**C++프로그래밍**

**프로젝트**

|  |  |
| --- | --- |
| 프로젝트 명 | *Snake* |
| 팀 명 | *CMake* |
| 문서 제목 | 결과보고서 |

|  |  |
| --- | --- |
| **Version** | 1.0 |
| **Date** | 2024-06-06 |

|  |  |
| --- | --- |
| **팀원** | 여수민(팀장) |
| 김강한 |
| 김혜인 |
| 임민석 |

|  |
| --- |
| **CONFIDENTIALITY/SECURITY WARNING**  이 문서에 포함되어 있는 정보는 국민대학교 소프트웨어융합대학 소프트웨어학부 및 소프트웨어학부 개설 교과목 C++프로그래밍 수강 학생 중 프로젝트 “Snake”를 수행하는 팀 “CMake”의 팀원들의 자산입니다. 국민대학교 소프트웨어학부 및 팀 “CMake”의 팀원들의 서면 허락없이 사용되거나, 재가공 될 수 없습니다. |

**문서 정보 / 수정 내역**

|  |  |
| --- | --- |
| **Filename** | 최종보고서-snakedoc |
| **원안작성자** | 여수민, 김강한, 김혜인, 임민석\_ |
| **수정작업자** | 여수민, 김강한, 김혜인, 임민석\_ |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 수정날짜 | 대표수정자 | Revision | 추가/수정 항목 | 내 용 |
| 2024-06-06 | 여수민 | 1.0 | 최초 작성 |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**목 차**

[1 개요 4](#_Toc43103653)

[2 개발 내용 및 결과물 5](#_Toc43103654)

[2.1 목표 5](#_Toc43103655)

[2.2 개발 내용 및 결과물 6](#_Toc43103656)

[2.2.1 개발 내용 6](#_Toc43103657)

[2.2.2 시스템 구조 및 설계도 6](#_Toc43103658)

[2.2.3 활용/개발된 기술 6](#_Toc43103659)

[2.2.4 현실적 제한 요소 및 그 해결 방안 6](#_Toc43103660)

[2.2.5 결과물 목록 7](#_Toc43103661)

[3 자기평가 8](#_Toc43103662)

[4 참고 문헌 8](#_Toc43103663)

[5 부록 8](#_Toc43103664)

[5.1 사용자 매뉴얼 8](#_Toc43103665)

[5.2 설치 방법 8](#_Toc43103666)

# 개요

|  |
| --- |
| **평가기준 (10점)**  **프로젝트를 완성하기 위해 사용한 개발 방법을 기술하세요.**  **또한 사용하고 있는 외부 라이브러리와 해당 라이브러리를 획득/설치하는 방법을 기술하세요.** |

**1.1 개발 구조 및 방법**

* + 개발환경은 ubuntu에서 진행되었으며, ncurses 기능을 활용하여 본 게임의 UI를 구현하였다. 조작의 경우, ‘w’, ‘a’, ‘s’, ‘d’로 각 상, 좌, 하, 우로 방향키를 설정하였다. 프로젝트의 구조는 크게 게임의 구성요소를 담당하는‘item’, ‘snake’, ‘snake\_game’와 윈도우창을 관리하는 ‘board’ ‘missionboard’, ‘leaderboard’ 등으로 개발 구조가 이루어져있다. Snake\_game내에 board, snake, item 등의 객체를 만들어 게임 조작이 이루어지고, 이때 게임을 진행하며 얻어지는 점수들은 missionboard를 통해 화면에 보여진다. 게임이 클리어 또는 종료가 되면 사용자와 점수가 leaderbord를 통해 기록된다.
    1. board: map을 형성하고, 맵 내에 있는 문자를 찾아내는 역할을 한다.
    2. item: gate 또는 아이템의 클래스를 관리한다.
    3. snake: 사용자가 직접 조작하는 메인 snake의 구조를 관리한다. 이때 queue를 사용하여 snake가 이동하는 모습을 구현하였다.
    4. snake\_game: 모든 요소들이 생성되며, 전체적인 게임을 조작하는 로직으로 구성되어 있다.
    5. missionboard: 점수와 스테이지를 관리한다.
    6. leaderboard: 게임의 사용자 및 점수 등의 기록을 관리한다.

**1.2 사용한 외부 라이브러리 설명 및 설치법**

**1.2.1 ncurses (version 6.2)**

* + Curses 라이브러리는 터미널 표시장치를 조작할 수 있는 함수를 제공하며, 윈도우 데이터 구조이다.
  + 설치방법은 다음과 같다. (다음 코드는 모두 터미널에서 진행된다.)
  1. Ncurses 다운로드

|  |
| --- |
| [사용자]:~$wget https://ftp.gnu.org/pub/gnu/ncurses/ncurses-6.2.tar.gz -P ~/Downloads |

(2) 다운받은 파일 압축 해제

* + 다운 받은 경로를 입력하고(ex: cd Downloads), 다음 명령어를 입력하여 압축 해제를 진행한다. 경로를 확인하고 싶다면 (ls ./ Downloads)를 입력한다.

|  |
| --- |
| [사용자]:~$ cd Downloads  [사용자]:~/Downloads$ tar xzfv ncurses-6.2.tar.gz |

(3) Ncurses 설정

* ‘ls’를 눌러 압축 해제된 파일을 확인하고 ‘cd ncurses-6.2’를 입력하여 디렉터리를 이동한다. 이때, make가 필요하므로, ‘sudo apt install make’를 입력하여 설치받는다.
* 이때, ‘./configure’만 입력하면 /usr/local 경로에 설치가 진행된다. 따라서 만약 opt/ncurses 경로로 설치하고 싶다면 ‘./configure --prefix/opt/ncurses’로 입력한다.

|  |
| --- |
| [사용자]:~/Downloads/ncurses-6.2$ ./configure |

(4) Ncurses 빌드

* Make를 이용해서 빌드라고, make install을 사용해서 인스톨을 수행한다.

|  |
| --- |
| [사용자]:~/Downloads/ncurses-6.2$ make  ...  [사용자]:~/Downloads/ncurses-6.2$ sudo make install  ... |

(5) 설치 확인

* + ‘ls -la /opt/ncurses’를 입력하여 설치가 제대로 진행되었는지 확인한다.

|  |
| --- |
| [사용자]:~$ ls –la/opt/ncurses |

### 1.2.2 queue

- queue라이브러리는 표준 템플릿 라이브러리(STL)에서 제공하는 FIFO(First-In-Frist\_Out) 자료 구조이다.

- 사용하기 위한 헤더 파일작성은 다음과 같다.

|  |
| --- |
| #include <queue> |

### string

- string 라이브러리는 C++ 표준 라이브러리에서 문자열 클래스 string을 제공한다.   
해당 프로젝트에 사용된 함수들은 다음과 같다.

* ‘length’: 문자열의 길이를 반환한다.
* ‘c\_str’: char 형태의 문자열로 반환한다.

- 사용하기 위한 헤더 파일 작성은 다음과 같다.

|  |
| --- |
| #include <string> |

### fstream

- fstream 라이브러리는 파일 입출력을 다루기 위한 헤더 파일이다.  
해당 프로젝트에 사용된 함수들은 다음과 같다.

* ‘ifstream’ : 파일 데이터를 읽는다.
* ‘ofstream’: 파일 데이터를 작성한다.

- 사용하기 위한 헤더 파일 작성은 다음과 같다.

|  |
| --- |
| #include <fstream> |

### sstream

- sstream 라이브러리는 문자열 스트림을 다루기 위한 헤더 파일이다.

해당 프로젝트에 사용된 함수들은 다음과 같다.

* + ‘stringstream’: 문자열에서 사용자가 원하는 자료형의 데이터를 추출 할 수 있다.
* 사용하기 위한 헤더 파일 작성은 다음과 같다.

|  |
| --- |
| #include < sstream > |

### chrono

- chrono 라이브러리는 STL중에서 시간 관련 기능을 하는 라이브러리로, 다양한 정확도로 시간을 측정할 수 있다. 이 라이브러리는 time라이브러와 다르게 나노 밀리 초 단위까지 측정 가능하여 특정 시간 구간에 걸린 시간을 나노 초까지 작은 단위로도 얻을 수 있다.

* 사용하기 위한 헤더 파일 작성은 다음과 같다.

|  |
| --- |
| #include < chrono > |

### set

- set 라이브러리는 STL(Standard Template Library)에서 제공하는 기능으로 고유한 값을 저장하고 정렬된 순서로 관리할 때 사용한다.

