

프로젝트 2. 스네이크 게임

핵심 키워드	스네이크 게임, 콘솔, time.h, stdlib.h, 랜덤, 시간
스네이크 게임은 매우 유명한 콘솔 게임 중 하나이다. 무작위로 생성되는 사과를 먹으면 뱀이 길어지고, 벽이나 자신의 몸에 닿으면 게임에서 패배한다. 게임의 목표는 뱀의 길이를 최대한 길게 만드는 것이다. 이번 프로젝트에선 14챕터에서 배운 콘솔 제어와 C 언어의 표준 함수들을 이용하여 스네이크 게임을 직접 구현해보자.	

1. 문제 파악하기

스네이크 게임의 규칙은 간단하지만 이를 구현하기 위한 요소들을 파악해보면 생각보다 그 수가 많다는 것을 느낄 수 있을 것이다.

(1) 요소 분할

- | |
|-------|
| 1. 뱀 |
| 2. 사과 |
| 3. 벽 |

게임을 이루는 요소는 뱀과 사과, 그리고 벽이 있다. 이 세 요소는 콘솔에서 보이면서 게임에 영향을 준다. 그렇기에 스네이크 게임은 이 요소들을 중심으로 구성되어야 한다.

이동
길이 증가
패배
사과 랜덤 생성

세 요소로부터 얻을 수 있는 핵심적인 기능은 이동과 길이 증가, 사과 랜덤 생성, 그리고 패배가 있다. 열거된 네 기능이 가장 중요한 기능이므로 앞으로의 설계는 네 기능을 중심으로 하되, 필요에 따라 함수를 더 나누어 구현한다.

(2) 함수 분할

요소에 대한 기능은 곧 하나의 함수가 된다. 위에서 정리한 기능을 바탕으로 기본적인 함수를 나열하면 다음과 같다.

이동()
길이_증가()
패배()
사과_생성()

출력()

입력()

벽과 과일, 뱀을 출력하는 기본적인 함수를 구상하고, 여기에 추가로 현재 뱀의 길이를 숫자로 확인할 수 있도록 점수 출력 함수를 추가한다. 입력 함수는 키보드로부터 방향키 입력을 받고 가장 최근에 입력한 방향을 저장하는 용도로 사용될 것이다.

2. 설계하기

게임을 구성하는 기본적인 함수를 구상했으니, 이제 이 함수를 연결하여 프로그램의 전체적인 틀을 만들 차례이다.

(1) 게임 루프

게임이 끝날 때까지 반복한다

입력

업데이트

렌더

일반적인 게임 루프는 입력과 업데이트, 렌더를 반복한다. 뼈대는 이 게임 루프를 바탕으로 하겠으나, 우리는 여기서 효과적인 처리를 위해 약간의 변화를 추가할 것이다.

게임이 끝날 때까지 반복한다

입력

업데이트 시간이 되었다면?

업데이트

렌더

스네이크 게임은 컴퓨터의 연산량을 100% 사용할 필요가 없다. 뱀이 1초에 한 칸씩 움직인다면 업데이트와 렌더를 1초에 한 번만 수행하면 되기 때문이다. 이를 위해 이전 업데이트에서부터 지금까지 시간이 얼마나 흘렀는지를 계산하고, 뱀의 이동 속도에 맞춰서 업데이트의 횟수를 제한한다.

(2) 함수 설계

상당수의 함수는 간단한 기능을 하지만 일부 함수는 다른 함수와 연계하여 연산이 이루어지는 경우가 있다.

이동

이동할 위치를 계산한다.

이동할 위치에 몸 또는 벽이 있다면?

패배한다.

이동할 위치에 사과가 있다면?

사과를 먹는다.
이동한다.

뱀의 이동은 패배하거나 사과를 먹는 함수와 연계되어 있다. 뱀이 이동할 위치에 무엇이 있느냐에 따라 각 연산으로 분기하기 때문이다.

사과 먹기
길이를 증가시킨다.
사과를 생성한다.

