**C++프로그래밍**

**프로젝트**

|  |  |
| --- | --- |
| 프로젝트 명 | Snake Game |
| 팀 명 | 저쩔스네이크 |
| 문서 제목 | 결과보고서 |

|  |  |
| --- | --- |
| **Version** | 2.0 |
| **Date** | 2022-Jun-17 |

|  |  |
| --- | --- |
| **팀원** | 김진재 (팀장) |
| 이현영 |
| 허승범 |
|  |
|  |

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

**문서 정보 / 수정 내역**

|  |  |
| --- | --- |
| **Filename** | 최종보고서-SnakeGame\_저쩔스네이크.doc |
| **원안작성자** | 이현영, 김진재, 허승범 |
| **수정작업자** | 이현영, 김진재, 허승범 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 수정날짜 | 대표수정자 | Revision | 추가/수정 항목 | 내 용 |
| 2022-06-16 | 이현영 | 1.0 | 최초 작성 | 상세 구현 내용 작성 |
| 2022-06-16 | 김진재 | 1.1 | 내용 수정 | 개요, 부록, 개발 목표 작성 |
| 2022-06-16 | 허승범 | 1.2 | 내용 수정 | 상세 구현 내용 작성 |
| 2022-06-17 | 모두 | 2.0 | 내용 수정 | 자기 평가 등 최종본 합본 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**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.**  **프로젝트의 전체적인 구조 및 개발 내용을 명확하게 기술한다.** |

2022학년도 1학기’ C++프로그래밍 과목’의 팀 프로젝트입니다.

프로젝트 완성을 위해 사용한 개발 언어는 C++이며, 리눅스 OS 환경에서 개발을 진행하였습니다.

일부 make utility와 외부 라이브러리인 ncurses를 사용하여 개발하였습니다. Make Utility는 기본적으로 GNU에 포함되어 있어, 리눅스 계열 OS에서는 별도의 설치 없이 사용할 수 있습니다. 그러나 외부 라이브러리인 ncurses는 리눅스 기준으로 다음과 같은 설치 과정이 필요합니다.

sudo apt-get update

sudo apt-get install libncurses5-dev libncursesw5-dev

프로젝트의 전체적인 구조는 다음 그림과 같습니다.

Diagram

Description automatically generated

main을 통해 프로그램을 실행하면, 크게 보면 3 종류의 반복문을 돌고 끝납니다. 첫 번째 반복문은 gamestartscene의 구현에 관한 내용으로, 게임 초기 실행 시 띄울 화면을 설정한 이후에 사용자로부터 키를 받으며 어떠한 조건으로 갈지 결정합니다.

사용자가 게임 스타트를 결정하면 for문 내로 들어가서 본격적으로 게임이 실행되며, 첫 번째 스테이지가 로드되어 snake가 죽기 전까지 while문을 무제한 받습니다. 이 때 gamescene(맵과 뱀 출력 및 맵의 조건들 체크)을 계속 업데이트 하면서 Item과 Gate도 설정합니다. 다만 while 문을 무제한으로 계속 돌려두면 끝나자마자 계속 처음으로 돌아가기 때문에 gamingscene에서 0.5초 프로그램을 지연시키는 함수를 넣어 마치 0.5초씩 스네이크가 움직이는 것처럼 보이게 구현하였습니다.

게임에서 snake가 죽는 경우는 크게 세 가지가 있습니다. snake가 벽 혹은 자신의 몸에 닿거나(자신의 몸 방향으로 키를 누르는 것 포함), snake 몸 길이가 poison을 먹어 3보다 줄어들었을 때, 그리고 snake가 미션 수행에 실패하였을 경우입니다. 여기서 미션 수행 실패란, poison을 스테이지 내에 주어진 미션 제한양 이상으로 먹어 스테이지를 통과할 수 없는 경우입니다. 게임을 성공하는 경우는 앞서 언급한 내용을 위반하지 않으면서 모든 주어진 미션에 체크가 될 때입니다.

snake가 죽은 경우, gameoverscene을 띄웁니다. 이 화면에서는 메인화면으로 가서 게임을 다시 할지, 혹은 프로그램을 그냥 종료할지 선택할 수 있습니다. 반대로 snake가 죽지 않으면서 모든 미션을 성공한 경우, gameclearscene이 구현됩니다. 이 화면에서는 아무 키나 입력 대기 모드로 진입한 후, 키가 들어오면 다음 스테이지로 넘어가는 for문에 진입합니다. 만약 stage4, 즉 마지막 스테이지에서 게임을 성공한 경우 congratulationsscene을 띄웁니다. 이후 키를 입력받으면 프로그램을 종료합니다.

깃허브 주소 : https://github.com/HyeonYoung-Lee/Snake-2022-1

노션 주소 : https://road-freedom-b61.notion.site/Snake-Project-881e8fb579674fbfa485d7046e2312d9

# 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 |
| 4단계 | Gate 요소의 구현 | 적용 |
| 5단계 | 점수 요소의 구현 | 적용 |

모든 단계에서의 구현은 강의 중 제공한 권고안의 내용을 최대한 반영하여 제작하였습니다. 구체적으로 단계별 권고안 내용 및 고려한 구현 목표들은 다음과 같습니다.

※ ← 해당 표시는 권고안의 내용 중 개발자가 직접 정해야 하는 내용이거나, 권고안의 내용을 해치지 않는 선에서 약간 변형된 내용을 담았습니다.

**1단계: Map의 구현**

ncurses 초기 프로그램 화면 출력 클래스 생성

2차원 배열로 map을 저장하는 클래스 생성 (이 때, 4단계의 stage 여러 개를 고려하여 파일 입력을 통해 Wall, Immune Wall 정보를 갖고 있는 stage파일 입출력을 통해 생성)

※ 초기 생성 파일에서 snake 위치 제외

**2단계: Snake 표현 및 조작**

snake 제작: snake 클래스 생성

snake 초기 위치 세팅 구현

화살표를 입력받으면 스네이크가 움직이도록 구현

snake의 규칙 구현 (Game Rule #1 준용)

**3단계: Item 요소의 구현**

Item 스테이지에 표시

Item 출현 매커니즘 설정 (Game Rule #2 준용) ※ 동시에 출현하는 아이템 수는 총 2개를 넘지 않도록 제한

snake 몸의 총 길이가 3보다 짧아질 경우 게임 종료

**4단계: Gate 요소의 구현**

Gate를 스테이지에 표시

Gate 출현과 초기화 매커니즘 설정 (Game Rule #3, #5 준용)

snake가 Gate 통과 시 Gate 위치에 따른 진출 순서 설정 (Game Rule #4 준용)

※ map 내 아이템 표현을 구체화

**5단계: 점수 요소의 구현**

게임 우측에 게임 점수를 표시하는 창 생성

게임 점수 설정 (Game Rule #6 준용)

Mission 달성값 설정

Mission 달성 시 다음 map으로 진행

## 개발 내용 및 결과물

### 개발 내용

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

**1단계: Map의 구현**

21 X 41 크기의 4개의 stage를 구성하였습니다. 0 (일반), 1 (wall), 2 (immune wall) 로 구분되는 파일을 작성하여 Stage 클래스에서 파일 입출력을 통해 2차원 배열로 맵 정보를 저장합니다. 초기 생성 파일에서 snake 위치를 제외하고 snake 구현 단계에서 초기 위치를 지정하여 snake와 map을 분리시켰습니다.

Stage 클래스에서는 map을 저장할 뿐 만아니라 map을 출력하기도 하는데, 2차원 배열로 저장된 맵 정보를 중첩반복문을 통해 요소를 출력합니다.

**2단계: Snake 표현 및 조작**

snake 클래스에서는 snake가 맵에 표시되어야 할 좌표 벡터 멤버 변수와 snake가 이동해야 하는 방향, snake 생존 여부를 포함합니다.

Stage(Mapset)클래스에 snake의 좌표를 저장한 벡터 멤버 변수를 받아와서 map 위의 해당 위치에 snake를 세팅할 수 있는 함수를 정의했습니다.

snake의 머리는 3, 꼬리는 4로 표시합니다.

방향키를 입력받으면 해당 방향키의 방향으로 스네이크가 움직이도록 구현했습니다.

snake의 규칙은 다음과 같습니다.

snake의 사망조건 : 자신의 몸에 닿음(진행방향의 반대 이동 포함), 벽에 닿음

같은 방향으로의 이동 무시 (키 입력 무시)

snake는 일정 시간마다 지속적으로 head 방향 무한 이동

방향키 입력이 있다면 방향키의 방향으로 이동

방향키 입력이 없다면 이전에 snake가 진행하고 있던 방향으로 이동

방향키 지정 ※ 방향키는 키보드의 실제 상하좌우 방향키로 지정하였습니다.

**3단계: Item 요소의 구현**

Item을 스테이지에 표시합니다. Growth Item은 5, Poison Item은 6으로 표시합니다.

아이템은 Snake Body와 벽으로 정의되지 않은 임의의 위치에 출현합니다. 아이템은 일정시간이 지나면 사라지고, 다시 벽과 뱀이 있는 위치가 아닌 다른 위치에 나타납니다.동시에 출현할 수 있는 Growth와 Poison 각각 1개 총 2개로 제한했습니다.

Snake가 표시된 아이템 위치에 접근할 때, 아이템을 획득합니다.

닿은 아이템이 Growth Item(5) 인 경우, 뱀의 vector 마지막(꼬리) 증가

닿은 아이템이 Poison Item(6)인 경우, 뱀의 vector 마지막(꼬리) 삭제

snake 몸의 총 길이가 3보다 짧아지면 게임 창을 더 이상 띄우지 않고 GameOverScene으로 이동합니다.

**4단계: Gate 요소의 구현**

Gate를 스테이지에 표시합니다. Gate는 7로 표시합니다.

한쌍의 Gate가 임의의 벽 위의 각자 다른 위치에 동시에 출현합니다. (Game Rule #3 준용)

Gate 출현 위치는 Immune Wall을 제외한 Wall (Game Rule #5 준용) 의 범위 내 입니다.

Gate는 Snake가 Item을 두 개 이상 먹었을 때부터 출현, 이후 Snake가 Item을 두 개 먹을 때마다 다른 위치에 생성되도록 설정하였습니다. immune wall 을 제외한 가장자리 벽과 내부 벽 어디서든 나타날 수 있습니다.

Gate와 위치한 벽에 따라 Gate의 진출 방향을 다르게 구현했습니다. (Game Rule #4 준용)

가장자리 벽에 위치한 Gate : 내부방향으로 진출

내부벽에 위치한 Gate: 기존 진출 방향의 시계방향으로 진출

※ map 내 아이템 표현을 구체화 했습니다 화면상에 표현시 Wall(1), ImmuneWall(2), SnakeHead(3), SnakeBody(4), GrowthItem(5), PoisonItem(6), Gate(7)은 권고안을 준수하였지만, 뱀이 다닐 수 있는 공간인 0은 플레이 시 가시성을 높이기 위해 출력 시 빈 공간(’ ‘)으로 출력하도록 변경하였습니다.

