C++프로그래밍 프로젝트

프로젝트 명	snake_game
팀 명	4분반 팀 I
문서 제목	결과보고서

Version	1.1
Date	2021.06.18

팀원	윤형준
	박성영

문서 정보 / 수정 내역

Filename	최종보고서-snake_game.doc
원안작성자	윤형준, 박성영
수정작업자	윤형준, 박성영

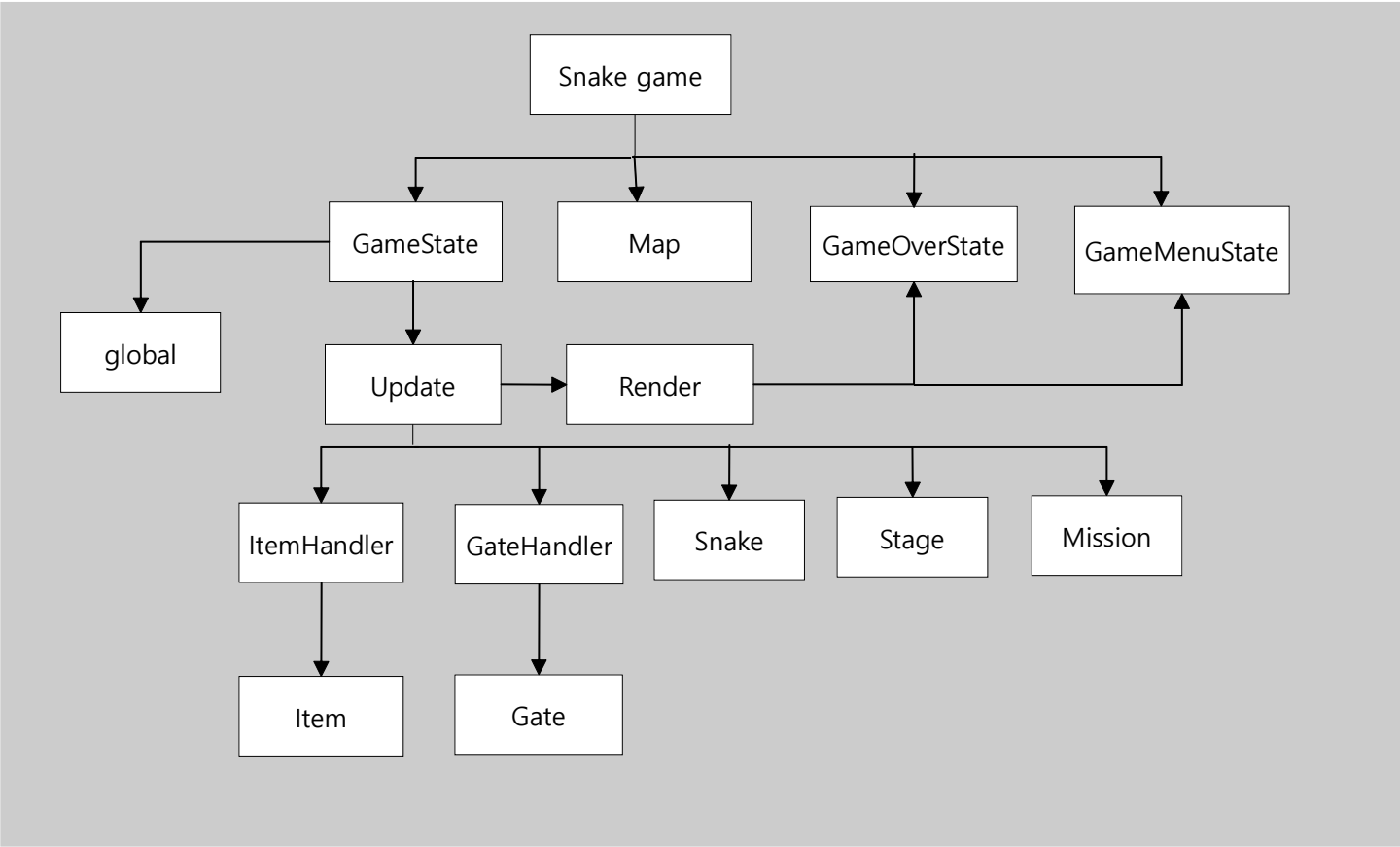
수정날짜	대표수정자	Revision	추가/수정 항목	내 용
2021-06-11	윤형준	1.0	최초 작성	
2021-06-18	박성영	1.1	내용 수정	수정된 연구내용 추가

목 차

1 개요

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

1.1 전체 구조



1.2 개발 내용

C++ 언어와 ncurses 라이브러리를 활용하여, snake game을 제작한다.

- 사용자에게 보이는 화면은 ncurses 윈도우를 이용해 구현한다.
- C++ 버전은 11을 사용하여 개발하였다.

1.3 사용한 외부 라이브러리

- ncurses

ncurses는 GNU에서 개발한 new curses 라이브러리이며, 1990년대 중반에 curses 라이브러리의 개발이 중단된 후 개발된 라이브러리이다. 이때, curses는 Cursor Optimization에서 유래했으며, 이를재미있게 발음한 것을 말한다. curses 라이브러리는 유닉스 계열 운영체제를 위한 제어 라이브러리 중하나이고 매우 유연하고 효율적인 Application Programing Interface를 제공한다. 또한, 커서를 움직이거나, 윈도우를 생성하고, 색깔을 표시하거나, 마우스 관련 함수를 제공하는 등의 TUI 응용프로그램들의 구성을 가능하게 하는 라이브러리이다.

