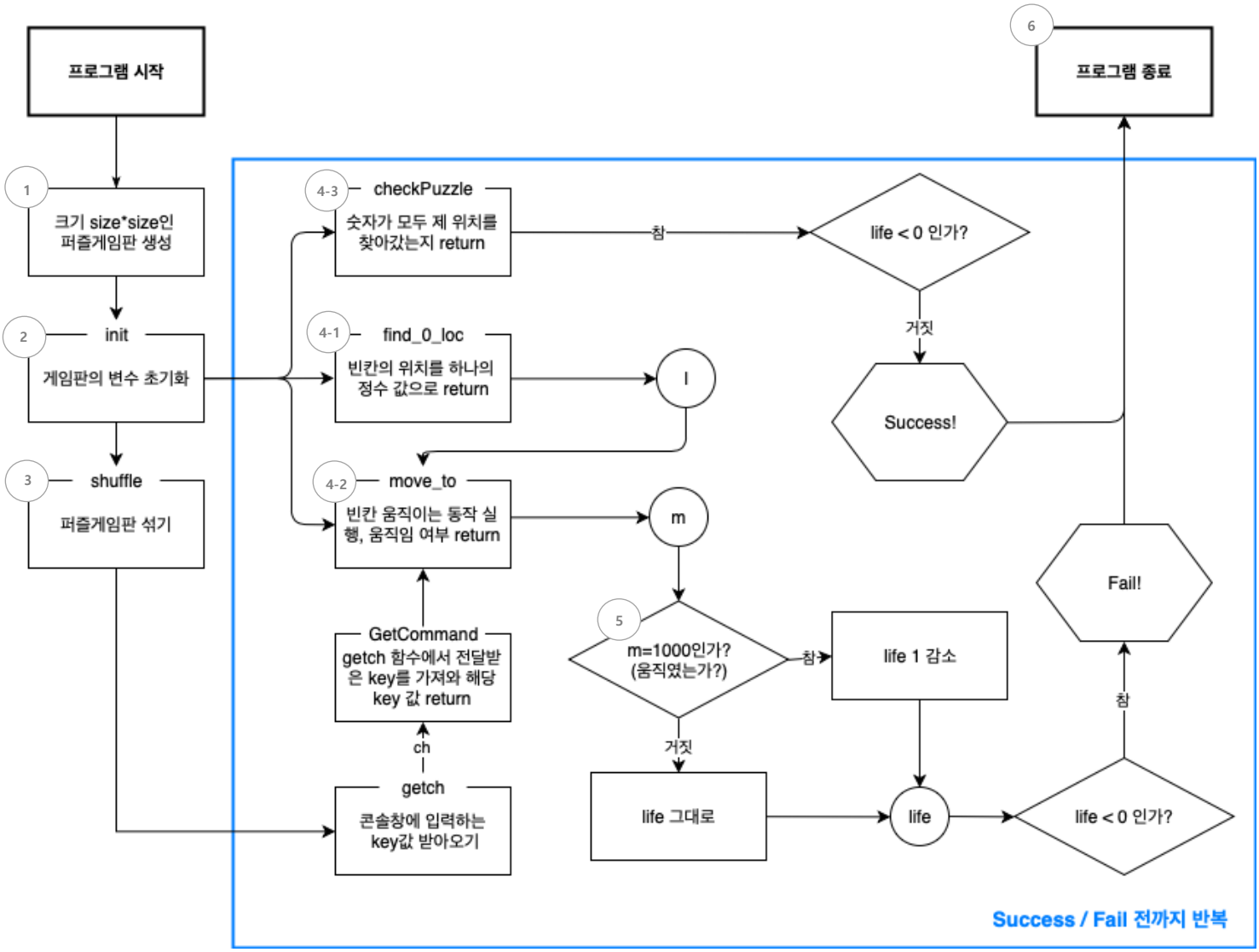
(숫자 퍼즐 설계 프로젝트 보고서)

과목: [CSE2003-03] 기초공학설계

담당교수: 서강대학교 컴퓨터공학과 김 지 환

개발자: 강상원 (20191559)

- **프로그램 실행 흐름도 및 설명**



1. main 함수에서 puzzle[size][size]인 퍼즐게임판 2차원 배열을 생성한다.

```
int main(void)
{
    int puzzle[size][size]; /* 크기가 size * size인 퍼즐게임판*/
```