사과를 먹으면 항상 길이가 증가하고 새로운 사과가 생성된다. 여기서 사과 먹기와 길이 증가는 항상 붙어있으므로 이 둘을 묶어 하나의 함수로 만든다.

사과 생성
반복한다:
 랜덤한 위치에 사과를 생성한다.
 그 위치에 뱀의 몸이 존재하지 않는다면?
 종료한다.

사과가 뱀의 몸에 생성되면 안되므로 뱀의 몸이 아닌 위치에 생성될 때까지 반복한다. 운이 나쁘다면 하나의 사과를 생성하기 위해서 많은 반복이 있을 수 있지만, 확률적으로 성능에 문제가 있을 만큼 반복이 될 가능성은 없고, 이 방법이 구현이 더 간단하다.

3. 구현하기

만들어야 할 함수들을 나열하고 그 함수들 사이의 관계를 정했으면, 이제 본격적으로 프로그램을 만들 차례이다. 여기서는 프로그램을 5개의 소스 파일로 나누어서 만들었으며 각 파일 사이의 값을 전달해야 하는 것은 전역 변수로 만들었다. 위에서 설명한 함수 외에 편의를 위한 함수도 포함되어 있다.

(1) input

```
input.h
typedef enum dir {
    RIGHT, LEFT, UP, DOWN
} dir;

void input(); // 입력 처리 함수
```

input은 키보드 입력을 담당한다.

```

input.c
#include <ncurses.h>
#include "input.h"

// 뱀이 이동할 방향
dir SNAKE_DIR = RIGHT;

void input()
{
    int code = getch();
    if (code == 259)
        SNAKE_DIR = UP;
    else if (code == 258)
        SNAKE_DIR = DOWN;
    else if (code == 260)
        SNAKE_DIR = LEFT;
    else if (code == 261)
        SNAKE_DIR = RIGHT;
}

```

input()은 getch()를 이용하여 방향키 입력을 받는다. 스네이크 게임은 한 번 입력한 방향은 다른 방향을 누르기 전까지 지속되므로, 이전의 입력을 저장하는 변수 SNAKE_DIR이 있다.

(2) render

```

render.h
void initRender(); // 시작 렌더 함수
void renderMove(); // 이동 렌더 함수
void renderEat(); // 사과를 먹었을 때의 렌더 함수

```

render는 콘솔에 대한 렌더 함수들이 모여있다. 렌더는 3종류로 시작했을 때, 이동했을 때, 그리고 사과를 먹었을 때이다.

```

render.c ... 1
#include <stdio.h>
#include <ncurses.h>
#include "render.h"

```

```
extern const int MAP_X;
extern const int MAP_Y;
extern const int START_X;
extern const int START_Y;
extern int SNAKE_LEN;
extern int APPLE_X;
extern int APPLE_Y;
extern int SNAKE_POS[128][2];
extern int PRE_POS[2];

void initRender()
{
    // 벽 렌더
    attron(COLOR_PAIR(1));
    for (int i = 0; i < MAP_X + 2; i++)
    {
        mvprintw(0, i * 2, " ");
        mvprintw(MAP_Y + 1, i * 2, " ");
    }
    for (int i = 0; i < MAP_Y + 2; i++)
    {
        mvprintw(i, 0, " ");
        mvprintw(i, (MAP_X + 1) * 2, " ");
    }

    // 뱀 렌더
    attron(COLOR_PAIR(2));
    for (int i = 0; i < SNAKE_LEN; i++)
        mvprintw(START_Y, 2 * (START_X + i), " ");

    // 사과 렌더
    attron(COLOR_PAIR(3));
    mvprintw(APPLE_Y, APPLE_X * 2, " ");
}
```

```

// 점수 렌더
char str[128];
sprintf(str, "score:%d", SNAKE_LEN);
attron(COLOR_PAIR(4));
mvprintw(MAP_Y + 3, 2, str);

refresh();
}