- ncurses 설치 방법

- CentOS7

```
$ sudo yum install ncurses-devel
```

- Ubuntu

```
$ sudo apt-get update
$ sudo apt-get install libcurses5-dev libcursesw5-dev
```

- make

make는 유닉스 계열 운영체제에서 주로 사용되는 프로그램 빌드 도구이다. 또한, 여러 파일들끼리의 의존성과 각 파일에 필요한 명령을 정의함으로써 프로그램을 컴파일 할 수 있으며 최종 여러 프로그램을 만들 수 있는 과정을 서술할 수 있는 표준 문법을 가지고 있다.

make는 파일 관련 유틸리티로써, 각 파일 간의 종속관계를 파악하여 기술파일에 기술된 대로 컴파일명령 혹은 쉘 명령을 순차적으로 실행한다. 즉, make를 활용하면 각 파일에 대한 반복적인 명령을 자동화시켜서 개발자의 수고를 덜고 시간을 절약할 수 있다.

2 개발 내용 및 결과물

2.1 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
6단계	추가 구현	적용

1단계 map 구현
: 각 스테이지별 맵 데이터를 stage 디렉토리에서 읽어 옴
: 파일로부터 읽어온 데이터를 터미널 화면에 렌더링
: 맵에는 Space, Wall, Immune Wall이 존재
: 모든 맵의 크기는 WIDTH 62, HEIGHT 32로 고정

2단계 Snake 표현 및 조작
: Snake를 화면에 출력
: 뱀의 초기 몸 길이는 머리를 포함해 5
: 진행 방향은 Head의 방향이며, 초기에는 왼쪽을 바라보고 있음.
: 매 틱마다 뱀이 진행 방향으로 전진
: 사용자는 방향키 입력을 통해 뱀의 진행 방향을 수정할 수 있음
: 이때 진행 방향의 반대 방향으로 움직이려 한 경우 실패
: 뱀이 Wall에 부딪힌 경우 실패
: Snake를 화면에 출력
: 뱀의 초기 몸 길이는 머리를 포함해 5
: 진행 방향은 Head의 방향이며, 초기에는 왼쪽을 바라보고 있음.
: 매 틱마다 뱀이 진행 방향으로 전진
: 사용자는 방향키 입력을 통해 뱀의 진행 방향을 수정할 수 있음

: 이때 진행 방향의 반대 방향으로 움직이려 한 경우 실패
: 뱀이 Wall에 부딪힌 경우 실패

3단계 Item 요소의 구현
: 아이템은 뱀 출현 이후 일정 시간 뒤에 랜덤한 빈 공간에 생성됨.
: 아이템 생성은 고정된 DROP_ITEM_INTERVAL 마다 생성
: 아이템은 맵의 빈 공간(BLOCK_SPACE) 중 하나에 랜덤하게 생성됨
: 아이템의 만료 시간인 DROP_ITEM_EXPIRE이 지나면 아이템은 소멸됨
: 아이템의 개수는 아이템 최대 개수인 MAX_ITEMS를 초과할 수 없음
: 아이템이 소멸되면 아이템 최대 개수를 넘지 않는 선에서 다시 랜덤하게 생성됨
: 아이템의 종류는 과일/독 두 종류
: 과일 아이템을 먹을 경우 진행 방향으로 몸길이 1 증가
: 독 아이템을 먹을 경우 꼬리부터 몸 길이 1 감소
: 이때 독 아이템으로 인해 몸길이가 3보다 작아지면 실패

4단계 Gate 요소의 구현
: 게이트는 뱀 출현 이후 일정 시간 뒤에 랜덤한 벽에 생성됨
: 게이트는 두 개 가 한 쌍을 이룸
: 게이트는 한 쌍 이상 출현할 수 없음
: 게이트는 만료시간인 DROP_ITEM_EXPIRE 시간이 지나면 소멸됨
: 뱀이 한 게이트에 들어간 경우 다른 게이트에서 머리부터 빠져나와야 함
: 뱀이 게이트를 통과 중일 경우에는 게이트가 사라질 수 없음
: 뱀이 게이트를 통과 중인데 만료 시간이 지난 경우 게이트가 소멸되지 않아야 함.
: 그 외에 규칙 #3, #4, #5를 모두 준수해야 함.

5단계 점수 요소의 구현
: 게임 화면 우측에 스코어 보드와 미션 보드를 출력
: 스코어 보드는 현재 점수를 매 틱마다 출력
: 미션 보드는 미션 목표와 달성 여부를 틱마다 출력
: 스코어 보드의 스코어는 미션 달성 여부에 따라 부여되는 점수의 총점
: 미션은 각 스테이지마다 난이도가 높아지도록 구현
: 미션은 뱀의 최대 길이, 과일 아이템 획득 횟수, 독 아이템 획득 횟수, 게이트 통과 횟수 로 이루어져 있음.
: 모든 미션을 클리어한 경우 다음 스테이지로 이동해야 함.
: 이때 뱀의 최대 길이는 현재 몸길이가 아닌 해당 게임 중 달성한 뱀의 최고 길이여여 함.

6단계 추가 구현

● **GameClearState 구현**

: 게임을 클리어한 경우 GameClearState로 전환

● **게임 클리어 시 점수와 미션을 출력**

: 게임 오버 상태에서 스코어 표시

: 게임 오버가 된 경우 미션 달성 여부와 스코어를 모두 표시.

