**C++프로그래밍**

**프로젝트**

|  |  |
| --- | --- |
| 프로젝트 명 | Snake Game |
| 팀 명 | 1조 |
| 문서 제목 | 2022 기말프로젝트 결과보고서 |

|  |  |
| --- | --- |
| **Version** | 1.1 |
| **Date** | 2022-06-15 |

|  |  |
| --- | --- |
| **팀원** | 20195301 이용훈 (팀장) |
| 20213062 이찬우 |
| 20213078 전예찬 |
|  |
|  |

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

**문서 정보 / 수정 내역**

|  |  |
| --- | --- |
| **Filename** | 2022-기말프로젝트-이용훈.docx |
| **원안작성자** | 이찬우 |
| **수정작업자** | 이용훈, 이찬우, 전예찬 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 수정날짜 | 대표수정자 | Revision | 추가/수정 항목 | 내 용 |
| 2022-06-14 | 이찬우 | 1.0 | 최초 작성 | 1. 개요, 2.2.2 시스템 구조 및 설계도를 포함한 보고서 틀 작성 |
| 2022-06-15 | 이찬우 | 1.1 | 내용 추가 | 2.2.2. 시스템 구조 및 설계도 추가 작성 |
| 2022-06-16 | 이찬우 | 1.2 | 보고서 1차 완성 | 2.2.2. 시스템 구조 및 설계도 작성 완료, 1. 개요 내용 추가, |
| 2022-06-16 | 이용훈, 전예찬 | 1.3 | 보고서 추가 | 개발 내용과 해결방안 추가 |
| 2022-06-16 | \* |  | 최종본 | 미흡한 부분, 최종본 작성 |
|  |  |  |  |  |
|  |  |  |  |  |

**본 양식은 소프트웨어학부 C++프로그래밍 과목의 프로젝트 결과보고서 작성을 위한 기본 양식입니다. 문서의 필수 항목을 제시하는 것이니 폰트, 문단 구조 등의 디자인 부분은 자유롭게 설정하기 바랍니다. 양식 내에 붉은 색으로 기술한 부분은 지우고 작성하기 바랍니다.**

**목 차**

[1 개요 4](#_Toc43103653)

[2 개발 내용 및 결과물 6](#_Toc43103654)

[2.1 목표 6](#_Toc43103655)

[2.2 개발 내용 및 결과물 7](#_Toc43103656)

[2.2.1 개발 내용 7](#_Toc43103657)

[2.2.2 시스템 구조 및 설계도 7](#_Toc43103658)

[2.2.3 활용/개발된 기술 34](#_Toc43103659)

[2.2.4 현실적 제한 요소 및 그 해결 방안 34](#_Toc43103660)

[2.2.5 결과물 목록 35](#_Toc43103661)

[3 자기평가 36](#_Toc43103662)

[4 참고 문헌 36](#_Toc43103663)

[5 부록 37](#_Toc43103664)

[5.1 사용자 매뉴얼 37](#_Toc43103665)

[5.2 설치 방법 39](#_Toc43103666)

# 개요

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

본 프로젝트는 2022-1학기 C++프로그래밍 강의를 통해 학습했던 생성자, 클래스, 상속 등의 개념들을 적용하고, ncurses 라이브러리를 이용하여 스네이크 게임을 구현하는 프로젝트이다.

우리 팀은 본 프로젝트를 수행하기 위한 협업 수단으로 git과 GitHub를 이용하였고, 이를 위해 GitHub에 새로운 원격 저장소를 생성하여 관리했다. 팀원마다 기여 권한을 부여했고, 이를 통해 커밋과 푸시를 반복했다. 커밋을 할 때는 각자 수정한 내용을 요약하여 커밋 메시지를 작성하고 적절한 위치에 푸시하며 주어진 단계를 수행했다. 주요 변경사항이 새로 구현될 때마다 원격 저장소에 브랜치를 추가했으며, 브랜치에 추가된 내용을 메인 브랜치에 추가하고 병합해가며 프로젝트를 수행했다.

필요한 수정 사항이나 변경된 수정 사항이 발생할 경우 카카오톡 그룹채팅방을 통해 지속적으로 의견을 전달했고 어려운 알고리즘을 구현하기 위해 해결 방법에 대한 의견을 주고받았다. 또한 학교 내에서 주기적으로 대면하며 회의하였다.

우선 게임을 구동하기 위해 main.cpp라는 파일을 두었으며, 이 파일을 컴파일하여 게임이 실행될 수 있도록 했다. 그리고 Board.hpp에 게임판을 설정하는 코드를 구현하고, SnakeGame.hpp에 게임 실행에 필요한 주요 알고리즘을 구현했다. 그리고 게임에 필요한 나머지 요소들은 각각 hpp파일 및 클래스를 생성하여 수행했다.

본 프로젝트에서 사용된 외부 라이브러리는 ncurses이다. ncurses를 설치하기 위해 팀원들 각각의 OS인 Ubuntu와 Mac OS의 터미널에 다음과 같이 입력했다.

$ sudo apt-get update

$ sudo apt-get install libcurses5-dev libncursesw5-dev

이를 통해 ncurses를 설치한 후 각 소스파일의 상단에 #include <ncurses.h>를 기입하여 사용했다.

# 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 | 미적용 |
| 5단계 | 점수 요소의 구현 | 적용 |

**프로젝트의 각 적용 단계별 구현 목표를 명확하게 제시한다. 제시한 권고안의 내용을 포함하여, 변경된 부분 등을 구체적으로 단계별 구현 목표를 작성해야 한다.**

**1단계 – Map의 구현**

* WINDOW\*와 ncurses 메소드를 이용하여 윈도우 3개를 생성해야 한다. 각각 메인 윈도우, 게임판 윈도우, 점수판 윈도우에 해당하며, 스네이크가 움직이고 게임이 진행되는 윈도우는 게임판 윈도우이다.
* 스네이크의 머리가 게임판 윈도우의 외곽에 닿으면 스네이크가 죽어 게임이 종료되어야 한다.

**2단계 – Snake 표현 및 조작**

* 스네이크는 설정된 틱(일정 시간)에 의해 이동해야 한다.
* 방향키를 통해 사용자로부터 키를 입력받아 조작할 수 있어야 한다.
* w키나 위쪽 방향키를 누르면 위로, s키나 아래쪽 방향키를 누르면 아래로, a키나 왼쪽 방향키를 누르면 왼쪽으로, d나 오른쪽 방향키를 누르면 오른쪽으로 방향을 바꾸어야 한다. 단, 진행방향과 같은 방향키가 입력될 경우는 무시한다.
* 진행방향과 반대되는 방향키가 입력되는 경우 게임이 종료되어야 한다.

**3단계 – Item 요소의 구현**

* Growth item에 해당하는 사과와, Poison item에 해당하는 독은 게임판 내에서 스네이크가 있지 않은 랜덤한 위치에서 생성되어야 한다. 이때, 사과는 게임이 시작한 직후부터 생성될 수 있지만, 독은 스네이크의 길이가 5를 넘겨야 생성될 수 있어야 한다.
* 아이템은 일정시간이 지나면 사라지고 다른 위치에 나타나야 한다.
* 동시에 출현할 수 있는 아이템의 수는 3개로 제한되어야 한다.
* 사과를 먹을 경우 스네이크의 몸 길이가 1만큼 증가해야 한다. 이때 스네이크의 진행방향으로 증가해야 한다.
* 독을 먹을 경우 스네이크의 몸 길이가 1만큼 감소해야 한다. 이때 꼬리 부분이 감소해야 한다. 만약 독을 먹어 몸 길이가 3보다 작아지면 게임이 종료되어야 한다.

**4단계 – Gate 요소의 구현 (구현 실패)**

* 게이트는 두 개가 한 쌍이며, 두 개가 겹치지 않도록 벽에 생성되어야 한다.
* 게이트에 스네이크가 진입하면 다른 게이트로 진출해야 한다.
* 게이트에 스네이크가 진입중일 때 게이트가 사라지지 않아야 한다.
* 게이트가 나타나는 벽이 가장자리에 있을 때 스네이크는 항상 맵의 안쪽 방향으로 진출해야 한다.
* 게이트의 위치가 가장자리가 아닐 때
  + 진출방향이 자유로운 경우 진행방향으로 진출되어야 한다.
  + 진출방향이 좌-우인 경우 오른쪽에서 진입 시 우로 진출해야 한다.
  + 진출방향이 좌-우인 경우 왼쪽에서 진입 시 좌로 진출해야 한다.
  + 진출방향이 상-하인 경우 위쪽에서 진입 시 위로 진출해야 한다.
  + 진출방향이 상-하인 경우 아래쪽에서 진입 시 아래로 진출해야 한다.