**5단계: 점수 요소의 구현**

게임 우측에 게임 점수와 시간을 표시하는 ScoreBoard를 구현했습니다. (Game Rule #6 준용)

B : (게임 중 몸의 현재 길이) / (스네이크가 가질 수 있는 최대 길이)

+ : 게임 중 snake가 획득한 Growth Item의 수

- : 게임 중 snake가 획득한 Poison Item 수

G : 게임 중 gate 사용 횟수

게임 시간 (초 단위로 계산)

※ 시간 표시에 대한 창이 ‘Snake Game의 구조’에 따로 존재하지 않아, mission 칸 아래에 시간 창을 구현하였습니다. Total Time은 게임 시작 후 진행된 시간, 그리고 Stage Time은 해당 스테이지 진행 시간을 보여줍니다.

Mission 달성값을 표시하는 MissionBoard를 구현했습니다. 주어진 Mission을 모두 달성하면 다음 스테이지로 넘어갑니다.

※ 미션 달성값은 해당 map 마다 임의의 고정 값을 주었습니다.

초기 스테이지는 길이 5 달성, GrowthItem 2개, PoisonItem 4개, Gate 1개입니다.

이후 스테이지 하나를 클리어할 때마다 달성 길이는 2 추가, GrowthItem은 달성해야 할 길이의 절반 추가, PoisonItem은 하나 감소, gate는 2배씩 증가합니다.

※ PoisonItem은 처음 게임 실행 시 ‘달성’으로 체크됩니다. 이후, PoisonItem을 미션에서 주어진 개수 이상 먹게 되면 길이와 상관없이 미션 실패 확정으로 판정하여 게임 오버로 넘어가게 설계하였습니다.

※ 다음 맵으로 진행할 조건을 체크하여 달성 시, 다음 map으로 즉시 진입하면 플레이어가 당황할 가능성이 있어 중간에 ‘스테이지 클리어’씬을 통해 아무 키를 입력받기 전까지 다음 스테이지를 대기하도록 구성하였습니다.

### 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.** |

**ETCScene.h** 개발자 : 김진재

|  |
| --- |
| #include <ncurses.h>  #include <iostream>  #include <fstream>  #include <string>  #include "Snake.h"  class GameStartScene  {  protected:  int width, height;  int selectMode; // Select Using Arrow Keys 0: START 1: EXIT  int selected; // Pressed Enter?  int pastKey;  public:  GameStartScene();  int GameStartStatus();  int getSelectMode() { return this->selectMode; }  int getSelected() { return this->selected; }  WINDOW \*renderGameStartScene(int key);  };  class GameOverScene  {  protected:  int width, height;  int selectMode; // Select Using Arrow Keys 0: MAINMENU 1: EXIT  int selected; // Pressed Enter?  int pastKey;  public:  GameOverScene();  int GameOverStatus();  int getSelectMode() { return this->selectMode; }  int getSelected() { return this->selected; }  WINDOW \*renderGameOverScene(int key);  };  class GameClearScene  {  protected:  int width, height;  public:  GameClearScene();  WINDOW \*renderGameClearScene();  };  class CongratulationsScene  {  protected:  int width, height;  public:  CongratulationsScene();  WINDOW \*renderCongratulationsScene();  }; |

ETCScene은 기본 ncurses 설정 화면을 띄우는 Scene과, 게임 화면을 관리하는 gameScene을 제외한 나머지 부가적인 창을 띄우는 역할을 합니다.

GameStartScene은 생성자\*\*에서 멤버변수들을 적절한 수로 초기화합니다. 멤버 변수 중 height와 width는 ncurses에서 창의 크기를 구현하기 위한 설정값들이며, selectedMode와 selected, 그리고 pastKey는 화면 내에서 키를 입력받아 사용자가 선택한 게임 옵션을 확인하고 그에 따라 창을 유지할지, 게임 창으로 넘어갈지, 아니면 게임을 종료할 지 결정하기 위해 필요한 멤버변수입니다.

GameStartStatus() 는 사용자가 입력한 키를 검사하여 어떠한 상황인지 return 하는 역할을 합니다. 사용자가 엔터를 누르지 않은 상황에서는 -1을 리턴하며, START를 활성화 해둔 상황에서 엔터를 입력받으면 0을 return, 그리고 EXIT을 활성화 해둔 상태에서 엔터키를 입력받으면 1을 return합니다. 이 return 값은 창을 관리하는 main 인자에서 사용합니다.

renderGameOverScene()은 WINDOW\*, 즉 ncurses에서 띄울 수 있는 창을 return하는 함수입니다. 이 함수를 호출하면 winGameStart라는 윈도우를 생성하여 필요한 내용들을 출력하는 창을 띄웁니다. 메인 화면들의 디자인은 Framework 폴더에 각각 맞추어서 구현이 되어 있으며, 이 함수에서는 해당 파일의 내용을 불러와서 mvwprintw 함수를 통해 창에 내용을 출력합니다. 여기서 구현한 winGameStart에서는 팀의 이름인 ‘저쩔스네이크(JEO JEOUL SNAKE)’와 함께 뱀 디자인, 게임 시작(GAME START)과 프로그램 종료(EXIT), 그리고 제작자 정보가 들어가 있습니다. 이후, getch() 함수를 통해 위 방향키/아래 방향키/엔터 를 입력 받을 수 있으며 방향키를 통해 게임 시작과 프로그램 종료 중 하나를 선택할 수 있습니다. 이후 엔터키를 누르면 해당 모드가 확정됩니다. 이 과정에서 게임 시작 선택 시에는 클래스의 멤버 변수인 selectedMode를 1, selected를 0, 그리고 프로그램 종료 선택 시에는 selectedMode를 0, selected를 0, 그리고 엔터 입력 시 selected를 1로 바꿔주는 역할을 합니다.

GameOverScene은 전반적인 멤버 변수와 기능이 GameStartScene과 매우 유사합니다. 다만 해당 Scene은 winGameOver라는 윈도우를 생성하며, 가져오는 파일의 내용에는 스네이크 오버(SNAKE OVER)라는 내용과 함께 메인 화면으로(BACK TO MAIN MENU), 종료(EXIT) 두 가지를 선택할 수 있습니다. 멤버 변수인 selectedMode와 selected를 통해 사용자가 입력한(getch()) 내용을 확인합니다.

GameClearScene은 생성자에서 GameStartScene에서 키 입력 기능을 없애 화면에 윈도우를 할당하고 표시하는 기능만 수행합니다. GameClearScene에는 스테이지 클리어 (STAGE CLEAR) 라는 내용과 함께 뱀 그림, 그리고 다음 스테이지 (NEXT STAGE) 가 표시됩니다.

CongratulationsScene은 GameClearScene과 매우 유사한 기능을 수행합니다. 게임 클리어 (GAME CLEAR)라는 내용과 함께 뱀 그림, 그리고 PRESS ANY KEY TO EXIT를 표시합니다.

**Mission.h** 개발자: 김진재

|  |
| --- |
| #include <ncurses.h>  #include <iostream>  #include <string>  #include "Snake.h"  class Mission  {  private:  int width, height;  int misLength;  int misGrowth;  int misPoison;  int misGate;  bool length;  bool growth;  bool poison;  bool gate;  public:  Mission(int length, int growth, int posion, int gate);  WINDOW \*updateMissionBoard(Snake &snake);  int getMisLength() { return misLength; }  int getMisGrowth() { return misGrowth; }  int getMisPoison() { return misPoison; }  int getMisGate() { return misGate; }  bool missionAllCleared();  }; |

Mission은 게임이 진행중인 동안 스테이지별 주어져야 할 미션을 파악하고 미션 창을 구현하는 역할을 합니다.

Mission의 생성자는 main에서 부르는데, 이 때 각 stage에 맞는 미션 수들을 가지고 와서 멤버 변수인 misLength, misGrowth, misPoison, misGate를 초기화합니다. 이후 이 변수들은 미션보드를 띄우는 함수인 updateMissionBoard\*\*()\*\*에서 활용합니다.

updateMissionBoard()는 WINDOW\*를 return하는 함수입니다. 이 윈도우는 게임 규칙 #6을 준용하여 만들어졌으며, Mission 이라는 창 이름 아래줄에 snake 객체의 정보를 받아와서 snake가 미션 성공을 위해 가져야 할 최소 길이, 최소 Growth Item 개수, 최대 Poison Item 개수, 그리고 마지막으로 gate 사용 횟수를 각 줄마다 출력합니다. 이 함수는 불러질 때마다 창이 refresh되어 상황에 맞는 정보를 표시합니다.

**ScoreBoard.h** 개발자 : 김진재

|  |
| --- |
| #include <ncurses.h>  #include <string>  #include "Snake.h"  class ScoreBoard  {  protected:  int width, height;  public:  ScoreBoard();  WINDOW \*updateScoreBoard(Snake snake);  }; |

ScoreBoard는 게임이 진행중인 동안 snake가 현재 어떠한 상태이고 미션을 얼마나 수행하였는지를 나타내는 board 역할을 합니다.

updateScoreBoard()는 WINDOW\*를 return하는 함수입니다. 이 윈도우는 게임 규칙 #6을 준용하여 만들어졌으며, ScoreBoard 라는 창 이름 아래줄에 snake 객체의 정보를 받아와서 snake의 현재 길이 / snake의 최대 길이, 그 아래에 획득한 Growth Item 개수, 그 아래에 획득한 Poison Item 개수, 그리고 Gate 통과 횟수를 각 줄에 출력합니다. 이 함수는 불러질 때마다 창이 refresh되어 상황에 맞는 정보를 표시합니다.

**TimeBoard.h** 개발자 : 김진재, 허승범

|  |
| --- |
| #include <ncurses.h>  #include <string>  #include <cmath>  class TimeBoard  {  protected:  int width, height;  double time;  double stagetime;  public:  TimeBoard();  void setTime();  WINDOW \*updateTimeBoard();  }; |

TimeBoard는 게임이 진행중인 동안 시간이 얼마나 흘렀는지 지속적으로 보여주는 클래스입니다. 생성자를 불러오면 time과 stagetime을 0으로 초기화합니다.

updateTimeBoard() 는 WINDOW\*를 return하는 함수입니다. Game Time 라는 창 이름 아래 줄에 Total time으로 현재 게임을 시작하고 지난 총 시간과 Stage time으로 현재 스테이지에서 게임을 시작하고 지난 총 시간을 측정합니다. 이 때, time과 stagetime은 main에서 게임을 돌리는 while문이 실행될 때(즉, 게임에서 snake가 이동할 때=0.5초)마다 0.5씩 더해집니다.

