과 제

Programming Assignment 2

과목명	C프로그래밍
분반	0451분반
담당교수님	박소영 교수님
학과	컴퓨터공학부
학번	202011353
이름	이호은
제출일	2020년 6월 16일

Programming Assignment II

Q. 보물찾기 게임 제작

1. 문제 정의와 제안 목적

2차원 배열을 이용한 보물찾기 게임을 구현한다. 각 깃발 아래에는 도움 혹은 위협이 되는 것과 하나의 깃발 아래에는 깃발이 있다. 벽(장애물)도 존재한다. 난이도를 다르게 구성하고, 개성있는 스토리, 화면 구성, 추가 기능들을 추가로 개발한다.

이 과제를 통해 게임이라는 장르의 프로그램의 전반적인 구성과 여러 기능들, 코드의 효율성, 함수형 프로그래밍의 장점 등 그동안 코딩으로 실습했던 프로그램의 종류와 완전히 다른 종류의 프로그램이기에 게임이라는 하나의 완전한 프로그램을 제작하며 실전 프로그래밍에 대해 배울 수 있는 시간이 될 것이라고 생각한다.

2. 주요 기능

게임의 컨셉은 비행기 조종사인 플레이어(◆)가 사막에 추락한 후 여러 가방들(▶)을 찾아보며 다음 지역으로 가는 지도(보물)를 찾는 것이다. 단계(Level)을 "지역(Region)"이라고 표현하고, 게임 룰을 "생존 가이드"라고 표현하여 게임 진행에 일체감을 주었다.

i) 메뉴

- 메뉴는 ① 게임 시작, ② 생존 가이드(게임 룰), ③ 통계, ④ 데이터 초기화, ⑤ 스토리 다시 보기, ⑥ 게임 종료로 구성되어 있다. 사용자는 방향키(↑, ↓)를 이용해 포커스를 이동한 후 <y>를 눌러 선택이 가능하다.

ii) 레벨 선택

- 단계(지역)는 총 6 개로 난이도가 정해져 있는 1~5 단계와 사용자가 직접 난이도를 조절할 수 있는(단, 5 단계보다는 어려움) Custom Region 으로 구성되어 있다. 메뉴 화면과 마찬가지로 방향키(↑, ↓)를 이용해 포커스를 이동한 후 <y>를 눌러 선택할 수 있다.

- <q>를 누르면 메인 메뉴로 돌아간다.

- 단계(지역)는 바로 이전 단계를 클리어해야 해금된다. (예를 들어 Region 2 를 클리어해야 Region 3 이 해금되며, Region 1 과 2 가 열려 있는 상태에서 Region 1 을 다시 클리어한다고 Region 3 이 해금되지는 않는다.) 잠긴 레벨은 (잠김)이 출력되어 플레이할 수 있는 단계와 없는 단계를 구분해주었다.

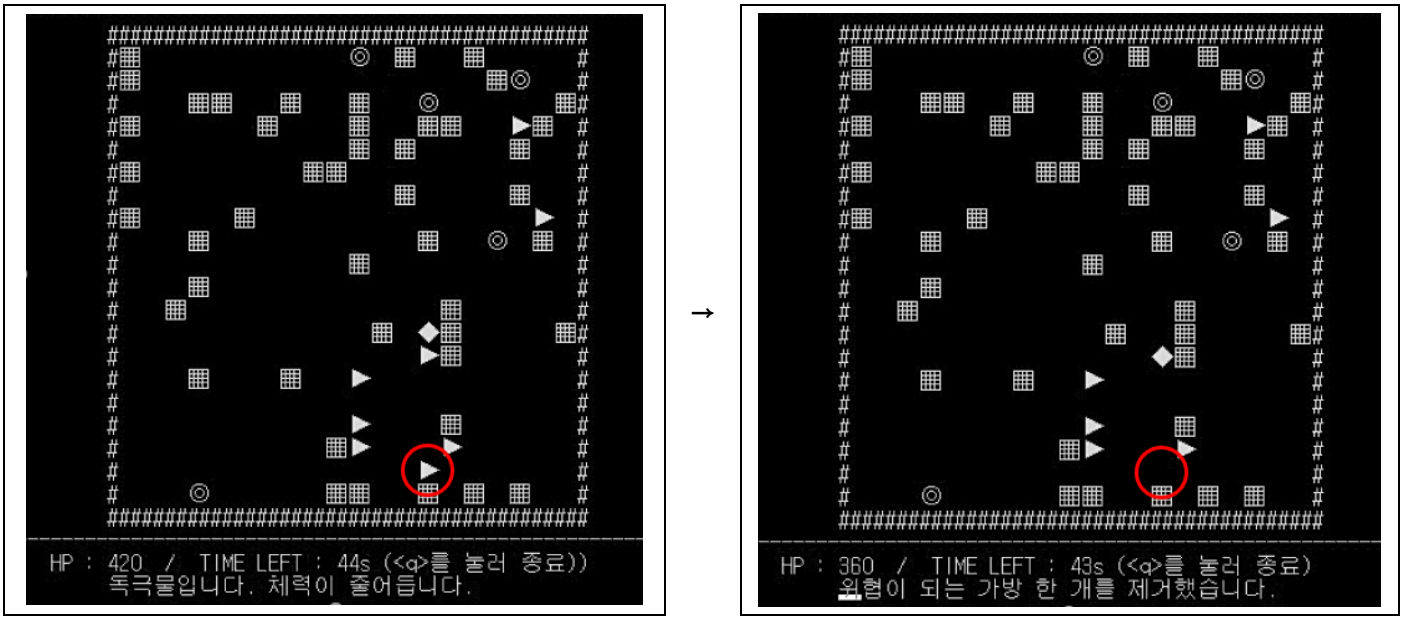
iii) 게임 진행

게임은 기본 요구사항과 같이 플레이어(◆)를 방향키로 이동해 가방(▶) 중 지도를 찾아야 한다.

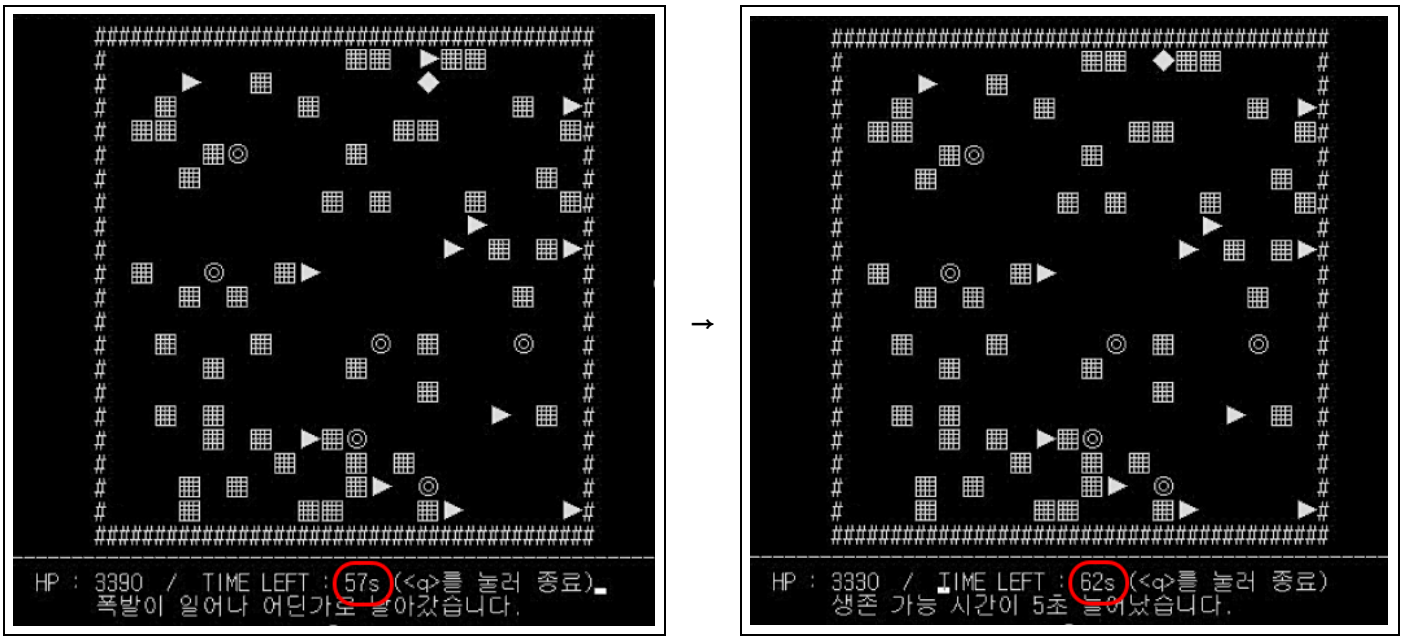
단계(지역)별 각 아이템 구성과 난이도는 아래 표와 같다.

	L1	L2	L3	L4	L5	LC
가방	4	6	8	10	12	15~30
도움	3	2	3	4	4	0~30%
위협	1	4	5	6	8	70%~100%
폭탄	2	3	4	5	6	2~12
벽	10	20	30	40	50	150%~250%
체력 (차감)	1000 (-20)	1500 (-30)	2000 (-40)	3000 (-50)	4000 (-60)	5000 (-80)
제한시간	20s	30s	40s	50s	60s	70s

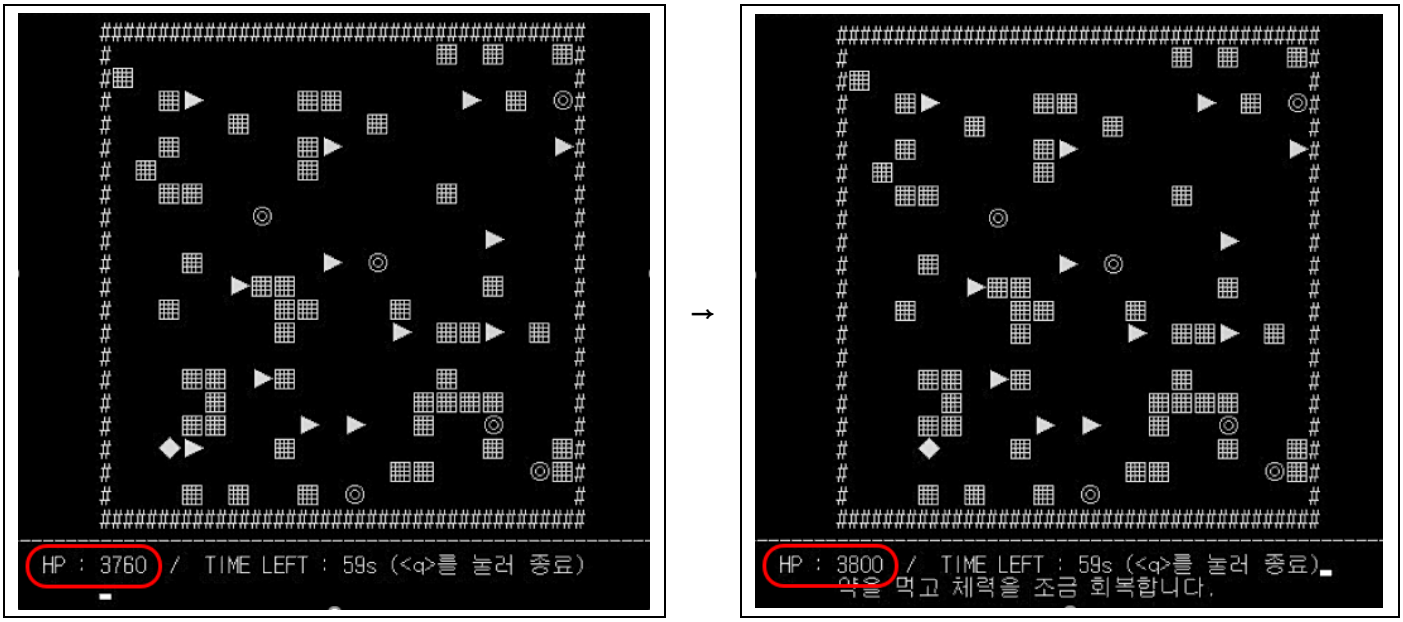
iii-1) 도움 가방



- 위협 가방 한 개 삭제 : 전체 가방 중 위협이 되는 깃발을 한 개 제거한다. (단, 남은 가방이 모두 도움이 되는 가방일 경우 이 이벤트는 발생하지 않는다.)