● **스테이지 선택 기능**

: 게임 메뉴에서 1, 2, 3, 4 스테이지 중 원하는 스테이지부터 시작할 수 있도록 선택 기능부여

● **머리가 몸에 닿으면 게임 실패**

: 뱀의 머리가 뱀의 몸통에 닿은 경우 뱀이 죽도록 구현

● **게임 오버 이유 출력**

: 게임 오버가 된 경우 게임 오버의 이유를 알려준다.

: 게임 오버가 가능한 경우는

1. 뱀의 머리가 벽에 닿은 경우
2. 뱀의 머리가 몸에 닿은 경우
3. 뱀의 길이가 3보다 작아진 경우
4. 뱀이 진행 방향의 반대 방향으로 이동하려 한 경우

● **최고 점수 시스템**

: 스코어 보드에서 현재 점수 뿐 아니라 게임의 최고 점수를 동시에 출력

: 게임의 최고 점수를 달성한 경우 파일에 최고 점수를 기록

: 현재 게임 점수가 최고 기록을 갱신하고 있는 경우 매 틱마다 최고 점수 갱신

2.2 개발 내용 및 결과물

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

2.2.1 개발 내용

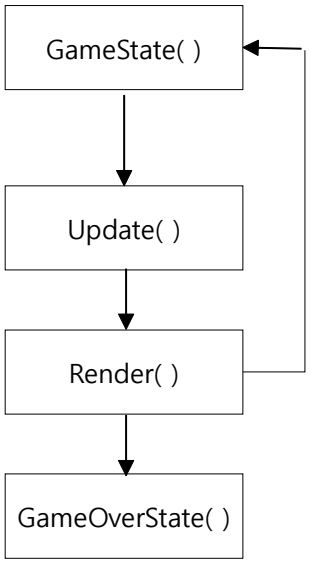
2.2.1.1 초기 설계 내용

- 객체 지향 프로그래밍 언어인 C++의 특징을 최대한 활용하기 위해 각 단계별 목표를 구현하기 전에 뱀 게임 전체에 대한 설계를 한다. 모든 게임관련 구현 요소는 ExcuteDriver()라는 함수가 담당한다. 이때 게임은 State라는 상태로 구분되며 ExcuteDriver는 이 State를 관리하는 드라이버다.

- State는 초기 게임 메뉴를 출력할 GameMenuState부터 게임 중을 나타내는 GameState, 게임 오버 상태인 GameOverSate 등으로 이루어져있다. 모든 State는 BaseState라는 추상 클래스를 상속받아 이루어지며, Update와 Render라는 두 메소드를 반드시 구현해야 한다. 이렇게 State를 구분할 경우 ExcuteDriver에서 각 State의 Update와 Render만 호출하는 형태로 게임 전체를 관리할 수 있다는 이점이 있다.

- 가장 기능이 많은 State는 게임 State. 하지만 게임 State 내에서도 ExecutorDriver처럼 간단한 형태로 모든 게임 요소(뱀, 아이템, 게이트, 점수 등)을 관리 할 수 있도록 하는 것이 전체의 구현 목표다. 따라서 게임 State 역시 ExecutorDriver처럼 각 요소(뱀, 아이템, 게이트, 점수 등)의 Update 함수를 호출하는 형태로 동작할 수 있도록 구현한다.

- 이를 위해 각 게임 요소는 Update 함수를 공통으로 구현해야 한다. 이런 공통점을 OOP의 개념과 엮어 모든 게임 요소는 BaseState가 그러하듯 BaseObject를 상속 받도록 구현하다. 따라서 게임은 다음과 같은 루프를 도는 것으로 간소화 할 수 있다. 또한 Update 함수 내부에서도 이와 같은 루프를 도는 간단한 형태로 게임 전체 목표를 달성할 수 있도록 하였다.