**Stage.h && Stage.cpp** 개발자 : 김진재, 허승범, 이현영

|  |
| --- |
| #include <vector>  #include <iostream>  #include <fstream>  #include <string>  #include <sstream>  #include "Snake.h"  #include "Item.h"  #include "Gate.h"  #include "Info.h"  class MapSet  {  int row;  int col;  char \*\*map;  public:  void LoadMap(int mapNum);  void setMap(int row, int col, int num);  void printSnake(Snake snake);  void printItem(Item item);  void printGate(Gate gate);  char \*getMap(int row);  char \*printMap(int row);  int getRow();  int getCol();  bool checkMapElement(std::vector<std::vector<int>> snakeBody);  friend std::ostream &operator<<(std::ostream &os, const MapSet &map);  }; |

|  |
| --- |
| #include "Stage.h"  void MapSet::LoadMap(int mapNum)  {  Info info;  Info::allWallLoc.clear();  Info::wallLoc.clear();  Info::planeLoc.clear();  std::ifstream mapfile;  srand((unsigned)time(NULL));  mapfile.open("./Framework/stage" + std::to\_string(mapNum), std::ios\_base::in | std::ios\_base::binary);  std::ofsream planeLog("./log/planeLog" + std::to\_string(mapNum));  if (mapfile.fail())  {  std::cout << "Map file does not exist.";  }  else  {  char ch;  std::string line;  std::vector<std::string> linearr;  while (getline(mapfile, line))  {  linearr.push\_back(line);  }  std::istringstream stream(linearr[0]);  std::vector<int> mapinfoarr;  std::string mapinfo;  stream >> row;  stream >> col;  map = new char \*[row];  for (int i = 0; i < row; i++)  map[i] = new char[col];  for (int i = 0; i < row; i++)  {  for (int j = 0; j < col; j++)  {  map[i][j] = linearr[i + 1][j];  if ((map[i][j] == '1') || (map[i][j] == '2'))  {  std::vector<int> temp{i, j};  info.setAllWallLoc(temp);  if (map[i][j] == '1')  info.setWallLoc(temp);  }  else if (map[i][j] == '0')  {  std::vector<int> temp{i, j};  info.setPlaneLoc(temp);  planeLog << i << " " << j << std::endl;  }  }  }  mapfile.close();  planeLog.close();  }  }  void MapSet::setMap(int row, int col, int num)  {  // char type map에 접근하여 변경. 사용법: row, column, 바꿀 수  this->map[row][col] = num + '0';  }  void MapSet::printSnake(Snake snake)  {  for (int i = 0; i < snake.getSnakeBody().size(); i++)  {  setMap(snake.getSnakeBody()[i][0],  snake.getSnakeBody()[i][1], snake.getSnakeBody()[i][2]);  }  }  void MapSet::printItem(Item item)  {  setMap(item.getX(), item.getY(), item.getValue());  }  void MapSet::printGate(Gate gate)  {  setMap(gate.getX(), gate.getY(), gate.getValue());  }  char \*MapSet::getMap(int row)  {  return (this->map[row]);  }  char \*MapSet::printMap(int row)  {  Info info;  char \*tmp = this->map[row];  for (int i = 0; i < getCol(); i++)  {  if (tmp[i] == '0')  tmp[i] = ' ';  }  return tmp;  }  int MapSet::getRow()  {  return (row);  }  int MapSet::getCol()  {  return (col);  }  std::ostream &operator<<(std::ostream &os, const MapSet &m)  {  for (int i = 0; i < m.row; i++)  {  for (int j = 0; j < m.col; j++)  {  os << m.map[i][j];  }  os << "\n";  }  return (os);  } |

Stage.h는 Snake Game에서 화면에 출력될 map의 정보를 갖고 있는 파일로 Mapset클래스입니다.

Mapset의 멤버 변수로는 map의 size정보를 담고 있는 row와 col, 그리고 char배열로 된 map이 있습니다.

\*\*LoadMap()\*\* 함수에서는 먼저 info 클래스에 있는 allWallLoc, wallLoc, planeLoc의 정보를 초기화해줍니다. 그리고 텍스트 파일로 된 map의 정보를 읽어 char배열인 map에 값을 넣어줍니다. 값을 넣어주면서 map의 값을 판단하여 초기화된 allWallLoc, wallLoc, planeLoc에 다시 값을 넣어줍니다.

**Snake.h** 개발자 : 허승범, 김진재, 이현영

|  |
| --- |
| #pragma once  #include <vector>  #include <string>  #include <ncurses.h>  #include <unistd.h>  #include <iostream>  class Snake  {  private:  // score board 변수  int maxLength = 12;  int growthItems = 0;  int poisonItems = 0;  int gateUses = 0;  std::vector<std::vector<int>> snakeBody;  int direction; // snake의 머리 진행 방향  // 1 // 1 북  // 4 2 // 4 서 2 동  // 3 // 3 남  bool isAlive;  int pastKey;  std::vector<int> pastTail;  public:  Snake();  void setAttribute();  // getter and setter  int getDirection() { return direction; }  void setDirection(int direction) { this->direction = direction; }  bool getIsAlive() { return isAlive; }  void setIsAlive(bool isAlive) { this->isAlive = isAlive; }  std::vector<std::vector<int>> getSnakeBody() { return this->snakeBody; }  std::vector<std::vector<int>> getSnakeInfo();  void addSnakeBody(int row, int col, int num);  void snakeGrown();  void snakePoisoned();  void clearSnake();  void moveSnakeHead();  int understandKey(int key);  void setPastKey(int key);  int getPastKey();  // score board accessor  int getCurrentLength() { return snakeBody.size(); }  void setMaxLength(int ml) { maxLength = ml; }  int getMaxLength() { return maxLength; }  void incGrowthItems() { growthItems++; }  int getGrowthItems() { return growthItems; }  void incPoisonItems() { poisonItems++; }  int getPoisonItems() { return poisonItems; }  void incGateUses() { gateUses++; }  int getGateUses() { return gateUses; }  }; |

Snake.h는 Snake Game에서 Player가 조종할 Snake가 가져야 하는 정보들과 기능들이 담겨있는 파일입니다.

**Snake**의 멤버 변수로는 snake의 머리부터 몸통까지의 정보를 갖고 있는 snakeBody 벡터, snake의 머리 진행 방향을 가지고 있고 direction, snake의 생존 여부 정보를 갖고 있는 isAlive, Player가 전에 keypad에 눌렀었던 key값을 갖고 있는 pastKey, snake의 과거 정보를 갖고 있는 pastTail 벡터와 score board에 넘겨줄 변수인 snake가 가질 수 있는 최대 크기인 maxLength, 그리고 growthItem과 posionItem을 먹은 개수인 grothItems과 posionItems 마지막으로 gate 사용 횟수를 저장해주는 gateUses변수가 있습니다.

여기서 directon 변수의 방향은 1일 때는 북, 2일 때는 동, 3일 때는 남쪽 그리고 4일 때는 서쪽을 의미합니다.

**Snake의 생성자**에서는 Snake Game이 시작됐을 때 처음 snake가 나타날 좌표와, 머리 그리고 몸통의 정보를 저장합니다.

**setAttribute()** 함수는 snake의 기본 설정인 direction, maxLength, growthItems, poisonItems, gateUses 변수들의 초기값을 설정해주는 함수입니다.

**addSnakeBody()** 함수는 실질적으로 객체 snake의 이동을 담당합니다. snake가 앞으로 나아가기 위해서는 snake의 방향정보를 받아와서 그 방향에 맞는 다음 위치로 머리를 옮긴 후, 기존 머리를 3으로 수정하며 마지막 위치에 있었던 snake의 꼬리를 삭제하면 실질적으로 플레이어에게 snake가 이동한 것처럼 보입니다. 이러한 과정을 수행하기 위해 snake 벡터에 있는 내용에 접근하여 방향의 앞 부분에 머리 좌표를 추가하고, 그 이후 기존 snake 머리였던 부분 표시의 ‘3’을 ‘4’로 변경하며, 마지막으로 vector 끝부분을 pop\_back 합니다.

**clearSnake()** 함수는 한 stage를 클리어할 때 마다 snake의 정보를 초기 상태로 설정하는 함수입니다.

**moveSnakeHead()** 함수는 snake의 현재 진행 direction을 받아와서 snake의 머리 부분이 어느 방향으로 꺾여야 하는지 판단하여 snake의 머리 부분을 addSnakeBody() 함수로 설정해주는 함수 입니다.

**understandKey()** 함수는 key를 매개 인자로 받아와서 key의 값에 따라 어떤 행동을 할지 판단하여 return해주는 함수입니다.

만약 key값이 바로 이전에 받았던 값과 같다면 -3을 return 해주고, key가 ERR값을 가지고 있는 경우에는 현재 진행 방향에 맞는 키 값을 key에 넣어주고, switch문을 사용해 key값을 판단하여 setDirection() 함수를 사용해 방향을 설정해 줍니다.

**Scene.h && Scene.cpp** 개발자 : 허승범, 이현영, 김진재

|  |
| --- |
| #include "Stage.h"  #include "ETCScene.h"  #include "Item.h"  #include "Gate.h"  #include <vector>  #include <algorithm>  #include <fstream>  #include <iostream>  #include <fstream>  class Scene  {  protected:  int width, height;  public:  Scene();  WINDOW \*changeScene(int stage, Snake snake);  WINDOW \*gamingScene(int stage, MapSet &mapset, Snake &snake, Item &growth, Item &poison, Gate &fGate, Gate &sGate);  }; |