**5단계 – 점수 요소의 구현**

* 스네이크의 현재 길이 / 최대 길이 가 표시되어야 한다.
* 사과를 먹은 횟수와 독을 먹은 횟수가 각각 점수로서 표시되어야 한다.
* 게임 시간이 표시되어야 한다.
* 최대 길이가 10 이상이고, 사과를 먹은 횟수가 10 이상이고, 독을 먹은 횟수가 2 이상이라면 게임이 클리어되어야 한다.

## 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**

Map은 Board.hpp와 Scoreboard.hpp에 있는 board\_win, score\_win에서 ncurse라이브러리에서 제공하는 newwin과 box를 통해 맵을 배치하고 refresh하여 시각화시켰다. 또한 main.cpp에 explain 윈도우를 추가하여 게임 설명도 넣었다. 그 결과 게임을 실행했을때 지렁이 게임을 할 수 있는 맵과 점수현황을 확인할 수 있는 맵, 게임 설명을 볼 수 있는 맵을 생성하였다.

**2단계 – Snake의 표현 및 조작**

Snake는 wasd나 방향키로 이동할 수 있으며, ‘A’를 먹었을때 길이가 1추가되고 ‘P’를 먹었을때 길이가 1감소되며 벽이나 반대 방향키, 자기 자신이 닿았을때 그리고 길이가 3이하가 되었을때 게임오버 되도록 하였다. 기본적인 모양은 ‘@’이다. queue를 활용하여 지렁이의 길이를 조절하였다. Direction 열거를 활용하여 방향키 코딩을 최적화 시켰다. 움직이는 방향에 대해 row와 col의 위치를 바꾸었고, 그 바꾼위치에 대해 지렁이 객체를 추가하고 꼬리를 삭제한후 빈 부분에 빈 문자 ‘ ‘를 넣었다. 또한 time.h안에 timeval을 활용하여 시스템 시간을 연산하여 지렁이가 같은 키를 여러 번 눌렀을때 부스팅되지 않도록 방지하였다.

**3단계 - item의 요소 구현**

Item요소는 Drawable.hpp에서 Apple과 Poison, empty를 상속시킨후 랜덤한 위치를 받아 만약 apple과 poison이 맵에 없을때 (NULL) 10초동안 맵에 생성하고 10초가 지나면 없어지고 새로운 위치에 생성되도록 하였다. 또한 지렁이가 아이템에 닿았을때 해당 아이템에 대한 효과를 지렁이와 점수판에 적용시키고 아이템은 새로운 위치가 추가될 수 있도록 하였다. 길이가 길어지는 아이템은 ‘A’고 길이가 작아지는 아이템은 ‘P’다. 지렁이 길이가 5를 넘을때 ‘P’가 생성하기 시작하도록 하였다.

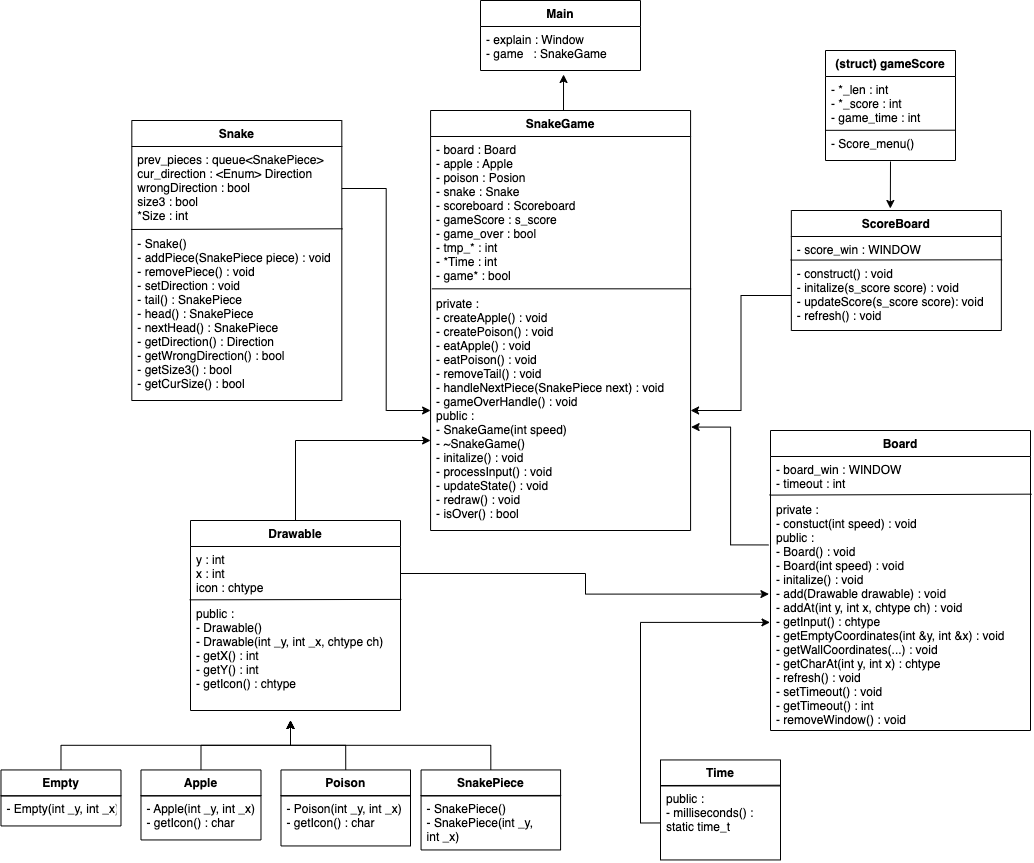
**5단계 - 점수 요소의 구현**

GameScore.hpp에 점수를 저장할 수 있는 구조체를 생성하였다. 현재길이/최대길이, 아이템을 먹은 횟수, 게임시간을 저장하였다. 그리고 게임이 진행되는 동안 해당 행동을 했을시 변수를 변경하였고 그 점수를 ScoreBoard에 있는 점수판에서 시각화 할 수 있도록 하였다. 또한 해당 점수가 미션조건을 만족했을시 게임클리어가 될 수 있도록 하였다.

### 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.** |

**시스템 설계도**



SnakeGame.hpp가 게임 구동의 메인 알고리즘을 담당하고, Snake는 스네이크의 이동에 관한 알고리즘을 담당한다. 그리고 Board.hpp는 게임판의 구현을 담당하고, ScoreBoard.hpp는 점수판의 구현을 담당한다. 나머지 게임 내 요소들도 각각의 파일과 클래스로 구성되어 있다. 게임의 실행은 main.cpp파일이 담당하며, main.cpp를 컴파일하여 실행하면 게임이 시작된다.

**main.cpp**

|  |
| --- |
| #include <ncurses.h>  #include "Board.hpp"  #include "SnakeGame.hpp"  #include <iostream>  #define extern TICK;  int main()  {  initscr();  refresh();  noecho();  curs\_set(0);  border('\*', '\*', '\*', '\*', '\*', '\*', '\*', '\*');  refresh();  WINDOW \*explain = newwin(10, 50, 30, 60);  box(explain, 0, 0);  mvwprintw(explain, 1, 3, "Game Rule");  mvwprintw(explain, 2, 3, "Eat 'A' makes your snake length + 1");  mvwprintw(explain, 3, 3, "Eat 'P' makes your snake length - 1");  mvwprintw(explain, 4, 3, "if current length less than 3, Game Over");  mvwprintw(explain, 5, 3, "if clear all mission, Game Clear");  wrefresh(explain);  SnakeGame game = SnakeGame(TICK);  while (!game.isOver())  {  game.redraw();  game.updateState();  game.redraw();  game.processInput();  game.redraw();  }  getch();  endwin();  if (game.isClear() && game.isOver())  std::cout << "GameClear" << std::endl;  else  std::cout << "GameOver" << std::endl;  return 0;  } |

main.cpp은 게임의 시작 및 종료를 포함한 게임 구동에 필요한 요소가 포함된 메인 파일이다. 따라서 게임을 실행하려면 이 파일을 컴파일하여 실행 파일을 만들어야 한다. Snake 게임의 게임판을 구성하는 Board.hpp파일과 게임 구동 알고리즘이 구성되어 있는 SnakeGame.hpp을 include한다. 그리고 TICK을 extern을 통해 불러와 정의한다.

main함수는 게임의 구동을 담당한다. ncurses 라이브러리의 initscr 함수와 refresh 함수를 통해 ncurses의 자료구조를 초기화하고 기본 크기의 window를 생성하고 초기 환경을 세팅했다. 그리고 noecho 함수를 통해 사용자로부터 입력받은 문자를 출력하지 않게끔 했고, curs\_set(0)를 통해 화면에 커서가 보이지 않게끔 설정했다. border 를 통해 게임판 외곽을 \*로 둘러싸도록 설정했고, refresh를 통해 설정한 것들이 반영되도록 했다.