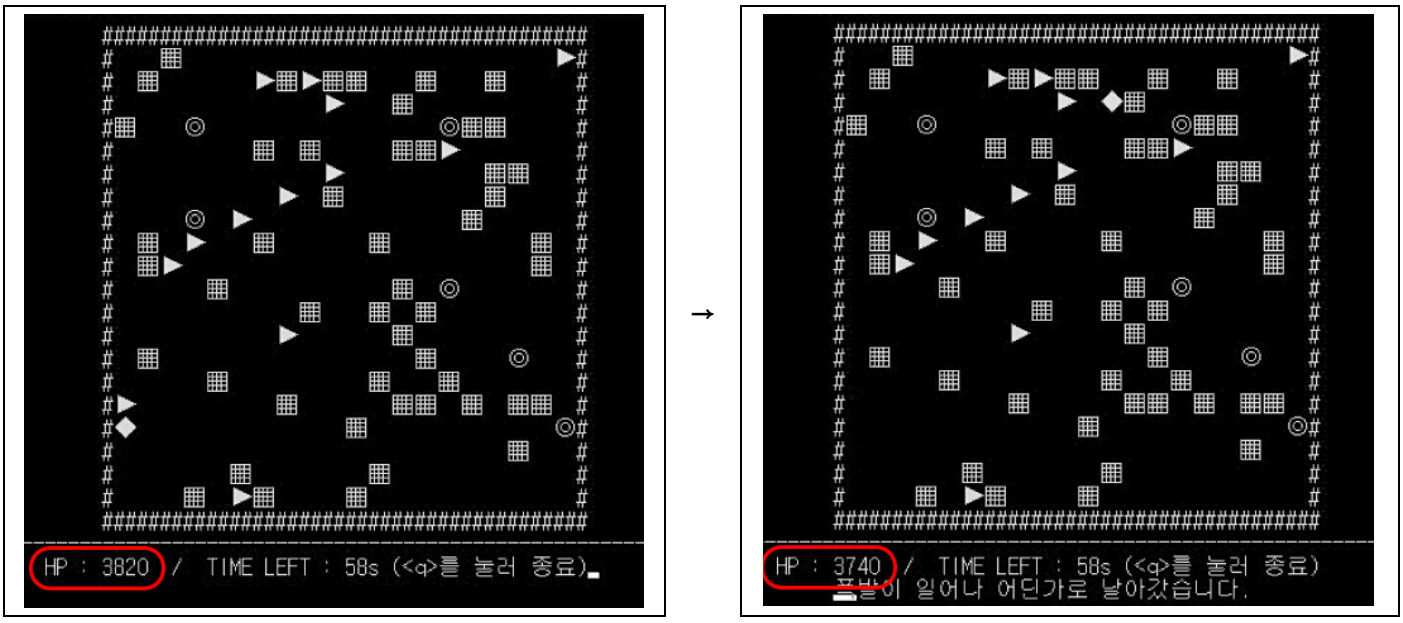


- 생존 시간 5 초 증가 : 남은 시간이 5 초 늘어난다.

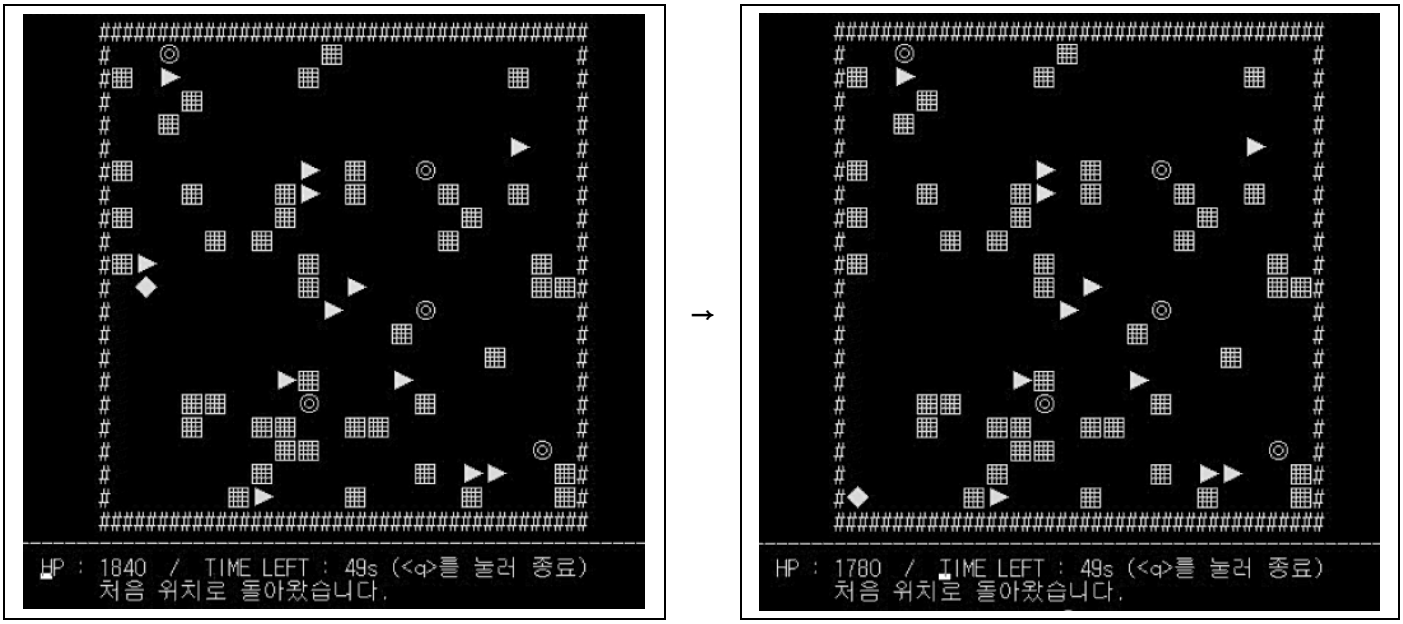


- 체력 증가 : 체력(점수)가 랜덤(최소 10P ~ 최대 100P)으로 증가한다.

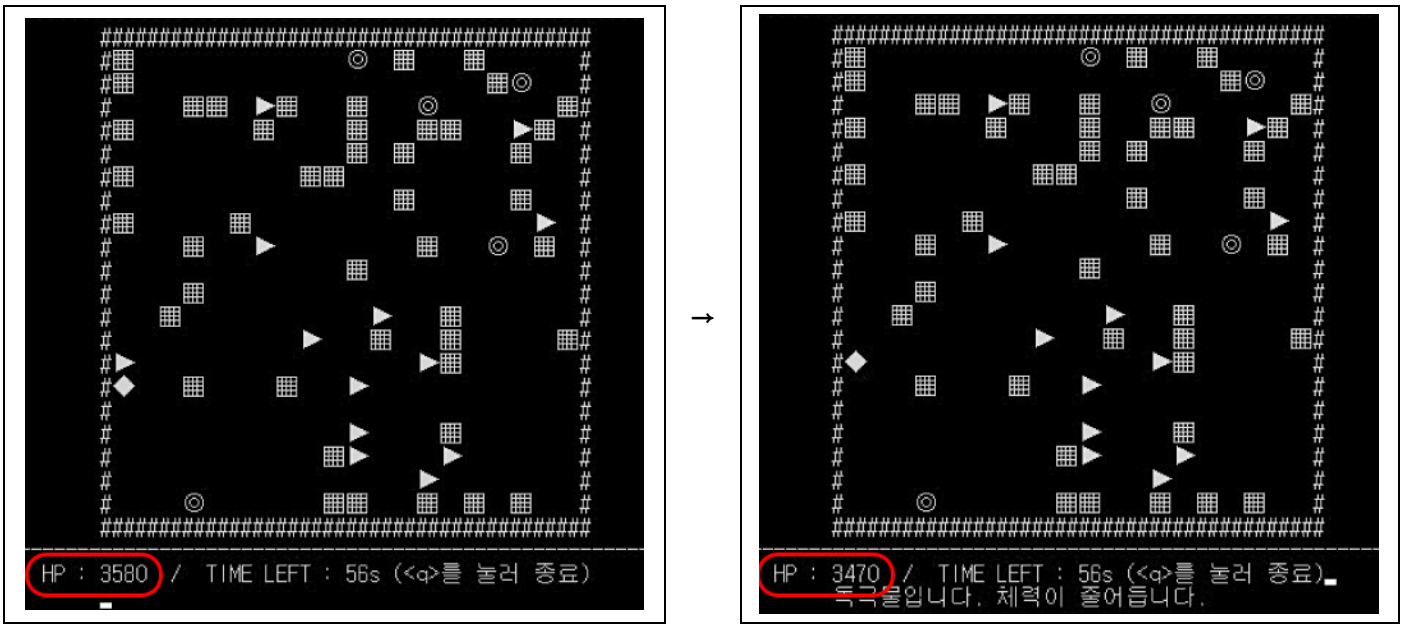
iii-2) 위협 가방



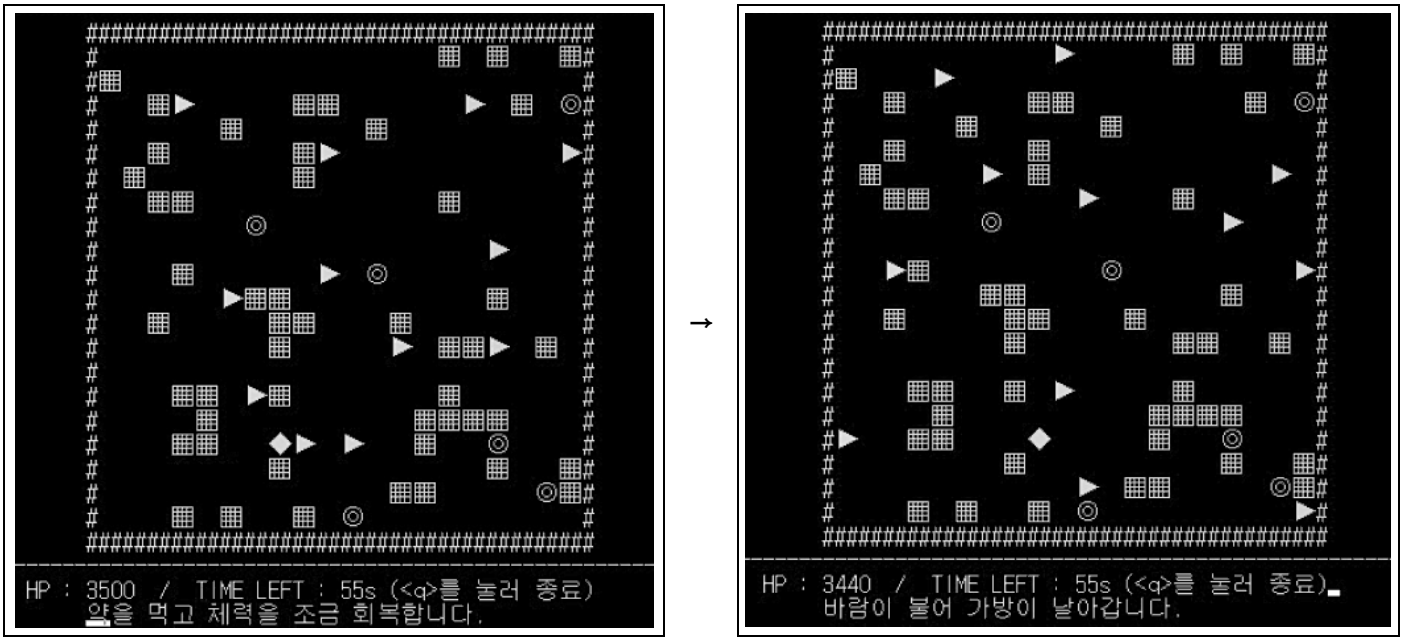
- 폭발 : 체력이 소량 랜덤(최소 10P ~ 최대 50P) 감소하며 플레이어의 위치가 랜덤으로 변한다.