|  |
| --- |
| #include "Scene.h"  Scene::Scene()  {  Scene::width = 100;  Scene::height = 100;  initscr();  resize\_term(height, width);  curs\_set(0);  noecho();  border('|', '|', '-', '-', '\*', '\*', '\*', '\*');  refresh();  return;  }  WINDOW \*Scene::gamingScene(int stage, MapSet &mapset, Snake &snake, Item &growth, Item &poison, Gate &fGate, Gate &sGate)  {  WINDOW \*winGaming;  winGaming = newwin(23, 58, 4, 4);  if (stage != 0)  {  Info info;  mapset.LoadMap(stage);  // Item //  if ((growth.getTime() % 50 == 0)) // 10?? ?? resetItem  {  growth.resetItem();  }  if ((poison.getTime() % 50 == 0)) // 10?? ?? resetItem  {  poison.resetItem();  }  mapset.printItem(growth);  mapset.printItem(poison);  // Gate  if (info.MakeGate == true)  {  mapset.printGate(fGate);  mapset.printGate(sGate);  }  info.setSnakeLoc(snake);  for (int i = 0; i < info.allWallLoc.size(); i++)  {  if (info.snakeLoc[0] == info.allWallLoc[i])  {  if (info.MakeGate == true)  {  if (info.snakeLoc[0] != info.gateLoc[0] && info.snakeLoc[0] != info.gateLoc[1])  {  snake.setIsAlive(false);  return winGaming;  }  }  else  {  snake.setIsAlive(false);  return winGaming;  }  }  }  for (int i = 1; i < info.snakeLoc.size(); i++)  {  if (info.snakeLoc[0] == info.snakeLoc[i])  {  snake.setIsAlive(false);  return winGaming;  }  }  if (info.MakeGate)  {  if (info.snakeLoc.at(0) == info.gateLoc.at(0) || info.snakeLoc.at(0) == info.gateLoc.at(1))  {  std::ofstream gateLog;  gateLog.open("./log/gateLog", std::ios::app);  std::vector<int> inGate = (info.snakeLoc.at(0) == info.gateLoc.at(0)) ? info.gateLoc.at(0) : info.gateLoc.at(1);  std::vector<int> outGate = (info.snakeLoc.at(0) == info.gateLoc.at(0)) ? info.gateLoc.at(1) : info.gateLoc.at(0);  int newHeadRow = outGate.at(0);  int newHeadCol = outGate.at(1);  gateLog << "in gate : " << inGate.at(0) << " " << inGate.at(1) << " ";  gateLog << "out gate : " << newHeadRow << " " << newHeadCol << std::endl;  // edge gate  if ((0 < newHeadRow && newHeadRow < 20) && (newHeadCol == 0)) // left edge -> direction is right  {  snake.addSnakeBody(newHeadRow, newHeadCol, 3);  snake.setDirection(2);  }  else if ((0 < newHeadRow && newHeadRow < 20) && (newHeadCol == 40)) // right edge -> direction is left  {  snake.addSnakeBody(newHeadRow, newHeadCol, 3);  snake.setDirection(4);  }  else if ((0 < newHeadCol && newHeadCol < 40) && (newHeadRow == 0)) // top edge -> direction is bottom  {  snake.addSnakeBody(newHeadRow, newHeadCol, 3);  snake.setDirection(3);  }  else if ((0 < newHeadCol && newHeadCol < 40) && (newHeadRow == 20)) // bottem edge -> direction is top  {  snake.addSnakeBody(newHeadRow, newHeadCol, 3);  snake.setDirection(1);  }  else // inner edge  {  int tempRow;  int tempCol;  if (snake.getDirection() == 1) // in direction == top -> row--  {  tempRow = newHeadRow - 1;  tempCol = newHeadCol;  // check can remain direction  std::vector<int> temp{tempRow, tempCol};  auto it = std::find(info.allWallLoc.begin(), info.allWallLoc.end(), temp);  if (it != info.allWallLoc.end())  snake.setDirection(2);  }  else if (snake.getDirection() == 2) // in direction == right -> col++  {  tempRow = newHeadRow;  tempCol = newHeadCol + 1;  std::vector<int> temp{tempRow, tempCol};  auto it = std::find(info.allWallLoc.begin(), info.allWallLoc.end(), temp);  if (it != info.allWallLoc.end())  snake.setDirection(3);  }  else if (snake.getDirection() == 3) // in direction == bottom -> row++  {  tempRow = newHeadRow + 1;  tempCol = newHeadCol;  std::vector<int> temp{tempRow, tempCol};  auto it = std::find(info.allWallLoc.begin(), info.allWallLoc.end(), temp);  if (it != info.allWallLoc.end())  snake.setDirection(4);  }  else if (snake.getDirection() == 4) // in direction == left -> col--  {  tempRow = newHeadRow;  tempCol = newHeadCol - 1;  std::vector<int> temp{tempRow, tempCol};  auto it = std::find(info.allWallLoc.begin(), info.allWallLoc.end(), temp);  if (it != info.allWallLoc.end())  snake.setDirection(1);  }  snake.addSnakeBody(newHeadRow, newHeadCol, 3);  }  snake.incGateUses();  gateLog.close();  }  }  if (info.snakeLoc[0] == info.growthLoc)  {  growth.resetItem();  snake.incGrowthItems();  snake.snakeGrown();  }  if (info.snakeLoc[0] == info.poisonLoc)  {  poison.resetItem();  snake.incPoisonItems();  snake.snakePoisoned();  if (snake.getCurrentLength() < 3)  {  snake.setIsAlive(false);  return winGaming;  }  }  snake.moveSnakeHead();  mapset.printSnake(snake);  std::string stage\_string = "Stage " + std::to\_string(stage);  auto charStage = stage\_string.c\_str();  mvwprintw(winGaming, 0, 0, charStage);  for (int i = 0; i < mapset.getRow(); i++)  mvwprintw(winGaming, i + 2, 0, mapset.printMap(i));  }  wrefresh(winGaming);  usleep(500000);  return winGaming;  }  WINDOW \*Scene::changeScene(int stage, Snake snake)  {  WINDOW \*winScene;  winScene = newwin(25, 60, 3, 3);  start\_color();  init\_pair(0, COLOR\_BLUE, COLOR\_BLACK);  init\_pair(1, COLOR\_RED, COLOR\_BLACK);  init\_pair(2, COLOR\_YELLOW, COLOR\_BLACK);  init\_pair(3, COLOR\_MAGENTA, COLOR\_BLACK);  init\_pair(4, COLOR\_WHITE, COLOR\_BLACK);  wborder(winScene, '\*', '\*', '\*', '\*', '\*', '\*', '\*', '\*');  // Scene window control  wbkgd(winScene, COLOR\_PAIR(stage));  wattron(winScene, COLOR\_PAIR(stage));    wrefresh(winScene);  return winScene;  } |

Scene.h는 Snake Game에서 가장 핵심 적이 부분을 맡고 있는 부분입니다. Scene.h에서는 stage가 바뀔 때마다 장면을 바꿔 주는 부분과 게임이 0.5초씩 업데이트 될 때 불러는 함수를 갖고 있는 클래스 입니다.

**Scene의 생성자**에서는 ncurses를 활용하기 위한 스크린을 띄어주는 함수를 불러 cmd창에 자신이 설정한 값에 따라 화면을 업데이트 해줍니다.

**gamingScene()** 함수 는 window를 하나 만들어 현재 맵의 상태를 cmd창에 업데이트를 해주는 함수입니다.

매개 인자로는 현재 스테이지의 index인 stage 변수,맵의 정보를 갖고 있는 mapset, main함수에서 다뤄지는 snake, 그외에 Item들과 Gate가 있습니다.

먼저 stage의 index에 따라 LoadMap()을 이용하여 mapset을 변경해줍니다. 그리고 item들의 리젠타임에 맞게 리셋을 해줄지 안해줄지 판단을 합니다. 판단을 한 후 map에 item들의 좌표를 설정해줍니다.

Gate를 출력해줄지 판단합니다. 이 때 Info의 MakeGate 변수를 이용해서 변수가 최초 생성이 되었는지 확인합니다. MakeGate 를 사용하지 않으면 Info의 Gate 정보가 생성된 적 없기 때문에 참조 오류를 일으켜 반드시 필요한 판단 조건 입니다.

이후 snake에게 이벤트가 일어났는지 판단하는 조건문을 실행시킵니다. 순서는 아래와 같습니다.

먼저 Info에서 새로 움직인 snake의 정보를 갱신시킵니다. 이후 snake의 머리가가 새로 움직인 좌표값이

1. 벽의 위치인지

1. 게이트가 생성되었는지

1. 게이트가 생성되었다면 게이트가 아닌 일반 벽의 위치인지

→ 참이면 snake의 alive 정보를 false로 설정 후 리턴

2. snake의 몸 위치인지

→ 참이면 snake의 alive 정보를 false로 설정 후 리턴

3. 게이트의 위치인지

1. 가장자리 벽 위치인지

→ 반대편 gate의 위치로 snake 머리 위치 조정

→ 벽의 진출방향에 따라 snake의 direction 조정

2. 내부 벽 위치인지

→ 반대편 gate의 기존 snake 방향 한 칸 앞이 벽이 아닌지 확인

→ 방향 시계방향으로 조정 후 snake 머리 위치 반대편 gate의 위치로 조정

→ snake의 gateuse 횟수 추가

4. snake의 머리 위치가 growth item의 위치인지

→ 참이면 growth item 초기화

→ snake의 growth item 획득 횟수 추가

→ snake의 몸 길이 추가

5. snake의 머리 위치가 poison item의 위치인지

→ 참이면 poison item 초기화

→ snake의 poison item 획득 횟수 추가

→ snake의 몸 길이 감소

모든 판단이 다 끝나고 설정을 마친 뒤에 wrefresh() 함수를 이용하여 윈도우를 cmd창에 업데이트 해줍니다.

마지막에 usleep() 함수를 이용하여 delay를 줍니다.

\*\*changeScene()\*\* 함수는 stage가 바뀔 때마다 구별하기 편하게 주변 테두리의 색을 바꿔 윈도우를 업데이트 해주는 함수입니다.

**Main.cpp** 개발자 : 김진재, 허승범, 이현영

|  |
| --- |
| #include <ncurses.h>  #include "Scene.h"  #include "Snake.h"  #include "ScoreBoard.h"  #include "Mission.h"  #include "TimeBoard.h"  #include "Item.h"  #include "Gate.h"  using namespace std;  int main()  {  // create default window including game screen, score board, user name  Info info;  Scene scene;  MapSet mapset;  Snake snake;  TimeBoard timeboard;  GameStartScene gameStartScene;  GameOverScene gameOverScene;  GameClearScene gameClearScene;  CongratulationsScene congratulationsScene;  Gate fGate;  Gate sGate;  int key;  int chkstage = 0;  WINDOW \*winGaming;  keypad(stdscr, TRUE);  key = KEY\_UP;  gameStartScene.renderGameStartScene(key);  while (gameStartScene.GameStartStatus() == -1)  {  key = getch();  gameStartScene.renderGameStartScene(key);  if (gameStartScene.GameStartStatus() == 0)  {  endwin();  return 0;  }  else if (gameStartScene.GameStartStatus() == 1)  break;  }  scene = Scene();  int len, grth, pois, gate;  len = 5, grth = 2, pois = 4, gate = 1;  for (int i = 1; i < 5; i++)  {  scene = Scene();  snake.clearSnake();  ScoreBoard scoreBoard;  Mission missionBoard(len, grth, pois--, gate);  len += 2, grth += (len / 2), gate \*= 2;  chkstage = 0;  Item growthItem(5);  Item poisonItem(6);  timeboard.setTime();  scene.changeScene(i, snake);  key = KEY\_RIGHT;  snake.setPastKey(key);  info.MakeGate = false;  while (snake.getIsAlive())  {  scoreBoard.updateScoreBoard(snake);  missionBoard.updateMissionBoard(snake);  // reset Gate when snake's length is evenNum  if ((!info.gateExistence) && (snake.getGrowthItems() != 0) && (snake.getGrowthItems() % 2 == 0))  {  info.gateLoc.clear();  fGate.resetGate();  sGate.resetGate();  }  if (snake.getGrowthItems() % 2 != 0)  {  info.gateExistence = false;  info.gateExistence = false;  }  nodelay(stdscr, TRUE);  key = getch();  int check = snake.understandKey(key);  if (check == -3)  continue;  if (missionBoard.missionAllCleared())  {  chkstage = 1;  break;  }  winGaming = scene.gamingScene(i, mapset, snake, growthItem, poisonItem, fGate, sGate);  timeboard.updateTimeBoard();  growthItem.upTime();  poisonItem.upTime();  }  nodelay(stdscr, FALSE);  scene = Scene();  if (chkstage == 1 && i == 4)  {  break;  }  if (chkstage == 1)  {  gameClearScene.renderGameClearScene();  getch();  continue;  }  else  {  key = KEY\_UP;  gameOverScene.renderGameOverScene(key);  while (gameOverScene.GameOverStatus() == -1)  {  key = getch();  gameOverScene.renderGameOverScene(key);  if (gameOverScene.GameOverStatus() == 0)  {  endwin();  return 0;  }  else if (gameOverScene.GameOverStatus() == 1)  {  main();  return 0;  }  }  }  snake.clearSnake();  }  congratulationsScene.renderCongratulationsScene();  getch();  delwin(winGaming);  endwin();  return 0;  } |

**Main.cpp** 는 전체적인 코드 실행에 관한 파일입니다.

**main**에서는 게임을 실행하는 부분에 있어 모든 부분을 제어하고 있습니다. 먼저 main에서 game을 진행하는 데에 필요한 Info, Scene, Mapset, Snake, Timeboard, GameStartScene, GameOverScene, GameClearScene, CongratulationScene, Gate의 인스턴스를 생성합니다.