이때 size의 크기는 코드 초반에 4로 정의되어 있다.

```
/* 퍼즐게임판 크기 */
#define size 4
```

2. init 함수에서 게임판 (2차원 배열)의 변수를 초기화한다.

```
void init(int puzzle[][size])
{
    int i,j;
    game_over = 0;
    for(i = 0; i < size; i++)
    {
        for(j=0; j<size; j++)
            puzzle[i][j] = size*size-1-size*i-j;
    }
}
```

3. shuffle 함수로 퍼즐게임판을 섞는다. iter=100이므로 move_to함수를 이용해 빈칸을 100번 옮긴다.

```
void shuffle(int puzzle[][size])
{
    int r,l;
    int iter = 100;     /* 필요시 횟수를 변경해 가면서 테스트 해보세요 */
    while(iter)
    {
        iter--;
        l = find_0_loc(puzzle);
        r = rand()%4;
        move_to(puzzle,l,r);
    }
}
```

4. checkPuzzle, find_0_loc, move_to 함수는 init 함수로부터 2차원 배열의 변수 값을 전달받는다.

4-1. find_0_loc 함수에서 빈칸의 위치를 하나의 정수 값으로 return한다.

```
int find_0_loc(int puzzle[][size])
{
        int i, j, loc;//TODO
        for (i=0 ; i<size ; i++)
                for (j=0 ; j<size ; j++)
                        if (puzzle[i][j]==0)
                                loc=10*i+j;
        return loc;
}
```

i \ j	0	1	2	3
0	0	1	2	3
1	10	11	12	13
2	20	21	22	23
3	30	31	32	33

4-2. move_to 함수는 find_0_loc 함수로부터 전달받은 빈칸의 위치 l과 GetCommand로부터 받은 key 값을 이용해 빈칸을 움직인다. 또한 빈칸이 모서리에 있어 움직일 수 없는 경우가 있으므로 움직임 여부를 return (1000)한다.

```
int move_to(int puzzle[][size],int loc,int command)
{
	int i, j;
	i=loc/10;
	j=loc%10;
	if (command==0)
	{
		if (i<size-1)
		{
			puzzle[i][j]=puzzle[i+1][j];
			puzzle[i+1][j]=0;
			return 1000;
		}
	}

	if (command==1)
	{
		if (j<size-1)
		{
			puzzle[i][j]=puzzle[i][j+1];
			puzzle[i][j+1]=0;
			return 1000;
		}
	}

	 if (command==2)
	 {
		if (i>0)
		{
			puzzle[i][j]=puzzle[i-1][j];
			puzzle[i-1][j]=0;
			return 1000;
		}
	 }

	 if (command==3)
	 {
		if (j>0)
		{
			puzzle[i][j]=puzzle[i][j-1];
			puzzle[i][j-1]=0;
			return 1000;
		}
	 }//TODO
}
```

4-3. checkPuzzle 함수에서 퍼즐판의 숫자가 모두 지정된 위치를 찾아갔는지 반복적으로 확인한다.

```
void checkPuzzle(int puzzle[][size])
{
        int result=1;
        for(int i=0 ; i<size ; i++)
                for (int j=0 ; j<size ; j++)
                {
                        if (puzzle[i][j] != size*size-1-size*i-j)
                                result *= 0;
                }
        if (result==1)
                game_over=1;//TODO
}
```

5. m=1000 (빈칸 움직임)의 경우에 life 값을 1 줄인다.

```
m=move_to(puzzle, l, command);

if (m==1000)
        life--;
if (life < 0)
        break;
```


6. 만약 life<0 이 되기 전에 checkPuzzle에서 참 값을 return하면 "Success!"를 출력하고 프로그램을 종료한다.

   checkPuzzle에서 참 값을 return 하기 전에 life<0 이 되면 "Fail!"를 출력하고 프로그램을 종료한다.

```
        if(game_over)
        printf(ANSI_RED"      Success!\n\n\n"ANSI_RESET);
        else
                if(life <=0)
                printf(ANSI_RED"      Fail!\n\n\n"ANSI_RESET);
        else
                printf(ANSI_RED"      Success!\n\n\n"ANSI_RESET);

    return 0;
}
```

위 4~5 과정은 6번 단계로 넘어가기 전까지 계속 반복된다.

- **프로그램의 각 함수 설명**

  - init 함수

    퍼즐게임의 전역변수와 퍼즐게임판을 초기화 해준다. (빈칸은 0으로 초기화)

```
void init(int puzzle[][size])
{
    int i,j;
    game_over = 0;
    for(i = 0; i < size; i++)
    {
        for(j=0; j<size; j++)
            puzzle[i][j] = size*size-1-size*i-j;
    }
}
```

i \ j	0	1	2	3
0	15	14	13	12
1	11	10	9	8
2	7	6	5	4
3	3	2	1	0

  - printPuzzle 함수

    퍼즐게임판 배열을 입력받고, 퍼즐게임판 모양을 출력한다.