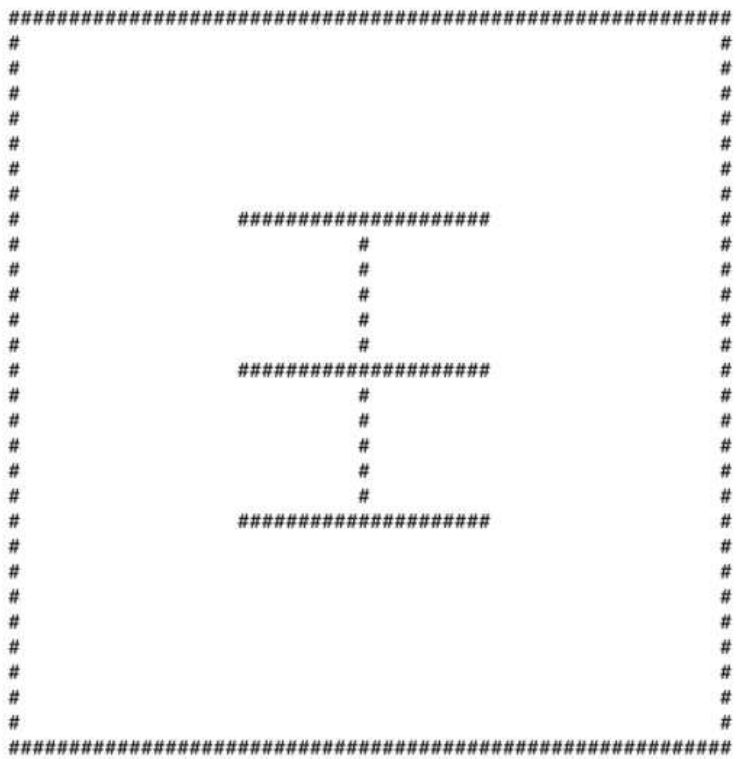
2.2.1.2 Map 구현

- 초기에 설정한 모든 목표를 구현하는 것이 본 절의 목표이며 모든 목표를 달성하였다. 필수 요구 사항이었던 4개의 서로 다른 게임 스테이지를 설계하고 Map 관련 클래스가 이를 화면에 맵으로 출력하도록 설계하였다. 이때 각 맵의 정보나 게임 요소는 모두 매크로로 만들어 관리하였다. 이때 맵의 정보를 램에 올려서 정적으로 사용하거나 하드디스크에서 읽어오는 두 가지 방법이 있었는데, 팀원과의 상의 후 하드 디스크에 파일로 저장한 뒤 이를 읽어오도록 결정하였다. 따라서 맵 데이터는 stage 디렉토리 하위에 있는 스테이지별 파일을 읽어오는 것으로 한다.

- 또한 GameState의 Render 함수 호출 시 맵 정보를 화면에 출력할 수 있도록 했다. 기존적인 맵에는 Space, Wall, Immune Wall이 존재하며 모든 데이터는 매크로로 정의되어있다. 모든 맵의 크기는 WIDTH 62, HEIGHT 32로 고정 크기를 사용하여 개발 시 매크로로 쉽게 변수에 접근할 수 있었다.

● 구현 결과

예시: 스테이지2의 맵을 화면에 렌더링한 결과



2.2.1.3 Snake 표현 및 조작

- 초기에 설정한 모든 목표를 구현하는 것이 본 절의 목표이며 모든 목표를 달성하였다. 뱀의 머리와 몸에 쉽게 접근이 가능해야 하며 각 요소마다 위치 정보를 포함해야 한다. 또한 게이트관련 내용 구현 시 몸의 모든 위치 정보에 각각 접근하는 것도 용이해야 할 것이라고 생각했다. 따라서 뱀의 요소는 STL의 vector를 사용하여 구현하였다. 뱀의 초기 길이는 5로 맵의 중하단에서 왼쪽을 바라보며 등장한다. 뱀은 사용자가 아무런 조작을 가하지 않아도 게임 전체의 틱마다 진행 방향으로 알아서 한 칸씩 이동한다. 이때 진행 방향은 뱀의 머리가 향하는 방향이다. 뱀이 진행 방향으로 전진하면 모든 몸통은 머리가 지나쳐간 대로 따라서 이동한다.

- 사용자는 키보드 입력을 통해 매 틱마다 뱀의 진행 방향을 결정할 수 있다. 뱀의 가능한 진행 방향은 상,하,좌,우 네 방향이며 사용자는 방향키를 이용해 네 방향을 입력할 수 있다. 이때 사용자의 입력방향이 진행 방향과 반대 방향인 경우 뱀이 죽으면서 게임 실패가 되도록 구현하였다.

- 또한 뱀이 벽에 부딪힌 경우 뱀이 죽어야 한다는 목표를 달성하기 위해 매 틱마다 뱀의 머리와 맵의 블록을 비교하는 함수를 구현하였다. 뱀의 머리가 SPACE에 있는 경우 아무일도 일어나지 않으며, 뱀의 머리가 벽과 일치하게 된 경우 뱀이 사망하도록 구현했다.

● 구현 결과

뱀이 중하단에서 시작하여 맵의 상단으로 이동중인 그림

```
############################################################
#                                                          #
#                                                          #
#                          H                               #
#                          B                               #
#                          B                               #
#                          B                               #
#                          B                               #
#                                                          #
#                                                          #
#                                                          #
#                                                          #
#                                                          #
#                                                          #
#                                                          #
#                                                          #
#                                                          #
#                                                          #
#                                                          #
#                                                          #
#                                                          #
#                                                          #
#                                                          #
#                                                          #
#                                                          #
#                                                          #
#                                                          #
#                                                          #
############################################################
```

2.2.1.4 Item 요소의 구현

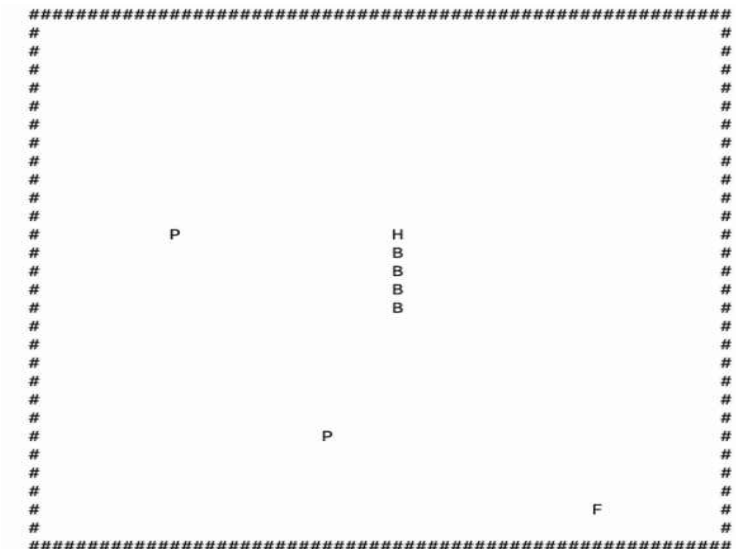
- 초기에 설정한 모든 목표를 구현하는 것이 본 절의 목표이며 모든 목표를 달성하였다.
아이템은 뱀 출현 이후 일정시간 뒤에 랜덤한 빈 공간에 생성된다. 아이템 생성은 고정된 DROP_ITEM_INTERVAL 마다 생성된다. 아이템이 생성될 수 있는 위치는 뱀의 몸이나 벽이 아닌 맵의 빈 공간(BLOCK_SPACE) 중 하나이다. 아이템은 랜덤한 위치에서 생성된다.