그리고 keypad() 함수를 사용하여 keypad의 사용을 true로 설정해줍니다.

게임을 시작하기 위해 gameStartScene을 사용하여 게임을 시작할건지 끝낼건지를 판단하기 위한 while문을 돕니다.

위에서 게임이 시작된다고 판단이 되면 Scene의 생성자로 window를 clear해줍니다.

게임이 시작되면 맵의 index를 결정해주는 for문에 들어갑니다. 먼저 snake 게임을 하는데에 필요한 snake, missionBoard, items, 등을 설정해줍니다.

설정이 끝난 뒤 snake가 죽었는지 살았는지를 계속해서 판단하며 게임에 필요한 window들을 update 해주면서 while문을 돕니다.

while문에서 먼저 scoreBoard와 missionBoard를 update 해줍니다. 이 후 gate가 존재하는지, snake의 growthitem의 개수가 0인지, 만약 0이 아니고 짝수 개를 먹었다면 gate를 만들어줍니다. snake가 먹은 growthitem의 개수가 홀수인 경우에는 gateExistence의 값을 false로 변경해줍니다.

key값을 받고 그 key값을 snake.understandKey(key)로 그 키값이 어떠한 행동을 할 건지에 대해 판단을 한 후 return 받은 값을 check에 넣어줍니다. 만약 check가 -3일 경우에는 이전 key값과 동일 하다는 의미로 아무런 행동을 하지않는다고 판단하여 continue를 해줍니다. 그리고 mission을 클리어했다면 break하고 다음 stage로 넘어갑니다. 이렇게 main에서 결정난 사항들을 scene.gamingScene() 함수를 불러 window를 업데이트 해줍니다.

gamingScene() 함수의 인자로는 현재 스테이지, 맵 객체, 스네이크객체, 두개의 아이템 객체, 두개의 게이트 객체를 넘겨주어 메인에서 만들어진 객체들이 Scene 안에서 동작하게끔 만들어줍니다.

정상적으로 while문을 나왔다면 죽었는지 살았는지를 판별하여 GameClearScene을 보여줄건지, GameOverScene을 보여줄건지 판별합니다. 모든 stage를 정상적으로 다 clear했다면 마지막에 CongratulationScene을 보여주며 게임이 끝납니다.

**Info.h && Info.cpp** 개발자 : 이현영

|  |
| --- |
| #pragma once  #include <vector>  #include "Snake.h"  using namespace std;  class Info  {  public:  static vector<vector<int>> snakeLoc;  static vector<vector<int>> allWallLoc;  static vector<vector<int>> wallLoc;  static vector<vector<int>> planeLoc;  static vector<int> growthLoc;  static vector<int> poisonLoc;  static vector<vector<int>> gateLoc;  static bool MakeGate;  static bool gateExistence;  Info();  static void setSnakeLoc(Snake snake);  void setAllWallLoc(vector<int> v);  void setWallLoc(vector<int> v);  void setPlaneLoc(vector<int> v);  void setGrowthLoc(int x, int y);  void setPoisonLoc(int x, int y);  void setGateLoc(vector<int> v);  }; |

|  |
| --- |
| #include "Info.h"  vector<vector<int>> Info::snakeLoc;  vector<vector<int>> Info::allWallLoc;  vector<vector<int>> Info::wallLoc;  vector<vector<int>> Info::planeLoc;  vector<int> Info::growthLoc = {0, 0};  vector<int> Info::poisonLoc = {0, 0};  vector<vector<int>> Info::gateLoc;  bool Info::MakeGate = false;  bool Info::gateExistence = false;  Info::Info(){};  void Info::setSnakeLoc(Snake snake)  {  snakeLoc = snake.getSnakeInfo();  }  void Info::setAllWallLoc(vector<int> v)  {  allWallLoc.push\_back(v);  }  void Info::setWallLoc(vector<int> v)  {  wallLoc.push\_back(v);  }  void Info::setPlaneLoc(vector<int> v)  {  planeLoc.push\_back(v);  }  void Info::setGrowthLoc(int x, int y)  {  growthLoc.clear();  growthLoc.push\_back(x);  growthLoc.push\_back(y);  }  void Info::setPoisonLoc(int x, int y)  {  poisonLoc.clear();  poisonLoc.push\_back(x);  poisonLoc.push\_back(y);  }  void Info::setGateLoc(vector<int> v)  {  gateLoc.push\_back(v);  } |

Info 클래스는 게임에서 필요한 요소들의 정보를 저장하는 클래스입니다.

Info 클래스의 멤버 변수는 모두 static으로 선언하여 모든 클래스에서 같은 정보를 사용할 수 있도록 했습니다. 따라서 어느 클래스에서 저장한 내용이든 다른 클래스에서도 그 정보에 접근하는 DB의 개념으로 사용했습니다.

Info 클래스를 정의한 목적은 snake가 벽, item, gate를 만났을 경우를 판단하기 위함와 Item 생성, Gate 생성시 좌표값을 지정하기 위해서 입니다.

멤버변수로는 다음과 같은 변수들이 있습니다

**static vector<vector<int>> snakeLoc**

현재 snake의 위치 좌표를 저장하는 변수입니다. vector<int>(snake의 머리~몸통의 좌표)를 요소로 가지는 vector로 구성되어 있습니다. snake의 몸통이 움직일 때 마다 정보가 업데이트 됩니다. static이지만 setter 메소드를 사용하여 책임을 클래스에 할당시켰습니다. setSnakeLoc(Snake snake) 메소드는 snake 객체를 전달받아 해당 snake 객체의 위치 정보를 변수에 저장하여 갱신합니다.

**static vector<vector<int>> allWallLoc**

immune wall을 포함한 전체 벽 위치를 저장하는 변수입니다. vector<int>(wall의 좌표)를 요소로 가지는 vector로 구성되어 있습니다. MapSet에서 Loadmap 을 할 때 주로 실행되며 각 스테이지마다 벽의 위치가 임의적으로 다르기 때문에 스테이지를 불러오며 출력함과 동시에 해당값이 1과 2일 경우 allWallLoc에 좌표값을 추가합니다. 이 변수는 snake가 벽을 만났을 경우를 판단하는데 사용됩니다.

**static vector<vector<int>> wallLoc**

immune wall을 제외한 벽 위치를 저장하는 변수입니다. vector<int>(wall의 좌표)를 요소로 가지는 vector로 구성되어 있습니다. MapSet에서 Loadmap을 할 때 스테이지를 불러오며 출력함과 동시에 해당값이 1일 경우에만 wallLoc에 좌표값을 추가합니다. 이 변수는 gate를 리셋할 때 범위를 지정하기 위해 사용됩니다.

**static vector<vector<int>> planeLoc**

모든 wall을 제외한 좌표값을 저장하는 변수입니다. vector<int>(맵의 좌표)를 요소로 가지는 vector로 구성되어 있습니다. MapSet에서 Loadmap을 할 때 스테이지를 불러오며 출력함과 동시에 해당값이 0일 경우에만 planeLoc에 좌표값을 추가합니다. 이 변수는 Item을 리셋할 때 범위를 지정하기 위해 사용됩니다.

**static vector<int> growthLoc**

Growth Item의 좌표 정보를 저장하기 위한 변수입니다. x, y 순서대로 저장하는 vector<int>로 구성되어 있습니다. growth item이 리셋될 경우, **setGrowthItem()** 메소드가 실행되어 기존에 저장되어있던 growth item 정보가 삭제되고, 새로 추가된 growth item의 x좌표값과 y좌표값을 순서대로 추가합니다.

**static vector<int> poisonLoc**

Poison Item의 좌표 정보를 저장하기 위한 변수입니다. x, y 순서대로 저장하는 vector<int>로 구성되어 있습니다.poison growth item이 리셋될 경우, \*\*setPoisonItem()\*\* 메소드가 실행되어 기존에 저장되어있던 poison item 정보가 삭제되고, 새로 추가된 poison item의 x좌표값과 y좌표값을 순서대로 추가합니다.

**static vector<vector<int>> gateLoc**

Gate의 좌표 정보를 저장하기 위한 변수입니다. vector<int>(gate의 좌표)를 요소로 가지는 vector로 구성되어 있습니다. Gate는 한 쌍이기 때문에 두개의 좌표정보를 한번에 저장해야 합니다. Gate가 리셋 될 경우 기존에 있던 gate 정보를 삭제하고 새로운 gate 좌표정보 배열을 추가합니다. setGateLoc 내부에서 gate 정보를 초기화하면 두번째 gate를 추가할 때에도 다시 초기화가 되기때문에 초기화는 gate를 리셋하는 로직에서 따로 실행합니다.

**static bool MakeGate**

Gate가 생성되었는지를 판단하는 변수입니다. Gate의 출현 조건은 snake가 아이템을 2개를 먹은 이후이고, 2개를 먹을 때 마다 리셋됩니다. 따라서 스테이지 시작 후 snake가 item을 두 개 먹기 전까지는 MakeGate를 false로 지정하고, gate의 최초 생성 이후에는 MakeGate를 true로 지정합닌다. 이는 snake의 머리가 있는 위치가 gate를 만났는지 판단하는 로직이 info에 아직 gate의 정보가 없는데에도 판단하는 로직이 실행되어 segmentation fault가 발생하는것을 막기 위해 정의한 변수입니다.

**static bool gateExistence**

Gate가 초기화 되었는지를 판단하는 변수입니다. Gate의 초기화 조건은 snake가 아이템을 2개 먹을 때 마다입니다. 따라서 snake 객체의 growthItems 변수값을 받아와 gate를 초기화하는데, 이 때 growthItems가 짝수일 경우에 reset을 하는 조건입니다. 하지만 이 조건만으로 gate를 초기화하게 된다면 growthItems가 계속 짝수인 상태인 경우 스네이크가 움직일 때 마다 gate가 초기화됩니다. 따라서 getGrowthItems으로 얻은 값이 짝수가 되어 gate가 초기화 되는 순간에 gateExistence를 true로 지정하고, 홀수가 되면 gateExistence를 false로 지정합니다. 그 뒤 gate 초기화 조건을 “gateExistence == false & getGrowthItem == 짝수” 일 경우로 설정하면 snake가 짝수개째의 growth Item을 먹은 순간에만 gate가 리셋됩니다.