### xterm(in run.sh)

- xterm은 terminal emulator인데, xterm에서 사용하는 control sequence를 이용해서 터미널의 크기를 조정하는 것이 가능하다.

|  |
| --- |
| *P s* = 8 ; *height ; width* → Resize the text area to [height;width] in characters. |
| In ascii: “\033[8;*h*;*w*t” |

# 개발 내용 및 결과물

## 목표

|  |
| --- |
| **작성요령 (10점)**  **프로젝트의 목표를 기술하세요. 각 단계별 목표를 구체적으로 쓰세요.** |

|  |  |  |
| --- | --- | --- |
| 적용단계 | 내용 | 적용 여부 |
| 1단계 | Map의 구현 | 적용 |
| 2단계 | Snake 표현 및 조작 | 적용 |
| 3단계 | Item 요소의 구현 | 적용 |
| Item 추가 구현1 & 새로운 wall 추가 구현 | 적용 |
| Bonus Map 요소 추가 | 적용 |
| Item update 구현 | 적용 |
| Item 추가 구현2 & item 개수 제한 | 적용 |
| 4단계 | Gate 요소의 구현 | 적용 |
| 5단계 | 점수 요소의 구현 | 적용 |
| 미션 및 스테이지 구현 | 적용 |
| 게임기록 및 점수판 구현 | 적용 |

**<목표별 부연 설명>**

* 1단계 목표는 일정 크기의 Snake Map을 ncurses 라이브러리를 이용하여 화면에 표시하는 것이다. Snake Map은 화면에 표시되는 문자와 마찬가지로 문자의 2차원 배열 형태의 자료 구조로 구성하고, Snake와 Wall, Item, Gate(Warp) 등의 구성요소를 표시하는 문자를 정해서 화면에 표시하기로 한다. Map은 따로 가져오는 것이 아니라 코드에 임베드하여 맵을 로딩하고 맵 별로 처리하는 과정이 컴파일 환경에서 충분히 예측 가능하도록 한다. 이때 마지막 Map은 보너스 스테이지이며, 기존의 구성한 맵의 마지막과 같은 구성이다. 보너스 스테이지는 기존의 구성한 맵을 다 클리어하고 나서 계속 반복하여 게임을 진행할 수 있도록 만든 스테이지이다.
* 2단계 목표는 Snake를 Map위에 나타내고, 화살표 키 입력을 통해 Snake가 움직이고 규칙 #1: 이동 방식을 따르도록 하는 것이다. 키 입력은 ncurses 라이브러리에서 제공하는 getch() 계열 함수를 이용해 구현하고, Snake는 입력 받은 키에 따라 방향을 전환하여 tick마다 해당 방향으로 진행하는 방식으로 구현한다. Tick 속도, 즉 snake가 움직이는 속도는 직접적으로 점점 빠르게 하는 것보다는 기본적으로 고정으로 하되, 맵이나 스테이지 진행 상황에 따라 변화하게 구현한다.
* 3단계 목표는 Map 위에 Item을 출현하도록 하고 규칙 #2: Item을 만족하면서 Snake와 상호작용하도록 구현하는 것이다. Item은 기본적으로 Apple(Growth Item), Bomb(Poison Item)에 더해 Strawberry(속도 증가), Melon(파괴가능한 벽 통과)을 구현하고, 이들이 임의 위치에 출현하며 일정 시간마다 최대 3개씩 재생성 되도록 한다. Melon 아이템의 효과를 이용하여, 보너스 스테이지를 매번 플레이할 때마다 통과가 가능한 Wall을 랜덤으로 생성하여 차별점을 둔다.
* 4단계 목표는 Snake가 통과할 수 있는 Gate(Warp) 한 쌍이 임의의 위치에 생성되고, 규칙 #3: Gate, 규칙 #4: Gate 심화, 규칙 #5: Wall을 모두 만족하도록 하는 것이다. Snake가 Gate로 이동하면 다른 하나의 Gate로 이동하고 규칙 #4에 지정된 것처럼 방향을 전환시키며, Snake가 Gate에서 빠져나온 이후에 Gate를 제거하고 새 한 쌍의 Gate를 다시 생성한다. 바로 Game Over되는 경우를 예방하기 위해 Gate는 벽 안에 생성되지 않도록 하며 이를 위해 Immune Wall을 미리 지정한다.
  + 5단계 목표는 Snake Game의 미션과 점수를 표시할 수 있는 Mission Board를 생성하고 미션 달성 시 다음 스테이지로 진행하는 등 관련 내용을 구현하는 것이다. 현재 길이나 이전 단계에서 구현한 Apple, Bomb, Strawberry, Melon을 먹은 횟수나 Gate(Warp)를 이용한 횟수 등을 미션으로 제시하고 모두 달성 시 다음 스테이지로 이동한다. 각 미션에 있는 것과 같은 항목에 따른 점수와 생존 시간을 더해 총 점수를 정산한다. 이렇게 정산한 점수는 게임 오버 시에 확인이 가능하며 Leaderboard에 게임 시작 전 입력한 이름으로 점수 기록이 가능하다. 기록된 점수는 파일로 저장되고 불러오게 되며 임의로 삭제하지 않는 한 최대 상위권 N명의 기록을 저장하도록 한다.

## 개발 내용 및 결과물

### 개발 내용

|  |
| --- |
| **작성요령 (10점)**  **프로젝트의 수행의 내용을 구체적으로 기술한다. 세부 목표별로 어떤 결과를 어떤 방법으로 달성하였는지를 자세히 기술한다.** |

**2.2.2.1 Map의 구현**

* 2차원 배열의 맵을 stage 변수 3차원 배열로 저장함으로써 인덱스를 통해 맵에 접근할 수 있도록 한다. 따라서 initialize 함수를 통해 stage에 저장된 맵을 인덱스로 접근해서 구성할 수 있게 된다. 마지막 인덱스 맵은 마지막 맵을 복사한 것으로 이는 보너스 스테이지에 해당한다. 클리어한 후에는 마지막 인덱스를 initialize 함수를 통해 접근해서 게임 오버가 될 때까지 반복을 시행할 것이다.
* Board 객체를 생성할 시 stage의 0번째 인덱스에 접근하여 맵을 구성한다. 또한 add 함수를 통해 Snake, Wall, Item, Gate 등 각각의 구성요소를 표시하는 문자를 맵에 나타날 수 있도록 한다.

**2.2.2.2 Snake 표현 및 조작**

* 게임이 진행되는 방식은 다음과 같다.
  + - 1. SnakeGame 객체를 생성한다. 이때 게임 창은 stage의 0번째 인덱스 맵으로 구성이 되고, 뱀은 왼쪽 대각선 위에 표현하게 된다. 변경 주기를 0.3초로 설정함으로써 뱀의 속도를 조정한다.
      2. 게임 오버가 될 때까지 getInputState -> playingState -> redraw 순으로 반복된다.
* 뱀을 표현하는 방법으로 큐를 사용하였고, 이 큐는 각각의 위치 값을 가지고 있는 조각들(SnakePiece)로 이루어져 있다.
* getInputState는 사용자가 입력한 값에 따라서 작동을 달리한다. w, a, s, d를 누르면 각각 위, 왼쪽, 오른쪽, 아래 방향으로 뱀의 현재 방향을 설정한다. 추가로 p를 누르면 일시정지, n을 누르면 다음 스테이지로 넘어갈 수 있도록 구현했다.
* playingState는 게임이 플레이 되는 부분을 담당한다. 이 부분에서 뱀이 다음으로 가야 할 위치를 뱀의 현재 방향에 따라 next 변수에 SnakePiece 객체로 저장하고handleNext(next)를 실행한다.
* handleNext는 next 변수를 가지고 뱀이 나아가는 로직을 수행한다. 만약 다음 위치가 ‘ ‘이라면 뱀의 꼬리 부분을 없애고 next를 push 함으로써 next는 새로운 뱀의 머리가 된다. 이 과정을 반복하면 뱀이 현재 방향으로 나아가는 것을 표현할 수 있다.

**2.2.2.3 Item 요소의 구현**

* 아이템이 만들어지는 과정은 다음과 같다.

1. 게임 만들어질 시 initialize 함수를 통해 createApple, createBomb을 진행한다. createApple과 createBomb은 getItemPos 함수를 통해 위치를 정하고, 그 위치에 생성하는 함수이다.
2. 뱀의 다음 위치에 apple를 나타내는 ‘A’가 있다면 뱀의 몸통 길이는 증가한다.
3. 뱀의 다음 위치에 bomb을 나타내는 ‘B’가 있다면 뱀의 몸통 길이는 감소한다.
4. 각각 아이템을 먹었다면 eatApple, eatBomb을 실행하여 apple과 bomb을 각각 delete하고 NULL로 한다.
5. playingState에서는 아이템이 NULL 상태인지 체크하여 만약 NULL 상태라면 아이템을 생성하는 과정을 진행한다.

* getItemPos는 y, x를 레퍼런스로 받아서 while 문을 사용하여 게임 창의 y행 x열 위치가 ‘ ‘이 될 때까지 랜덤으로 y, x 값을 변경한다. while 문을 빠져나왔다는 의미는 y행 x열 위치가 ‘ ‘이라는 의미이므로 즉, ‘ ‘의 랜덤 한 위치를 y, x에 저장하는 효과가 나타난다.
* create할 때마다 아이템은 동적 할당이 된다. 따라서 eat을 진행할 시, 동적 할당 된 아이템을 delete해줘야 한다. 마찬가지 이유로 SnakeGame의 소멸자에서도 아이템을 delete하는 과정을 진행해야 한다.