- 초기 위치로 이동 : 플레이어의 위치가 처음 시작 위치로 이동된다. (최좌측하단)

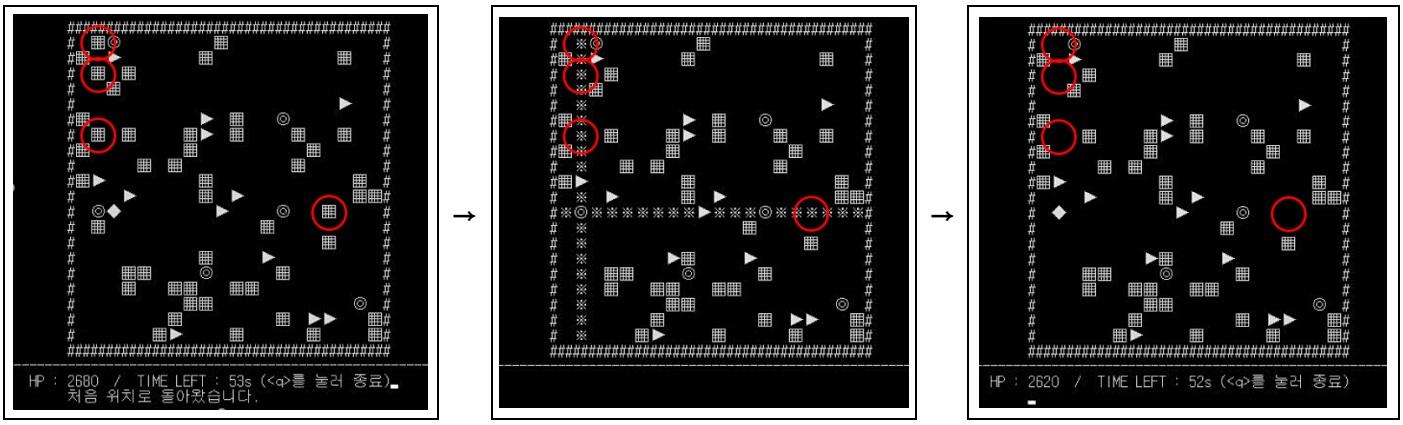


- 체력 감소 : 체력(점수)가 랜덤(최소 10P ~ 최대 100P)으로 감소한다.



- 가방 위치 랜덤 배치 : 모든 가방(지도 포함)의 위치가 랜덤으로 변한다. (플레이어, 벽, 아이템 제외)

iii-3) 폭탄



- 폭탄(◎)은 십자(+) 모양으로 선(※)이 그려지며 벽(▦)을 파괴한다. 플레이어는 이를 잘 활용해 벽을 파괴해 지름길을 만들 수 있다. 더욱 전략적인 플레이가 가능하도록 추가한 요소이다.

iii-4) 점수(체력)와 시간

플레이어의 체력은 곧 점수이다.

- 게임 하단에 현재 체력과 남은 시간이 표시된다. 주어지는 초기 체력과 시간은 단계에 따라 다르며 높은 단계일수록 초기 체력과 시간이 많이 주어진다. 단, 한 번 이동 시 차감 체력도 증가한다.

- 높은 단계일수록 시작 체력을 높게 해 높은 단계에서 더욱 높은 점수를 달성할 수 있도록 하였다.

iii-5) 게임 클리어와 실패

- 제한시간과 체력이 0 이 되기 전에 가방들 중 지도가 들어있는 가방을 찾으면 게임이 클리어 된다. 화면이 출력되고 3 초 후 자동으로 다음 단계로 넘어간다. (단, 5 단계 클리어 시 엔딩이, Custom 단계 클리어 시 메인 화면으로 이동한다.)

- 제한시간 혹은 체력이 0 이 되면 게임 실패로 종료된다. 실패한 경우는 시작화면으로 돌아간다.

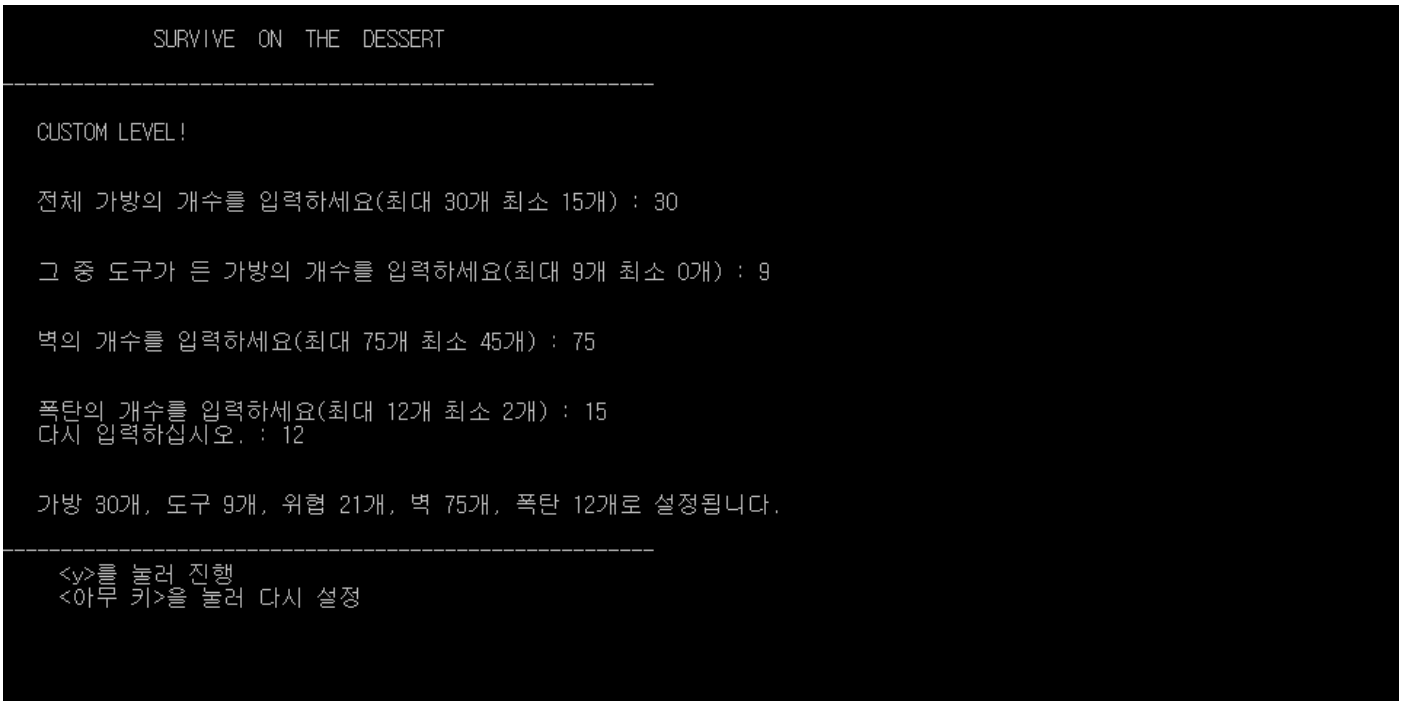
- 게임 실패한 경우 다음 팁 중 하나를 랜덤으로 출력해준다.

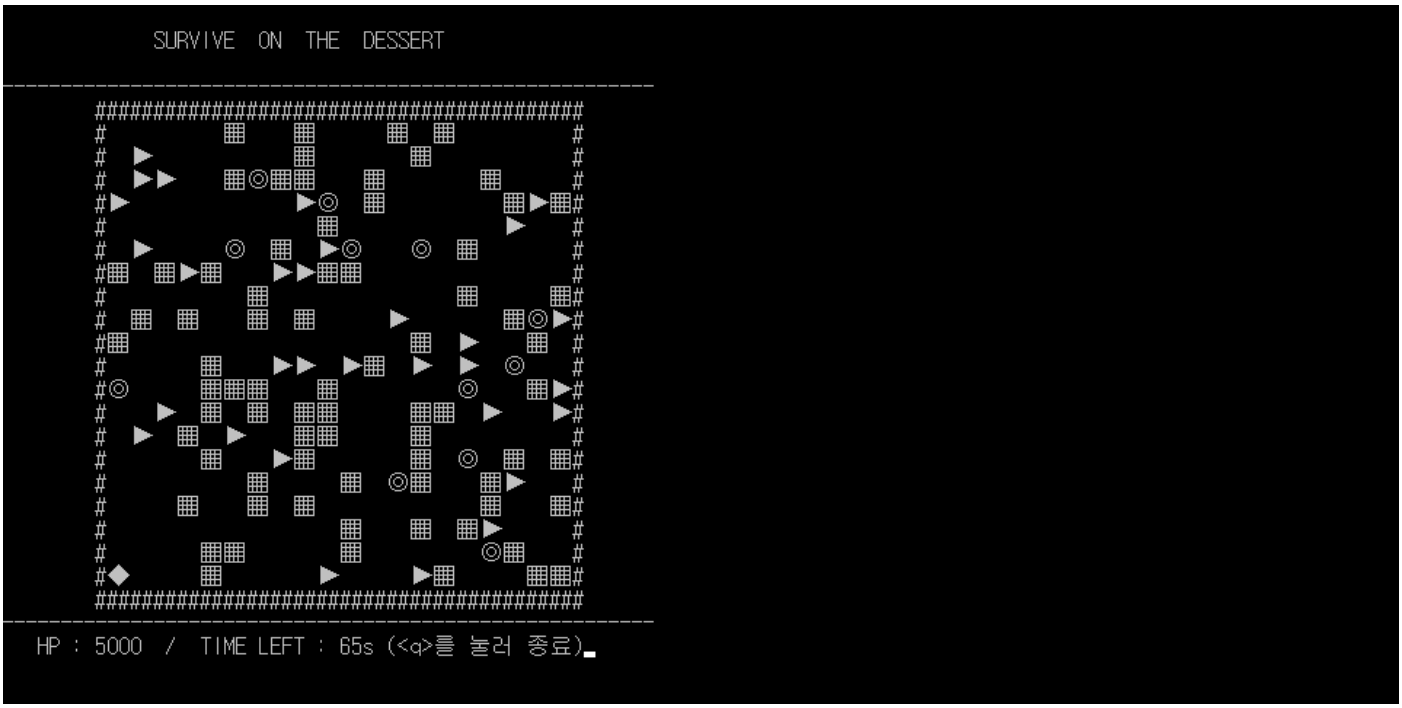
이동을 효율적으로 해야 합니다.
도구가 든 가방은 생존에 큰 도움이 됩니다.
다음 지역을 해금하려면 바로 이전 지역을 클리어해야 합니다.
폭탄을 이용하면 벽을 뚫어 빠른 길을 만들 수 있습니다.
메인 메뉴에서 '통계' 메뉴를 통해 실적을 확인해보세요.

iv) 사용자 커스텀

커스텀 단계는 5 단계까지 클리어했을 경우 해금된다. 사용자들은 전체 가방의 개수, 도움이 되는 가방의 개수, 벽의 개수, 폭탄의 개수를 직접 입력하여 난이도를 조절할 수 있다. (위협이 되는 가방의 개수는 전체 가방의 개수 - 도움이 되는 가방의 개수와 같다)

가방의 개수는 최대 30 개 ~ 최소 15 개로 정해지며, 나머지의 수는 가방 개수의 비례하여 변한다.