    +, −, | 를 이용하여 표 모양을 출력한다.

```
void printPuzzle(int puzzle[][size])
{
    int i,j;
    system("clear");
    for(i=0;i<size;i++)
    {
        for(j=0;j<size;j++)
            printf("+--");
        printf("+\n");
        for(j=0;j<size-1;j++)
            if(puzzle[i][j] == 0)
                printf("|   ");
            else
                printf("|"ANSI_YELLOW"%2d"ANSI_RESET,puzzle[i][j]);
        if(puzzle[i][j] == 0)
            printf("|   |\n");
        else
            printf("|"ANSI_YELLOW"%2d"ANSI_RESET"|\n",puzzle[i][j]);
    }
    for(j=0;j<size;j++)
        printf("+--");
    printf("+\n");
}
```

➢ checkPuzzle 함수

퍼즐게임판을 입력받고, 퍼즐판의 숫자가 모두 지정된 위치를 찾아갔는지 확인한다. 2차원 배열의 수가 모두 일치하면 전역변수 game_over에 1을 저장한다.

```
void checkPuzzle(int puzzle[][size])
{
        int result=1;
        for(int i=0 ; i<size ; i++)
                for (int j=0 ; j<size ; j++)
                {
                        if (puzzle[i][j] != size*size-1-size*i-j)
                                result *= 0;
                }
        if (result==1)
                game_over=1;//TODO
}
```

➢ find_0_loc 함수

퍼즐게임판을 입력받은 후, 빈칸의 위치(0이 저장된 곳)를 찾아 하나의 정수 값으로 return한다.


```
int find_0_loc(int puzzle[][size])
{
        int i, j, loc;//TODO
        for (i=0 ; i<size ; i++)
                for (j=0 ; j<size ; j++)
                        if (puzzle[i][j]==0)
                                loc=10*i+j;
        return loc;
}
```

i \ j	0	1	2	3
0	0	1	2	3
1	10	11	12	13
2	20	21	22	23
3	30	31	32	33

➢ getch 함수

콘솔창에 입력하는 key의 값을 받아오고, 입력받은 key를 return한다.

key를 입력할 때, Enter로 입력할 필요가 없다.

```
int getch(void)
{
    char   ch;
    int   error;
    static struct termios Otty, Ntty;
```

```
    fflush(stdout);
    tcgetattr(0, &Otty);
    Ntty = Otty;
    Ntty.c_iflag = 0;
    Ntty.c_oflag = 0;
    Ntty.c_lflag &= ~ICANON;
#if 1
    Ntty.c_lflag &= ~ECHO;
#else
    Ntty.c_lflag |= ECHO;
#endif
    Ntty.c_cc[VMIN] = 0;
    Ntty.c_cc[VTIME] = 1;

#if 1
#define FLAG TCSAFLUSH
#else
#define FLAG TCSANOW
#endif

    if (0 == (error = tcsetattr(0, FLAG, &Ntty)))
    {
        error = read(0, &ch, 1);
        error += tcsetattr(0, FLAG, &Otty);
    }

    return (error == 1 ? (int)ch : -1);
}
```

- move_to 함수

GetCommand 함수에서 전달받은 command 값에 대한 동작을 수행한다.

현재 빈칸의 위치를 find_0_loc 함수로부터 전달받고, 이를 다시 [i] [j] 형태로 변환한다. (loc=10*i+j)

전달받은 command값에 따라 빈칸을 옮기나, 빈칸이 모서리에 있어 더는 움직일 수 없는 경우가 발생한다. 따라서 움직임 여부를 출력해줘야 한다. if문을 사용해 (빈칸이 왼쪽 모서리에 있는데 왼쪽으로 옮기려 하는 경우), (맨 위에 있는데 위로 옮기려 하는 경우), …, ( ) 를 제외하고 1000을 return한다. 즉, 빈칸이 움직인 경우에만 1000을 return한다.

```
int move_to(int puzzle[][size],int loc,int command)
{
        int i, j;
        i=loc/10;
        j=loc%10;
        if (command==0)
        {
                if (i<size-1)
                {
                        puzzle[i][j]=puzzle[i+1][j];
                        puzzle[i+1][j]=0;
                        return 1000;
                }
        }

        if (command==1)
        {
                if (j<size-1)
                {
                        puzzle[i][j]=puzzle[i][j+1];
                        puzzle[i][j+1]=0;
                        return 1000;
                }
        }

         if (command==2)
         {
                  if (i>0)
                  {
                          puzzle[i][j]=puzzle[i-1][j];
                          puzzle[i-1][j]=0;
                          return 1000;
                  }
         }

         if (command==3)
         {
                  if (j>0)
                  {
                          puzzle[i][j]=puzzle[i][j-1];
                          puzzle[i][j-1]=0;
                          return 1000;
                  }
         }//TODO
}
```