**2.2.2.4 Item 추가 구현1 & 새로운 wall 추가 구현**

* 추가로 구현한 item으로, melon으로 class정의하여 map상에는 ‘M’으로 나타나고, Item 파일내에 melon class로 구현하였다. 이 아이템의 기능은 아이템을 먹은 경우 새로 정의한 벽(부서지는 벽’3’)을 부술 수 있는 기능으로, 아이템을 먹은 수만큼 축적하여 벽을 그 수만큼 부술 수 있다.
* 새로운 wall은 위에서 언급한 대로 부서지는 벽으로, map 상에 있는 벽’1’ 중에서 랜덤한 위치에 3개가 생성된다. 벽을 통과한 후에는 warp를 ‘1’로 복구시키는 방식과 동일하게 틱 수를 계산하여 ‘1’로 다시 복구된다. 벽을 부수는 동작에 대한 자세한 설명은 다음과 같다.
* ‘M’ ,즉 아이템을 먹은 상태로 snake가 ‘3’을 만났을 때, 공백‘ ‘을 만나는 것과 같은 방식으로 동작하도록 구현하였다. 반대로 아이템을 먹지 않은 상태에서는 벽‘1’과 같이 만나면 game over가 되도록 하였다.

**2.2.2.5 Bonus Map 요소 추가**

* 보너스 스테이지 진행 시 같은 맵만 반복해서 나오는 것보다 맵 구성 요소를 달리하는 것이 낫다고 판단되어 추가하였다.
* Melon을 먹으면 벽 ‘3’을 부술 수 있는 기능을 이용해 매 보너스 스테이지 시작할 때마다 벽 ‘3’이 랜덤으로 20개 생성되게 만들어 보너스 스테이지 간 차별성을 두었다.

**2.2.2.6 Item update 구현**

* Map내에 있는 아이템들은 일정시간이 지나면 다른 위치에 랜덤하게 다시 나타나야한다. 해당 구현을 위해 이번 chrono라이브러리를 사용하여 일정시간을 설정하였다.
* Main.cpp 내에 게임이 진행되는 while문 안에서 구현을 하였고, 랜덤하게 재생성하는 시간은 7초로 설정하였다. While문 시작과 동시에 시작 시간이 start\_time이라는 변수에 저장되고, 본 게임이 진행되는 playingState(); 함수가 실행되고 나서의 시간을 end\_time이라는 변수에 저장한다. While문이 반복하는 동안 start\_time-end\_time의 값이 정해둔 시간인 7보다 커질 때까지 반복해서 계산한다. 그러다 7초가 되면, snakegame파일 내에 정의해둔 아이템을 초기화하는 함수 ItemUpdate()을 실행시키므로써, map을 초기화하여 아이템을 재생성한다. 그후, start\_time은 end\_time시간으로 초기화하여 다시 7초를 카운트하도록 하였다.

**2.2.2.7 Item 추가 구현2 & item 개수 제한**

* Item 추가 구현으로 Strawberry를 만들었다. Strawberry는 먹을 시 게임 창의 변경 주기가 50ms 줄어든다. 즉, 게임 속도가 50ms 빨라진다. 이때 최대 게임 속도는 50ms로 아무리 Strawberry를 먹어도 50ms보다 빨라지지는 않는다.
* 한 게임 창에 아이템 3개 넘게 표시되면 안 되므로 deleteItem 함수를 통해 4개 아이템 중 하나의 아이템을 골라 게임 창에 표시되지 않게 한다. 다만, 스테이지 1에서는 Melon 아이템이 없으므로 사용하는 아이템이 3개이다. 따라서 스테이지 1을 제외하고deleteItem 함수를 실행하도록 만들었다.
* deleteItem 함수는 아이템 생성 과정 다음에 실행한다.

**2.2.2.8 gate 구현**

* gate도 하나의 item으로 보고, item 파일내에 warp class를 구현하였다. (gate를 코드내에서는 warp로 정의하였다.)동작 방식은 snake의 head가 warp를 만났을 때, 다른 warp의 상,하, 좌, 우를 검사하여 snake가 다음으로 이동할 방향을 결정하는 방식이다. 방향을 검사하는 방식은 다음과 같다.
* 우선 snake의 방향에 따라 검사해야하는 좌표값을 지정해 둔다. 이때, 방향을 결정하는 규칙은 현 방향, 시계방향, 반시계방향, 현 방향의 반대 방향 순으로 고정되어 있다. 그러고 나서 snake의 head와 만난 warp를 warp1로, 나오는 warp를 warp2라고 했을 때, 미리 지정해둔 좌표값에 따라 warp2의 상, 하, 좌, 우를 확인하여 빈 곳’ ‘을 찾는다. 지정해둔 방향순서대로 검사를 진행하는데 snake가 지나갈 수 없는 좌표(맵 밖, 벽 등)이면 다음 방향으로 넘어가서 검사하는 식으로 진행된다. 방향이 결정되면 snake의 head가 놓여져야할 좌표를 저장한 후 알고리즘은 끝나도록 구현하였다. 이 방식의 경우, 오셀로게임에서 돌의 8개 방향을 검사하는 로직에서 아이디어를 얻었으며, 이전에 구현한 오셀로게임을 참고하여 코드를 작성하였다.
* Gate를 통과한 후 다시 벽으로 복원되어야 하므로, 현재 snake의 길이와 warp를 만난 순간부터 다 통과한 순간까지의 틱 수가 같을 때, gate가 사라지고 벽이 복원되도록 구현하였다.

**2.2.2.9 Mission board 및 Socre 구현**

* Mission borad 시각적 표현: 미션 보드판의 각 미션의 현황을 구현하기 위해서는 현황을 표현하기 위한 x 축의 기준점과 y축의 기준점을 잡는다. 그후 각 달성 해야 할 미션 문구 배치에 따른 x, y값의 변화를 주어야 한다. 따라서 문구의 길이이는 차이를 length()를 이용하여 x의 값의 변화를 준다.
* 점수 계산: 각 점수계산은 시간 변화에 따라 1점식 획득, 아이템 획득 시 100점을 부여하는 식으로 구성하였다. 게임 진행이 시간 변화에 따라 업데이트가 된다. 게임 진행 시 아이템을 먹게 되면 calculate\_score()에 각 아이템에게 할당된 인덱스가 입력이 되면 switch문으로 인해 socre가 업데이트가 된다. 시간 변화에 따른 업데이트 되기 전 calculate\_score()함수를 호출하여 1틱이 변화하였다는 인덱스 값을 입력해 1점을 추가하는 방식으로 점수가 계산된다.

### 시스템 구조 및 설계도

|  |
| --- |
| **작성요령 (30점)**  **프로젝트의 각 세부 목표의 주요 기능(알고리즘 등)에 대해서 기술한다. 세부 목표별로 수정한 프로그램 소스 파일을 나열하고, 해당 파일에서 세부 목표를 달성하기 위해 작성한 클래스/함수에 대해 나열하고, 각 요소에 대해 간략한 설명을 작성한다. 또한 각 요소의 개발자를 명시한다.** |

**2.2.2.1 Map 구현**

* Developer: 임민석

1) board.hpp & board.cpp

|  |  |  |  |
| --- | --- | --- | --- |
| class | Access Modifier | Function & variable | Purpose |
|  | 전역 | Int stage[][21][21] | 미리 구성한 21x21 크기의 2차원 배열의 맵을 저장하는 변수 |
| Board | public | Board() | 생성자 호출 시 initialize() 함수를 사용하여 stage 0번째 인덱스 맵을 구성 |
| void add(int y, int x, char ch) | gameboard의 y행 x열 위치에 문자 ch를 표시 |
| int getStageNum() | 현재 맵이 stage의 몇 번째 인덱스인지 반환 |
| void initialize(int num) | num 번째 stage 인덱스 맵을 구성 |
| private | int xMax, yMax | 실행 창의 최대 너비, 최대 높이를 저장하는 변수 |
| Window \*gameboard | 게임 창을 저장하는 WINDOW 타입의 포인터 변수 |
| int stageNum | 현재 게임이 몇 번째 stage 인덱스인지 저장하는 변수 |

**2.2.2.2 Snake 표현 및 조작**

* Developer: 임민석

1) board.hpp & board.cpp 추가

|  |  |  |  |
| --- | --- | --- | --- |
| class | Access Modifier | Function & variable | Purpose |
| Board | Public | Board() | gameboard의 변경 주기(= Snake 이동 속도)를 0.3ms로 설정 |
| char getInput() | 사용자가 입력하는 키 값을 반환 |
| void setTimeout(int tick) | gameboard의 변경 주기를 tick ms로 설정 |
| void refresh() | gameboard를 새로고침 |