이어서 WINDOW 타입을 가리키는 포인터 변수인 explain 변수를 설정했는데, 이 변수에는 newwin을 통해 생성된 윈도우 주소가 할당된다. 이 윈도우는 화면 하단에 나타나는 게임 규칙 안내 박스에 해당한다. 생성된 윈도우 주소를 바탕으로 윈도우에 대한 설정을 진행해주었다. box로 윈도우의 외곽선을 그려주었고, mvwprintw로 각 줄에 게임 규칙을 적어주었다. 규칙 작성을 완료한 후 wrefresh를 통해 반영했다.

SnakeGame 생성자를 통해 game 변수를 생성했다. 매개변수에는 TICK값이 들어간다. game 변수를 while문으로 돌려 게임이 종료되기 전까지 무한히 게임이 진행되도록 했다. game.updateState를 사용하여 게임 상태가 지속적으로 업데이트되도록 했고, game.processInput를 사용하여 사용자의 입력을 지속적으로 받도록 했다. 그리고 두 메서드를 사용할 때마다 game.redraw를 통해 화면이 새로고침될 수 있도록 했다. 만약 게임이 종료되면 game.isOver는 참값을 갖게 되고 while문을 빠져나온다.

game.isover에 참값이 들어가 게임을 클리어한 것으로 판단되면 GameClear를 출력하고, 그렇지 않다면 GameOver를 출력한다. 이 출력을 마지막으로 게임은 종료된다.

**Board.hpp**

|  |
| --- |
| #pragma once  #include "Drawable.hpp"  #include <stdlib.h>  #include "Time.hpp"  #define HEIGHT 25  #define WIDTH 60  class Board  {  private:  WINDOW \*board\_win;  int timeout;  void construct(int tick)  {  getmaxyx(stdscr, yMax, xMax);  board\_win = newwin(HEIGHT, WIDTH, 5, 30);  wborder(board\_win, '0', '0', '0', '0', '/', '\\', '\\', '/');  wrefresh(board\_win);  timeout = tick;  setTimeout(tick);  keypad(board\_win, true);  }  protected:  int xMax, yMax;  public:  Board()  {  }  Board(int tick)  {  construct(tick);  }  void initalize()  {  clear();  refresh();  }  void add(Drawable drawable)  {  addAt(drawable.getY(), drawable.getX(), drawable.getIcon());  }  void addAt(int y, int x, chtype ch)  {  mvwaddch(board\_win, y, x, ch);  }  chtype getInput()  {  time\_t time\_last\_input = Time::milliseceonds();  chtype input = wgetch(board\_win);  chtype new\_input = ERR;  setTimeout(0);  while (time\_last\_input + timeout >= Time::milliseceonds())  {  new\_input = wgetch(board\_win);  }  setTimeout(timeout);  if (new\_input != ERR)  input = new\_input;  return input;  }  void getEmptyCoordinates(int &y, int &x)  {  while ((mvwinch(board\_win, y = rand() % HEIGHT, x = rand() % WIDTH)) != ' ')  ;  }  void getWallCoordinates(int &y1, int &x1, int &y2, int &x2)  {  while (1)  {  while ((mvwinch(board\_win, y1 = rand() % HEIGHT, x1 = rand() % WIDTH)) == ' ')  ;  while ((mvwinch(board\_win, y2 = rand() % HEIGHT, x2 = rand() % WIDTH)) == ' ')  ;  if (y1 != y2 || x1 != x2)  {  break;  }  }  }  chtype getCharAt(int y, int x)  {  return mvwinch(board\_win, y, x);  }  void refresh()  {  wrefresh(board\_win);  }  void setTimeout(int timeout)  {  wtimeout(board\_win, timeout);  }  int getTimeout()  {  return timeout;  }  void removeWindow()  {  wclear(board\_win);  }  }; |

Board.hpp는 게임판을 설정하는 파일이다. 그리기 요소들에 대한 인터페이스가 담긴 Drawable.hpp 파일과 time을 관리하는 파일인 Time.hpp 파일을 include한다. Board 클래스를 통해 작동하며, 이 클래스에 대한 설명은 다음과 같다.

Board 클래스의 멤버 변수는 WINDOW 타입을 가리키는 포인터 변수 board\_win와 int 타입의 변수 timeout이다. board\_win은 게임판 윈도우의 주소값이 할당되며, timeout에는 TICK값이 할당된다.

private 멤버 함수는 construct이다. 설정된 tick을 매개변수로 받고, getmaxyx를 통해 윈도우의 좌표를 가져온다. wborder를 통해 외곽을 설정하는데, 게임판의 꼭짓점에 해당하는 네 부분은 슬래시(/)와 역슬래시(\)로 처리하고 게임판의 변에 해당하는 부분들은 0의 반복으로 나타나도록 처리한다. wrefresh로 새로고침하고, 매개변수로 받았던 tick을 timeout으로 설정하고 setTimeout를 사용하여 timeout을 설정한다. 그리고 keypad로 키패드의 입력을 활성화한다. protected 멤버 함수는 xMax와 yMax이다.

public 멤버 함수는 Board, add, addAt, getInput, getEmptyCoordinates, getWallCoordinates, getCharAt, refresh, setTimeout, getTimeout, removeWindow이다. Board는 클래스의 생성자로, 매개변수로 tick이 입력될 경우 그 값으로 construct 함수를 실행한다. initialize는 게임판을 초기화하는 함수로, clear와 refresh로 게임판 윈도우를 초기화한다. add는 Drawable 타입을 입력으로 받는데, addAt함수를 불러와 그리기 요소에 해당하는 것을 윈도우에 추가하는 역할을 한다. addAt 은 y, x, 그리고 chtype 타입의 ch를 매개변수로 받아 지정된 좌표에 ch를 추가하는 역할을 한다.

getInput 은 윈도우 내의 문자를 읽고 그 값을 반환하는 함수이다. 우선 Time::milliseconds를 통해 밀리초 단위의 시간을 time\_last\_input 변수에 할당한다. 그리고 wgetch를 통해 윈도우에서 문자를 읽어 input 변수에 저장하고 new\_input 변수는 ERR를 기본값으로 세팅한다. 그리고는 setTimeout함수로 타임아웃을 0으로 설정하고 Time::milliseconds를 통해 다시 계산한 시간보다 time\_last\_input과 timeout을 더한 값이 더 클 동안 while문을 도는데, 이 때 다시 wgetch로 문자를 읽어 new\_input에 저장한다. while 루프를 돈 후, 다시 타임아웃을 기존의 timeout으로 설정한다. 만약 루프를 통해 정상적으로 값을 읽어서 new\_input에 문자가 할당되었다면 new\_input을 input에 할당한다. 만약 값을 읽지 못해서 new\_input에 문자가 할당되지 않았더라면 기존 input값을 그대로 유지하고, 다른 변경을 취하지 않는다. 그리고 최종적인 input값을 반환한다.

getEmptyCoordinates는 매개변수로 y값과 x값을 레퍼런스로 입력받아 윈도우 내에서 공백 문자에 해당하는 좌표들 중 랜덤한 좌표 하나를 설정한다. getWallCoordinates는 매개변수로 y값 2개와 x값 2개를 레퍼런스로 입력받아 윈도우 내에서 공백 문자에 해당하지 않는 좌표들, 즉 벽에 해당하는 좌표들 중 랜덤한 좌표를 각각 설정한다. 이때, 두 좌표가 겹치지 않을 때 while 무한루프를 빠져나온다.

getCharAt은 게임판 윈도우에서 매개변수로 입력된 y, x좌표에 해당하는 문자를 mvwinch를 통해 chtype 타입으로 반환해준다. refresh는 wrefresh를 통해 윈도우를 새로고침하고, setTimeout은 설정할 타임아웃 값을 매개변수로 받아 wtimeout을 통해 타임아웃을 설정한다. getTimeout는 설정된 타임아웃 값을 반환한다. removeWindow는 게임판 윈도우를 wclear를 통해 제거한다.