➢ shuffle 함수

퍼즐게임판을 입력받아 빈칸을 옮기는 방식으로 게임판을 랜덤하게 섞는다. iter=100이므로 move_to함수를 이용해 빈칸을 100번 옮긴다. r=rand()%4는 0, 1, 2, 3 중 하나의 숫자이다.

```
void shuffle(int puzzle[][size])
{
    int r,l;
    int iter = 100;     /* 필요시횟수를 변경해 가면서 테스트 해보세요
    while(iter)
    {
        iter--;
        l = find_0_loc(puzzle);
        r = rand()%4;
        move_to(puzzle,l,r);
    }
}
```

➢ GetCommand 함수

W, w를 입력하는 경우에는 0을 return, A, a를 입력하는 경우에는 1을 return, S, s를 입력하는 경우에는 2를 return, D, d를 입력하는 경우에는 3을 return한다.

아무 key도 입력되지 않는 동안은 −1을 return해 게임판이 움직이지 않도록 한다.

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

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

    return -1;
}
```

➢ **main 함수**

크기가 size*size인 퍼즐게임판을 생성한다, (2차원 배열)

init 함수를 실행하여 퍼즐게임판을 초기화해주고, shuffle 함수를 실행하여 퍼즐게임판을 섞는다.

```
int main(void)
{
    int puzzle[size][size]; /* 크기가 size * size인 퍼즐게임판 */
    int life = life_count; /* 게임 생명 카운트 */
        int command = -1;
        // 필요한 변수는 자유롭게 추가 가능

        srand(time(NULL));              /* 난수를 생성하기 위한
    init(puzzle);                       /* 퍼즐게임판 초기화 */
    shuffle(puzzle);            /* 퍼즐게임판 섞기 */
```

game_over=1이 되기 전까지 (배열의 수가 모두 제 자리를 찾기 전까지) key값 입력받기, life값 줄이기, checkPuzzle을 반복한다.

```
        while(!game_over)
    {
                /*
                        입력이 일어날때마다 퍼즐게임판을 새롭게

                */

                command = GetCommand();
                int l;
                int m;
                l=find_0_loc(puzzle);
                m=move_to(puzzle, l, command);

                if (m==1000)
                        life--;
                if (life < 0)
                        break;
                /*
                   TODO LIST

                   퍼즐게임판에서 빈칸을 찾는다. ( find_0_loc() )
                   key에 대한 동작을 수행한다.
                */
                printf(ANSI_GREEN"up    : w\nleft   : a\ndown   : s
\nright :d\n"ANSI_RESET);

        printf(ANSI_RED"Press Command...\n"ANSI_RESET);
```

만약 life<0이 되기 전에 game_over=1이라면 “Success!”를 출력하고 프로그램을 종료하며, life<0 이 먼저 조건 만족하면 “Fail!”을 출력하고 프로그램을 종료한다.

```
                /*
                        퍼즐게임의 종료 조건을 체크한다.
                */
                checkPuzzle(puzzle);
    }
    /*
                퍼즐게임 결과 출력
        */
    printf("\n\n\n\n\n");

        if(game_over)
        printf(ANSI_RED"      Success!\n\n\n"ANSI_RESET);
        else
                if(life <=0)
                printf(ANSI_RED"      Fail!\n\n\n"ANSI_RESET);
        else
                printf(ANSI_RED"      Success!\n\n\n"ANSI_RESET);

    return 0;
}
```


- **결과 평가**
  - 방향키(w, s, d, a)를 눌렀을 경우 숫자 판이 정상적으로 움직였다.
  - 숫자가 정렬되면 프로그램이 정상적으로 종료되었다.
  - 프로그램 종료 후, terminal로 정상적으로 원상 복귀되었다.
  - 게임이 끝날 때까지 오류가 발생하지 않았다.
  - (추가구현 $\boldsymbol{a_1}$) 숫자 판이 움직일 때 마다 life가 1씩 감소하고, life가 0일 때 숫자 판이 정렬되지 않은 상태이면 Fail을 출력하고 프로그램을 종료하는 추가 구현 사항을 만족하였다.