2) snake.hpp & snake.cpp

|  |  |  |  |
| --- | --- | --- | --- |
| class | Access Modifier | Function & variable | Purpose |
|  | 전역 | enum Direction | 위, 아래, 왼쪽, 오른쪽 각각의 방향을 특정 정수를 가진 상태로 다음과 같이 열거형으로 저장  up = -1, down = 1, left = -2, right = 2 |
| SnakePiece | public | SnakePiece(int y=0, int x=0, char ch=’#’) | SnakePiece를 생성하면 y, x, ch 값을 각각 row, col, icon 변수에 저장 |
| int getX() | col 변수 값을 반환 |
| int getY() | row 변수 값을 반환 |
| void setIcon(char ch) | ch 값을 icon 변수에 저장 |
| private | Int row, col | 위치 값을 행과 열로 각각 저장하는 변수 |
| char icon | Snake를 표시하는 문자를 저장하는 변수 |
| Snake | public | Snake() | 생성자 호출 시 cur\_direction을 right로 설정 |
| void initialize() | snake 큐와 cur\_direction을 기본 설정으로 초기화 |
| void addBody(SnakePiece piece) | piece 객체를 snake 큐에 push 진행 |
| void removeBody() | snake 큐에 pop진행 |
| SnakePiece tail() | 뱀의 꼬리에 해당하는 SnakePiece 객체(= snake 큐의 front 부분) 반환 |
| SnakePiece head() | 뱀의 머리에 해당하는 SnakePiece 객체(= snake 큐의 back 부분) 반환 |
| bool setDirection(Direction direct) | direct가 뱀의 현재 방향과 정반대라면 false, 그게 아니라면 뱀의 현재 방향을 direct로 설정하고 true 반환 |
| int getSize() | 뱀의 몸통 길이를 반환 |
| Direction getDirection() | 뱀의 현재 방향을 반환 |
| SnakePiece nexthead() | 뱀이 다음으로 가야 할 위치를 저장한 SnakePiece 객체 반환 |
| private | std::queue<SnakePiece>snake : queue | SnakePiece 객체를 요소로 저장하는 큐 |
| Direction cur\_direction | 현재 뱀의 방향을 저장하는 변수 |

3) snake\_game.hpp & snake\_game.cpp

|  |  |  |  |
| --- | --- | --- | --- |
| class | Access Modifier | Function & variable | Purpose |
| SnakeGame | public | SnakeGame() | 생성자 호출 시 게임 초기 설정으로 구성 |
| void initialize(int stageNum) | 게임 창을 stage의 stageNum 번째 인덱스 맵으로 구성하고, 게임 기본 설정으로 초기화 |
| void getInputState() | 사용자가 입력한 키 값에 따라 진행하는 함수 |
| void playingState() | 게임을 하는 동안 진행되는 로직을 담은 함수 |
| void handleNext(SnakePiece next) | Next SnakePiece 객체를 통해 다음 위치의 아이콘에 따라 뱀을 조종하는 함수 |
| void setStage(int stageNum) | Stage의 stageNum번째 맵으로 재구성 |
| Int getStageNum() | 현재 게임이 stage의 몇 번째 인덱스 맵인지 반환 |
| bool isOver() | 게임 오버가 되었는지 확인 |
| void redraw() | 게임 창을 새로고침 |
| private | Board board | 게임 창을 저장하는 변수 |
| Snake snake | 뱀을 저장하는 변수 |
| bool game\_over | 게임 오버의 여부를 저장하는 변수  기본 값은 false |

**2.2.2.3 Item 요소의 구현**

* Developer: 임민석

1) board.hpp & board.cpp

|  |  |  |  |
| --- | --- | --- | --- |
| class | Access Modifier | Function & variable | Purpose |
| Board | public | char getCharAt(int y, int x) | 게임 창의 y행 x열 위치에 있는 문자를 반환 |
| void getItemPos(int &y, int &x) | 아이템이 놓일 수 있는 위치를 찾아 그 위치로 y, x 값을 변경 |

2) item.hpp & item.cpp

|  |  |  |  |
| --- | --- | --- | --- |
| class | Access Modifier | Function & variable | Purpose |
| Item | public | Item(int y=0, int x=0) | 생성자 호출 시 row, col에 각각 y, x 값을 저장하고, icon은 ‘ ‘ 으로 설정 |
| int getX() | col 값을 반환 |
| int getY() | row 값을 반환 |
| void setIcon(char ch) | icon을 ch 값으로 설정 |
| protected | int row, col | 각각 행과 열을 저장하는 변수 |
| char icon | 게임 창에 표시할 문자 저장하는 변수 |
| Apple  (inheritance by Item) | public | Apple(int y=0, int x=0) | Item 생성자를 호출하고, icon을 ‘A’로 설정 |
| Bomb  (inheritance by Item) | public | Bomb(int y=0, int x=0) | Item 생성자를 호출하고, icon을 ‘B’로 설정 |

3) snake\_game.hpp & snake\_game.cpp

|  |  |  |  |
| --- | --- | --- | --- |
| class | Access Modifier | Function & variable | Purpose |
| SnakeGame | public | ~SnakeGame() | 동적 할당 된 아이템 변수 delete 진행 |
| void initialize() | apple과 bomb을 만든다 |
| void createApple() | apple을 동적 할당하고 게임 창에 표시 |
| void createBomb() | bomb을 동적 할당하고 게임 창에 표시 |
| void playingState() | 만약 아이템이 NULL이라면 게임 창에 아이템을 만듬 |
| void handleNext(SnakePiece next) | 다음 위치가 apple 또는 bomb일 때 각각 뱀 길이가 늘어나거나 줄어드는 과정을 진행 |
| void eatApple() | apple을 delete하고 NULL로 설정 |
| void eatBomb() | bomb을 delete하고 NULL로 설정 |
| private | Apple \*apple | Apple 객체 변수 |
| Bomb \*bomb | Bomb 객체 변수 |

**2.2.2.4 Item 추가 구현1 & 새로운 wall 추가 구현**

* Developer: 김혜인

1) item.hpp & item.cpp

|  |  |  |  |
| --- | --- | --- | --- |
| class | Access Modifier | Function & variable | Purpose |
| Melon  (inheritance by Item) | public | Melon(int y, int x) | Item 생성자를 호출하고, icon을 ‘M’으로 설정 |

2) board.hpp & board.cpp

|  |  |  |  |
| --- | --- | --- | --- |
| class | Access Modifier | Function & variable | Purpose |
| Board | public | getDWallPos(int &y, int &x) | 벽인 ‘1’인 랜덤의 좌표를 찾아내고 저장함.(가장자리에 있는 ‘1’을 제외하고 찾음) |

3) snake\_game.hpp & snake\_game.cpp

|  |  |  |  |
| --- | --- | --- | --- |
| class | Access Modifier | Function & variable | Purpose |
| SnakeGame | public | ~SnakeGame() | melon의 동적할당 해제 |
| initialize(int stageNum) | 스테이지 1일때를 제외하고createMelon()함수 호출하여 melon 생성 및 새로운 벽‘3’ 생성.  tick\_count2 초기화 |
| createMelon() | getItemPos () 함수를 호출하여 ‘M’를 생성할 좌표를 얻고, 객체 melon 동적할당 |
| playingState() | 이 함수가 호출될 때마다 카운트한 틱 수에 따라 통과한 '3’삭제후, 재생성 |
| handleNext(SnakePiece& next) | snake가 ‘M’을 만났을때와 ‘M’을 먹을 상태로 ‘3’을 만났을때, ‘3’만 만났을때의 경우들을 처리하는 로직 추가 |
| private | Melon \*melon | melon객체 생성 |
| DWall\_y, DWall\_x | 3개의 ‘3’중 부서진 ‘3’의 좌표가 저장된 변수 |
| melon\_counter | 저장된 melon의 수 |
| tick\_count2 | '3’를 통과한 순간부터 빠져나오기 까지의 틱 수 (=snake의 길이) |

**2.2.2.5 Bonus Map 요소 추가**

* Developer: 임민석

1) board.hpp & board.cpp

|  |  |  |  |
| --- | --- | --- | --- |
| class | Access Modifier | Function & variable | Purpose |
| Board | public | getBonusWallPos(int &y, int &x) | ‘ ‘위치를 반환. 단, 뱀이 워프를 나가자마자 게임 오버 되는 것을 방지하기 위해 가장자리 위치는 제외 |

2) snake\_game.hpp & snake\_game.cpp

|  |  |  |  |
| --- | --- | --- | --- |
| class | Access Modifier | Function & variable | Purpose |
| Snake  Game | public | initialize(int stageNum) | 보너스 스테이지 진행될 때마다 20개의 벽 ‘3’이 랜덤으로 만들어짐 |

**2.2.2.6 Item update 구현**

* Developer: 김혜인

1) snake\_game.hpp & snake\_game.cpp

|  |  |  |  |
| --- | --- | --- | --- |
| class | Access Modifier | Function & variable | Purpose |
| SnakeGame | public | ItemUpdate() | 일정 시간이후 아이템 재생성하기 위해 호출되는 함수. |

2) main.cpp

|  |
| --- |
| #게임 시작시간 저장  auto start\_time = std::chrono::steady\_clock::now();  # playingState()호출할때의 특정시간 저장  auto end\_time = std::chrono::steady\_clock::now();  # (end\_time-start\_time)시간 계산 후 저장  auto temp= std::chrono::duration\_cast<std::chrono::seconds>(end\_time-start\_time).count();  # temp 값이 7초 이상이 되면 ItemUpdate()함수 호출하여 업데이트 이후, start\_time을 end\_time으로 초기화하여 다시 7초를 계산하도록 하였다. |