**SnakeGame.hpp**

|  |
| --- |
| #pragma once  #define TICK 250  #define ITEMTICK 10000  #include <ncurses.h>  #include <stdlib.h>  #include <time.h>  #include <iostream>  #include "Board.hpp"  #include "Empty.hpp"  #include "Apple.hpp"  #include "Drawable.hpp"  #include "Snake.hpp"  #include "Scoreboard.hpp"  #include "Poison.hpp"  #include "gameScore.hpp"  class SnakeGame  {  private:  Board board;  Apple \*apple;  Poison \*poison;  Snake snake;  Scoreboard scoreboard;  s\_score gameScore;  bool gamePause, gameOver, gameClear;  int tmp\_apple\_x, tmp\_apple\_y;  int tmp\_poison\_x, tmp\_poison\_y;  int snakeTime, appleTime, poisonTime; // handle Item respon time  void createApple()  {  int y, x;  board.getEmptyCoordinates(y, x);  apple = new Apple(y, x);  board.add(\*apple);  tmp\_apple\_x = x;  tmp\_apple\_y = y;  appleTime = 0;  }  void createPoison()  {  int y, x;  board.getEmptyCoordinates(y, x);  poison = new Poison(y, x);  board.add(\*poison);  tmp\_poison\_x = x;  tmp\_poison\_y = y;  poisonTime = 0;  }  void eatApple()  {  delete apple;  apple = NULL;  gameScore.apple\_score += 1;  gameScore.cur\_len += 1;  if (gameScore.max\_len < gameScore.cur\_len)  gameScore.max\_len = gameScore.cur\_len;  appleTime = 0;  scoreboard.updateScore(gameScore);  }  void eatPoison()  {  delete poison;  poison = NULL;  gameScore.poison\_score += 1;  gameScore.cur\_len -= 1;  poisonTime = 0;  scoreboard.updateScore(gameScore);  }  void removeTail()  {  int emptyRow = snake.tail().getY();  int emptyCol = snake.tail().getX();  board.add(Empty(emptyRow, emptyCol));  snake.removePiece();  }  void hanleNextPiece(SnakePiece next)  {  // const char appleIcon = apple->getIcon(); // segment error occured  if ((apple != NULL) || (poison != NULL))  {  char boardCharAt = board.getCharAt(next.getY(), next.getX());  // game over handler  switch (boardCharAt)  {  case 'A':  eatApple();  break;  case 'P':  eatPoison();  removeTail();  removeTail();  break;  case ' ':  {  removeTail();  break;  }  default:  gameOver = true;  break;  }  }  else  {  if (appleTime == TICK)  removeTail();  }  board.add(next);  snake.addPiece(next);  }  void gameOverHandle()  {  if (snake.getWrongDirection())  {  gameOver = true;  }  if (snake.getSize3())  {  gameOver = true;  }  if (gameScore.max\_len >= 10 && gameScore.apple\_score >= 10 && gameScore.poison\_score >= 2)  {  gameOver = gameClear = true;  }  }  public:  SnakeGame(int speed)  {  board = Board(speed);  scoreboard = Scoreboard();  initalize();  srand(time(NULL));  }  ~SnakeGame()  {  delete apple;  }  void initalize() // 수정  {  apple = NULL;  poison = NULL;  board.initalize();  scoreboard.initialize(gameScore);  snakeTime = appleTime = poisonTime = 0;  gamePause = gameOver = gameClear = false;  srand(time(NULL));  // snake make  snake.setDirection(down);  hanleNextPiece(SnakePiece(1, 1));  hanleNextPiece(snake.nextHead());  hanleNextPiece(snake.nextHead());  if (apple == NULL)  createApple();  }  void processInput()  {  chtype input = board.getInput();  int old\_timeout = board.getTimeout();  // process input  switch (input)  {  case KEY\_UP:  case 'w':  snake.setDirection(up);  break;  case KEY\_DOWN:  case 's':  snake.setDirection(down);  break;  case KEY\_RIGHT:  case 'd':  snake.setDirection(right);  break;  case KEY\_LEFT:  case 'a':  snake.setDirection(left);  break;  case 'p':  gamePause = true;  board.setTimeout(-1);  while (board.getInput() != 'p')  ;  gamePause = false;  board.setTimeout(old\_timeout);  break;  default:  break;  }  }  void updateState() // 수정  {  // 6/15  snakeTime += TICK;  appleTime += TICK;  poisonTime += TICK;  if (snakeTime % 1000 == 0)  {  gameScore.game\_time += 1;  scoreboard.updateScore(gameScore);  }  gameOverHandle();  hanleNextPiece(snake.nextHead());  if (apple == NULL)  {  createApple();  }  if ((gameScore.max\_len > 5) && (poison == NULL))  {  createPoison();  }  if (appleTime == ITEMTICK)  {  board.addAt(tmp\_apple\_y, tmp\_apple\_x, ' ');  appleTime = 0;  delete apple;  apple = NULL;  }  if (poisonTime == ITEMTICK)  {  board.addAt(tmp\_poison\_y, tmp\_poison\_x, ' ');  poisonTime = 0;  delete poison;  poison = NULL;  }  }  void redraw()  {  board.refresh();  scoreboard.refresh();  }  bool isOver()  {  return gameOver;  }  bool isClear()  {  return gameClear;  }  }; |

SnakeGame.hpp는 게임 구동에 대한 세부적인 알고리즘이 작성된 파일이다. 게임판 설정 파일인 Board.hpp파일과 점수판 설정 파일인 Scoreboard.hpp 파일, 그리기 요소들에 대한 인터페이스가 담긴 Drawable.hpp파일, 게임 내 각종 요소들에 대한 구현 파일(Empty.hpp, Apple.hpp, Snake.hpp, Poison.hpp, gameScore.hpp)을 include한다. 또한 TICK(틱)값을 250으로 정의하고, ITEMTICK값은 10000으로 정의한다. SnakeGame 클래스를 통해 작동하며, 이 클래스에 대한 설명은 다음과 같다.

SnakeGame 클래스의 멤버 변수는 Board 타입의 board, Apple 타입을 가리키는 포인터 apple, Poison 타입을 가리키는 포인터 poison, Snake 타입의 snake, Scoreboard 타입의 scoreboard, s\_score 타입의 gameScore, bool 타입의 gamePause, gameOver, gameClear, int 타입의 tmp\_apple\_x, tmp\_apple\_y, tmp\_poison\_x, tmp\_poison\_y, snakeTime, appleTime, poisonTime이다.

SnakeGame클래스의 private 멤버 함수는 createApple, createPoison, eatApple, eatPoison, removeTail, hanleNextPiece, gameOverHandle이다. createApple과 createPoison은 각각 게임판에 사과와 독을 생성하는 함수로, getEmptyCoordinates를 통해 랜덤한 빈 공간의 좌표를 받아온 후, 그 y좌표와 x좌표를 Apple 객체 생성자의 인자로 넘겨 각각 사과와 독을 생성한다. 그리고 다른 함수(updateState)에서 사용할 목적으로 tmp\_apple\_x, tmp\_apple\_y, tmp\_poison\_x, tmp\_poison\_y에 각각 사과가 위치한 좌표와 독이 위치한 좌표를 할당한다. 그리고 각각 appleTime과 poisonTime을 초기화한다.

eatApple과 eatPoison은 각각 스네이크가 사과와 독을 먹었을 때 호출되는 함수로, 함수가 호출되면 각각 apple과 posion은 삭제되고 거기에 NULL이 할당된다. 사과를 먹었을 때는 gameScore.apple\_score를 증가시킴으로써 사과를 먹은 카운트를 1 증가시키고, 독을 먹었을 때는 gameScore.poison\_score를 증가시킴으로써 독을 먹은 카운트를 1 증가시킨다. 그리고 각각 gameScore.cur\_len에 +1과 -1을 더하여 스네이크의 길이를 업데이트한다. 사과를 먹어서 길이가 gameScore.cur\_len이 1 증가했을 때 gameScore.max\_len보다 값이 커지면 gameScore.max\_len의 값을 업데이트한다. 기존에 있던 사과, 독이 없어졌으므로 appleTime과 poisonTime을 0으로 초기화한다. 그리고 scoreboard를 바뀐 점수로 업데이트한다.

removeTail은 스네이크의 꼬리를 제거하는 함수이다. snake.tail()에 getX와 getY를 주어 꼬리부분의 좌표를 구하고, 그 좌표로 Empty 객체를 만든 후, board에 add한다. 그리고 removePiece 를 실행시켜 마무리한다. hanleNextPiece는 SnakePiece타입의 매개변수 next를 입력받고, apple이나 poison이 NULL일 때, 다음 좌표의 문자를 입력받는다. 그리고 다음 좌표에 위치한 문자가 ‘A’이면 사과를 먹는 함수를 호출하고, ‘P’이면 독을 먹은 것으로 간주한 후 removeTail을 두 번 실행하여 꼬리를 제거하면서 이동하고, ‘ ‘이면 removeTail을 한 번만 실행하여 단순 이동을 하고, 그 외의 케이스는 비정상적인 문자로 간주하고 gameOver에 true를 할당한다. 만약 apple이나 poison이 NULL이 아닐 때, appleTime이 TICK과 같다면 removeTail을 한 번 실행하여 이동한다. 조건문을 빠져나왔다면 next를 board에 add하고, snake 객체에 next를 addPiece한다.

gameOverHandle은 게임 종료와 관련된 동작에 대한 함수이다. 스네이크가 잘못된 방향이라면, 스네이크의 사이즈가 3보다 작다면 gameOver에 true를 할당하여 게임을 종료시킨다. 최대 길이가 10 이상이고, 사과를 먹은 횟수가 10 이상이고, 독을 먹은 횟수가 2 이상이라면 게임 클리어로 간주하고 gameOver와 gameClear에 모두 true를 할당하여 게임을 종료시킨다.