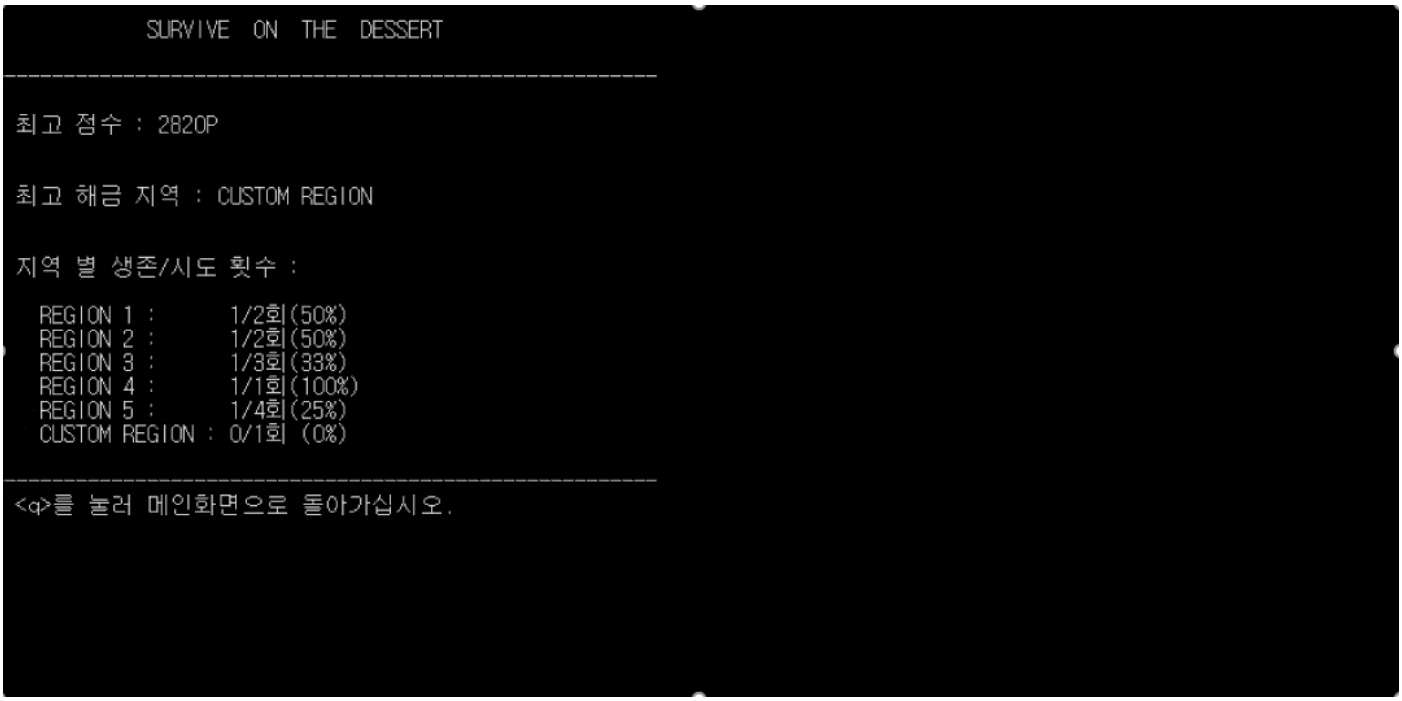


- 범위에 벗어나는 수를 입력하면 "다시 입력하십시오 : "를 출력하여 다시 입력 받는다.

- 모두 입력 후 <y>를 제외한 아무 키를 누르게 되면 처음부터 다시 입력 받는다.

v) 게임 통계

메인 메뉴에서 <통계>를 선택한다.



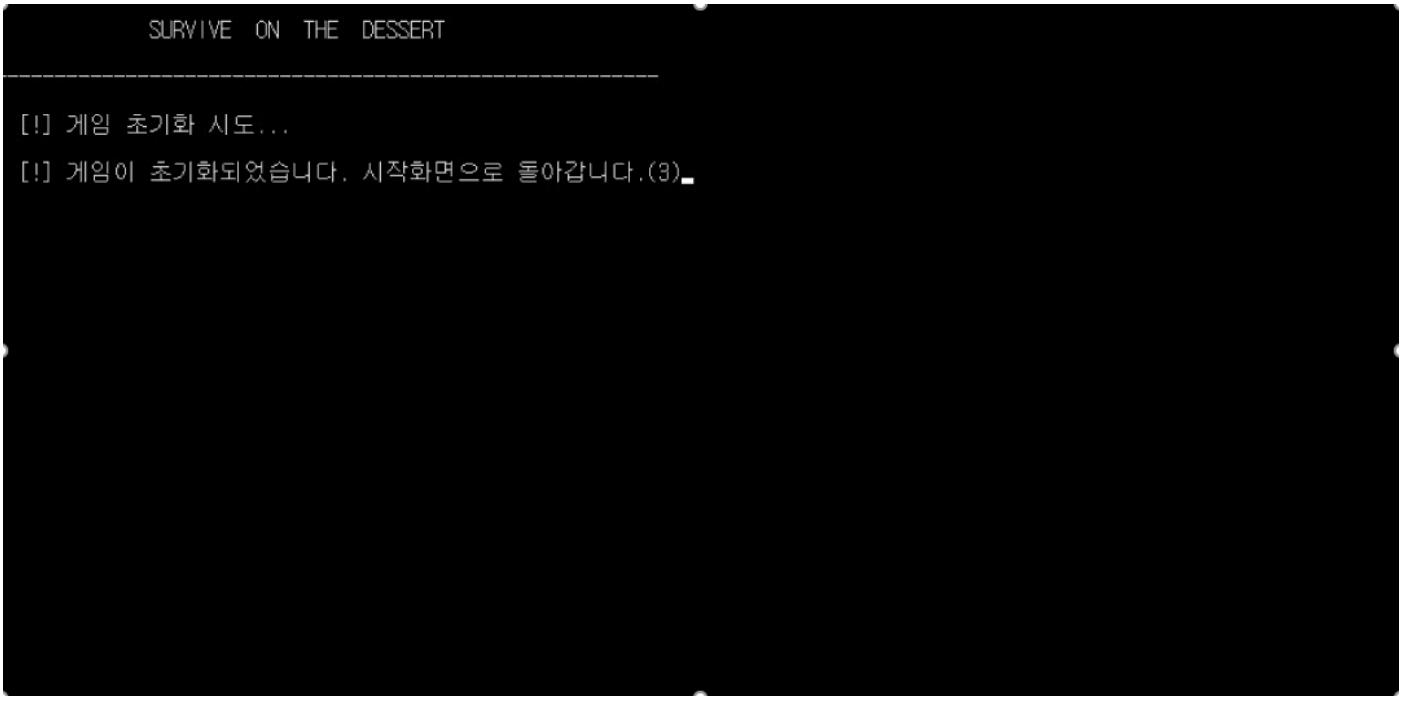
- 단계별로 지금까지 클리어 횟수와 플레이 횟수, 성공률(%)을 보여준다.

- 최고 해금 지역(어느 단계까지 클리어했는지)과 최고 점수를 보여준다.

vi) 게임 초기화

메인 메뉴에서 <데이터 초기화>를 선택한다. 지금까지의 진행 상황(클리어/플레이 횟수, 최고점수, 최고레벨)을 전부 초기화한다.

- <y>를 입력해 초기화를 진행하거나 <n>을 입력해 취소한다.



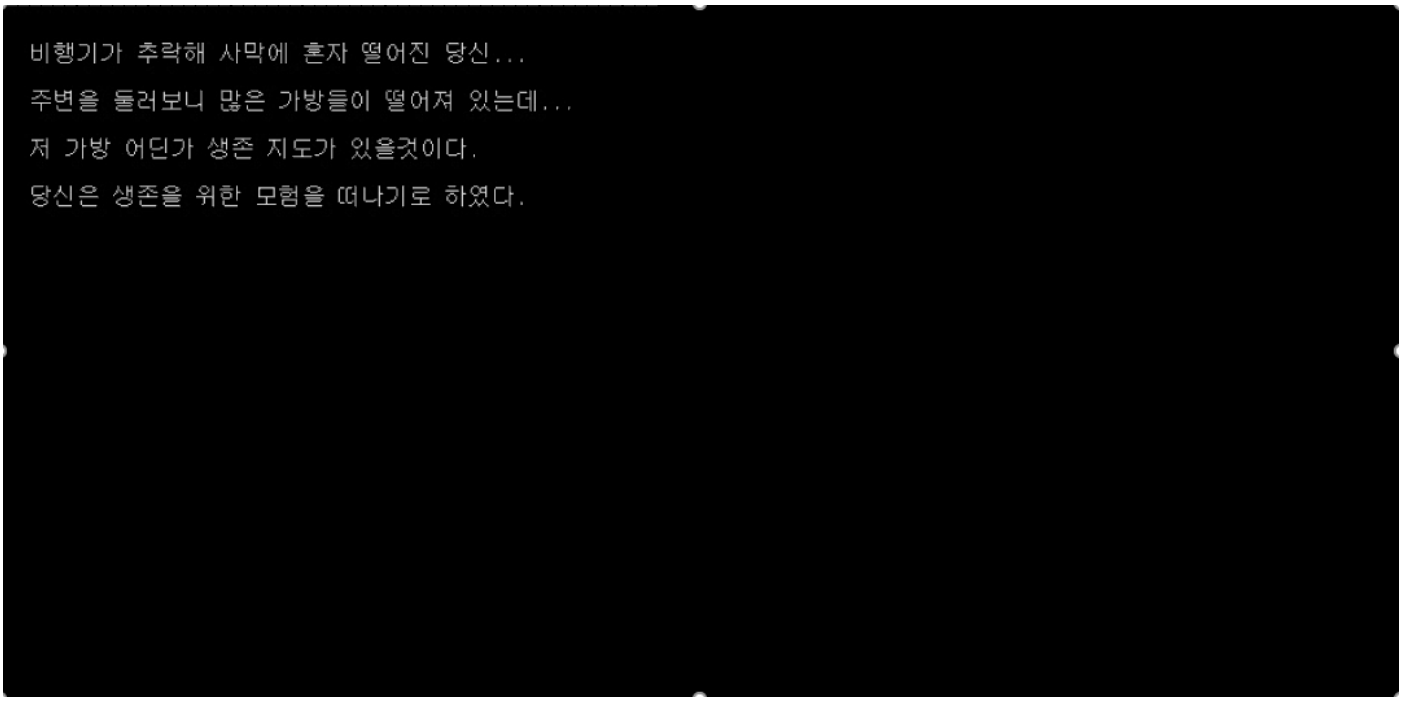
- 초기화를 시도하여 데이터를 전부 초기화하고 성공하면 성공 메시지를 출력한 후 메인 화면으로 돌아간다.

vii) 스토리 연출

스토리는 게임을 맨 처음 실행했을 때 연출되며, 5 단계를 클리어했을 때 엔딩 스토리가 연출된다. 또한 메인 메뉴의 <스토리 다시보기> 메뉴를 통해 다시 볼 수 있다.

게임의 전체적인 메뉴명과 연출을 스토리에 관련되게 하였다.