**2.2.2.7 Item 추가 구현2 & item 개수 제한**

* Developer: 임민석

1) item.hpp & item.cpp

|  |  |  |  |
| --- | --- | --- | --- |
| class | Access Modifier | Function & variable | Purpose |
| Strawberry  (inheritance by Item) | public | Strawberry(int y, int x) | Item 생성자 호출하고, icon을 ‘S’로 설정 |

2) board.hpp & board.cpp

|  |  |  |  |
| --- | --- | --- | --- |
| class | Access Modifier | Function & variable | Purpose |
| Board | public | Board() | 생성자 호출 시 speedTick 값에 300ms로 저장하고 변경 주기를 300ms로 설정 |
| initialize(int num) | speedTick 값을 300ms로 설정하고 변경 주기를 300ms로 초기화 |
| void setTimeout(int tick) | speedTick 변수에 tick 값을 저장 |
| int getSpeedTick() | 변수 speedTick 값 반환 |
| private | int speedTick | 게임 변경 주기를 저장하는 변수 |

3) snake\_game.hpp & snake\_game.cpp

|  |  |  |  |
| --- | --- | --- | --- |
| class | Access Modifier | Function & variable | Purpose |
| SnakeGame | public | ~SnakeGame() | strawberry의 동적할당 해제 |
| initialize(int stageNum) | 스테이지 1일때를 제외하고 deleteItem 함수 호출. strawberry 아이템을 생성 |
| createStrawberry() | getItemPos () 함수를 호출하여 ‘S’를 생성할 좌표를 얻고, 객체 strawerry 동적할당 |
| playingState() | strawberry가 NULL일시 생성 |
| handleNext(SnakePiece& next) | snake가 ‘S’를 만났을 때 게임 속도 50ms씩 빨라짐. 게임 속도는 50ms가 최대. |
| itemUpdate() | 스테이지 1일 때를 제외하고 아이템 재생성할 시 deleteItem 함수를 실행 |
| deleteItem() | 아이템 4개중 하나를 랜덤으로 골라서 게임 창에 아이템 표시를 지우는 함수 |
| private | Strawberry \*strawberry | strawberry객체 생성 |

**2.2.2.8 gate 구현**

* Developer: 김혜인

1) item.hpp & item.cpp

|  |  |  |  |
| --- | --- | --- | --- |
| class | Access Modifier | Function & variable | Purpose |
| Warp  (inheritance by Item) | public | Warp(int y, int x) | 생성자 |

2) board.hpp & board.cpp

|  |  |  |  |
| --- | --- | --- | --- |
| class | Access Modifier | Function & variable | Purpose |
| Board | public | getWarpPos(int &y, int &x) | 벽인 ‘1’인 랜덤의 좌표를 찾아내고 저장함. |

3) snake\_game.hpp & snake\_game.cpp

|  |  |  |  |
| --- | --- | --- | --- |
| class | Access Modifier | Function & variable | Purpose |
| SnakeGame | public | SnakeGame(int stageNum) | 생성자로, 초기화 함수를 호출 |
| ~SnakeGame() | warp의 동적할당 해제 |
| initialize(int stageNum) | createWarp()함수 호출하여 warp 생성, tick\_count1 초기화 |
| createWarp() | GetWarpPos() 함수를 호출하여 ‘$’를 생성할 좌표를 얻고, 두 개의 객체warp1, 2 동적할당 |
| playingState() | warp인 ‘$’를 만나면 checkWarp()호출해서 계산함. 해당 함수가 호출될 때마다 카운트한 틱수 에따라 warp삭제후, 재생성함. |
| endWarp() | 객체 warp1, 2를 삭제함. |
| checkWarp(SnakePiece& next, Warp \*warp) | 출구 워프의 4방향(상, 하, 좌, 우) 검사 후, 진행방향과 다음 head의 좌표를 결정함. |
| private | Warp \*warp1 | warp1 객체 생성 |
| Warp \*warp2 | warp2 객체 생성 |
| tick\_count1 | Warp를 통과한 순간부터 빠져나오기 까지의 틱 수 (=snake의 길이) |

**2.2.2.9 5단계 미션 보드판 및 Score 구현**

* Developer: 김강한

1) missionboard.hpp & missionboard.cpp

|  |  |  |  |
| --- | --- | --- | --- |
| class | Access Modifier | Function & variable | Purpose |
| MissionBoard | public | MissionBoard():  hegit, width, start\_y, start\_x, goal | 생성자 |
| init() | 초기화 함수 |
| drawBoarder() | 미션 보드판 생성 |
| drawScore(int Score) | 미션 보드판 내 점수 표시 |
| drawCurrent\_mission(int Current\_Length) | 미션 보드판에 현재 뱀 길이 출력 |
| drawApple\_mission(int Apple) | 미션 보드판에 먹은 Apple 개수 출력 |
| drawBomb\_mission(int Bomb) | 미션 보드판에 먹은 Bomb 개수 출력 |
| drawGateway\_mission(int Gateway) | 미션 보드판에 먹은 Gateway 개수 출력 |
| drawStrawberry\_mission(int Strawberry) | 미션 보드판에 먹은 Strawberry 개수 출력 |
| drawMelon\_mission(int Melon) | 미션 보드판에 먹은 Melon 개수 출력 |
| drawMongu\_mission(int y, int mission, int goal, int mission\_index) | 미션 보드판에 각 index에 따른 아이템 문구 출력 |
|  |  | mission\_index\_char[6] = {"Current\_Length", "Apple","Bomb","Gateway","Strawberry","Melon"}; | 미션 보드에 출력할 각 아이템 문구 저장 배열 |

* Developer: 김강한

2) snake\_game.hpp snake\_game.cpp

|  |  |  |  |
| --- | --- | --- | --- |
| class | Access Modifier | Function & variable | Purpose |
| SnakeGame | public | apple\_counter | 게임 진행 내 apple 저장 변수 |
| bomb\_counter | 게임 진행 내 bomb 저장 변수 |
| warp\_counter | 게임 진행 내 warp 저장 변수 |
| strawberry\_counter | 게임 진행 내 strawberry 저장 변수 |
| melon\_counter | 게임 진행 내 melon 저장 변수 |
| get\_apple\_Counter() | 게임 진행 시 apple 개수 반환 함수 |
| get\_bomb\_Counter() | 게임 진행 시 bomb 개수 반환 함수 |
| get\_warp\_Counter() | 게임 진행 시 warp 개수 반환 함수 |
| get\_strawberry\_Counter() | 게임 진행 시 strawberry 개수 반환 함수 |
| get\_melon\_Counter() | 게임 진행 시 melon 개수 반환 함수 |
| calculate\_score(int index) | 게임 진행시에 아이템 획득 및 시간 변화에 따른 점수 계산하여 점수를 리턴하는 함수 |

**2.2.2.10 5단계 게임기록 및 점수판 구현**

* Developer: 여수민

1) leaderboard.hpp & leaderboard.cpp

|  |  |  |  |
| --- | --- | --- | --- |
| class | Access Modifier | Function & variable | Purpose |
| LeaderBoard | public | void init() | 점수 기록이 담긴 파일을 읽어서 로드하는 등 초기화 작업을 함. |
| void show() | 점수판을 출력하기 위한 윈도우를 초기화하고 점수판을 출력한다. |
| void close() | 점수판을 출력한 윈도우를 제거하고 점수판을 닫는다. |
| void addRank(Rank rank) | rank를 ranking에 추가한다. |
| void clearRank() | ranking의 모든 항목을 제거한다. |
| void load() | rankingFile 위치에 저장된 파일에서 점수 기록을 읽어서 ranking에 저장한다. |
| void save() | ranking에 추가된 점수 기록을 rankingFile 위치에 파일을 만들어서 저장한다. |
| private | void drawBorder() | 점수판의 테두리를 그린다. |
| void drawHeader() | 점수판의 각 항목의 이름이 있는 헤더를 출력한다. |
| void drawScore(int idx, std::string name, int score) | idx번째 위치에 name이 score점수를 냈다는 기록을 출력한다. |
| public static | const int rankingFile | 기록을 저장할 파일의 Path명 |
| const int rankRange | 기록 저장시 최대 몇 명까지 저장할지 |
| const char delimiter | 파일 저장시 각 항목을 나누는 구분자(기본값: ‘,’) |
| private | const int height, width | 점수판을 출력할 크기, 각각 세로와 가로 |
| const int start\_y, start\_x | 점수판을 출력할 기준 좌표 |
| std::set<snake::Rank>ranking | 점수 기록을 저장할 자료 구조 |
| Rank | public | Rank(std::string n, int s) | 이름을 n, 점수를 s으로 갖는 Rank 객체 생성 |
|  | Bool operator<(const Rank& rhs) | Rank 객체가 내림차순으로 정렬되게 하기 위한 중복 함수 |
| std::string name | 점수 기록자의 이름 |
| int score | 점수 기록의 기록한 점수 |

점수를 기록하고 점수판을 출력하고 파일과 동기화하는 기능을 가진 Leaderboard 클래스를 정의하였다. std::set을 이용해 중복이 없고 내림차순으로 정렬된 목록을 기반으로 순위 기능을 구현하고, std iostream 라이브러리를 이용해서 점수 기록을 파일에 저장하고 읽어오는 기능을 구현했다.