```

render는 다른 파일의 여러 값들을 바탕으로 렌더링한다. initRender()에선 게임이 시작되고 나서 필요한 벽, 사과, 뱀, 점수를 그려내는 역할이다.

```

render.c ... 2
void renderMove()
{
    // 꼬리 제거
    attron(COLOR_PAIR(4));
    mvprintw(PRE_POS[1], PRE_POS[0] * 2, " ");

    // 머리 추가
    attron(COLOR_PAIR(2));
    mvprintw(SNAKE_POS[SNAKE_LEN - 1][1], SNAKE_POS[SNAKE_LEN - 1][0] * 2, " ");

    refresh();
}

```

renderMove()는 사과를 먹지 않고 이동했을 때를 렌더링하는 함수이다. 사과를 먹지 않았다면 뱀의 길이는 변하지 않고, 뱀의 전체 모습을 보면 꼬리가 한 칸 줄고 머리가 한 칸 늘어난다. 변하지 않는 다른 모든 몸을 다시 그리는 것은 비효율적이므로 여기서는 꼬리를 제거하고 머리를 추가하는 두 부분만 다시 그린다.

```

render.c ... 3
void renderEat()
{
    // 머리 추가
    attron(COLOR_PAIR(2));

```

```

    mvprintw(SNAKE_POS[SNAKE_LEN - 1][1], SNAKE_POS[SNAKE_LEN - 1][0] *
2, " ");

    // 사과 렌더
    attron(COLOR_PAIR(3));
    mvprintw(APPLE_Y, APPLE_X * 2, " ");

    // 점수 렌더
    char str[128];
    sprintf(str, "score:%d", SNAKE_LEN);
    attron(COLOR_PAIR(4));
    mvprintw(MAP_Y + 3, 2, str);

    refresh();
}

```

반대로 사과를 먹었다면 길이가 하나 증가하여 꼬리가 없어지지 않는다. 따라서 이 경우 꼬리를 제거하지 않고 머리를 추가한 후 사과의 위치와 점수를 업데이트하도록 한다.

(3) snake

```

snake.h
typedef enum moveResult {
    EAT, MOVE, FAIL
} moveResult;

void init(); // 뱀 시작 설정 함수
moveResult moveSnake(); // 이동 함수
void newApple(); // 사과 생성 함수
void eat(int x, int y); // 사과 섭취 함수

```

스네이크 게임의 핵심 기능인 snake이다. 앞선 설계에서 설명한 함수가 그대로 있는데, 한 가지 특징으로 렌더링 자체가 사과를 먹는 경우와 먹지 않는 경우로 나뉘므로 이를 알아낼 수 있도록 이동의 결과를 반환하도록 하였다.

```

snake.c ... 1
#include <stdlib.h>
#include <time.h>

```

```

#include "snake.h"
#include "input.h"

const int START_X = 3; // 뱀의 시작 위치 x
const int START_Y = 8; // 뱀의 시작 위치 y
int SNAKE_LEN = 3; // 뱀의 길이
int SNAKE_POS[128][2];
int PRE_POS[2];

int APPLE_X = 12; // 사과의 위치 x
int APPLE_Y = 8; // 사과의 위치 y

extern dir SNAKE_DIR;
extern const int MAP_X;
extern const int MAP_Y;

void init()
{
    srand(time(NULL)); // 랜덤 함수의 시드값을 설정
    for (int i = 0; i < SNAKE_LEN; i++)
    {
        // 뱀 몸 위치 설정
        SNAKE_POS[i][0] = START_X + i;
        SNAKE_POS[i][1] = START_Y;
    }
}

```

snake.c에는 뱀의 길이와 몸의 위치, 사과의 위치를 나타내는 전역 변수가 있다. init()은 뱀의 길이와 시작 위치에 맞게 몸 위치 배열을 초기화하는 역할을 한다. init()의 상단에 있는 srand()는 stdlib.h에 있는 함수로, 무작위 값을 뽑아내기 위한 시드를 설정한다. 시드가 같다면 랜덤 함수의 결과가 똑같이 나오기 때문에 시드 값을 현재 시간으로 설정하여 매 게임마다 다른 위치에 사과가 생성되도록 유도한다.