- 이때 각 아이템은 BaseObject를 상속받은 Item 클래스의 객체이다. 아이템은 아이템의 속성, 생성시간, 위치 정보를 멤버 변수로 가지고 있다. 이때 위치 정보를 하나의 클래스로 구현하여 다른 게임 요소들 역시 사용할 수 있도록 구현했다. 아이템의 만료 시간인 DROP_ITEM_EXPIRE이 지나면 아이템은 소멸된다. 아이템의 개수는 필수 요구사항에 따라 아이템 최대 개수인 MAX_ITEMS(3개)를 초과할 수 없다. 아이템이 소멸되면 아이템 최대 개수를 넘지 않는 선에서 다시 랜덤하게 생성된다.

- 아이템의 종류는 과일/독 두 종류뿐이다. 뱀이 과일 아이템을 먹을 경우 진행 방향으로 몸길이 1 증가하고, 독 아이템을 먹을 경우 꼬리부터 몸길이가 1 감소하도록 구현했다. 이때 독 아이템으로 인해 몸길이가 3보다 작아지면 실패하도록 구현했다. 전술하였듯이 Item은 각 아이템 하나에 대한 클래스이며 모든 아이템들을 관리하기 위한 또다른 클래스가 필요했다. 이 클래스를 ItemHandler라고 명명하였으며 이 아이템 핸들러는 멤버 변수로 아이템의 vector를 갖는다. 아이템 핸들러는 각 아이템을 벡터의 형태로 관리하며 GameState와 같은 외부 함수에서 직접 접근이 가능하도록 설계했다.

● 구현 결과

맵에 과일 아이템 F와 독 아이템 P 최대 개수인 3개까지 모두 생성된 그림



2.2.1.5 Gate 요소의 구현

- 초기에 설정한 모든 목표를 구현하는 것이 본 절의 목표이며 모든 목표를 달성하였다. 게이트는 뱀 출현 이후 일정 시간 뒤에 랜덤한 벽에 생성된다. 이때 벽의 모서리에서는 생성되지 않는다. 게이트는 두 개 가 한 쌍을 이루며, 한 쌍 이상 동시에 맵에 존재할 수 없다.

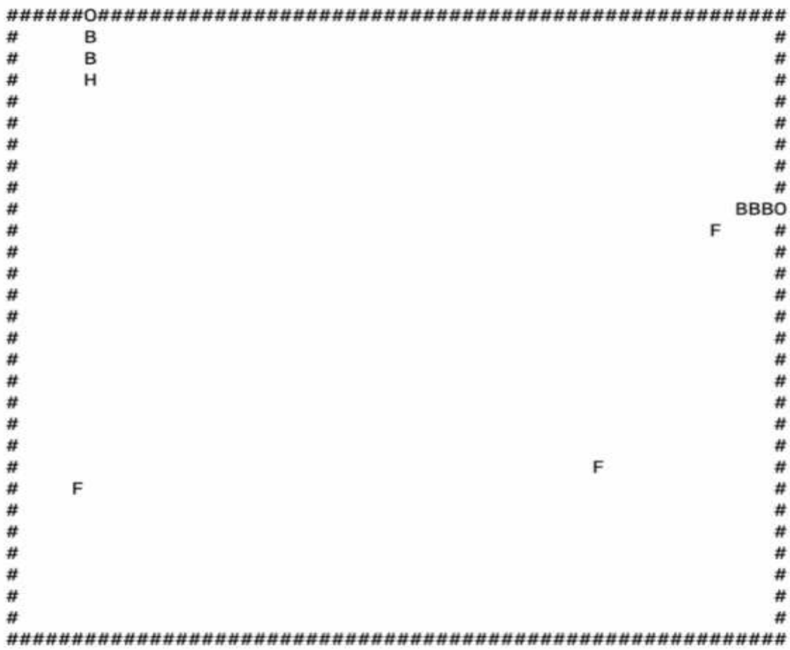
- 게이트는 아이템과 마찬가지로 만료시간인 DROP_ITEM_EXPIRE 시간이 지나면 소멸된다. 뱀이 한 게이트에 들어간 경우 다른 게이트에서 머리부터 빠져나와야 하며, 뱀이 게이트를 통과 중일 경우에는 게이트가 사라질 수 없다. 만약 뱀이 게이트를 통과 중일 때 게이트 만료 시간이 지난 경우 게이트가 소멸되지 않고 유지되도록 구현하였다. 또한 뱀의 꼬리가 완전히 통과한 이후에 게이트가 소멸될 수 있도록 구현하였다. 게이트를 통과한 뱀의 진행 방향은 필수 구현 사항인 규칙을 모두 준수할 수 있도록 설계했다.

- 게이트는 아이템과 마찬가지로 Gate, GateHandler의 형태로 구현하였다. Gate는 하나의 게이트를 의미하는 클래스이며 GateHandler는 이 게이트들을 관리한다. Item을 vector 형태로 관리하면 아이템 핸들러와 달리, 게이트는 최대 두 개만 동시에 존재할 수 있으므로 게이트 핸들러는 게이트 한 쌍만을 관리한다. 이때 게이트 핸들러의 멤버 변수를 게이트들의 벡터가 아닌 두 개의 게이트로 제한하면서 메모리/속도 관점에서 이점이 있다.