2) main.cpp

|  |
| --- |
| std::cout << "please enter your name:";  std::cin >> name;    name.erase(std::remove(name.begin(), name.end(), LeaderBoard::delimiter), name.end()); |

사용자가 delimiter로 지정된 문자를 이름에 포함시키는 경우를 대비해서 위와 같이 이름을 읽어들일 때 delimiter를 문자열에서 필터링하는 작업을 거친다.

### 활용/개발된 기술

|  |
| --- |
| **작성요령 (10점)**  **프로젝트 수행에 사용한 외부 기술/라이브러리를 나열하여 작성한다. 각각 기술을 이 프로젝트에 적용할 때, 도움 받거나 해결하고자 하는 기능에 대해 상세히 설명한다.**  **NCURSES / STL 라이브러리 등을 포함하여 설명한다.**  **또한, 이 프로젝트를 수행하면서, 새롭게 고안한 알고리즘 등이 있다면 설명한다.** |

### 2.2.3.1 Makefile

|  |
| --- |
| CC := g++  CCFLAGS := -Wall -I/usr/include/ncurses  LDFLAGS := -L/usr/lib -lncurses    SRC\_PATH := src  OBJ\_PATH := obj  BUILD\_PATH := build    # target(entry point)와 target-specific dependency 정의  TARGET := main main2  TARGET\_DEPENDENCY := main.cpp main2.cpp    SOURCES := $(filter-out $(addprefix $(SRC\_PATH)/,$(TARGET\_DEPENDENCY)), $(wildcard $(SRC\_PATH)/\*.cpp) $(wildcard $(SRC\_PATH)/\*.h\*))  OBJECTS := $(patsubst $(SRC\_PATH)/%.cpp,$(OBJ\_PATH)/%.o,$(filter-out %.h %.hpp,$(SOURCES)))  DEPENDENCY = $(OBJECTS)    # target-specific dependency 정의  main: DEPENDENCY += $(SRC\_PATH)/main.cpp  main2: DEPENDENCY += $(SRC\_PATH)/main2.cpp    .SECONDEXPANSION:    all : $(TARGET)    $(TARGET): $$(DEPENDENCY)  $(CC) -o $@ $(filter %.c %.cpp %.o, $^) $(CCFLAGS) $(LDFLAGS)  @mkdir -p $(BUILD\_PATH) && mv $@ $(addprefix $(BUILD\_PATH)/,$@)    $(OBJ\_PATH)/%.o : $(SRC\_PATH)/%.cpp $(wildcard $(SRC\_PATH)/%.hpp)  @mkdir -p $(OBJ\_PATH)  $(CC) -c $< -o $@ $(CCFLAGS)    # 의존성 목록 출력  list:  # ...    # 깨끗하게 청소  clean:  # ... |

해당 프로젝트에서는 프로그램을 효율적으로 빌드하기 위해 make(Makefile)을 이용했다. 위 Makefile은 entry point를 포함하는 소스 파일을 제외하고 src 디렉토리 이하에 있는 모든 소스코드를 컴파일해서 오브젝트 파일로 만드는 작업과 entry point를 포함하는 소스 파일을 컴파일하고 각 오브젝트 파일과 링크하는 작업을 정의하여 축적식 컴파일이 가능하도록 한다. 이외에 list, clean 등의 명령을 추가하여 소스 파일 목록을 확인하고 빌드시에 생성되는 파일을 정리하는 기능을 추가했다.

또한 여러 개의 executable을 생성해야 할 경우를 대비해서 각 target 별로 종속성을 설정할 수 있도록 했다. 위의 main2가 그 예시이며 main2를 컴파일할 때에는 main.cpp가 종속성에 포함되지 않는다.

### 2.2.3.2 std::set

|  |
| --- |
| class Rank {  public:  std::string name;  int score;    Rank(std::string n, int s);  bool operator<(const Rank& rhs) const;  };    bool Rank::operator<(const Rank& rhs) const {  return (score != rhs.score) ? (score > rhs.score) : (name > rhs.name);  }    class Leaderboard {  public:  void addRank(Rank rank)    private:  std::set<snake::Rank> ranking;  } |

LeaderBoard에서 점수 기록을 정렬된 상태로, 중복 없이 저장하기 위해 std::set을 사용하였다. 점수 기록을 저장하기 위한 클래스 Rank를 정의하고 < 연산자를 재정의해서 set이 Rank를 (score, name)에 대해서 내림차순으로 정렬하도록 했다.

### 2.2.3.3 std::ifstream, std::ofstream, std::stringstream

|  |
| --- |
| void LeaderBoard::load() {  std::ofstream dfs("debug", std::ios\_base::out | std::ios\_base::app);    std::ifstream ifs(rankingFile, std::ios\_base::in);  std::stringstream ss;  std::string line;    std::string name;  int score;  while (std::getline(ifs, line)) {  ss << line;  std::getline(ss, name, delimiter);  ss >> score;  std::stringstream().swap(ss);    addRank(Rank(name, score));  dfs << "loaded " << name << "(" << score << ")" << std::endl;  }  ifs.close();  dfs.close();  }  void LeaderBoard::save() {  std::ofstream dfs("debug", std::ios\_base::out | std::ios\_base::app);    std::ofstream ofs(rankingFile, std::ios\_base::out);  for(auto it=ranking.begin(); it!=ranking.end(); it++) {  ofs << it->name << delimiter << it->score << std::endl;  dfs << "saved " << it->name << "(" << it->score << ")" << std::endl;  }  ofs.close();  dfs.close();  } |

파일에서 점수 기록을 읽기 위해 std::ifstream을 이용해서 파일을 열고 점수 기록을 쓰기 위해 std::ofstream으로 파일을 연다. 한편 디버그 로그를 남기기 위한 dfs는 std::ofstream을 append 모드로 열어서 후행에 추가해주는 식으로 사용되고 있다.

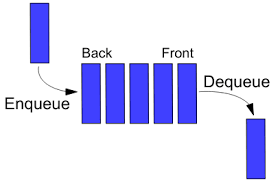
이후 파일에서 std::getline으로 각 line을 읽고, 해당 라인에서 다시 std::getline으로 delimiter로 분리된 각 항목을 다시 읽기 위해 stringstream을 사용했다. 해당 용도로 std::stringstream을 사용한 이후에는 std::stringstream().swap(ss)를 이용해 새롭게 초기화해줘야 정상적으로 동작한다.

### 2.2.3.4 queue

주요 맴버 함수로는 다음과 같다.

1. push: Back으로부터 요소를 추가한다 (그림에서는 Enqueue)
2. pop: Front으로부터 요소를 삭제한다. (그림에서는 Dequeue)
3. front: Front 부분의 요소를 반환한다.
4. back: Back 부분의 요소를 반환한다.
5. empty: queue가 비어 있는지 여부를 판단하여 반환한다.
6. size: queue의 크기를 반환한다.

그림으로 나타내면 다음과 같다.



### 2.2.3.5 string

해당 프로젝트에 사용된 함수들은 다음과 같다.

* ‘length’: 문자열의 길이를 반환한다.

|  |
| --- |
| std::string mission\_index\_char[6] = {"Current\_Length","Apple","Bomb","Gateway", "Strawberry","Melon"}; // missionboard.hpp에 선언 된 문자열  int goal\_x = mission\_check\_x + mission\_index\_char[mission\_index].length(); /\*  goal\_x: 미션목표의 x좌표 변수  mission\_check\_x: 입력된 문자열 x좌표  mission\_index\_char: missionboard.hpp에 선언된 문자열을 인덱스에 맞춰서 할당  mission\_index: 각 아이템에 할당된 인덱스들은 저장하고 있는 변수  \*/ |

미션 보드의 작성되는 문구들을 문자열의 길이에 맞처서 x좌표에 변화를 주어야 할 때 length 함수를 이용하여서 고정값이 아닌 유동적으로 좌표변화를 준다.

### 2.2.3.6 Chrono

* 해당라이브러리는 시간과 관련한 라이브러리로, 관련함수는 다음과 같다.

|  |
| --- |
| auto 변수 = std:: chrono:: steady\_clock:: now(); //현재 시간 저장  //특정 시간 계산  auto temp= std::chrono::duration\_cast<std::chrono::seconds>(end\_time-start\_time).count(); |

### 2.2.3.7 ncurses

해당 프로젝트에서 사용된 함수들은 다음과 같다.

1. initscr:  curses mode로 터미널을 초기화 한다. 몇몇 구현에서는 화면을 클리어하고 빈 화면을 보여주기도 한다. curses 패키지를 이용해서 스크린 처리를 하려면 이 함수를 반드시 먼저 선언해야 한다. 이 함수는 curses 시스템을 초기화 하고 'stdscr' 이라 불리는 우리의 현재 윈도우와 몇몇 데이터들-구조체- 들을 위한 메모리를 할당한다.
2. endwin:  curses의 하위 시스템들과 데이터 구조체가 점유했던 메모리를 해제하고 터미널을 일반적인 모드로 바꿔 놓는다. 이 기능은 curses 모드에서 작업을 마치게 되면 반드시 호출해야 한다.
3. refresh / wrefresh: stdscr(또는 특정 창)의 버퍼에 있는 내용들을 화면에 표시하기 위해 쓴다.
4. noecho: 사용자들이 입력한 문자들을 윈도우에 표시 안되게 한다.
5. curs\_set: 인자에 따라서 커서를 제어한다.