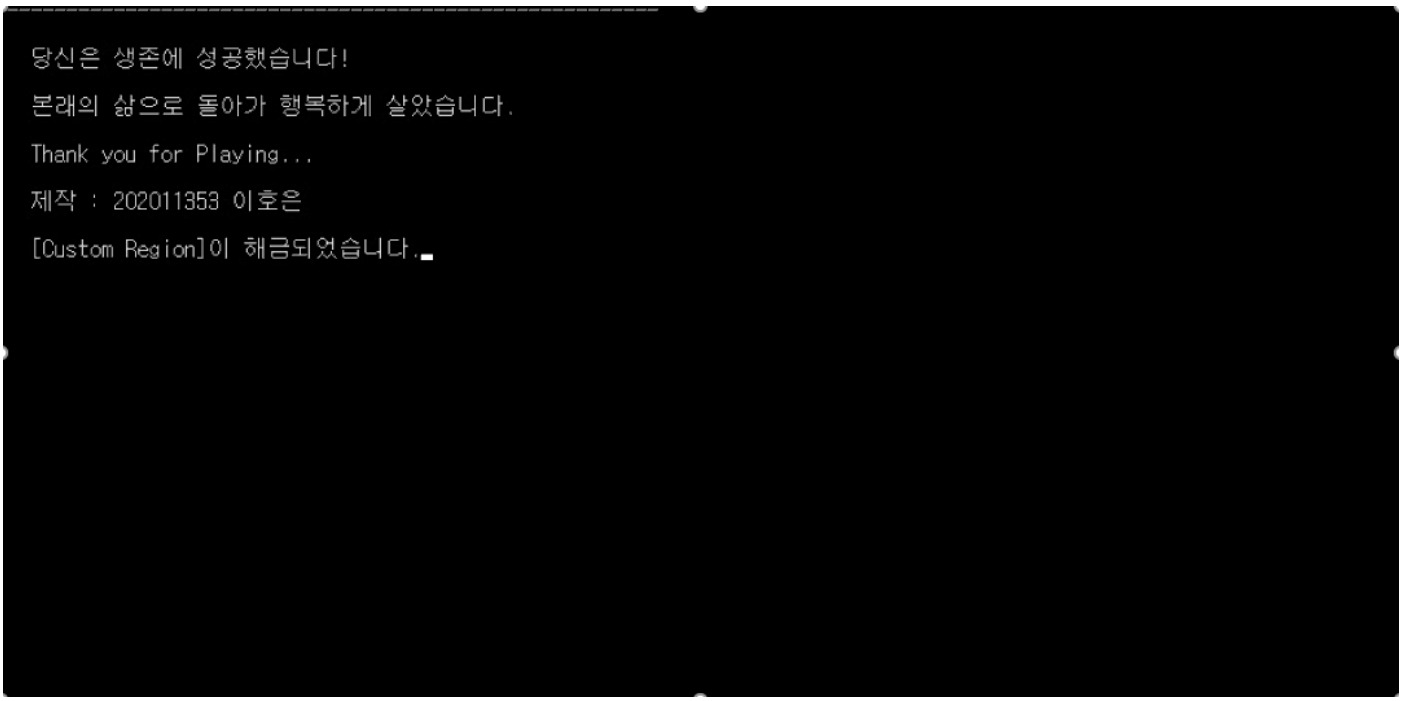
vii-1) 시작 스토리



- 스토리는 한 글자씩 출력되도록 하여 글을 읽어주는 듯한 연출을 하였다.

- 스토리 이후 게임의 룰을 보여주도록 되어있다.

vii-2) 엔딩 스토리



- 5 단계를 클리어하면 엔딩 스토리가 출력된다.

- 추락한 플레이어가 결국 생존에 성공하였다는 스토리이다.

- 시작 스토리와 마찬가지로 한 글자씩 출력되도록 하여 글을 읽어주는 듯한 연출을 하였다.

viii) UI

게임의 타이틀과 선을 적절하게 사용하고 여러 기능을 추가하여 게임의 느낌을 내었다.

- 실제 게임처럼 방향키(↑, ↓)를 이용해 화살표의 포커스를 이동시키고 기존 콘솔 프로그램처럼 선택지의 문자를 입력하는 방식이 아닌 특정 키를 눌러 선택하는 방식을 채택, 구현하였다.

- 게임 하단에 방금 발동된 도움 혹은 위협이 되는 가방이 어떤 종류인지 표시하여 준다.

3. 주요 기능 구현을 위한 기술

게임 기능과 개발의 효율성을 위해 기능에 따라 함수로 나누어 프로그래밍 하였고, 비슷한 기능을 가진 함수끼리 c 파일로 분리하여 프로그램을 작성하였다. 배열의 크기는 기호상수로 정의해주었다.

```c
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include <time.h>
#include <conio.h>
#include <string.h>

/* LEVEL */
#define MIN_LVL 1
#define MAX_LVL 6

/* GAME MAP */
#define MAP_HORZ 20
#define MAP_VERT 20

#define ARR_HORZ 42
#define ARR_VERT 22

/* main.c */
void gotoXY(int x, int y);
int main(void);

/* function.c */
void printLine(void);
void printTitle(void);
void gameTitle(void);
void clear(int type, char(*mapPtr)[ARR_HORZ]);
int getMove(int startX, int startY, int interval, int startV, int minV, int maxV);

/*data.c */
int database(char type[20], int data);
int printData(void);

/* game.c */
int explain(void);
int startGame(char(*mapPtr)[ARR_HORZ], int selectedLevel);
void drawScreen(char(*mapPtr)[ARR_HORZ]);
void tutorial(void);
int trap(char(*mapPtr)[ARR_HORZ], int *score, int *x, int *y, int baseX, int baseY, int hintN, int trapN);
void hint(char(*mapPtr)[ARR_HORZ], int *score, int *plusTime);
void item(char(*mapPtr)[ARR_HORZ], int x, int y);
void ending(void);

/* menu.c */
int mainMenu(void);
int selectLevel(int level);
```

i) 메뉴 선택 : 메인 화면과 단계 선택 화면의 메뉴에서 방향키 이동을 감지하고 항목을 선택할 수 있도록 한다.

- 구현 기능 : 방향키(↑, ↓)를 이용해 포커스를 이동시킨다. <y>를 누르면 포커스가 있는 항목을 선택한다.

```
int getMove(int startX, int startY, int interval, int startV, int minV, int maxV)
```

- 매개변수

변수명	자료형	설명
startX	int	커서 초기 위치의 x 값(콘솔)
startY	int	커서 초기 위치의 y 값(콘솔)
interval	int	커서가 상하로 이동할 때의 이동 간격(y 값)
startV	int	선택지가 시작하는 값
minV	int	선택지가 될 수 있는 최솟값
maxV	int	선택지가 될 수 있는 최댓값

메인 화면과 단계 선택 화면의 메뉴에서 각 인자를 제외하고 동일한 기능(방향키를 통한 이동, <y>를 눌러 선택)을 제공하므로 함수로 제작하였다.

```
    while (1) {
        printf("\b\b  ");
        gotoXY(x, y);
        printf("◀");

        c = _getch();
        if (c == 224) {
            c = _getch();
            switch (c) {
```

while 문 루프 내에 _getch 함수로 방향키를 입력 받는다. 루프가 돌 때마다 화살표(◀)를 지우고 출력한다.

```
                case 72: {
                    if (now > minV) {
                        y -= interval;
                        now--;
                    }
                    break;
                }
```

입력된 방향키에 따라 y 값을 변경해 이동한다. 다만 현재 선택지가 최솟값이거나 최댓값이면 더 이상 이동할 수 없도록 한다.

```
    else if (c == 'y' || c == 'q')
        break;
}

if (c == 'q') return -1;
else return now;
```

만약 <y> 또는 <q>를 입력하면 루프를 탈출하며, <q>일 경우 메인 화면으로 돌아가는 것이므로 -1 을 반환하고, <y>일 경우 선택한 것이므로 현재 포커스 된 선택지의 값을 반환한다.

ii) 데이터베이스 : 정적 변수(static)를 지역변수로 갖는 함수를 선언해 데이터베이스로 사용했다.

- 구현 기능 : 사용자의 최고 점수 SAVE & LOAD, 최고 해금 단계 UP & LOAD, 각 단계별 플레이 횟수 SAVE & LOAD, 각 단계별 클리어 횟수 SAVE & LOAD, 모든 데이터 초기화

- 매개변수

변수명	자료형	설명
type	char[]	데이터베이스 제어용 코드 전달(SAVE or LOAD 등)
data	int	데이터

정적 변수(static)는 함수가 종료되어도 변수는 유지되므로 이를 활용해 값을 저장하는 함수를 제작하였다.

type 에 인자로 명령어를 전달하고, data 에 인자로 데이터를 저장하면 명령어를 시행한다.

```
static int level = 1;
static int bestScore = 0;
static int levelP[6] = { 0,0,0,0,0,0 };
static int levelC[6] = { 0,0,0,0,0,0 };
int i;
```

상단부터 사용자의 최고 해금 단계, 최고 점수, 단계 별 플레이 횟수(배열), 단계 별 클리어 횟수(배열)을 저장하는 정적 변수를 선언, 초기화해주었다.

```
/* returnLevel : 레벨 반환 */
if (!strcmp(type, "returnLevel")) {
    return level;
}
/* saveLevel : 레벨 한 단계 업 */
if (!strcmp(type, "upLevel")) {
    if (level < 6) {
        level++;
        return 0;
    }
    else return -1;
}
/* 점수 기록 */
if (!strcmp(type, "writeScore")) {
    if (data > bestScore)
        bestScore = data;
}
/* 최고 점수 반환 */
if (!strcmp(type, "returnScore")) {
    return bestScore;
}
/* 플레이 횟수 */
if (!strcmp(type, "played")) {
    levelP[data - 1] += 1;
}
/* 클리어 횟수 */
if (!strcmp(type, "cleared")) {
    levelC[data - 1] += 1;
}


/* clearAll : 모든 데이터 초기화 */
if (!strcmp(type, "clearAll")) {
    level = 1;
    bestScore = 0;
    for (i = 0; i < 6; i++)
        levelP[i] = 0;
}
```

```c
/* 플레이 횟수 출력 */
if (!strcmp(type, "printLevelP")) {
    for (i = 0; i < 5; i++) {
        printf("    REGION %d :      %d/%d회", i + 1, levelC[i], levelP[i]);
        if (levelC[i] == 0) printf(" (0%%)");
        else printf("(%d%%)", (int)((double)levelC[i] / (double)levelP[i] * 100));
        printf("\n");
    }
    printf("    CUSTOM REGION : %d/%d회", levelC[i], levelP[i]);
    if (levelC[i] == 0) printf(" (0%%)");
    else printf("(%d%%)", (int)((double)levelC[i] / (double)levelP[i] * 100));
    printf("\n");
}
```

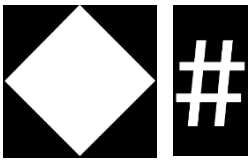
strcmp 함수를 이용해 해당 명령과 일치하는 명령을 실행하도록 하였다.

반환을 요구하는 병령은 return 을 이용해 해당 데이터값을 반환해주고, 저장과 변경을 요구하는 명령은 데이터를 이용해 변수의 값을 변경한다.

iii) 게임 실행 : main 함수에서 게임의 배경이 되는 배열을 선언해주고 이를 이용해 game 함수에서 진행한다.

```c
char map[ARR_VERT][ARR_HORZ];
char(*mapPtr)[ARR_HORZ] = map;
```

- 여기서 특수문자는 콘솔 창에서 두 칸을 차지하고 이외 문자의 경우도 세로의 길이가 가로의 길이의 두 배이므로 실제 게임에 사용되는 배열의 크기는 20x40 으로 설정하였다.