● 구현 결과

뱀이 우측의 게이트(O)로 들어가서 상단의 게이트에서 빠져나오는 모습.

게이트 페어에서 뱀의 머리부터 빠져나오는 모습

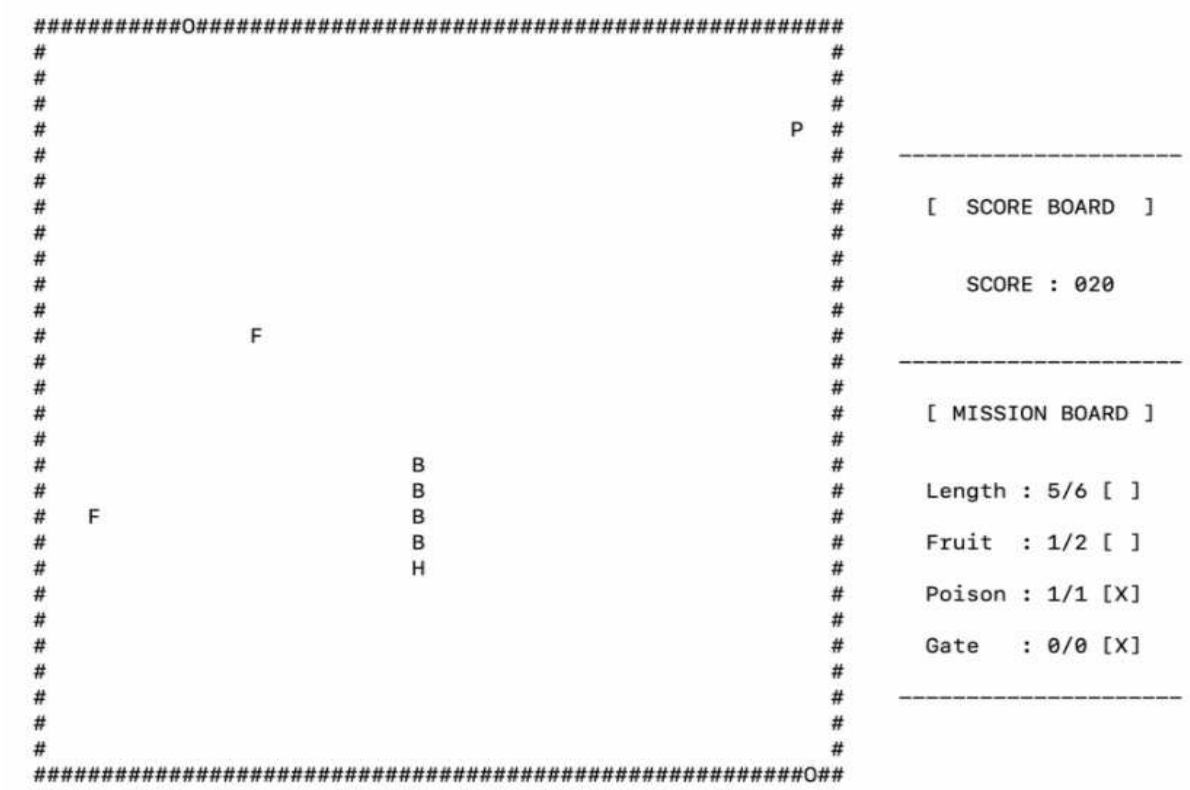


2.2.1.6 점수 요소의 구현

- 초기에 설정한 모든 목표를 구현하는 것이 본 절의 목표이며 모든 목표를 달성하였다. 게임 화면 우측에 스코어 보드와 미션 보드를 출력하였다. 스코어 보드는 현재 점수를 매 틱마다 출력하고 미션 보드는 미션 목표와 달성 여부를 틱마다 출력 한다. 스코어 보드의 스코어는 미션 달성 여부에 따라 부여되는 점수의 총점이다. 총점은 각 미션에 일정 weight를 곱하여 더한 값이다.

- 미션은 각 스테이지마다 난이도가 높아지도록 구현하였다. 각 미션은 뱀의 최대 길이, 과일 아이템 획득 횟수, 독 아이템 획득 횟수, 게이트 통과 횟수로 이루어져 있음.
이때 뱀의 최대 길이는 현재 몸길이가 아닌 해당 게임 중 달성한 뱀의 최고 길이이도록 필수 구현 사항을 충족한다. 모든 미션을 클리어한 경우 다음 스테이지로 이동할 수 있도록 구현하였다. 미션 보드에서는 각 미션의 충족 여부를 X 표시를 통해 표현한다.

● 구현 결과
게임 화면 우측에 스코어 보드와 미션 보드가 출력된 모습



2.2.1.7 추가 구현

● GameClearState 구현

게임을 클리어한 경우 GameClearState로 전환되도록 구현하였다.

게임 클리어 시 점수와 미션을 출력한다.

● 게임 오버 상태에서 스코어 표시

게임 오버가 된 경우 미션 달성 여부와 스코어를 모두 표시.