SnakeGame 클래스의 public 멤버 함수는 SnakeGame, ~SnakeGame, initialize, processInput, updateState, redraw, isOver, isClear이다. SnakeGame과 ~SnakeGame은 각각 클래스의 생성자와 소멸자로, 생성자가 호출되면 Board 객체와 Scoreboard 객체가 만들어지고 이를 통해 게임판과 점수판이 생성된다. initialize로 게임 내 각종 요소들을 기본값으로 초기화하고, srand를 통해 랜덤 난수 생성이 가능하도록 한다.

initialize는 게임 내 각종 요소들을 기본값으로 초기화하는 함수이다. apple과 poison에 각각 NULL을 할당하고, board.initialize와 scoreboard.initialize로 게임판과 점수판도 초기화한다. snakeTime, appleTime, poisonTime에는 모두 0을 할당하고, gamePause, gameOver, gameClear에는 모두 false를 할당한 후 srand를 통해 랜덤 난수 생성이 가능하도록 한다. snake.setDirection으로 스네이크 객체의 최초 진행 방향을 아래 방향으로 설정하고, hanleNextPiece를 통해 스네이크의 형태를 최초 생성한다. apple이 NULL이면 사과를 생성하도록 한다.

processInput은 게임판 상에서 키보드의 입력에 따라 스네이크의 이동 방향을 제어하는 함수이다. 키보드의 입력을 board.getInput을 통해 받고, 이를 chtype 변수인 input에 할당한다. 그리고 board.getTimeout을 통해 타임아웃값을 불러와 old\_timout에 할당한다. input값이 w이거나 위쪽 방향키이면 스네이크의 방향을 up으로 설정하고, s이거나 아래쪽 방향키이면 스네이크의 방향을 down으로 설정하고, d이거나 오른쪽 방향키이면 스네이크의 방향을 right로 설정하고, a이거나 왼쪽 방향키이면 스네이크의 방향을 left로 설정한다. 그리고 pause를 의미하는 p가 입력되면 gamePause에 true를 할당하고 타임아웃을 -1로 설정하여 게임을 일시정지한다. p가 다시 입력되면 gamePause에 false를 할당하고 타임아웃을 다시 기존값으로 설정한다. 그 외 다른 입력이 들어오면 무시한다.

updateState는 상태를 업데이트하는 함수이다. updateState 함수가 실행될 때마다 snakeTime과 appleTime, poisonTime에 계속 TICK값을 더한다. 만약 SnakeTime을 1000으로 나눈 나머지 값이 0이라면 gameScore.game\_time에 1을 더함으로써 1초가 지났음을 나타낸다. 그리고 scoreboard.updateScore로 업데이트한다. gameOverHandle과 hanleNextPiece를 호출하여 updateState가 호출될 때마다 게임 종료 여부를 감지하고 다음 위치에 따라 이동할 수 있도록 한다. apple이 NULL이면 사과를 생성한다. gameScore.max\_len이 5보다 크고 poison이 NULL이면 독을 생성한다. 이로 인해 스네이크는 한번이라도 길이 5를 넘겨야 poison을 먹을 수 있다. appleTime이나 poisonTime이 ITEMTICK과 같아지면 각각 tmp로 저장해두었던 좌표의 사과와 독을 제거한다. 따라서 updateState이 다음에 호출되면 새로 사과와 독을 랜덤한 위치에 생성하도록 한다.

redraw는 refresh를 통해 게임판과 점수판을 새로고침한다. isOver은 gameOver값을 반환하며, isClear는 gameClear값을 반환한다.

**Drawable.hpp**

|  |
| --- |
| #pragma once  class Drawable  {  public:  Drawable()  {  y = x = 0;  icon = ' ';  }  Drawable(int \_y, int \_x, chtype ch)  {  y = \_y;  x = \_x;  icon = ch;  }  int getX()  {  return x;  }  int getY()  {  return y;  }  chtype getIcon()  {  return icon;  }  protected:  int y, x;  chtype icon;  }; |

Drawable.hpp는 그리기 요소들(Apple, Poison, Gate, Snake 등)의 인터페이스가 작성된 파일이다. Drawable 클래스를 통해 작동하며, 다른 그리기 요소에 해당하는 클래스들은 이 클래스를 필수적으로 상속해야 한다. 이 클래스에 대한 설명은 다음과 같다.

Drawable 클래스의 멤버 변수는 int 타입을 가리키는 x, y와 chtype 타입의 icon이다. x, y에는 각각 그리기 요소의 x좌표와 y좌표가 할당되고, icon에는 그리기 요소에 해당하는 문자가 할당된다.

public 멤버 함수는 Drawable, getX, getY, getIcon이다. Drawable은 클래스의 생성자로, 매개변수로 \_x, \_y 좌표와 \_icon이 입력될 경우, 그대로 각각 x, y, icon에 값을 세팅한다. getX, getY, getIcon은 각각 x, y, icon의 getter함수로, 각각 설정된 x, y, icon을 반환한다.

**Apple.hpp**

|  |
| --- |
| #pragma once  #include "Drawable.hpp"  class Apple : public Drawable  {  public:  Apple(int \_y, int \_x)  {  y = \_y;  x = \_x;  icon = 'A';  }  char getIcon()  {  return this->icon;  }  }; |

Apple.hpp는 게임 내 요소 중 하나인 사과를 생성하는 파일이다. 따라서 그리기 요소들의 인터페이스가 작성된 파일인 Drawable.hpp를 include한다. Drawable 클래스를 상속한 Apple 클래스를 통해 작동하며, 이 클래스의 멤버 변수는 Drawable 클래스의 멤버 변수인 x, y, icon을 그대로 사용한다. 클래스의 생성자는 매개변수로 \_x, \_y 좌표가 입력될 경우, 그대로 각각 x, y에 값을 세팅하며, icon은 기본적으로 사과를 의미하는 ‘A’로 세팅한다. getIcon은 icon의 getter 함수로, 사과의 icon인 A를 반환한다.

**Empty.hpp**

|  |
| --- |
| #pragma once  #include "Drawable.hpp"  class Empty : public Drawable  {  public:  Empty(int \_y, int \_x)  {  y = \_y;  x = \_x;  icon = ' ';  }  }; |

Empty.hpp 는 공백을 생성하는 파일이다. 따라서 그리기 요소들의 인터페이스가 작성된 파일인 Drawable.hpp를 include한다. Drawable 클래스를 상속한 Empty 클래스를 통해 작동하며, 이 클래스의 멤버 변수는 Drawable 클래스의 멤버 변수인 x, y, icon을 그대로 사용한다. 클래스의 생성자는 매개변수로 \_x, \_y 좌표가 입력될 경우, 그대로 각각 x, y에 값을 세팅하며, icon은 기본적으로 ‘ ‘로 세팅한다.

**Gate.hpp**

|  |
| --- |
| #pragma once  #include "Drawable.hpp"  class Gate : public Drawable  {  private:  public:  Gate(int \_y, int \_x)  {  y = \_y;  x = \_x;  icon = 'G';  }  }; |

Gate.hpp는 게임 내 요소 중 하나인 게이트를 생성하는 파일이다. 따라서 그리기 요소들의 인터페이스가 작성된 파일인 Drawable.hpp를 include한다. Drawable 클래스를 상속한 Gate 클래스를 통해 작동하며, 이 클래스의 멤버 변수는 Drawable 클래스의 멤버 변수인 x, y, icon을 그대로 사용한다. 클래스의 생성자는 매개변수로 \_x, \_y 좌표가 입력될 경우, 그대로 각각 x, y에 값을 세팅하며, icon은 기본적으로 게이트를 의미하는 ‘G’로 세팅한다.

**Poison.hpp**

|  |
| --- |
| #pragma once  #include "Drawable.hpp"  class Poison : public Drawable  {  public:  Poison(int \_y, int \_x)  {  y = \_y;  x = \_x;  icon = 'P';  }  char getIcon()  {  return this->icon;  }  }; |

Poison.hpp는 게임 내 요소 중 하나인 독을 생성하는 파일이다. 따라서 그리기 요소들의 인터페이스가 작성된 파일인 Drawable.hpp를 include한다. Drawable 클래스를 상속한 Poison 클래스를 통해 작동하며, 이 클래스의 멤버 변수는 Drawable 클래스의 멤버 변수인 x, y, icon을 그대로 사용한다. 클래스의 생성자는 매개변수로 \_x, \_y 좌표가 입력될 경우, 그대로 각각 x, y에 값을 세팅하며, icon은 기본적으로 독을 의미하는 ‘P’로 세팅한다. getIcon은 icon의 getter 함수로, 독의 icon인 P를 반환한다.