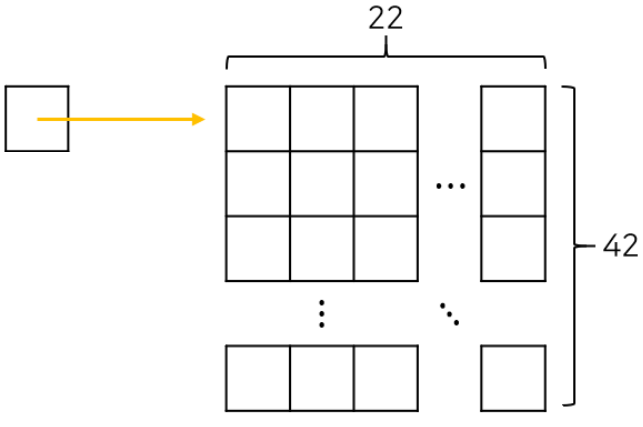


(실제 콘솔에서 출력되는 문자의 비율 - 좌측 특수문자는 2 칸, 우측 이외 문자는 1 칸)

- startGame 함수에서는 main 함수의 지역변수인 2 차원 배열을 참조해야 하므로 2 차원 배열 포인터를 넘겨준다.

```c
int startGame(char(*mapPtr)[ARR_HORZ], int selectedLevel) {
    int x = 1, y = 20;
    const int scoreX = 3, scoreY = 27;
    const int baseX = 8, baseY = 4;
    int level, score, fin = 0;
    int flagN = 0, hintN = 0, trapN = 0, disN = 0, itemN = 0;
    int i, j, tempX, tempY;
    time_t t1 = 0, t2 = 0;
    int plusTime = 0, timeL = 0;
    int c, type = -1;
```

2 차원 배열은 결국 1 차원 배열이 계속 이어진 것이므로 각 행마다 각 행의 첫 주소를 통해 접근할 수 있다. 즉, 2 차원 배열의 포인터 변수는 1 차원 배열이 되어야 한다.



char(*mapPtr)[42] = map char map[22][42]

즉 이 배열을 이용해야 하는 함수는 포인터 배열을 넘겨받는다.

- 매개변수

매개변수명	자료형	설명
mapPtr	(char*)[]	main 함수의 지역변수, 게임의 맵인 이차원 배열
selectedLevel	int	이전 단계를 클리어했다면 자동으로 다음 단계로 넘어가기 위함

- 게임의 진행을 담당하는 startGame 함수

```
int startGame(char(*mapPtr)[ARR_HORZ], int selectedLevel) {
    int x = 1, y = 20;
    const int scoreX = 3, scoreY = 27;
    const int baseX = 8, baseY = 4;
    int level, score, fin = 0;
    int flagN = 0, hintN = 0, trapN = 0, disN = 0, itemN = 0;
    int i, j, tempX, tempY;
    time_t t1 = 0, t2 = 0;
    int plusTime = 0, timeL = 0;
    int c, type = -1;
```

- 변수

변수명	자료형	설명
x	int	사용자 캐릭터(◆)의 위치(x 값이자 인덱스)
y	int	사용자 캐릭터(◆)의 위치(y 값이자 인덱스)
scoreX	const int	점수와 시간을 출력하는 라인의 커서 위치(x 값)
scoreY	const int	점수와 시간을 출력하는 라인의 커서 위치(y 값)
baseX	const int	사용자 캐릭터가 움직일 수 있는 배열의 영역이 콘솔 원점(0, 0)으로부터 거리(x 값)
baseY	const int	사용자 캐릭터가 움직일 수 있는 배열의 영역이 콘솔 원점(0, 0)으로부터 거리(y 값)
level	int	사용자가 선택한 레벨(단계)
score	int	체력(점수)
fin	int	게임이 끝난 종류(time over, 체력 고갈, 성공)
flagN	int	깃발(가방)의 총 개수
hintN	int	가방 중 도움이 되는 가방의 개수
trapN	int	가방 중 위협이 되는 가방의 개수
disN	int	장애물(벽)의 개수
itemN	int	아이템(폭탄)의 개수
i	int	for 반복문의 index(x 용)
j	int	for 반복문의 index(y 용)
tempX	int	x 값에 대한 임시 저장용 변수
tempY	int	y 값에 대한 임시 저장용 변수
t1	time_t	게임이 시작된 시간을 저장
t2	time_t	while 루프가 돌아갈 때마다 현재 시간을 저장
plusTime	int	도움이 되는 가방에서 추가 시간을 얻었을 경우 추가되는 시간
timeL	int	해당 단계에서 주어지는 시간
c	int	사용자가 키보드에 입력하는 키의 ASCII 값을 저장
plus	int	도움이 되는 가방 혹은 위협이 되는 가방의 유형

- 맵 설정하기

```
/* 레벨에 따른 배치 */
if (level == 1) {
    flagN = 4;
    hintN = 3;
    trapN = 1;
    itemN = 2;
    score = 1000;
    timeL = 20;
}
```

각 단계 별로 깃발, 도움, 위협, 아이템, 시작 체력(점수)의 값을 대입해준다. 벽의 경우 `level * 10` 이다.

```
printf("\n   CUSTOM LEVEL!\n\n\n   전체 가방의 개수를 입력하세요(최대 30개 최소 15개) : ");
while (1) {
    scanf("%d", &flagN);
    if ((flagN < 15 || flagN > 30)) {
        printf("   다시 입력하십시오. : ");
    }
    else break;
}
```

커스텀 단계는 이와 같이 직접 입력 받는다.

```
// 깃발 배치(HINT)
for (i = 0; i < hintN; i++) {
    while (1) {
        tempX = (rand() % 20) * 2 + 1;
        tempY = (rand() % 20) + 1;

        if (mapPtr[tempY][tempX] == 0) {
            mapPtr[tempY][tempX] = 'H';
            break;
        }
    }
}
```

가방, 보물, 장애물, 아이템은 모두 이와 같은 알고리즘을 통해 배치한다. 이 때, 가로 즉 x 값은 홀수만 할당한다.

가로는 두 칸을 한 칸으로 처리하기 때문에 x 값은 1 부터 39 사이의 홀수로 배치해야 20 칸이 나오게 된다.

특수문자를 배열에 저장하는 것이 어렵기 때문에 배열에 실제로는 알파벳 대문자를 넣고, 이를 출력하는 과정에서 대치하기로 하였다.

도움 가방	위협 가방	보물(지도)	사용자 캐릭터	장애물	폭발 광선	아이템(폭탄)
H	T	X	O	D	Y	I
▶	▶	▶	◆	▦	※	◎

- `void drawScreen(char(*mapPtr)[42])` 함수는 다음과 같이 배열을 읽어 화면을 출력하여준다.

```
for (i = 0; i < ARR_VERT; i++) {
    for (j = 0; j < ARR_HORZ; j++) {
        if (mapPtr[i][j] == 'H') {
            printf("▶");
            j++;
        }
```

위와 같은 이중 for 문을 사용해 배열에 있는 문자에 따라 특수문자를 출력한다. 이때, 특수문자는 가로로 두 칸을 차지하므로 특수문자 출력 시 x 인 j 의 인덱스를 2 칸 할당하여 준다.

```
if (_kbhit()) {
    c = _getch();
    if (c == 'q') return 0;
    if (c == 224) {
        c = _getch();
        switch (c) {
            // UP
        case 72: {
            if (y > 1) {
                if (mapPtr[y - 1][x] != 'D') {
                    gotoXY(x + baseX, y + baseY);
                    printf("  ");
                    gotoXY(x + baseX, y - 1 + baseY);
                    printf("  ");
                    if (mapPtr[y - 1][x] == 'T') {
                        type = trap(mapPtr, &score, &x, &y, baseX, baseY, hintN, trapN);
                    }
                    else if (mapPtr[y - 1][x] == 'H') {
                        hint(mapPtr, &score, &plusTime);
                    }
                    else if (mapPtr[y - 1][x] == 'I') {
                        item(mapPtr, x, y - 1);
                    }
                    else if (mapPtr[y - 1][x] == 'X') {
                        fin = 1;
                    }
                    if (type != 1 && type != 2) y--;
                    gotoXY(x + baseX, y + baseY);
                    printf("◆");

                    if (level == 1) score -= 20;
                    else if (level == 2) score -= 30;
                    else if (level == 3) score -= 40;
                    else if (level == 4) score -= 50;
                    else if (level == 5) score -= 60;
                    else if (level == 6) score -= 80;

                    mapPtr[y][x] = 0;
                }
            }
            break;
        }
```

게임이 본격적으로 시작되는 `do-while` 문 내부 코드이다. `switch-case` 문 중 상향 방향키 부분 코드이다.

`_kbhit` 함수를 이용해 키가 입력되었을 때, <q>라면 게임을 종료하고, 방향키라면 코드를 실행한다.

먼저 진행하고자 하는 방향이 장애물(D)이라면 진행되지 않는다. 즉, 장애물이 아닐 때 먼저 커서를 이동한다. 상향은 y 축 방향으로 -1 만큼 이동하는 것이므로 이동하기 전 표시되어 있던 문자를 지우고 커서를 위치로 이동한다. 이후 이동한 이동한 곳의 배열 내부 문자를 읽는다.

1) 만약 보물(X)이라면 : fin 변수의 값을 1 로 변경한다.(fin 이 1 인 경우 게임 클리어)

2) 만약 아이템(I)이라면 :

- void item(char(*mapPtr)[42], int x, int y) 함수

매개변수명	자료형	설명
mapPtr	(char*)[]	main 함수의 지역변수, 게임의 맵인 이차원 배열
x, y	int	해당 아이템의 위치(즉, 현재 사용자의 위치)의 x, y 값

```
void item(char(*mapPtr)[ARR_HORZ], int x, int y) {
    int i, j;

    for (i = 1; i <= 20; i++) {
        for (j = 1; j < 40; j += 2) {
            if (i == y || j == x) {
                if (mapPtr[i][j] == 'D' || mapPtr[i][j] == 0)
                    mapPtr[i][j] = 'Y';
            }
        }
    }

    drawScreen(mapPtr);
    Sleep(1000);

    for (i = 1; i <= 20; i++) {
        for (j = 1; j < 40; j += 2) {
            if (mapPtr[i][j] == 'Y')
                mapPtr[i][j] = 0;
        }
    }
    drawScreen(mapPtr);
    return;
}
```

첫 번째 for 문에서는 해당 x, y 줄에 벽이나 빈칸이라면 광선(※)을 출력하는 'Y'로 채워준다. 광선이 십자로 뻗어 나가는 것 같이 보이기 위해서이다. 이 상태로 화면을 출력하고 1 초 후, 'Y'인 곳을 비워준다. 이렇게 되면 벽이 있던 곳은 비게 되어 해당 좌표 기준 십자로 벽이 사라진다.

3) 만약 도움이 되는 가방(H)라면 :

- void hint(char(*mapPtr)[42], int* score, int* plusTime) 함수

매개변수명	자료형	설명
mapPtr	(char*)[]	main 함수의 지역변수, 게임의 맵인 이차원 배열
scroe	int*	사용자의 체력(점수) 변수의 주소
plusTime	int*	추가 시간 변수의 주소

```
void hint(char(*mapPtr)[ARR_HORZ], int* score, int *plusTime) {
    switch (rand() % 3) {
```

rand 함수를 이용해 0~3 의 값을 얻는다.

3-i) 도움이 되는 가방 1(생존 시간 증가)

```
//시간 증가
case 0: {
    *plusTime += 5;
    gotoXY(8, 28);
    printf("생존 가능 시간이 5초 늘어났습니다.    ");
    return;
}
```

포인터를 이용해 추가 시간 변수의 값을 5 증가한다.