**Gate.h && Gate.cpp** 개발자 : 이현영

|  |
| --- |
| #ifndef \_\_GATE\_\_  #define \_\_GATE\_\_  #include <vector>  #include <random>  #include <algorithm>  #include "Item.h"  class Gate  {  private:  int gate[3];  public:  Gate();  void resetGate();  int getX();  int getY();  int getValue();  };  #endif |

|  |
| --- |
| #include "Gate.h"  using namespace std;  Gate::Gate()  {  gate[2] = 7;  }  void Gate::resetGate()  {  Info info;  info.MakeGate = true;  info.gateExistence = true;  std::random\_device rd;  std::mt19937 gen(rd());  std::uniform\_int\_distribution<int> disIdx(0, info.wallLoc.size() - 1);  int randomIdx = disIdx(gen);  std::vector<int> temp = info.wallLoc.at(randomIdx);  if (info.gateLoc.size() > 0) // if second gate - avoiding overlap  {  while (true)  {  if (temp != info.gateLoc.at(0))  break;  randomIdx = disIdx(gen);  }  }  gate[0] = temp.at(0);  gate[1] = temp.at(1);  info.setGateLoc(temp);  }  int Gate::getX()  {  return gate[0];  }  int Gate::getY()  {  return gate[1];  }  int Gate::getValue()  {  return gate[2];  } |

Gate 클래스는 게임 플레이 중 출현할 gate의 객체를 정의하는 클래스입니다.

Gate가 가진 멤버 변수로는 int값을 가지는 3크기의 배열입니다. 이 배열의 요소로는 순서대로 x좌표값, y좌표값, gate요소의 값(7)을 가집니다.

\*\*Gate의 생성자\*\*는 gate가 가질 값 7을 초기화합니다.

\*\*resetGate()\*\* 함수는 gate의 위치를 랜덤값의 좌표로 지정합니다.

Gate의 출현은 다음과 같은 조건을 만족해야합니다.

1. 두개의 한 쌍이 겹치지 않는다

2. 임의의 위치에 있는 벽에서 나타난다

따라서 resetGate() 함수에서는 먼저 Gate가 만들어졌음과 Gate가 초기화된 상태임을 판단하는 Info 객체의 static 논리형 변수 MakeGate 와 gateExistence 변수를 true로 만들어줍니다.

그 후 <random> 라이브러리의 random\_device 객체를 이용하여 운영체제에서 제공하는 난수를 생성하는 mt19937 엔진객체를 초기화합니다. 하지만 gate의 경우 항상 동시에 생성되는데 이 때 시간을 시드값으로 난수를 발생시키면 gate의 위치가 같은 위치에 나올 수 밖에 없어 random\_device 객체를 사용했습니다.

uniform\_int\_distribution<int> disIdx(0, info.wallLoc.size() - 1) \*\*\*\*에서는 랜덤값을 얻어올 범위를 지정합니다. Gate의 2번 출현 요소를 만족하기 위해 벽의 좌표 위치를 저장한 info 클래스의 wallLoc vector배열을 범위로 지정합니다. 이 난수 생성 엔진은 정수를 반환하기 대문에 wallLoc 배열의 인덱스를 범위로 난수를 반환하도록 지정합니다. 생성된 난수를 인덱스 값으로 가지는 wallLoc의 좌표가 gate의 좌표가 됩니다.

gate는 두개가 생성되는데, 이때 두개의 위치가 겹쳐서는 안됩니다. 따라서 info 클래스에 gate 정보가 1개라도 저장되어있다면 (즉 gate 정보 저장하는 배열의 크기가 0보다 크다면) 현재 만들어지고 있는 gate는 두번째 gate이기 때문에 while loop을 통해 gate 정보에 저장된 첫번째 요소와 비교를 거친 후 일치하지 않을 때 까지 반복문을 실행하며 난수를 발생시킵니다.

모든 조건을 만족한다면 Gate 객체의 멤버 변수 배열의 0번 인덱스와 1번 인덱스에 각각 x좌표와 y좌표를 저장하고, info 객체에 gate 정보를 저장합니다.

Gate 객체의 멤버 변수의 접근 제어자는 private이기 때문에 이를 얻을 수 있는 getter 메소드 \*\*getX(), getY(), getValue()\*\*를 만들었습니다. 이들은 각각 Gate 객체의 멤버 변수 배열의 지정된 요소값을 반환합니다.

**Item.h && Item.cpp** 개발자 : 이현영, 허승범(시간관리)

|  |
| --- |
| #ifndef \_\_ITEM\_\_  #define \_\_ITEM\_\_  #include <random>  #include <time.h>  #include <vector>  #include <algorithm>  #include "Info.h"  class Item  {  private:  int item[3];  int time;  public:  Item(int value);  void resetItem();  int getX();  int getY();  int getValue();  int getTime();  void upTime();  };  #endif |

|  |
| --- |
| #include "Item.h"  using namespace std;  Item::Item(int value)  {  time = 0;  item[2] = value;  }  void Item::resetItem()  {  Info info;  while (true)  {  std::random\_device rd;  std::mt19937 gen(rd());  std::uniform\_int\_distribution<int> disX(0, info.planeLoc.size() - 1);  int randomIdx = disX(gen);  std::vector<int> temp;  int tempX = info.planeLoc.at(randomIdx).at(0);  int tempY = info.planeLoc.at(randomIdx).at(1);  temp.push\_back(tempX);  temp.push\_back(tempY);  bool isNotSnake = false;  auto it2 = find(info.snakeLoc.begin(), info.snakeLoc.end(), temp);  if (it2 == info.snakeLoc.end())  isNotSnake = true;  if (isNotSnake)  {  item[0] = tempX;  item[1] = tempY;  break;  }  }  time = 0;  if (item[2] == 5)  info.setGrowthLoc(item[0], item[1]);  else if (item[2] == 6)  info.setPoisonLoc(item[0], item[1]);  }  int Item::getX()  {  return item[0];  }  int Item::getY()  {  return item[1];  }  int Item::getValue()  {  return item[2];  }  int Item::getTime()  {  return time;  }  void Item::upTime()  {  time++;  } |

Item 클래스는 게임 도중 출현할 Item 객체를 정의하는 클래스입니다. Item은 시간에 따라 재생성되기 때문에, 생성 후 얼마나 시간이 지났는지를 저장할 변수 time과 item의 x좌표와 y좌표, 요소의 값 (5 혹은 6)을 저장할 배열을 멤버 변수로 가집니다.

**Item의 생성자**는 각 Item객체의 time을 0으로 초기화하고 파라미터로 받은 숫자(5 or 6)로 멤버변수 배열의 3번째 요소를 초기화합니다. 해당 숫자에 따라 growth 혹은 poison이 결정됩니다.

**resetItem()**은 Item의 위치를 초기화 시킵니다. 먼저 <random> 모듈의 그 후 <random> 라이브러리의 random\_device 객체를 이용하여 운영체제에서 제공하는 난수를 생성하는 mt19937 엔진객체를 초기화합니다. Item의 경우 초기 게임 시작시에는 무조건 동시에 생성되기 때문에, 이 때 시간을 시드값으로 난수를 발생시키면 item의 위치가 같은 위치에 나올 수 밖에 없어 random\_device 객체를 사용했습니다.

uniform\_int\_distribution<int> disX(0, info.planeLoc.size() - 1) 에서는 item을 지정할 난수를 발생시킬 범위를 지정합니다. 난수 생성 범위는 info클래스에 저장된 static 변수 placeLoc 배열의 크기입니다. placeLoc은 벽이 아닌 지점의 좌표를 저장해 놓은 배열로, 이 배열에서의 무작위의 인덱스값을 받아 그 위치를 item의 위치로 지정합니다.

Item은 snake 의 위치와도 겹쳐서는 안되기 때문에 해당 시점의 snake 좌표 정보를 저장한 info 클래스의 snakeLoc 배열의 요소값들과 일치하지 않는지를 while loop을 통해 검사합니다. 만약 일치하지 않을 경우에 loop을 탈출하고 item의 좌표값을 객체의 멤버변수 배열에 저장합니다.

이후 item의 요소값이 5인지 6인지에 따라 info 클래스의 growthLoc와 poisonLoc에 새로 생성된 item 좌표 값을 저장합니다.

Item의 멤버변수의 접근제어자는 private이기 때문에 이에 접근할 getter 메소드가 필요합니다. 따라서 **getX(), getY(), getValue(), getTime()**을 지정했고, 게임 실행 중 item의 실행 시간을 증가시킬 \*\*upTime()\*\* 메소드를 지정했습니다.

### 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.** |

이번 프로젝트 수행에 사용한 외부 기술/라이브러리는 다음과 같습니다.

**NCURSES**

ncurses는 터미널 스크린에 최적화를 진행하며 글자를 띄워주는 역할을 하는 라이브러리입니다. 게임 시작 창부터 게임 진행 창, 그리고 게임 내부에 존재하는 점수 창과 미션 창, 시간 초를 띄우는 창과 게임 종료 창들을 각 상황에 맞게 터미널에 윈도우 포인터 형식으로 띄워주는 역할을 위해 사용하였습니다. 해당 라이브러리를 사용하여 각 창을 스네이크 매 틱마다 일일이 직접 설정해줘야 하는 부담감을 덜고, WINDOW 포인터로 불러와서 터미널 내 해당 위치에 띄워주기 위해 사용하였습니다.

**VECTOR**

vector는 c++ STL 라이브러리 중 하나입니다. vector는 동적 배열 구조를 c++ style로 구현한 것으로 배열의 맨 끝에서의 삽입과 삭제가 효율적인 자료구조입니다. 동적 배열이기 때문에 크기가 동적으로 변하며 연속적인 메모리 공간을 차지합니다. 개발자가 신경쓰지 않아도 자동적으로 배열의 크기를 조정할 수 있고 객체의 추가 및 삭제가 유연하다는 점에서 자주 쓰이는 라이브러리입니다. 하지만 중간에 있는 데이터를 삭제할 경우에는 복잡도가 늘어나 적합하지 않은 구조입니다.

vector를 사용하기 위해서는 <vector> 헤더파일을 포함해야 합니다.

해당 프로젝트에서는 스테이지별로 임의의 개수를 가지는 일반 좌표와 벽 좌표의 정보를 저장하고, 동적으로 꼬리가 늘어나고 감소하는 뱀 객체를 처리하기 위해 동적 배열 형태의 vector가 적합하다고 판단, 사용하였습니다. 또한, game 의 구성 요소들의 위치 정보를 수시로 갱신하고 저장하기 위한 Info class에서도 삽입/삭제가 빠른 vector를 사용했습니다.