**Scoreboard.hpp**

|  |
| --- |
| #pragma once  #include "gameScore.hpp"  class Scoreboard  {  WINDOW \*score\_win;  void construct()  {  score\_win = newwin(20, 40, 5, 100);  box(score\_win, 0, 0);  wrefresh(score\_win);  }  public:  Scoreboard()  {  construct();  }  void initialize(s\_score initial\_score)  {  updateScore(initial\_score);  refresh();  }  void updateScore(s\_score gameScore)  {  mvwprintw(score\_win, 1, 1, "MISSON");  mvwprintw(score\_win, 2, 1, "Grow your length : %d / 10 ", gameScore.max\_len);  mvwprintw(score\_win, 3, 1, "Eat Apple Count : %d / 10 ", gameScore.apple\_score);  mvwprintw(score\_win, 4, 1, "Eat Posion Count : %d / 2 ", gameScore.poison\_score);  mvwprintw(score\_win, 5, 1, "Game Time : %d / 60", gameScore.game\_time);  mvwprintw(score\_win, 7, 1, "Score Board");  mvwprintw(score\_win, 8, 1, "Current Length/Max Length : %d / %d", gameScore.cur\_len, gameScore.max\_len);  mvwprintw(score\_win, 9, 1, "Eat Apple Count : %d ", gameScore.apple\_score);  mvwprintw(score\_win, 10, 1, "Eat Poison Count : %d ", gameScore.poison\_score);  mvwprintw(score\_win, 11, 1, "Game Time : %d", gameScore.game\_time);  }  void refresh()  {  wrefresh(score\_win);  }  }; |

Scoreboard.hpp는 점수판을 설정하는 파일이다. 게임 점수를 다루는 파일인 gameScore.hpp를 include한다. Scoreboard 클래스를 통해 작동하며, 이 클래스에 대한 설명은 다음과 같다.

Scoreboard의 멤버 변수는 WINDOW 타입을 가리키는 포인터 변수인 score\_win이다. score\_win은 점수판 윈도우의 주소값이 할당된다. private 멤버 함수는 construct이다. construct로 해당 좌표에 윈도우를 생성하고, box로 외곽을 설정한 후, wrefresh로 새로고침한다.

public 멤버 함수는 Scoreboard, initialize, updateScore, refresh이다. Scoreboard는 클래스의 생성자로, 실행되면 private 멤버 함수인 construct를 실행하여 점수판 윈도우를 새로 만든다. initialize는 s\_score 타입의 Score\_menu 구조체 변수 initial\_score를 매개변수로 입력받아 updateScore를 통해 점수를 업데이트한다. updateScore는 s\_score타입의 Score\_menu 구조체 변수 gameScore를 매개변수로 입력받아 게임 스코어를 각 항목별로 업데이트한 후 mvwprintw를 통해 출력한다. refresh는 wrefresh를 통해 점수판 윈도우를 새로고침한다.

**Snake.hpp**

|  |
| --- |
| #pragma once  #include <queue>  enum Direction  {  up = -1,  down = 1,  left = -2,  right = 2  }; |

Snake.hpp는 스네이크 객체의 움직임과 방향 등을 포함한 전반적인 스네이크의 이동 알고리즘을 담당하는 파일이다. 스네이크를 queue를 통해 표현하기 위해 queue를 include한다. 방향을 직관적으로 나타내야 하므로 enum을 통해 up, down, left, right를 설정했다. SnakePiece 클래스와 Snake 클래스를 통해 작동하며, 각 클래스에 대한 설명은 다음과 같다.

|  |
| --- |
| class SnakePiece : public Drawable  {  public:  SnakePiece()  {  this->x = this->y = 0;  this->icon = '@';  }  SnakePiece(int \_y, int \_x)  {  x = \_x;  y = \_y;  icon = '@';  }  }; |

SnakePiece 클래스는 그리기 요소로서의 스네이크를 담당하는 클래스이다. 따라서 그리기 요소들의 인터페이스가 작성된 클래스인 Drawable 클래스를 상속한다. SnakePiece의 생성자는 매개변수로 아무 것도 입력되지 않았을 경우 y와 x를 0으로 초기화하고, 매개변수로 y좌표와 x좌표가 입력되면 y와 x를 입력된 좌표값으로 설정한다. 두 경우 모두 아이콘은 스네이크를 가리키는 ‘@’으로 설정한다.

|  |
| --- |
| class Snake  {  private:  std::queue<SnakePiece> prev\_pieces;  Direction cur\_direction;  bool wrongDirection = false;  bool size3 = false;  int maxSize, curSize;  public:  Snake()  {  cur\_direction = down;  maxSize = curSize = prev\_pieces.size();  }  void addPiece(SnakePiece piece)  {  prev\_pieces.push(piece);  curSize++;  if (curSize > maxSize)  {  maxSize = curSize;  }  }  void removePiece()  {  prev\_pieces.pop();  curSize--;  if (curSize < 2)  {  size3 = true;  }  }  SnakePiece tail()  {  return prev\_pieces.front();  }  SnakePiece head()  {  return prev\_pieces.back();  }  Direction getDirection()  {  return cur\_direction;  }  bool getWrongDirection()  {  return this->wrongDirection;  }  bool getSize3()  {  return this->size3;  }  int getCurSize()  {  return this->curSize;  }  void setDirection(Direction d)  {  if (cur\_direction == d)  {  return;  }  else if (cur\_direction + d != 0)  {  cur\_direction = d;  }  else  {  wrongDirection = true;  }  }  SnakePiece nextHead()  {  int row = head().getY();  int col = head().getX();  switch (cur\_direction)  {  case down:  row++;  break;  case up:  row--;  break;  case left:  col--;  break;  case right:  col++;  break;  }  return SnakePiece(row, col);  }  }; |

Snake 클래스는 스네이크의 이동 알고리즘을 담당하는 클래스이다. private 멤버 변수는 queue<SnakePiece> 타입의 prev\_pieces, Direction 타입의 cur\_direction, bool 타입의 wrongDirection와 size3, int 타입의 maxSize와 curSize가 있다. prev\_pieces는 큐 형식으로서의 스네이크 몸체를 가리킨다. cur\_direction은 현재 스네이크가 진행 중인 방향을 가리킨다. wrongDirection은 잘못된 방향인지를 가리킨다. size3은 스네이크의 길이가 3보다 작은지를 가리킨다. maxSize와 curSize는 게임 진행을 통해 누적된 스네이크 몸체의 최고 길이와 현재 스네이크 몸체의 길이가 각각 할당된다.

Snake클래스의 public 멤버 함수는 Snake, addPiece, remove, removePiece, tail, head, getDirection, getWrongDirection, getSize3, getCurSize, setDirection, nextHead이다. Snake는 클래스의 생성자로, 매개변수를 입력받지 않는다. 생성자가 호출되면 cur\_direction을 down으로 초기화하고, maxSize와 curSize는 현재 prev\_pieces의 길이를 그대로 할당받는다.

addPiece는 스네이크의 길이를 1만큼 증가시키는 함수이다. 매개변수로 SnakePiece타입의 piece를 입력받으며, prev.pieces에 push를 통해 큐의 원소를 추가, 즉 스네이크 몸체의 길이를 1만큼 늘린다. 늘린 후 curSize를 1 증가시키고, 만약 curSize가 maxSize보다 크다면 maxSize를 변경된 curSize로 업데이트한다.

removePiece는 스네이크의 길이를 1만큼 감소시키는 함수이다. prev.pieces에 pop을 통해 큐의 원소를 제거, 즉 스네이크의 몸체의 길이를 1만큼 줄인다. 줄인 후 curSize를 1 감소시키고, 만약 curSize가 2보다 작다면, 즉 스네이크 전체의 길이가 3보다 작다면 size3에 True를 할당한다.

tail은 스네이크의 꼬리, 즉 queue의 맨 앞 원소를 반환한다. head는 스네이크의 머리, 즉 queue의 맨 뒤 원소를 반환한다. getDirection은 현재 방향 cur\_direction을 반환한다. getWrongDirection은 잘못된 방향인지의 여부인 wrongDirection을 반환한다. getSize3은 스네이크의 길이가 3보다 작은 지의 여부인 size3을 반환한다. getCurSize는 현재 길이인 curSizes를 반환한다.

setDirection은 Direction 타입의 매개변수 d를 입력받아, 입력된 방향에 맞게 cur\_direction을 재설정한다. d가 cur\_direction과 일치하면 할당하지 않고, cur\_direction과 일치하지 않으면 d를 cur\_direction에 할당한다. d로 잘못된 방향이 입력되면 wrongDirection에 true를 할당한다.

nextHead는 스네이크 머리가 다음에 이동해야 할 좌표를 구하고 그 좌표에 스네이크를 나타내는 문자인 ‘@’가 위치하도록 해서 스네이크가 이동되도록 하는 함수이다. 현재 스네이크 머리의 y좌표와 x좌표를 각각 row와 col에 할당하고, cur\_direction이 아래 방향인지, 위 방향인지, 왼쪽 방향인지, 오른쪽 방향인지에 따라 row와 col을 수정함으로써 그 방향으로 한 칸 이동시킨다. 그리고 SnakePiece를 통해 그 위치에 적용하고, 이를 리턴한다.