3-ii) 도움이 되는 가방 2(위협이 되는 가방 한 개 제거)

```
//remove one trap flag
case 1: {
    int i, j, n = 0, t = 0;
    for (i = 1; i <= 20; i++) {
        for (j = 1; j < 40; j += 2) {
            if (mapPtr[i][j] == 'T') n++;
        }
    }
    if (n == 0) { ... }
    else if (n != 1) {
        n = rand() % n + 1;
    }
    for (i = 1; i <= 20; i++) {
        for (j = 1; j < 40; j += 2) {
            if (mapPtr[i][j] == 'T') t++;
            if (t == n) {
                mapPtr[i][j] = 0;
                drawScreen(mapPtr);
                gotoXY(8, 28);
                printf("위협이 되는 가방 한 개를 제거했습니다.    ");
                return;
            }
        }
    }
    return;
}
```

먼저 남은 위협이 되는 가방의 개수를 센다. 만약 남은 위협이 되는 가방이 없을 경우(n = 0) 다른 이벤트를 발생시킨다. 만약 있을 경우 한 개의 위협이 되는 가방의 위치를 찾은 후 배열의 'T'를 0 으로 바꾸어 제거한다.

3-iii) 도움이 되는 가방 3(체력 랜덤 증가)

```
//plus score
case 2: {
    *score += (rand() % 10) * 10 + 10;

    gotoXY(8, 28);
    printf("약을 먹고 체력을 조금 회복합니다.     ");
    return;
}
```

rand 함수와 포인터를 이용해 점수(체력) 변수의 값을 10~100 중 랜덤으로 증가시킨다.

4) 만약 위협이 되는 가방(T)라면 :

- `void trap(char(*mapPtr)[42], int* score, int* x, int* y, int baseX, int baseY, int hintN, int trapN)` 함수

매개변수명	자료형	설명
mapPtr	(char*)[]	main 함수의 지역변수, 게임의 맵인 이차원 배열
scroe	int*	사용자의 체력(점수) 변수의 주소
x, y	int*	사용자의 위치 변수의 각 주소
baseX	int	사용자 캐릭터가 움직일 수 있는 배열의 영역이 콘솔 원점(0, 0)으로부터 거리(x 값)
baseY	int	사용자 캐릭터가 움직일 수 있는 배열의 영역이 콘솔 원점(0, 0)으로부터 거리(y 값)
hintN	int	가방 중 도움이 되는 가방의 개수
trapN	int	가방 중 위협이 되는 가방의 개수

```
int trap(char(*mapPtr)[ARR_HORZ], int *score, int* x, int* y, int baseX, int baseY, int hintN, int trapN) {
    int h = 0, t = 0;

    switch (rand() % 4) {
```

rand 함수를 이용해 0~4 의 값을 얻는다.

4-i) 위협이 되는 가방 1(모든 가방 랜덤 배치)

```
// 깃발 랜덤 배치
case 0: {
    int i, j, tempX, tempY;
    for (i = 1; i <= 20; i++) {
        for (j = 1; j < 40; j += 2) {
            if (mapPtr[i][j] == 'X' || mapPtr[i][j] == 'H' || mapPtr[i][j] == 'T') {
                if (mapPtr[i][j] == 'H')
                    h++;
                if (mapPtr[i][j] == 'T')
                    t++;
                mapPtr[i][j] = 0;
            }
        }
    }
```

배열 중 가방(도움, 위협, 지도 포함)들을 전부 삭제한다. 이 때 변수 h 와 t 를 이용해 남아 있는 도움 가방과 위협 가방의 개수를 센다.

이후 위 최초로 배치할 때와 마찬가지로 남은 개수에 대해 재배치한다.

4-ii) 위협이 되는 가방 2(초기 위치로 이동)

```
// 초기 위치로 이동
case 1: {
    gotoXY(*x + baseX, *y + baseY);
    printf("  ");
    *x = 1;
    *y = 20;
    gotoXY(8, 28);
    printf("처음 위치로 돌아왔습니다.            ");
    return 1;
}
```

사용자의 시작 위치인 (1, 20)으로 포인터를 이용해 위치를 이동한다.

4-iii) 위협이 되는 가방 3(체력 소량 감소 후 플레이어 캐릭터 위치 랜덤)

```
// 캐릭터 랜덤
case 2: {
    int tempX, tempY;
    mapPtr[tempY][tempX] == 0;
    gotoXY(*x + baseX, *y + baseY);
    printf("  ");

    while (1) {
        tempX = (rand() % 20) * 2 + 1;
        tempY = (rand() % 20) + 1;

        if (mapPtr[tempY][tempX] == 0) {
            *x = tempX;
            *y = tempY;
            break;
        }
    }

    gotoXY(8, 28);
    *score -= (rand() % 5) * 10 + 10;
    printf("폭발이 일어나 어딘가로 날아갔습니다.    ");
    return 2;
}
```

for 문을 이용해 배열 중 빈 칸을 찾아 랜덤으로 이동시킨다. 이후 체력을 10~50 랜덤으로 감소시킨다.

4-iv) 위협이 되는 가방 4(체력 랜덤 감소)

```
// 점수 차감
case 3: {
    *score -= (rand() % 10) * 10 + 10;
    gotoXY(8, 28);
    printf("독극물입니다. 체력이 줄어듭니다.       ");
    return 3;
}
```

rand 함수와 포인터를 이용해 점수(체력) 변수의 값을 10~100 중 랜덤으로 감소시킨다.

위의 모든 코드를 실행해 해당 턴(Turn)이 끝나면 사용자의 위치를 랜덤으로 이동하거나 처음 위치로 이동하게 되는 것이 아닌 상황일 때 현재 위치에 마름모(◆)를 출력해준다. 사용자의 위치가 변하는 경우에는 이동한 곳에 출력해야 하기 때문에 겹쳐진다. 또한 각 단계에 따라 전체 체력(점수)에서 일정 포인트를 차감한다.

```
    gotoXY(scoreX, scoreY);
    printf("HP : %d  /  ", score);
    type = -1;
    t2 = time(NULL);
    printf("TIME LEFT : %ds (<q>를 눌러 종료)", (timeL + plusTime) - (int)(t2 - t1));
    gotoXY(8, 28);

    if ((int)(t2 - t1) > (timeL + plusTime)) fin = -1;
} while (fin == 0 && score > 0);
```

시간이 초과될 경우 fin 값이 -1 이 되며 보물을 찾은 경우 0 이 된다. 혹은 점수가 0 점 이하로 떨어진 경우 do-while 루프를 탈출하게 된다.

루프를 탈출하면 게임 결과에 따라 각기 다르게 출력한다. 먼저 database 함수를 "played" 명령어로 호출해 data 매개변수에 현재 플레이한 레벨을 인자로 주어 플레이 횟수를 1 늘린다.

```
database("played", level);

if (fin == 1) {
    printf("\n     생존 성공!\n\n");
    printf("     점수(남은 HP) : %dP!", score);
    database("writeScore", score);
    database("cleared", level);
    if (database("returnScore", 0) == score) printf(" (최고 점수 달성!)");
    printf("\n\n");

    if (database("returnLevel", 0) == level && level != 5 && level != 6) {
        database("upLevel", 0);
        printf("     REGION %d(이)가 해금되었습니다!\n\n", database("returnLevel", 0));
        printf("     다음 지역으로 이동합니다! ");
        for (i = 3; i >= 0; i--) {
            printf("(%d)", i);
            Sleep(1000);
            printf("\b\b\b   \b\b\b");
        }

        return database("returnLevel", 0);
    }
    else if (level == 5) {
        ending();
        system("cls");
        printTitle();
        printLine();
        database("upLevel", 0);
        return 0;
    }
    else if (level == 6) {
        printf("     CUSTOM REGION 클리어! \n\n");
    }
}
```

먼저 fin 의 값이 1 인 경우 보물(지도)를 찾아 게임 클리어이다.

생존 성공 문구와 점수(잔여 체력)를 출력한다. database 함수를 호출해 점수를 저장하고 (함수에서 만약 최고 점수보다 낮다면 기록하지 않을 것이다) 해당 레벨의 클리어 횟수를 1 늘린다. 만약 최고 점수가 현재 점수와 같으면 "최고 점수 달성" 문구를 출력한다. 만약 플레이 한 레벨이 최고 해금 레벨과 같은데, 5, 6(Custom)이 아니라면 다음 단계를 해금해야 한다. database 함수를 "upLevel" 명령어로 호출해 레벨을 올린다. 그리고 해금된 내용을 출력하며 자동으로 다음 지역으로 이동하도록 한다.

```
/* 게임 시작 */
case 1: {
    temp = 0;
    do {
        temp = startGame(mapPtr, temp);
    } while (temp != 0);
    break;
}
```

main 함수의 코드이다. 이때 최고 해금 레벨 즉 다음 단계를 반환하여 자동으로 다음 단계로 넘어가도록 한다.

만약 클리어 한 레벨이 5 레벨이라면 엔딩 스토리를 출력과 함께 최고 해금 레벨도 1 증가시킨다.

클리어 레벨이 6(Custom)이라면 Custom Region 을 클리어했다는 문구를 출력해준다.

```
else {
    printf("\n     실패! 죽었습니다.. ");
    if (fin == -1) printf(" 시간이 너무 오래 지났습니다.\n\n");
    else printf(" 체력이 다 닳았습니다.\n\n");
    switch (rand() % 5) {
        case 0: {
            printf("     TIP : 이동을 효율적으로 해야 합니다. \n\n");
            break;
        }
        case 1: {
            printf("     TIP : 도구가 든 가방은 생존에 큰 도움이 됩니다. \n\n");
            break;
        }
        case 2: {
            printf("     TIP : 다음 지역를 해금하려면 바로 이전 지역을 클리어해야 합니다. \n\n");
            break;
        }
        case 3: {
            printf("     TIP : 폭탄을 이용하면 벽을 뚫어 빠른 길을 만들 수 있습니다. \n\n");
            break;
        }
        case 4: {
            printf("     TIP : 메인메뉴에서 \'통계\' 메뉴를 통해 실적을 확인해보세요. \n\n");
            break;
        }
    }
}
```

`fin` 의 값이 1 이 아닌 경우 게임 실패이다. 이 경우 중 `fin` 의 값이 -1 이라면 시간이 다 되어 실패한 것이므로 이에 대한 문구를 출력해준다. 모두 아닌 경우는 체력(점수)이 0 이하가 되어 죽은 경우이다.

`rand` 함수를 이용해 팁 5 개 중 1 개를 랜덤으로 출력해준다.

iv) 스토리 연출: `main` 함수에서 게임의 배경이 되는 배열을 선언해주고 이를 이용해 `game` 함수에서 진행한다.

```
char subtitle1[] = "당신은 생존에 성공했습니다!";
char subtitle2[] = "본래의 삶으로 돌아가 행복하게 살았습니다.";
char subtitle3[] = "Thank you for Playing...";
char subtitle4[] = "제작 : 202011353 이호은";
char subtitle5[] = "[Custom Region]이 해금되었습니다.";


printf("\n   ");
for (i = 0; i < strlen(subtitle1); i++) {
    putchar(subtitle1[i]);
    Sleep(80);
}
Sleep(1200);
```

스토리 연출을 위해 한 글자씩 출력하도록 하였다. 미리 배열에 문자열을 이용해 초기화한다.