0 보이지 않게 한다. 1 기본이다. 2 매우 잘 보이게 한다.

1. getch / wgetch: stdscr(또는 특정 창)에서 사용자가 입력한 문자를 ASCII 정수형으로 반환한다.
2. printw / mvprintw / wprintw / mvwprintw: stdscr(또는 특정 위치, 특정 창, 특정 창의 특정 위치)에서 문자열을 출력한다.
3. clear / wclear: stdscr(또는 특정 창)을 아무것도 없는 창으로 만든다.
4. addch / mvaddch / waddch / mvwaddch: stdscr(또는 특정 위치, 특정 창, 특정 창의 특정 위치)에 문자 하나를 추가한다.
5. timeout / wtimeout: stdscr(또는 특정 창)의 변경 주기를 설정한다.
6. newwin: 높이, 너비, 생성되는 행 위치, 생성되는 열 위치를 받아 윈도우를 생성한다.
7. delwin: 인자로 받은 창 포인터를 삭제하고, 해당 창과 관련된 모든 메모리를 해제한다.
8. wborder: 4개의 줄과 4개의 코너에 대해 주어진 문자로 특정 창의 테두리를 그린다.
9. start\_color: 색을 사용하기 위해 먼저 호출하는 함수이다.
10. init\_pair: 색상 쌍을 정의한다. 쌍의 번호와 텍스트 색, 배경 색을 인자로 받는다. 설정된 색상 쌍은 COLOR\_PAIR를 통해 사용된다.
11. attron / attroff: 속성을 활성화 시키거나 비활성화 시킬 수 있다. 이 함수를 통해 설정한 색상 쌍을 활성화하거나 비활성화 시킬 수 있다.
12. getmaxyx: 인자로 받은 창의 최대 높이, 최대 너비를 가져올 수 있다.

### 현실적 제한 요소 및 그 해결 방안

|  |
| --- |
| **작성요령 (5점)**  **제안된 프로젝트의 단계 별 수행에 있어, 제한 요소를 찾아 작성한다. 해당 제한 요소를 해결하기 위해서 어떤 방법으로 해결하였는지 작성한다.** |

* 터미널 위에서의 동작: 해당 프로젝트 수행에 있어서 가장 큰 제한 요소는 터미널 위에서 동작하는 프로그램이라는 점이었다. 우선 출력 가능한 문자열이나 색상이 제한되고, 입력으로 문자열을 받아서 처리하는 것도 쉽지 않았다. 이를 해결하기 위해서 1단계 ~에서는 ....
* 그래픽 구현 제한: UI를 구현하는 데 있어서 문자 외에는 사용할 수 없는 제한이 있어 구분이 쉽지 않았다. 그래서 최대한 아이템 변수명을 과일이름으로 짓거나, 벽 종류에 따라 1, 2, 3등으로 구별짓는 등 쓸 수 있는 문자를 최대한 활용하여 구분지을 수 있도록 함으로써 해결하였다.

### 결과물 목록

|  |
| --- |
| **작성요령 (5점)**  **결과물 목록을 작성한다. 목록은 제출하는 파일과 각 파일의 역할을 간략히 설명한다.** |

[**https://youtu.be/sjidQMEU5Qk**](https://youtu.be/sjidQMEU5Qk)

# 자기평가

|  |
| --- |
| **작성요령 (5점)**  **프로젝트를 수행한 자기 평가를 서술한다. 팀원 개개인의 자기 평가가 포함되어야 하며, 본인의 역할, 프로젝트 수행 시 어려운 점, 도움이 되었던 점, 이 프로젝트 운영에 개선이 필요하다고 생각하는 점을 충분히 서술한다.** |

## 여수민(팀장)

* 나는 프로젝트 관리, 즉 회의 내용 정리나 빌드 환경 정립, Github repo 및 버전 관리를 맡았다. 먼저 Makefile 환경을 구축했는데 결과적으로 모두가 통일된 개발 환경에서 작업할 수 있어서 좋은 선택이었던 것 같다. 그리고 Git 버전 관리를 맡아 각 팀원들의 구현물을 합치는 작업을 했는데, 팀원들 대부분이 Git 사용에 익숙하지 않아 merge 작업이 다소 까다로웠으나 Git 사용법이나 팁 등을 알려주거나 커밋이 꼬였을 때의 문제 해결 방법을 익히거나 단순히 합치는 squash merge외에 rebase, cherry-pick 등의 방식에 대해 배우기도 하고 서로에게 득이 되는 방향으로 진행된 것 같아서 좋았다.
* 조금 아쉬웠던 점은 시간이 부족해서 코드 리뷰를 자주 진행하지 못했던 점이나 리팩터링 작업을 진행하지 못한 점, 추가 기능을 많이 구현하지 못한 점이었다. 만약 다음에 비슷한 프로젝트에 참여해서 같은 역할을 맡게 된다면 프로젝트를 효율적으로 진행하기 위한 여러가지 방법에 대해 고민해봐야겠다고 느꼈다.

## 김강한

* 저는 이 프로젝트에서 아이템 미션에 따른 점수 클래스 및 메소드 미션 인터페이스 코드를 구현하는 역할을 맡았습니다. 구체적으로는 플레이어가 아이템을 획득할 때마다 점수가 변화하는 로직을 구현했고, 플레이어가 미션을 클리어할 때마다 새로운 맵으로 변경되는 로직을 추가했습니다. 또한, 게임 맵의 단계가 변화할 때마다 미션 보드를 초기화하는 기능과 각 미션이 달성되면 시각적으로 이를 표현하기 위해 색이 변하는 기능을 구현했습니다. 프로젝트를 수행하면서 가장 어려웠던 점은 각 기능들을 클래스로 어떻게 깔끔하게 묶을 것인가에 대한 고민이었습니다. 객체 지향 프로그래밍의 원칙을 지키면서도 코드의 가독성과 유지보수성을 높이기 위해 많은 고민을 했습니다. 또한, ncurses 라이브러리를 이용한 메소드 구현에서도 어려움을 겪었습니다. 이 라이브러리를 처음 사용해보는 터라, 문서를 참고하며 여러 시행착오를 겪어야 했습니다. 프로젝트를 진행하면서 어려운 점을 해결해나가는 과정이 매우 도움이 되었습니다. 특히, 코드 정리에 대한 고민과 구현을 통해 코드의 가독성을 높이는 방법을 배울 수 있었습니다. 이를 통해 클래스와 메소드를 깔끔하게 분리하고, 각각의 역할을 명확히 할 수 있었습니다. 팀원들과의 협업을 통해 다양한 해결책을 모색할 수 있었던 점도 큰 도움이 되었습니다. 프로젝트 운영에 있어 개선이 필요하다고 생각되는 점은 이미지를 snake 및 아이템의 퀄리티를 높이는 과정입니다. 현재 구현된 그래픽은 기본적인 수준에 머물러 있어, 좀 더 시각적으로 매력적인 게임을 만들기 위해서는 그래픽 퀄리티를 높이는 것이 필요합니다. 이를 위해 전문 디자이너와 협업하거나, 그래픽 관련 리소스를 좀 더 탐색하는 과정이 필요할 것 같습니다.

## 김혜인

* 본 프로젝트에서 담당한 부분은 gate 생성, item 추가 등 로직과 관련한 파트를 담당하였다. 담당한 gate, item추가, 새로운 wall생성 등 맵상에 등장하는 요소들을 주로 다루었다.

프로젝트를 진행하면서, 아무래도 사용하는 함수의 종류가 많다보니 하나씩 다 생각하면서 코드를 작성하는 것이 쉽지는 않았다. 그리고 초반에 git을 사용하는 데에 있어 익숙치 않은것들이 많아 팀원들의 도움과 개인적인 공부를 통해 개선해나갔다.

개인적으로 시간관계상 유니코드를 활용한 개선된 UI를 구현하지 못한 점이 아쉬었다. 그리고 C++이라는 언어를 다루는데 부족한 점이 많아 개선이 필요하다고 느꼈다. 그래도 이번 프로젝트를 통해 git사용, makefile 구조 등 새로운 정보를 얻을 수 있어 의미있는 활동이었다고 생각한다. 따라서 다른 프로젝트를 진행하게 되었을 때 이번 경험과 아쉬움을 바탕으로 더 개선된 작업물을 만들 수 있을 것같다.