snake.c ... 2

```

moveResult moveSnake()
{
    int headX = SNAKE_POS[SNAKE_LEN - 1][0], headY =

```



```

SNAKE_POS[SNAKE_LEN - 1][1];

if (SNAKE_DIR == RIGHT)
    headX++;
else if (SNAKE_DIR == LEFT)
    headX--;
else if (SNAKE_DIR == DOWN)
    headY++;
else if (SNAKE_DIR == UP)
    headY--;

// 머리가 벽에 닿았다면 패배
if (headX == 0 || headY == 0 || headX == MAP_X + 1 || headY == MAP_Y
+ 1)
    return FAIL;

// 머리가 몸에 닿았다면 패배
for (int i = 1; i < SNAKE_LEN; i++)
    if (SNAKE_POS[i][0] == headX && SNAKE_POS[i][1] == headY)
        return FAIL;

// 머리가 사과에 닿았다면 먹음
if (APPLE_X == headX && APPLE_Y == headY)
{
    eat(headX, headY);
    return EAT;
}

// 제거될 꼬리를 설정하고, 모든 몸을 한 칸 옮기고, 머리를 추가
PRE_POS[0] = SNAKE_POS[0][0];
PRE_POS[1] = SNAKE_POS[0][1];
for (int i = 1; i < SNAKE_LEN; i++)
{
    SNAKE_POS[i - 1][0] = SNAKE_POS[i][0];
    SNAKE_POS[i - 1][1] = SNAKE_POS[i][1];
}

```

```
SNAKE_POS[SNAKE_LEN - 1][0] = headX;
SNAKE_POS[SNAKE_LEN - 1][1] = headY;
return MOVE;
}
```

게임의 핵심 기능인 move()는 input()의 결과를 통해 다음으로 이동할 위치를 구하고, 그 위치가 몸 또는 벽이라면 FAIL, 사과라면 EAT, 무엇도 아니라면 MOVE를 반환한다. 만약 사과라면 사과를 먹었을 때의 함수인 eat()을 호출하며 그냥 이동이라면 뱀 몸 위치 배열의 값을 갱신한다.

```
snake.c ... 3
void newApple()
{
    int appleX, appleY, isCreated = 0;

    while (!isCreated)
    {
        // 랜덤한 값으로 사과의 위치를 설정
        appleX = rand() % MAP_X + 1;
        appleY = rand() % MAP_Y + 1;
        isCreated = 1;

        // 사과의 위치에 뱀의 몸이 있다면 다시 시도
        for (int i = 0; i < SNAKE_LEN; i++)
        {
            if (SNAKE_POS[i][0] == appleX && SNAKE_POS[i][1] == appleY)
            {
                isCreated = 0;
                break;
            }
        }
    }

    // 뱀의 몸이 없는 위치에 사과 생성
    APPLE_X = appleX;
    APPLE_Y = appleY;
}
```

```

}

void eat(int x, int y)
{
    // 사과의 위치에 새로운 몸을 추가
    SNAKE_POS[SNAKE_LEN][0] = x;
    SNAKE_POS[SNAKE_LEN][1] = y;
    SNAKE_LEN++;

    // 사과 생성
    newApple();
}

```

newApple()은 사과를 생성하는 기능, eat()은 길이를 증가시키고 newApple()은 사과를 생성한다. rand()는 srand()로 설정된 시드를 바탕으로 무작위 값을 뽑아내는데, 여기서 나머지 연산을 사용하면 0부터 내가 원하는 값까지의 무작위 값을 얻을 수 있다. 여기서 설정된 무작위 위치에 뱀의 몸이 있다면 다시 위치를 설정하고, 뱀의 몸이 없다면 사과의 위치를 설정한 후 종료한다.