**gameScore.hpp**

|  |
| --- |
| #ifndef GAMESCORE\_HPP  #define GAMESCORE\_HPP  typedef struct Score\_menu  {  int cur\_len;  int max\_len;  int apple\_score;  int poison\_score;  int game\_time;  Score\_menu()  {  cur\_len = max\_len = 3;  apple\_score = poison\_score = game\_time = 0;  }  } s\_score;  #endif |

gameScore.hpp는 게임 점수를 다루는 파일이다. 이를 위해 Score\_menu라는 구조체를 정의하였다. Score\_menu의 멤버 변수는 cur\_len, max\_len, apple\_score, poison\_score, game\_time이고 각각 스네이크의 현재 길이, 스네이크의 최대 길이, 사과를 먹은 수, 독을 먹은 수, 게임 시간을 의미한다. 생성자가 실행되면 각 멤버변수 값들을 초기값인 3, 3, 0, 0, 0으로 세팅한다.

**Time.hpp**

|  |
| --- |
| #pragma once  #include <chrono>  #include <sys/time.h>  #include <time.h>  class Time  {  public:  static time\_t milliseceonds()  {  struct timeval time\_now;  gettimeofday(&time\_now, nullptr);  time\_t msecs\_time = (time\_now.tv\_sec \* 1000) + (time\_now.tv\_usec / 1000);  return msecs\_time;  }  }; |

Time.hpp는 게임 내 시간을 다루는 파일이다. 시간 관련 설정을 위해 시간 측정에 괗나 C++ 표준 라이브러리 헤더 파일인 chrono를 include했고, sys/time.h와 time.h를 include했다. Time 클래스 내에 time\_h 구조체를 구성하였으며, 구조체 내에서time.h 헤더의 timeval 구조체를 불러와 time\_now로 설정한 후, gettimeofday의 인자로 time\_now의 레퍼런스를 입력하여 time\_now에 현재 시간을 입력하였다.

msecs\_time에는 time.now의 초 단위 시간정보에 1000을 곱한 값과 time.now의 마이크로초 단위 시간정보를 1000으로 나눈 값을 더한 값이 할당되고, 이것을 그대로 time\_h 타입의 현재 시간을 반환한다.

**Makefile**

|  |
| --- |
| all : main  main : \*.cpp  g++ main.cpp -lncurses -o main && ./main |

Makefile을 이용한 컴파일은 컴파일 방법 중의 하나로, 컴파일 과정 자동화를 통해 시간을 절약하는 방법이다. 이 프로그램을 실행하기 위해서는 터미널에 $ g++ main.cpp -lncurses -o main && ./main을 모두 입력해야 하므로 사용자 입장에서는 번거롭다. 따라서 사용자의 편의를 위해 Makefile 파일에 위와 같이 입력하였고 이를 통해 make만 입력해도 main부분이 실행될 수 있도록 했다.

### 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.** |

**Ncurses 라이브러리**

ncurses라이브러리는 이번 프로젝트를 수행하는 데에 있어 핵심이 된 라이브러리였다. 이 라이브러리를 통해 스네이크의 이동 양상, 획득 점수, 게임판, 점수판 등을 화면에 GUI 형식으로 사용자에게 보일 수 있었다. 이를 위해 ncurses 내의 다양한 메소드들을 활용했으며, 많은 요소들을 사용했지만 주로 사용된 데이터 타입, 메소드들은 다음과 같다. (메소드의 경우 입력 인자는 생략)

* **WINDOW \***

특정 윈도우를 가리키는 포인터 데이터 타입이다. 또한 stdscr는 기본 윈도우의 포인터이다. 본 프로젝트에서는 게임판, 점수판 등을 만드는 데 사용하였다.

* **initscr**

ncurses 자료구조들을 초기화하고 terminfo 파일을 읽는 데 사용된다. 본 프로젝트에서는 main.cpp 함수에서 초기 환경을 설정하는 데 사용하였다.

* **refresh**

일종의 새로고침 역할을 한다. 어떤 출력을 수행한 뒤에 refresh 메소드를 호출해야 정상적으로 화면에 반영된다. 본 프로젝트에서는 출력 코드를 넣을 때마다 아랫줄에 추가함으로써 추가한 내용이 바로 반영될 수 있도록 했다.

* **noecho**

사용자로부터 입력받은 문자를 출력하지 않도록 설정한다. 본 프로젝트에서는 main.cpp 함수에서 초기 환경을 설정하는 데 사용하였다.

* **wborder**

특정 윈도우의 경계를 설정해주는 메소드이다. 본 프로젝트에서는 Board.hpp에서 윈도우의 외곽을 꾸미는 데 사용하였다.

* **wrefresh**

refresh를 WINDOW에 대해 수행하고자 할 때 사용되는 메소드이다. 본 프로젝트에서는 윈도우 단위에서 새로고침을 수행하고자 할 때 refresh와 비슷한 용도로 사용하였다.

* **newwin**

새로운 윈도우를 생성하는 메소드이다. 윈도우의 사이즈와 생성할 위치 y, x좌표 값을 인자로 요구한다. 본 프로젝트에서는 main.cpp, Board.hpp, Scoreboard.hpp에서 각각 메인 윈도우와 게임판 윈도우, 점수판 윈도우를 생성하기 위해 사용하였다.

* **box**

wborder와 마찬가지로 특정 윈도우의 경계를 설정해주는 메소드이다. 본 프로젝트에서는 main.cpp, Scoreboard.hpp 에서 각각 메인 윈도우와 점수판 윈도우의 외곽을 꾸미는 데 사용하였다.

* **mvwprintw**

특정 윈도우의 주어진 위치에서부터 문자열을 출력한다. 본 프로젝트에서는 main.cpp와 Scoreboard.hpp에서 각각 메인 윈도우와 점수판 윈도우에 문자열을 출력하는 용도로 사용하였다.

* **keypad**

표준 스크린에서 입력을 받도록 설정할 때 사용되는 메소드이다. 본 프로젝트에서는 Board.hpp에서 키패드의 입력을 활성화하기 위해 사용하였다.

* **getch**

터미널 창에서 문자를 가져와 읽는다. 본 프로젝트에서는 main.cpp에서 문자를 받아들이기 위해 사용하였다.

* **mvwinch**

특정 윈도우의 주어진 위치에서 chtype 유형의 문자를 반환한다. 본 프로젝트에서는 Board.hpp의 게임판 윈도우에서 chtype 문자를 반환하기 위해 사용하였다.

### 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.** |

- 처음 Map을 작성했을때 좌표위치나 WINDOW값이 해당에 제대로 전달이 되지않아 Segement fault 및 Abort가 빈번히 발생하였다. c++수업에서 제공된 ncurse설명서를 다시한번 꼼꼼히 살펴보고 ncurse레퍼런스의 설명을 집중해서 이해하면서 오류를 해결할 수 있었다.

- 지렁이가 일정시간이 되면 자동으로 길이가 늘어나는 버그가 있었다. 늘어난 길이가 점수판에 적용되지않았고 오랜 코드 분석에도 버그를 찾아내기 쉽지않았다. 하지만 계속된 확인을 통해 길이가 늘어나는 코드를 발견하였고 이를 수정하여 버그를 고쳤다.

- 아이템또한 일정시간 생성되면 지워져야하는데 지워지지않고 무한생성되는 버그가 있었으나 이 또한 지속적인 피드백을 통해 오류를 수정할 수 있었다.

- 포탈을 생성할 때 벽에 랜덤하게 포탈을 생성하고 지렁이가 닿았을때 순간이동 시키도록 했어야하는데 포탈생성에서 세그먼트 폴트가 계속 발생하였고 해결후에도 포탈이 남은 부분에 공백이 생겼으며 포탈 알고리즘을 구현하지 못하였다.

- 게임 시간을 시각화해야하는데 처음엔 chrono라이브러리를 활용하려했으나 해당 라이브러리에 대한 이해도 부족으로 엉성하게 구현되었다. 이를 수정하여 지렁이 속도를 초점으로 시간을 구현하였고 거의 정확하게 구현에 성공할 수 있었다.

- ncurses.h 한글 지원이 안 된다.

방법 1

locale.h를 인클루드 하고

로케일 설정을 한다.