**ALGORITH**

algorithm은 c++ STL 라이브러리 중 하나입니다. algorithm 라이브러리에는 원소들에 대해서 검색, 정렬, 수정, 카운트 등의 작업을 할 수 있는 함수가 정의되어 있습니다. 반복자나 포인터를 통해 작업할 원소들을 가리킵니다.

algorithm을 사용하기 위해서는 <algorithm> 헤더파일을 포함해야 합니다.

해당 프로젝트에서는 Info 클래스에 정의된 game 구성 요소들의 위치 정보를 검색하기 위해 사용되었습니다.

1. Snake의 현재머리 좌표값이 Info 클래스의 allWallLoc 배열에 속해있는지 검색

2. Snake 가 Gate를 만났을 시 Gate의 진행방향 한 칸 앞의 좌표가 Info 클래스의 allWallLoc 배열에 속해있는지 검색하여 새로운 방향 결정

3. Item 생성시 Info 클래스의 snakeLoc 배열에 속하지 않는지 검색

**RANDOM**

c++<random> 라이브러리는 메르센 트위스터 알고리즘을 사용합니다. 메르센 트위스터는 유사 난수 생성 알고리즘으로 컴퓨터 계산을 통해 난수를 발생시는데, 현재 컴퓨터의 구조상 정확하게 난수를 발생할 수 없기 때문에 유사 난수를 사용하고 있습니다. seed를 포함한 624개의 정수를 이용하여 난수를 생성하는데, 난수 발생속도가 빠르고 메모리를 적게 차지하며 난수 발생 주기가 메르센소수라는 특징이 잇습니다.

난수 발생엔진에는 MT19937, MT19937-64가 있는데, 발생시키는 난수의 비트수에 따라 나뉩니다.

메르센 트위스터의 동작 원리는 seed에서 624개 크기의 MT벡터를 생성하고, MT 벡터를 이용하여 624개의 유사 난수를 발생시킨 후 노이즈를 주어 다시 유사난수 발생을 반복합니다.

<random> 헤더파일을 포함하고 std 네임 스페이스를 선언해야 합니다.

해당 프로젝트에서는 Item과 Gate를 임의의 위치에 출현시키는 경우에 사용했습니다.

기존의 C style 난수 생성 알고리즘은 현재 시간의 초를 seed 값으로 지정하여 호출 할 때마다 해당 seed 값을 기반으로 무작위로 보이는 수열을 생성합니다. 하지만 이 방법의 경우 ‘초’를 기반으로 생성되기 때문에 동시에 Item 두개와 Gate 두개가 생성되는 코드에서는 seed 값이 이 속도를 따라가지 못합니다. 따라서 같은 두개의 Item, 두개의 Gate가 같은 seed 값을 가지게 되어 적절치 못한 방법이 됩니다.

따라서 <random> 라이브러리의 random\_device 객체를 사용하여 난수 발생 엔진을 초기화 시키고 지정된 범위 내에서 랜덤한 값을 추출하여 서로 다른 위치에 두 Item 객체 혹은 두 Gate 객체가 위치하도록 설계했습니다.

**FSTREAM**

<fstream> 은 하나의 파일에 대해 읽기와 쓰기를 동시에 할 때 사용하는 c++ 표준 파일 입출력 라이브러리입니다. 스트림 입출력 방식을 지원하며, 파일 입출력스트림과 프로그램을 연결해줍니다.

<fstream> 헤더파일을 포함하고 std 네임스페이스를 선언해야 합니다.

해당 프로젝트에서는 stage 파일을 읽어와서 프로그램의 맵 정보에 저장, 화면에 출력할 때 입력 스트림이 사용되었고, 디버깅이 필요한 부분에서 로그파일을 남기기 위해 출력 스트림이 사용되었습니다.

**UNISTD.H**

저수준 (운영체제 단) 코딩을 할 수 있는 함수들이 들어가 있는 라이브러리라는 배경 지식이 있습니다. 해당 파일에는 write 등 특정 byte에 접근하여 이 byte로부터 얼마만큼의 값을 수정한다와 같은 우리가 흔히 아는 출력함수보다 더 근본적인 차원들의 문제를 다루는 헤더입니다.

윈도우를 재울 수 있는 usleep() 함수 또한 해당 헤더 파일 내에 들어가 있어, 해당 헤더 파일을 프로젝트에 포함하였습니다. usleep()은 인자로 받은 microseconds 동안 실행을 연기시키는 함수입니다. 이를 이용하여 snake 가 이동한 이후 다음 while문에 들어가기까지 0.5초를 연기시킬 수 있었습니다.

**STRING**

수업시간에도 배웠던 string은 C언어 및 C++ 언어에서 한 글자 이상의 문자열을 관리할 수 있는 객체입니다. 기존 C 언어에서 관리하던 문자열의 방식은 문자 하나하나를 배열에 담아 char \*를 생성한 후, 마지막에 null이 나올 때까지 일일이 접근하여 문자를 읽어와야만 했습니다. string은 C++에서 문자열의 자유롭게 붙이거나 제거할 수 있고, 원하는 문자열의 위치에 접근하여 해당 위치의 문자를 char로 반환하는 등 굉장히 자유롭게 문자열을 사용할 수 있습니다.

mvwprintw 함수는 char \* 배열을 window의 특정 위치에 띄워주는 역할을 수행하는데, 이 때 파일 등에서 문자열을 불러와서 읽는 경우, 혹은 Board에 다양한 내용들을 결합한 문자열을 보여주어야 하는 경우에서 매우 유용하게 사용하였습니다.

**SSTREAM**

sstream은 c++ 언어에서 문자열을 자를 수 있는 기능을 찾으면서 함께 찾아낸 라이브러리입니다. 처음 map 파일을 설계하면서 map 크기의 유동성을 주고 싶은 마음에 각 stage 파일의 첫 줄에 맵의 행 크기와 열 크기를 작성하였고, 향후 이를 받아서 언제든 배열을 수정할 수 있도록 구현해보고 싶었습니다. 해당 내용까지는 프로그램에 구현하지 못했지만, 맵 크기가 유동적이더라도 배열 접근 오류 없이 화면에 띄우기 위해 row 개수와 column 개수를 문자열로 받아 (ex: “21 41”) 이를 공백 기준으로 나누어서 각각 저장하였습니다.

### 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.** |

**delay 넣기**

snake를 이동시키는 데에는 성공했지만, 이 함수를 재우지 않으면 while문이 한 번 돌 때마다 정말 엄청난 속도로 스네이크가 돌진해 바로 에러가 나왔습니다. 여기서 while문을 잠시 지연시키기 위해 정말 많은 것들을 찾아보았는데, 이전에 unistd.h 헤더 파일을 사용했던 경험이 있어 해당 헤더 파일 함수를 찾아본 결과 usleep이 존재하였습니다. 약간의 희망을 걸고 usleep을 주어 직접 넣어보았는데 의도한대로 프로그램이 멈추었다가 다시 깨어나게 되어, 이대로 구현하기로 결정하였습니다.

**난수생성**

Item과 Gate는 랜덤한위치에 생성되어야 하기 때문에 난수생성 알고리즘을 사용해야 합니다. 처음에는 알고있던 C 스타일의 난수생성 알고리즘을 적용하여 현재 시간의 초를 시드값으로 좌표값을 지정했는데, 프로그램이 컴파일되고 실행되는 속도는 초단위보다 훨씬 빠르고, 따라서 Item 두개와 Gate 두개는 동시에 생성되기 때문에 시드값이 같을 수 밖에 없었습니다.

그러한 이유로 각 요소의 위치가 겹치는 현상이 일어났고, 메르센 트위스터 유사난수 알고리즘을 사용하는 <random> 라이브러리를 조사, 사용하여 문제를 해결했습니다.

**Item이 벽을 제외한 위치에 뜨도록 하기**

처음에 item이 랜덤한 위치에 뜨도록 구현하는것은 성공했지만, item이 내부에 위치한 벽 위에 생성되는 경우가 있었습니다. 처음에 구현한 로직은 while 문 안에서 item을 리셋하고 리셋한 item 값이 벽 위의 위치와 일치하지 않을 때, 그리고 snake의 몸의 위치와 일치하지 않을 때 반복문을 탈출하도록 구상했습니다. 그 후 확정된 item의 값을 Info에 저장하고 맵에 출력하는 방식이었습니다.

처음에 찾은 문제점은 적절하지 않은 Item 값이 이미 저장되어 있었고, 따라서 제대로 생성되었음에도 불구하고 폐기해야했던 Item 좌표값이 읽혀 맵에 출력되는 것 이었습니다.

그 후 리셋 할 때 Info의 Item 값을 항상 초기화 했고, Item이 벽이 아닌 위치에서 나오도록 판단하는 로직을 벽과 일치하는지 검사하기보다는, 애초에 범위를 벽이 아닌곳으로 지정했습니다. 따라서 Info 클래스에 새로운 planeLoc 배열 변수를 추가하여 벽이 아닌 좌표를 저장해두로 Item의 리셋 위치를 이 범위 내로 지정했습니다. 이후 snake와 일치하는지 여부만 검사한 후 문제를 해결했습니다.

### 결과물 목록

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

- **Framwork** : 게임 시작, 스테이지 클리어, 게임 완료, 게임 오버, stage 정보가 작성된 파일이 포함된 디렉토리

- **src** : main.cpp 파일을 포함하여 게임을 구성하는 헤더파일과 cpp 파일이 포함된 디렉토리

- .**gitignore** : git push 시 포함하지 않을 ide 생성 파일, 로그파일 등 작성 파일

- **Makefile** : 프로그램 컴파일 자동화 makefile

- **최종보고서-Snake Game-저쩔스네이크.docx** :최종 보고서 파일

# 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.** |

**김진재**

Scene의 전환과 Snake의 알고리즘 구상, Stage(map 파일 불러와서 읽기), Mission, ScoreBoard, TimeBoard 구현을 하였습니다. git을 통해 버전 관리를 하며 모르는 사항이나 궁금한 사항, 업데이트한 사항에 대해 지속적으로 질문하고 이해하였으며 프로젝트에서 부족한 역할이 되지 않도록 정말 많은 노력을 하였습니다.

프로그램 제작 과정 측면에서 어려운 점들은 대부분 ‘2.2.4 현실적 제한 요소 및 그 해결 방안’에서 기술하였습니다. 그와 별개로 개인적인 어려움이 있었다면 올해 겨울방학부터 코딩을 처음 배우기 시작하였는데, 당시 C언어로 배우기 시작하여 클래스라는 객체지향을 이해하는 데에 어려움이 있었습니다. 그래도 객체지향 이해를 위해 많이 노력하였고, 프로젝트를 온전하게 완성하는 데에 도움이 되어 정말 행복합니다.

이번 팀 프로젝트를 하면서 C++의 구석구석을 정말 깊게 이해한 것 같습니다. 이전 대외 프로그램 활동을 바탕으로 man 파일을 보는 습관이 되어있어 함수를 모를 때마다 수십 번은 man을 참고하였습니다. 특히 처음에 ncurses를 이해할 때에는 ncurses의 정의 파일에 직접 가서 파일도 보면서 써보고 이해했던 것이 기억에 남습니다.