## 임민석

* Map 구현, Snake 구현, Item과 Map 상호작용 등 Snake게임을 구성하는 파트를 주로 담당하였다. 비록 게임을 만드는데 있어서 어떻게 시작해야 할지 감이 안 잡혔지만, 프로젝트-내용 pdf에 있는 단계별 조건과 이 조건을 적용하기 위해서 어떤 기능을 구현해야 하는지 정리하는 것이 도움이 됐다. 특히 단계별로 서술된 내용이 게임 구성을 시작하는데 있어서 큰 도움이 됐다. ncurses 라이브러리를 공부하면서 본인이 원하는 방향으로 윈도우를 제어할 수 있다는 점에서 좋은 라이브러리이며, 신기한 함수들이 많아 꽤 흥미로움을 느꼈다. 이번 프로젝트를 통해 ncurses라는 유용한 라이브러리를 공부할 수 있어서 좋았다. 또한, 필요한 로직을 고민하고 구현하는 과정에서 재미를 느꼈으며, 덕분에 프로젝트를 즐겁게 진행하였던 것 같다.

# 참고 문헌

**참고한 서적, 기사, 기술 문서, 웹페이지를 나열한다.:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 번호 | 종류 | 제목 | 출처 | 발행년도 |
| 1 | 웹페이지 | NCURSES Programming HOWTO | https://tldp.org/HOWTO/NCURSES-Programming-HOWTO/index.html | 2005-06-20 |
| 2 | 웹페이지 | NCURSES Programming HOWTO | https://wiki.kldp.org/wiki.php/NCURSES-Programming-HOWTO | 2014-03-22 |
| 3 | 웹페이지 | Linux man page | https://linux.die.net/man/3/timeout | - |
| 4 | 웹페이지 | std::queue | https://cplusplus.com/reference/queue/queue/ | - |
| 5 | 웹페이지 | queue 그림 | https://ko.wikipedia.org/wiki/%ED%81%90\_%28%EC%9E%90%EB%A3%8C\_%EA%B5%AC%EC%A1%B0%29 |  |
| 6 | 웹페이지 | GNU make | https://www.gnu.org/software/make/manual/make.html | 2023-02-26 |
| 7 | 웹페이지 | stdlib.h | https://cplusplus.com/reference/cstdlib/rand/#google\_vignette | - |
| 8 | 웹페이지 | Making Snake in Ncurses – Tutorial 7 – The Snake Class (and SnakePiece Class) | https://www.youtube.com/watch?v=oZfRXM4GQN4&list=PL2U2TQ\_\_OrQ\_TV2-wuHqGaK8qlnxgKUvK&index=11 | 2021-07-27 |
| 9 | 웹페이지 | xterm(1) - Linux man page | https://linux.die.net/man/1/xterm | - |

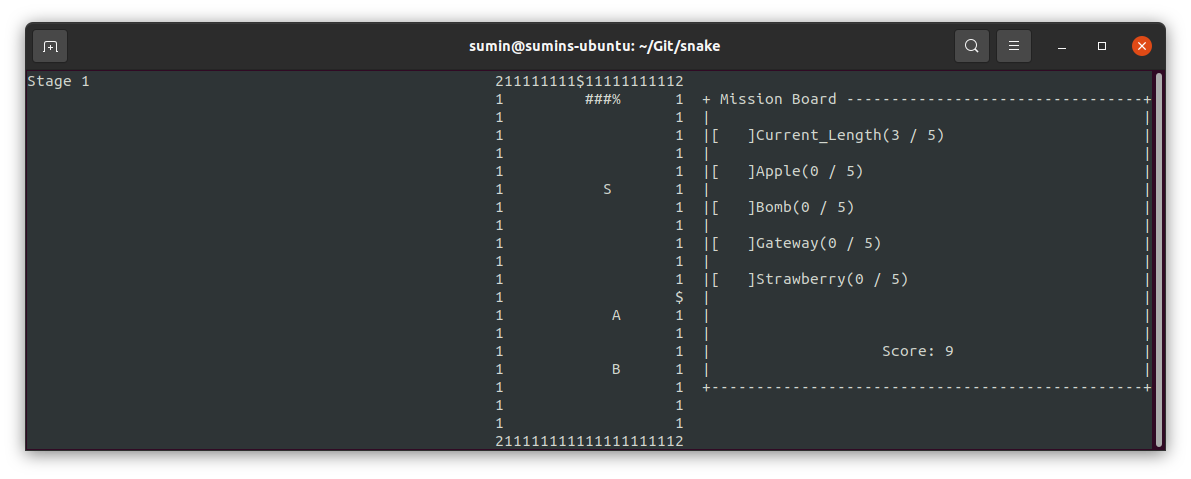
# 부록

|  |
| --- |
| **작성요령 (15점)**  **프로젝트의 결과물을 사용하기 위한 방법에 대해서 작성하세요.** |

## 사용자 매뉴얼



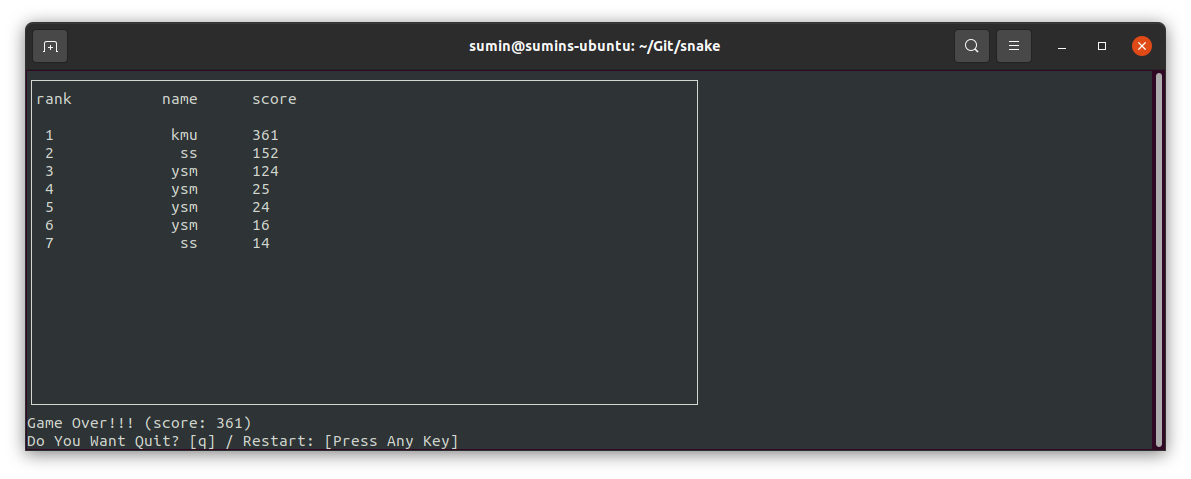
Run.sh 스크립트를 실행하면 자신의 이름을 입력하라는 메세지가 출력되며, 이름을 입력하고 엔터를 입력하면 바로 게임이 시작된다.

게임을 시작하면 위와 같은 모습의 화면이 출력된다. 화면 상에서 움직이는 ###%가 snake에 해당하며, 1과 2, 3이 벽, A나 B, S, M가 각각 Apple, Bomb, Strawberry, Melon 아이템이고, $가 게이트이다.

Wasd 키를 이용하여 snake의 방향을 변경할 수 있으며, snake는 일정 시간마다 보고 있는 방향으로 움직이며, 벽 등의 장애물과 만나거나 길이가 3보다 작아지면 게임이 종료된다.

각 아이템은 다음과 같이 각기 다른 효과를 가지고 있는데, 해당 스테이지를 클리어하기 위해서는 각각 일정 개수 이상을 획득할 필요가 있다.

* Apple 아이템을 먹으면 snake의 길이가 한 칸 늘어난다.
* Bomb 아이템을 먹으면 snake의 길이가 한 칸 줄어든다.
* Strawberry 아이템을 먹으면 snake가 자동으로 움직이는 속도가 빨라진다.
* Melon 아이템을 먹으면 부술 수 있는 벽(3으로 표시됨)을 한 번에 한해 부수고 통과할 수 있다.

게임 오버가 되면 자기 자신의 점수와 함께 전에 플레이했던 사람들의 기록 및 순위가 출력된다. 이후 게임을 종료하려면 q, 게임을 다시 플레이하려면 그외 아무 키를 입력하면 된다.

## 설치 방법

해당 프로그램을 컴파일하기 위해서는 선행 조건으로 make, gcc, ncurses 라이브러리가 설치되어 있어야 한다.

|  |
| --- |
| ~$ git clone https://github.com/soominyeo/snake.git |

위 조건을 만족한다면 해당 프로젝트의 소스 코드를 git clone 등의 방법으로 로컬에 다운로드한다.

|  |
| --- |
| ~$ cd snake ~/snake$ make g++ -c src/snake.cpp -o obj/snake.o -Wall -I/usr/include/ncurses g++ -c src/leaderboard.cpp -o obj/leaderboard.o -Wall -I/usr/include/ncurses g++ -c src/snake\_game.cpp -o obj/snake\_game.o -Wall -I/usr/include/ncurses  ... |

이후 다운로드한 소스의 최상단 디렉토리에서 터미널을 통해 make 명령어를 입력하면 실행파일 main이 build/ 이하 디렉토리에 생성된다.

|  |
| --- |
| ~/snake$ ./build/main  or  ~/snake$ ./run.sh |

이 실행파일을 직접 실행해도 되지만 소스 코드에 동봉된 run.sh bash 스크립트를 이용하면 터미널의 크기를 최적의 크기로 조정한 이후 실행할 수 있다.