● 스테이지 선택 기능

게임 메뉴에서 1, 2, 3, 4 스테이지 중 원하는 스테이지부터 시작할 수 있도록 선택 기능 부여

● 머리가 몸에 닿으면 게임 실패

뱀의 머리가 뱀의 몸통에 닿은 경우 뱀이 죽도록 구현

● 게임 오버 이유 출력

게임 오버가 된 경우 게임 오버의 이유를 알려준다.
게임 오버가 가능한 경우는

1. 뱀의 머리가 벽에 닿은 경우
2. 뱀의 머리가 몸에 닿은 경우
3. 뱀의 길이가 3보다 작아진 경우
4. 뱀이 진행 방향의 반대 방향으로 이동하려 한 경우

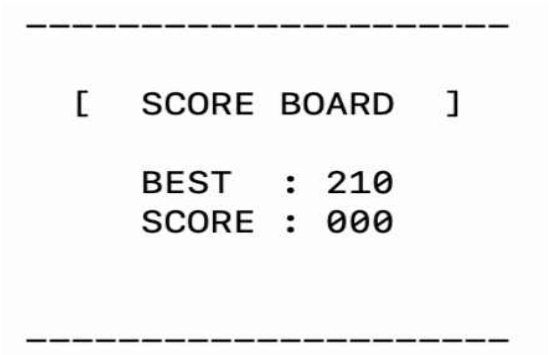
얘를 들어 뱀의 머리가 몸통에 닿아서 죽은 경우 아래와 같이 출력

● 최고 점수 시스템

스코어 보드에서 현재 점수 뿐 아니라 게임의 최고 점수를 동시에 출력

게임의 최고 점수를 달성한 경우 파일에 최고 점수를 기록

현재 게임 점수가 최고 기록을 갱신하고 있는 경우 매 틱마다 최고 점수 갱신



2.2.2 시스템 구조 및 설계도

작성요령 (30 점)

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 BaseObject 클래스

- 개발자 : 윤형준

- 소스파일 : BaseObject.h

- 게임에서 State가 아닌 대부분의 요소들이 상속받는 추상 클래스이다. 공통적으로 Update와 Render가 필요한 모든 요소를 위한 추상 클래스이다. 핸들러가 존재하는 Gate, Item 뿐만 아니라 게임의 주요 요소 중하나인 Snake를 포함한 대부분의 요소들이 상속받는다. 또한 게임의 요소가 아닌 GateHandler, ItemHandler, Map, Display, Mission와 같은 요소들도 BaseObject의 서브 클래스이다.

- method

access modifier	return	identifier	description
public		BaseObject( )	생성자
public		~BaseObject( )	소멸자
public	void	Update(float tic)	게이트, 아이템, 뱀을 맵에 새로 반영한다.
public	void	Render( )	게이트, 아이템, 뱀을 화면에 렌더링한다.

2.2.2.2 BaseState 클래스

- 개발자 : 윤형준

- 소스파일 : BaseState.h

- 게임의 모든 State들이 상속받는 추상 클래스. 공통적으로 Update와 Render가 필요한 모든 요소를 위한 추상 클래스. 게임 메뉴, 게임(게임 내부), 게임 오버, 게임 클리어가 이 클래스를 상속받는다. 모든 State들이 StateHandler에게 관리받기 쉽도록 구현하였다.

- method

access modifier	return	identifier	description
public		BaseState( )	생성자
public		~BaseState( )	소멸자

public	void	Update(float tic)	게임의 state들을 맵에 새로 반영한다.
public	void	Render( )	게임의 state들을 화면에 렌더링한다.

2.2.2.3 global 소스코드

- **개발자 : 박성영**

- **소스파일 : global.h**

- 게임에서 사용하는 모든 매크로와 위치 정보 클래스를 정의했다. 개발의 이점과 게임 최적화를 위해(예: string의 비교연산을 회피하고 int, char의 비교 연산으로 대체) 가능한 변수들을 매크로로 정의하여 사용하였다. 맵, 게임, 아이템과 같은 데이터 뿐 아니라 사용자 입력, 뱀의 방향, 아이템 만료 시간 등 개발 중 쉽게 수정해야하거나 여러 함수 내에서 자주 호출하는 변수를 정의하였다. 또한 게임에서 필요한 위치 정보를 쉽게 관리하기 위해 Position이라는 클래스를 만들었는데, 이 클래스 역시 global 헤더에 포함되어 있다.

- **method**

access modifier	return	identifier	description
public		Position(int x, int y)	아이템 게이트의 위치를 지정한다.

2.2.2.4 Main

- **개발자 : 윤형준**

- **소스파일: main.cpp**

- 게임 실행 시 가장 먼저 호출되는 함수. 게임 전체 실행을 담당하는 ExecuteDriver를 호출한다. 이때 ExecuteDriver는 StateHandler 내부에 구현되어 있다.

- **method**

access modifier	return	identifier	description
public		ExecuteDriver	게임 전체 실행을 담당하는 드라이버이다.

2.2.2.5 StateHandler 소스코드

- **개발자 : 박성영**

- **소스파일: StateHandler.h, StateHandler.cpp**

- 모든 State를 총괄하는 핸들러 클래스. 게임 전체의 실행을 담당하는 ExecuteDriver를 포함한다. 또한 초기 상태로 초기화하거나 게임 종료 시 필요한 메모리들을 정리한다. 게임 전체 실행에 필요한 틱을 계산하기도 한다. StateHandler가 관리하는 모든 State들을 BaseState를 상속받았다는 공통점이 있다. 덕분에 StateHandler는 현재 State가 무엇인지 고려하지 않고 Update와 Render만 반복하여 호출하기만 하면 된다는 이점이 있다.