이후 `for` 문을 이용해 한 자씩 읽고, 그 사이에 80ms 간격을 주어 글자가 한 글자씩 출력되도록 연출하였다.

v) 단계 선택 화면

```
int selectLevel(int level) {
    int i, j, slc;

    system("cls");
    printTitle();
    printLine();

    for (i = MIN_LVL; i <= MAX_LVL - 1; i++) {
        printf("  ▷ Region %d", i);
        if (i > level)
            printf("       (잠김) \n  ");
        else printf("\n  ");

        for (j = 1; j <= i; j++) {
            printf("★");
        }
        for (j = 5; j > i; j--) {
            printf("☆");
        }
        printf("\n\n");
    }
    printf("  ▷ Custom Region");
    if (i > level)
        printf("  (잠김) \n  ");
    else printf("\n  ");
    printf("? ? ? ? ? \n\n");
    printLine();
    printf("  <y>를 눌러 선택하십시오.\n");
    printf("  <q>를 메인메뉴로 돌아가기.");

    while ((slc = getMove(28, 4, 3, 1, 1, 6)) > level);

    return slc;
}
```

for 문을 이용해 자동으로 각 단계의 난이도 표시(별 5 개)와 이름, 잠김 표식을 출력되도록 하였다.

vi) 커서 이동

```
void gotoXY(int x, int y) {
    COORD Cur = { x, y };
    SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), Cur);
}
```

함수의 좌표를 인자로 전달하여 커서를 이동하는 역할을 하였다.

vii) 배열 및 데이터 초기화 함수

- `void clear(int type, char(*mapPtr)[42])` 함수 : 메인 화면에서 초기화를 담당한다.

```
/* 배열 초기화 */
if (type == 1 || type == 3) {
    for (i = 0; i < ARR_VERT; i++) {
        for (j = 0; j < ARR_HORZ; j++) {
            if (i == 0 || i == ARR_VERT - 1 || j == 0 || j == ARR_HORZ - 1) {
                mapPtr[i][j] = '#';
            }
            else {
                mapPtr[i][j] = 0;
            }
        }
    }
}
```

배열을 초기화 할 때 맨 끝에는 '#'을 입력해주어 화면 상에서도 출력되도록 하였다. 나머지는 0 으로 비워준다.

```
/* 데이터 초기화 */
else if (type == 2 || type == 3) {
    system("cls");
    printTitle();
    printLine();

    printf("\n  정말 게임을 초기화하시겠습니까?\n\n");
    printLine();
    printf("  예 <y>  아니오 <n> ");

    do {
        i = _getch();
    } while (i != 'y' && i != 'n');

    if (i == 'n') return;

    system("cls");
    printTitle();
    printLine();
    printf("\n  [!] 게임 초기화 시도...\n\n");
    Sleep(100);

    /* 데이터베이스 초기화 */
    database("clearAll", 0);

    printf("  [!] 게임이 초기화되었습니다. 시작화면으로 돌아갑니다.");
    for (i = 3; i >= 0; i--) {
        printf("(%d)", i);
        Sleep(1000);
        printf("\b\b\b   \b\b\b");
    }
}
```

실수로 선택하는 것을 대비해 다시 한 번 확인문을 넣었다. <y>를 누르면 진행하고, <n>을 누르면 취소한다.

게임을 초기화하면 database 함수의 "clearAll" 명령어를 통해 호출해 database 함수 내의 정적 변수의 값을 초기화한다.

4. 문제점과 해결책, 새롭게 발견한 점

i) 특수문자 출력에 관한 문제

위에서 언급하였듯 특수문자는 콘솔에서 두 칸을 차지하기 때문에 가로, 세로 크기를 정사각형으로 하는 데에 문제가 있었다.

이를 해결하기 위해 가로 40 칸, 세로 20 칸을 할당하였고, 가로로는 홀수 번째 인덱스에만 배치하도록 하였다. 이렇게 하면 마치 가로로 두 칸을 하나로 묶어 쓰는 것과 같은 효과를 낼 수 있었다.

콘솔 창에서 출력 뿐만 아니라 실제 char 형 배열에서도 특수문자는 두 칸을 차지했다. 게임에서 사용하는 대부분의 특수문자는 ASCII 코드 범위 내 문자로 표현이 불가능해서 문자열로 표현을 해야 했다. 이러한 이유로 배열에 직접 특수문자를 직접 대입하기에는 너무 알고리즘이 복잡하고 데이터 낭비가 될 것이라고 생각해 위에 언급하였듯 대문자를 대입하고 그 자리에 특수문자를 출력하는 알고리즘으로 해결할 수 있었다.

ii) 게임 시스템에 관한 문제

프로그램 실행을 성공한 것을 세 번째 시도 때였다. 맨 처음에는 배열을 활용해 배열 내에 x, y 도움, 위협 여부, 도움 위협 타입까지 전부 넣었고, 가방과 장애물들을 배치할 때 그 모든 배열과 겹치는 좌표가 있는지 확인해야 했다. 이렇게 되니 적어도 n^3 정도의 복잡도를 갖는 프로그램이 되었다. 한 배열을 채우기 위해 배열 내부를 이중 for 문으로 검사해야 했기 때문이다. 더욱이 값이 겹치는 경우라면 두 세 번 더 실행해야 했기 때문이다.

이를 혁신적으로 고치게 된 방법은 생각보다 간단한 방법이였다. 배열에 직접 넣는 방식이다. 만약 랜덤으로 뽑아낸 위치에 이미 다른 알파벳이 들어가 있으면 다시 찾으면 된다. 이것은 아무리 커도 상수의 복잡도를 가져 어마어마하게 프로그램 실행 속도가 빨라졌다. 이전의 경우 custom 단계를 할 때 가방의 개수를 최대(30 개)로 하면 거의 한 시간 넘게 걸리는 문제가 있었지만 이를 해결하자 프로그램 속도도 빨라지고 이미 존재하는 배열을 쓰니 메모리 관리 차원에서도 효율적인 프로그램이 되었다.

iii) 막을 수 없는 예외에 대한 문제

프로그램 실행 중 메뉴 입력과 키 입력에 관한 모든 예외는 조건문과 반복문 등을 이용해 해결하였지만 단 한가지 해결하지 못한 예외처리가 있다. 바로 다른 자료형의 데이터가 입력되는 경우이다. 실제 정수를 입력 받는 Custom 단계에서는 각각의 개수를 정수로 입력 받아야 하지만 사용자가 문자(작은 따옴표 없이) 혹은 문자열, 특수문자 같은 다른 자료형의 데이터를 입력하게 된다면 프로그램은 오류를 출력하며 종료된다. 이는 다른 프로그램처럼 예외처리 키워드를 이용해 구현할 수 있겠으나 배운 내용이 아니므로 생략하였다.

iv) 병렬 프로그래밍 문제

C 프로그래밍 수업 13 주차에 언급해주신 병렬 프로그래밍과 스레드(Thread)에 대해 고민해보니 지금 프로그램도 마찬가지고 키 입력과 깃발(가방)의 효과 처리, 시간 처리, 점수 처리 등 각자 독립적으로 실행되야 할 프로세스들이 하나의 do-while 문 안에서 순차적으로 실행되다 보니 출력에 있어 자그마한 문제들을 많이 겪었다. 실제로 이와 관련된 문제를 해결하기 위해 프로그램을 두 차례 갈아 엎기도 하였다. 이 문제는 교수님께서 언급해주신 스레드를 활용해야 더욱 완벽한 프로그램이 될 것 같다.

v) 불가능한 게임이 생기는 문제

게임을 실행하다 보면 사용자 캐릭터 주변에 벽이 형성되어 움직이지도 못하는 경우나 보물이 위치한 깃발로 가는 경로가 하나도 없던지, 주어진 횟수를 이용해 이동할 수 없는 경우가 존재한다. 사실 이는 검사를 하려면 캐릭터의 시작 위치부터 보물(지도)까지 막힘이 없는지 검사하여야 하나 이를 구현하기는 매우 어렵다.

일단 눈에 보이도록 불가능한 경우를 막기 위해 맵에 벽, 가방을 배치하기 전에

```
for (i = 17; i <= 20; i++) {
    for (j = 1; j < 6; j += 2) {
        mapPtr[i][j] = 'x';
    }
}
```

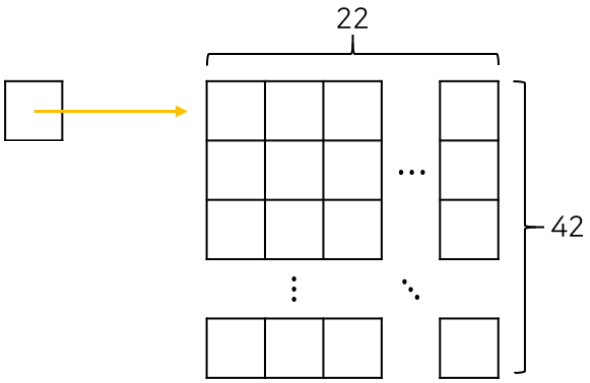
→

```
for (i = 17; i <= 20; i++) {
    for (j = 1; j < 6; j += 2) {
        mapPtr[i][j] = 0;
    }
}
```

이를 이용해 사용자의 캐릭터 초기 위치 주변 3x3 크기에 임시로 값을 넣어 주변에 보물(지도) 혹은 장애물들이 배치될 수 없도록 하였다. 모든 요소 배치 이후 다시 지워주었다.

vi) 2 차원 배열 포인터에 관한 문제

2 차원 배열을 다른 함수에 넘길 때 어떻게 넘겨야 할지 많이 고민했다. 수업 내에선 2 차원 배열에 대한 포인터 내용을 직접적으로 배운 적은 없기에 개인적으로 생각을 해보았다. 수업 내용에서 2 차원 배열은 결국 1 차원 배열을 알기 쉽게 표시한 것이므로 각 행의 맨 앞 주소만 저장한 배열을 이용하면 결국 배열 이름은 그 자체가 맨 앞 주소를 나타내기에 동일한 효과를 지닐 것이라고 생각하여 사용하게 되었다.



`char(*mapPtr)[42] = map` `char map[22][42]`

실제로 이러한 구조를 생각하고 포인터 배열을 선언하여 사용하였다.

굳이 배경이 되는 배열을 `main` 함수에 선언한 이유는 `main` 함수는 프로그램이 한 번 실행되면 종료될 때까지 항상 실행 중이기 때문이며, 게임을 실행하는 함수 외에도 화면 출력, 배열 초기화와 같은 함수에서도 이 배열을 이용하기 때문에 번거롭더라도 `main` 함수에 선언해주었다.

vii) 함수형 프로그래밍과 포인터 매개변수에 관한 문제

프로그램을 단일 함수 내에 작성하는 것보다 기능 별로 함수로 나누어 작성하는 것이 프로그램 효율과 코딩 효율에 좋다. 이러한 이유로 C 파일과 함수를 기능 별로 나누어 프로그래밍 하였으나 게임이 복잡하기 때문에 함수의 매개변수가 굉장히 많아지는 경우가 생겼다. 또한, 다른 함수의 지역변수 값을 변경하기 위해서는 포인터 값을 넘겨야 하고, 특히 2 차원 배열을 사용하는 게임이므로 2 차원 배열의 주소를 저장하는 배열 포인터 변수를 넘겨야 한다. 이러한 점이 처음엔 다소 어렵게 느껴졌으나 효율적으로 프로그래밍을 하기 위해서는 필수적인 요소라고 생각했고, 많은 시간을 이 프로그램 코딩에 투자한 결과 어느 정도 익숙해졌다.

5. 배우고 느낀 점

코딩은 고등학교 1 학년 때 구글, MIT 의 "앱인벤터 2"를 통해 접해 고등학교 2 학년 때 정보과학(C 언어), 자료구조를 배우며 코딩을 익혔다. 교내·외 대회에 참가하며 실력도 길렀지만 대학교에 진학하여 수업을 들어보니 C 언어에 대해 정말 모르는 부분이 많았다는 생각이 들었다. 특히 이번 프로젝트를 하며 배웠던 점을 세세하게 다시 복습할 수 있어서 정말 많이 성장했다고 생각한다.