저희 프로젝트에서 개선해보고 싶은 점은 다음과 같습니다. 가장 개선하고 싶은 점은 현재 WIDE CHAR에 대한 지원을 하지 않는다는 것입니다. 프로그램 구동에는 문제가 없지만 한글이나 다른 언어를 지원하는 기능을 게임에 추가하고자 한다면 lncursesw를 구현해야 했지만, M1 macOS에서 구현상의 문제와 함께 시간이 부족하여 lncursesw를 사용하지 못했습니다. 그리고 map을 프로젝트 초기에 짜서 그런지 원대한 꿈을 갖고 맵 에디터를 제공하기 위해 배열 크기를 파일에 작성하면서 맵 배열을 갖고 와 구현하고자 하였는데, 결국 프로젝트 일정상 구현하지 못하게 된 점이 아쉽습니다. 또한, 현재까지 작성한 프로그램을 바탕으로 2인 플레이어가 함께 플레이하는 기능도 따로 추가해보고 싶습니다. 프로젝트 제출 기한 내에 해보고 싶었으나 2인 플레이를 가능하게 하기 위한 함수가 아예 따로 정의가 필요하여 시간상 포기하였습니다. 이 점들은 프로젝트 기한과 관계없이 조금씩이라도 개선해보고 싶습니다.

**이현영**

Item의 출현 및 갱신, Gate의 출현 및 갱신의 알고리즘을 구상하고, Info 클래스를 구현하는 아이디어와 이벤트 처리 알고리즘을 구상했습니다. 또한 git 의 master 브랜치를 관리하여 팀원들의 코드를 병합하고 버전 관리를 주도적으로 맡고, 팀 노션 페이지를 작성하여 마일스톤 달성 여부를 실시간으로 모두가 확인 가능하게 운영했습니다.

Info 클래스는 게임 중 계속해서 갱신되는 요소들의 위치 정보를 어떻게 알아내서 효율적으로 snake와 item, gate, 벽 이 만났을 경우의 이벤트 처리 알고리즘을 작성할 수 있을지를 고민하며 구현한 클래스입니다. 각 요소들이 갱신됨과 동시에 Info 클래스의 지정된 변수에 좌표값을 업데이트하도록 설계하고, Info 클래스의 변수는 static으로 선언하여 모든 클래스에서 동일한 객체를 사용할 수 있도록 설계했습니다. 또한 이벤트 처리를 위해 snake가 한 칸 움직이는 순간에 snake 머리가 어떠한 요소의 위치와 일치하는지를 모두 검사하는 알고리즘을 고안하고 적용하도록 도왔습니다.

프로그램을 수행하며 어려웠던 점은 처음에 git 브랜치 관리를 제대로 하지 못했던 점이었습니다. 팀원 모두가 git에 익숙하지 않았지만 팀 프로젝트 경험이 있어 git을 조금이라도 사용해 본 제가 git master 브랜치 관리를 맡았는데, 처음에는 클래스 별로 브랜치를 나누어 커밋 내역을 명확히 하려고 했지만, 거대 규모의 팀 프로젝트가 아니기 때문에 오히려 인원수로 브랜치를 나누는 것이 효율적이라는 것을 프로젝트 진행 중반부부터 알게 되었습니다. 따라서 처음에 git을 사용하는 방식을 온전히 이해하지도 못했는데, 풀과 푸시가 꼬이는 상황이 많이 생겨 master 브랜치 관리자로서의 책임감을 많이 느꼈습니다. 이후에는 개별 브랜치를 작성하여 관리했고, 이런 시행착오 끝에 지금은 git 사용법을 많이 익히게 되었고 git의 동작원리도 이해하게 되어 뿌듯했습니다.

프로젝트에서 개선해보고 싶은 점은 조금 더 효율적인 알고리즘을 사용하여 프로그램을 최적화 하는 것입니다. 대표적으로 보이는 문제점은 main의 while문에 의존하여 게임이 진행되는데, while 루프가 실행될 때 마다 모든 맵의 요소가 갱신→ 출력 되고 있습니다. 그에 따라 Loadmap을 하여 파일에서 stage 파일을 불러오는 작업도 매 출력마다 작동하고 있습니다. Loadmap 안에서는 맵 정보를 Info에 저장하는 로직을 매번 실행하기 때문에 이 또한 while 루프마다 실행된다고 볼 수 있습니다. 따라서 현재 많은 오버헤드가 발생하고 있습니다. 비록 보기에는 문제가 없지만 프로그램을 완성했으니 시간을 더 투자한다면 조금 더 가벼운 프로그램으로 만들어 오버헤드를 줄일 수 있을 것 같습니다. 이 뿐만 아니라 시간상 효율성은 충분히 고려하지 못한 부분이 많다고 생각이 드는데, 조금 더 효율성을 고려하여 로직을 수정하면 좋을 것 같으며, 다른 프로젝트를 할 때 에도 설계 단계에서부터 효율성을 고려하는 습관을 들이고 싶습니다.

**허승범**

Main, Snake, Scene 의 전체적인 구조와 item(시간관리)를 구현했습니다. 저는 이번 프로젝트를 하면서 제가 많이 부족하다는 것을 느꼈고, 팀원들로부터 정말 많은 것을 배웠습니다. git이라는 것을 아예 다뤄본 적도 없던 저는 git을 사용해본 팀원들에게 무작정 메달리기만 했던 것 같습니다. 덕분에 저는 git을 통한 협업에 어느정도 눈을 뜰 수 있었습니다.

프로젝트 수행 시 어려웠던 점은 ncurses라는 것을 처음 접했을 때입니다. 처음 ncurses라는 것을 접했을 때, 처음 보는 라이브러리를 사용하여 프로젝트를 해야 한다는 것이 이해가 안되었습니다. 하지만 프로젝트를 마친 이 시점에서 이 과제의 의도를 어느 정도는 이해한 것 같아 얻어 가는 것이 많은 것 같습니다. 이 프로젝트를 통해 제가 얼마나 부족한지, 문서화가 얼마나 중요한지, 팀원 간의 협업에서 커뮤니케이션이 얼마나 중요한지 등에 대해 많은 것을 깨달았습니다.

또한 makefile이 개발하는데에 얼마나 편한지도 알게 되었고, 잘 활용할 줄 알게됐습니다.

제가 이 프로젝트에서 개선하고 싶은 점은 조금 더 Class별로 해야 할 일을 명확하게 나눈 후에 코드를 더 깔끔하게 정리하고 싶습니다. 지금 저희의 코드는 제가 느끼기에는 정리가 덜 된 느낌인 것 같습니다. 코드를 크게 보면 class별로 하는 일들을 갖고 있지만, 하는 일들이 명확하지 않았다고 느끼기 때문입니다. 또한 먼저 해야 할 일들을 세세하게 나누어 문서화 해서 정리를 한 다음에 코드를 작성하기 시작했더라면 협업을 하는 데에 있어서 더 편리하지 않았을 까라는 생각이 듭니다.

앞으로는 위와 같이 개선하고 싶은 점들을 항상 생각하며 개발하는 개발자가 되기 위해 더 많은 노력을 해야 할 것 같습니다.

# 참고 문헌

**참고한 서적, 기사, 기술 문서, 웹페이지를 나열한다.:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 번호 | 종류 | 제목 | 출처 | 발행년도 | 저자 | 기타 |
|  | 웹페이지 | cppreference | <https://en.cppreference.com/w/> |  |  |  |
|  | 웹페이지 | 씹어먹는 C++ - <17 - 3. 난수 생성(<random>)과 시간 관련 라이브러리(<chrono>) 소개> | <https://modoocode.com/304> |  |  |  |
|  | 웹페이지 | [Git] git add 취소하기, git commit 취소하기, git push 취소하기 | <https://gmlwjd9405.github.io/2018/05/25/git-add-cancle.html> |  |  |  |
|  | 웹페이지 | ncurses 프로그래밍 | <https://m.blog.naver.com/PostView.naver?isHttpsRedirect=true&blogId=koreteck&logNo=49955493> |  |  |  |
|  | 웹페이지 | 리눅스 프로그래머를 위한 가이드 | <http://www.xevious7.com/linux/lpg_8_3.html> |  |  |  |
|  | 웹페이지 | Linux man page | <https://linux.die.net/man/3/cbreak> |  |  |  |

# 부록

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

## 사용자 매뉴얼

메뉴얼에 앞서, 직접 게임을 실행하고 구현한 영상에 대한 링크를 첨부하였습니다. 참고하여 함께 보시면 프로그램 시연 방법을 직관적으로 알 수 있을 것 같습니다.

<https://www.youtube.com/watch?v=RktE-FaoHrI>

5.2.에 언급되어 있는 ‘설치방법’을 통해 설치하여 실행하면, 아래와 같은 초기화면이 생성됩니다. 이 초기 화면에서는 위/아래 방향키로 옵션을 선택하고, 엔터 키를 통해 확인을 누르면 해당 옵션이 실행됩니다. 이 때, GAME START는 게임을 실행하고, EXIT는 프로그램을 종료합니다.

Text

Description automatically generated

GAME START 를 선택하여 게임 화면에 진입하면, 아래와 같은 창이 뜹니다. 대부분의 구현 위치와 구현 내용은 권고안과 동일하며, 왼쪽이 게임 주요 화면, 그리고 오른쪽에 3개의 구역이 순서대로 점수판, 달성해야 할 미션과 미션 달성 여부 ((v) 표시), 그리고 게임 시간이 표시되고 있습니다.

Graphical user interface

Description automatically generated

게임에서 스네이크는 방향키를 통해 움직이며, 조건이 성공하면 Stage Clear 창이 뜬 후에 다음 스테이지로 진행하고, 조건이 실패하면 Game Over 화면을 띄웁니다.

Text

Description automatically generated

Text

Description automatically generated

게임 오버 화면에서는 메인 메뉴로 돌아가기(BACK TO MAIN MENU) 버튼과 종료 (EXIT) 버튼이 있습니다. 메인 메뉴로 돌아가기 버튼은 처음 게임 실행 화면으로 돌아가서 게임을 다시 시작할 수 있으며, 종료 버튼은 프로그램을 종료합니다.

Text

Description automatically generated

4개의 stage를 모두 클리어하면, 축하 창과 함께 아무 키나 눌러 프로그램을 종료할 수 있습니다.

## 설치 방법

**필수 선행 조건**

외부 라이브러리인 ncurses가 시스템에 설치되어 있어야 합니다. ncurses는 ubuntu OS 기준으로 다음과 같이 설치합니다.

sudo apt-get update

sudo apt-get install libncurses5-dev libncursesw5-dev

프로젝트 파일 내 src 폴더와 Framework 폴더, 그리고 Makefile을 필수적으로 다운로드 해야 합니다.

**설치 방법**

터미널에서, 앞서 언급한 필수 파일들이 있는 경로로 이동합니다.

make

이후, 위처럼 make 명령어를 실행하면 모든 파일이 컴파일되면서 게임이 실행됩니다.

Text

Description automatically generated