#include <curses.h>

#include <locale.h>

setlocale(LC\_CTYPE, "ko\_KR.utf-8");

setlocale(LC\_ALL, "ko\_KR.utf-8");

방법 2

ncurses 라이브러리 배포판 중에 데비안 패키지인 libncurses5-dev는 유니코드를 지원하지 않는다. 한글을 제대로 출력하려면 유니코드를 지원하는 버전인 libncursesw5-dev를 사용해야 한다.

설치: 데비안 기반 배포판인 경우

sudo apt-get install libncursesw5-dev

헤더 인클루드: 다른 라이브러리를 쓰는 것이므로 curses.h 대신 ncursesw/curses.h를 인클루드해야 한다.

#include <ncursesw/curses.h>

소스코드를 UTF-8으로 저장

링크: -lncurses 대신 -lncursesw 옵션으로 링크하면 된다.

g++ SOURCE.c -o APP -lncursesw

g++ -I "/home/jyc/Desktop/SnakeGame" main.cpp -o output -lncursesw

출처

https://bakyeono.net/post/2015-05-12-ncurses-korean-utf-8.html

https://www.debianusers.or.kr/node/118

https://onecellboy.tistory.com/32

### 결과물 목록

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

|  |  |  |
| --- | --- | --- |
| **구분** | **파일명** | **설명** |
| 소스 코드 | Apple.hpp | 사과 객체 생성을 위한 파일 |
| Board.hpp | 게임판을 설정하는 파일 |
| Drawable.hpp | 그리기 요소의 인터페이스 파일 |
| Empty.hpp | 공백 생성을 위한 파일 |
| Gate.hpp | 게이트 객체 생성을 위한 파일 |
| Makefile | Makefile 파일 |
| Poison.hpp | 독 객체 생성을 위한 파일 |
| Scoreboard.hpp | 점수판을 설정하는 파일 |
| Snake.hpp | 스네이크 객체 및 스네이크의 이동 알고리즘 파일 |
| SnakeGame.hpp | 스네이크 게임의 구동 알고리즘 파일 |
| Time.hpp | 게임 내 시간을 다루는 파일 |
| gameScore.hpp | 게임 내 각종 점수에 대한 파일 |
| main.cpp | 게임 실행에 관한 파일 |
| 실행 파일 | main | Makefile의 make에 의해 생성된 바이너리 파일 |
| README | README.md | README 파일 |
| 결과보고서 | 2022-기말프로젝트-이용훈.docx | 프로젝트 결과보고서 |

# 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.** |

**20195301 이용훈**

구글링을 통해 스네이크 게임의 기본적인 수도 코드를 작성하였다. 또한 팀원들이 추가한 코드 혹은 기능에 대해 오류나 개선사항에 대해 지속적으로 피드백을 하는 시간을 만들었다. 실시간 코드 수정으로 오류를 수정했고 어려운 점은 팀과 상의하여 해결하였다. 시간이 더 있었더라면 더 멋있는 프로젝트를 완성할 수 있었을텐데 그 부분이 조금 아쉬웠지만 팀원들과 협력하여 프로젝트를 한 것은 매우 의미있는 시간이었다.

**20213062 이찬우**

보고서 작성을 담당했다. 원래 게이트 생성도 같이 담당했었으나 보고서는 시간을 들여 쓸 수 있었고, 게이트 생성까지는 완료했으나 게이트 구현은 계속 시도해보아도 번번이 실패하였다. 이번 프로젝트를 통해 내가 지닌 코딩 능력이 좋지 않다는 것을 체감할 수 있었으며, 다음에는 팀프로젝트에 더 기여하기 위해 더 많은 코딩 연습을 통해 협업 능력을 개선해야겠다는 생각이 들었다.

**20213078 전예찬**

외부 라이브러리를 컴파일하는 과정에서 오래 막혀있었다. 팀원이 도와줘서 우분투에서 컴파일하는 방법을 익힐 수 있었다. 이전에 '벽돌깨기' 게임을 프로젝트 결과물로 제출한 적이 있어서 어렵지 않게 풀어나갈 수 있을거라 생각했다. 하지만 파이썬과 다르게 메모리를 직접 관리해야 하는 것과 ncurses에 대한 이해도가 낮다 보니 진행 과정에서 난항이 많았다. 진행 과정에서 git을 이용해 진행했는데 이 과정에서 git 명령어를 몇 개 더 익힐 수 있었다.

# 참고 문헌

**참고한 서적, 기사, 기술 문서, 웹페이지를 나열한다.:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 번호 | 종류 | 제목 | 출처 | 발행년도 | 저자 | 기타 |
| 1 | 동영상 강의 | Making Snake in ncurses | https://www.youtube.com/watch?v=MH6QlYJ2SwU&list=PL2U2TQ\_\_OrQ\_TV2-wuHqGaK8qlnxgKUvK | 2021 | Casual Coder | 프로젝트의 기초적인 틀을 만드는데 참고함 |
| 2 | 레퍼런스 | Ncurses programming guide | https://jbwyatt.com/ncurses.html | 2019 | Western University | Ncurses의 메소드들을 사용하는 데 참고함 |

# 부록

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

## 사용자 매뉴얼

**프로젝트 실행 후 사용안내, 따라하기 등 포함**

|  |
| --- |
| **< 게임 규칙>**   1. **진행방향과 같은 반대 방향키를 입력하는 경우 실패. 방향키는 직접 정함.** 2. **Snake의 이동 방향에 Item이 놓여 있는 경우**  * Snake가 Item을 획득 * Growth Item의 경우 -> 몸 길이 1 증가 -> 진행방향으로 증가 * Poison Item의 경우 -> 몸 길이 1 감소 -> 꼬리 부분이 감소 -> 몸 길이 3보다 작아지면 실패 2-1. Growth Item과 Poison Item의 출현 * Snake Body가 있지 않은 임의의 위치에 출현 * 출현 후 일정시간이 지나면 사라지고 다른 위치에 나타나야 함. * 동시에 출현할 수 있는 Item의 수는 3개로 제한함.  1. **Gate**  * Gate는 두 개가 한 쌍 * Gate는 겹치지 않음. * Gate는 임의의 위치에 있는 벽에서 나타남. * Gate는 한번에 한쌍만 나타남. * Gate에 Snake가 진입 시, 다른 Gate로 진출. * Gate에 Snake가 진입 중인 경우: Gate는 사라지지 않음. 다른 위치에서 Gate가 나타나지 않음.  1. **Gate 진출 방향**   **4-1. Gate가 나타나는 벽이 가장자리에 있을 때**   * 항상 Map의 안쪽 방향으로 진출. * 상단벽 -> 아래 방향, 하단벽 -> 위 방향, 좌측벽 -> 우측 방향, 우측벽 -> 좌측 방향   1. **Gate가 나타나는 벽이 Map의 가운데 있을 때 다음 순서로 진출**   **진출 방향이 자유로운 경우:**   * 진입 방향으로 진출   **진출 방향이 좌우인 경우:**   * 우측 진입 시 -> 우로 진출 * 아래로 진입 시 -> 좌로 진출 * 좌측 진입 시 -> 좌로 진출 * 위로 진입 시 -> 좌로 진출   **진출 방향이 상하인 경우:**   * - 우측 진입 시 -> 위로 진출 * 위쪽으로 진입 시 -> 위로 진출 * 좌측 진입 시 -> 아래로 진출 * 아래로 진입 시 -> 아래로 진출  1. **Wall과 Immune Wall**  * Wall: Gate로 변할 수 있다. * Immune Wall: Gate로 변할 수 없다. * All Wall: Snake가 통과할 수 없다. Snake Head와 충돌 시 실패 * Gate의 출현 방법 결정: 게임 시작 후 일정 시간 후 출현 or 몸의 최대 길이 X 초과 시 출현  1. **점수 계산**  * 게임 중 몸의 최대 길이 계산 -> 최종 뱀 길이 / 최대 뱀 길이 * 게임 중 획득한 Growth Item의 수 * 게임 중 획득한 Poison Item의 수 * 게임 중 Gate 사용 횟수 * 게임 시간 (Seconds로 계산) * 게임 방법: 주어진 미션 달성하기 * 미션: 좌측의 각 점수 항목별(최대길이, Growth Item 획득 수, Poison Item 획득 수, Gate 사용 횟수, 게임 시간)로 목표치 도달 시 게임 종료 |

## 설치 방법

**프로그램 컴파일 및 실행 방법 포함**

1. $ sudo apt-get update

$ sudo apt-get install libcurses5-dev libncursesw5-dev

Ubuntu나 MacOS의 터미널에서 위의 명령어를 순서대로 입력하여 Ncurses 라이브러리를 설치한다.

1. SnakeProject의 모든 소스코드 파일들이 있는 zip파일을 다운로드하고, 압축을 해제한다.
2. 생성된 폴더로 터미널에서 cd 명령어를 통해 이동한다.
3. $ make 또는 $ g++ main.cpp -lncurses -o main && ./main 를 입력하여 게임을 실행한다.