(4) main

```

main.c ... 1
int main()
{
    initNcurses();
    init(); // 뱀 기본값 설정
    initRender(); // 시작하기 전의 렌더
    gameLoop(); // 게임 루프
    endwin();
}

```

main 함수가 있는 main.c는 게임을 시작하기 전 설정 함수들을 호출한 다음 게임 루프를 시작한다.

```

main.c ... 2
void initNcurses()
{
    initscr();
}

```

```

start_color();
curs_set(FALSE);
noecho();
keypad(stdscr, TRUE);
nodelay(stdscr, TRUE);

init_pair(1, COLOR_WHITE, COLOR_WHITE); // 벽
init_pair(2, COLOR_WHITE, COLOR_BLUE); // 뱀
init_pair(3, COLOR_WHITE, COLOR_RED); // 사과
init_pair(4, COLOR_WHITE, COLOR_BLACK); // 기본
}

```

initNcurses()는 프로그램에서 필요한 ncurses의 함수들을 호출하는 부분이다. 여기에는 입력을 기다리지 않는 nodelay(), 색을 설정하는 init_pair(), 입력한 키를 출력하지 않는 noecho() 등을 모두 호출한다.

```

main.c ... 3
void gameLoop()
{
    clock_t pre = clock(); // 이전 틱의 시간 (ms 단위)
    while (1)
    {
        input(); // MSPT와 관계 없이 입력을 받음

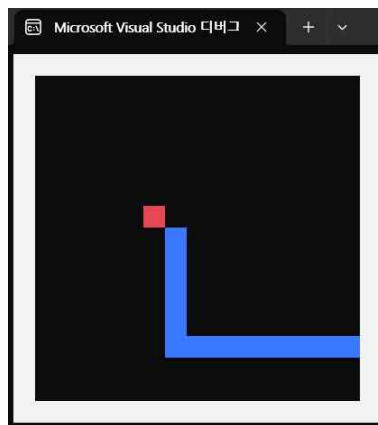
        clock_t now = clock(); // 현재 시간 (ms 단위)
        if (now - pre >= MSPT * CLOCKS_PER_SEC / 1000) // 이전 틱으로부터
지금까지 MSPT 이상 지났을 경우
        {
            moveResult result = moveSnake();
            if (result == FAIL) // 게임 패배
                break;
            else if (result == EAT) // 사과를 먹었을 때
                renderEat();
            else if (result == MOVE) // 먹지 않았을 때
                renderMove();
            pre = now;
        }
    }
}

```

```
}
}
```

위에서 정의된 MSPT는 1틱에 몇 초의 시간이 소요되는지를 나타내는 변수이다. time.h의 clock()은 현재 시간을 클락 단위로 얻을 수 있다. 클락을 CLOCKS_PER_SEC로 나누면 그 시간을 초 단위로 바꿀 수 있고, 이전의 한 시점에서 저장한 clock()과 지금 시간의 clock()을 계산하여 몇 초가 지났지를 알 수 있다. 여기서는 이 값을 1000으로 곱하여 보다 상세하게 시간을 설정할 수 있도록 밀리초로 사용하였다.

(6) 실행 결과



프로그램을 시작하면 흰 벽과 빨간 사과, 그리고 파란 뱀으로 이루어진 콘솔 게임이 시작된다. 방향키를 누르면 뱀의 이동 방향이 전환되고, 뱀이 사과를 먹으면 길이가 증가하여 점수가 오른다.

이번 프로젝트에서 만든 스네이크 게임은 콘솔 게임들 중에서는 구현이 쉬운 편에 속하지만, C 언어를 배우면서 활용한 많은 지식들이 한 번에 녹아들었음을 알 수 있었을 것이다. 완전히 새로운 작품을 만드는 것도 훌륭하지만, 다른 누군가가 만들어낸 작품을 따라서 만드는 것 또한 여러분의 실력을 키우는 좋은 연습이 된다. 기회가 된다면 스네이크 게임을 넘어서 다양한 프로젝트에 도전해보고, 어떻게 설계를 해야 더 나은 프로그램을 만들 수 있는지 고민하면서 연습하는 시간을 갖길 바란다.