- **method**

access modifier	return	identifier	description
public	void	ExecuteDriver( )	게임 전체 실행을 담당한다.
public	void	InitState( )	State를 초기화한다.
public	void	DelayState( )	게임 전역 tic만큼 지연한다.
public	void	ExitProcess( )	전체 프로세스를 종료한다.
public	void	Update(float tic)	각 State에 해당하는 Update를 호출한다.
public	void	Render( )	각 State들을 화면에 렌더링한다.
public	void	InitGlobalTimer( )	게임 전역 타이머 초기화한다.
public	float	GetTic( )	tic을 계산한다.
public	void	ChangeState(BaseState * state)	현재 State를 변경한다.

2.2.2.6 GameMenuState 클래스

- **개발자 : 박성영**

- **소스파일 : GameMenuState.h, GameMenuState.cpp**

- 게임을 실행하면 가장 먼저 보여지는 게임 메뉴 상태. 원하는 스테이지를 선택하여 실행할 수 있으며, 스테이지 선택 시 게임 State로 전환된다.

- **method**

access modifier	return	identifier	description
public		~GameMenuState(	소멸자
public	void	Update(float eTime)	게임 메뉴를 맵에 새로 반영한다.

public	void	Render( )	게임 메뉴를 화면에 렌더링한다.
public	void	Load( )	GameMenuState 데이터를 파일에서 읽어온다.
public	int	GetUserInput( )	사용자로부터 입력을 받는다.
public	int	SelectMenu( )	메뉴 선책 메시지를 출력한다.

2.2.2.7 GameOverState 클래스

- 개발자 : 윤형준

- 소스파일 : GameOverState.h, GameOverState.cpp

- 게임이 종료되면 전환되는 State. 구현 6단계인 추가 구현 상태를 반영하여 게임 오버 이유를 항상 출력한다. 게임을 다시 시도할지 물어본 뒤 y/n 입력에 따라 게임을 다시 실행한다.

- method

access modifier	return	identifier	description
public		~GameOverState(	소멸자
public	void	Update(float tic)	게임 종료를 맵에 새로 반영한다.
public	void	Render( )	게임 종료를 화면에 렌더링한다.
public	char	UserInput( )	사용자의 확인을 받는다.
public	char	AskUserToPlayAgain( )	다시 게임시작할건지에 대한 여부 출력한다.
public	void	Load( )	GameOverState 데이터를 파일에서 읽어온다.

2.2.2.8 GameClearState 클래스

- 개발자 : 윤형준

- 소스파일 : GameClearState.h, GameClearState.cpp

- 게임을 클리어하면 전환되는 State. 구현 6단계인 추가 구현 상태를 반영하여 추가하였다. 게임의 최고 점수와 현재 점수를 출력해준다.

- method

access modifier	return	identifier	description
public		GameClearState( )	생성자
public		~GameCelarState( )	메소드

public	void	Update(float tic)	게임 종료 다음 상태를 맵에 새로 반영한다.
public	void	Render( )	게임 클리어 장면을 화면에 렌더링한다.
public	void	Load( )	GameClearState 데이터를 파일에서 읽어온다.
public	char	UserInput( )	사용자의 확인을 받는다.

2.2.2.9 GameState 클래스

- 개발자 : 윤형준

- 소스파일 : GameState.h, GameState.cpp

- 게임 실행 내부 State. 게임이 실행되는 도중 일어나는 모든 상태를 관리한다. 모든 게임 요소(맵, 뱀, 아이템 핸들러, 게이트 핸들러, 미션, 디스플레이)를 모두 관리한다. 모든 요소가 공통적으로 BaseObject를 상속 받았으므로 간단히 Update 함수를 호출하는 것으로 게임을 진행할 수 있다.

- method

access modifier	return	identifier	description
public		~GameState( )	소멸자
public	void	Update(float tic)	게임 화면을 맵에 새로 반영한다.
public	void	Render( )	게임 화면을 화면에 렌더링한다.
public	void	InitGameStateWindow( )	게임 화면 초기화한다.
public	void	ActiveObject( )	아이템, 뱀의 길이, 게이트의 블록을 체크한다.

2.2.2.10 Snake 클래스

- 개발자 : 박성영

- 소스파일 : Snake.h, Snake.cpp

- 뱀의 상태를 관리하는 클래스. 다른 게임 요소들과 마찬가지로 BaseObject를 상속받는다. 뱀의 머리와 몸을 포함한 전체 데이터를 포지션의 벡터로 관리한다. 또한 뱀의 상태를 멤버변수로 가지고 있는데, 방향이나 성장/축소 여부 등을 포함한다. 특히 목표의 2단계 구현 사항을 모두 포함한다.

- method

access modifier	return	identifier	description
public		Snake( )	생성자
public		~Snake( )	소멸자
public	void	Update(float eTime)	snake 상태를 업데이트
public	void	PushData( )	뱀에 데이터를 전달한다.
public	void	SetDirection(int a)	뱀의 방향을 저장한다.
public	void	SetDirectionWithCheck (int a, int b)	뱀의 진행 반대 방향으로 인한 뱀의 상태를 저장한다.
public	bool	IsCollision( )	뱀이 충돌여부를 확인
public	void	SetGrow(bool flag)	뱀의 꼬리가 자라나는 여부 확인
public	void	Shrink( )	독을 먹었을 때 꼬리를 없앤다.
public	CharPosition	GetHead( )	뱀 머리의 현재 위치 반환
public	CharPosition	GetTail( )	뱀 꼬리의 위치 반환
public	void	SetHeadPos(int y, int x)	뱀 머리의 현재 위치를 저장한다.
public	void	SpawnSnake( )	처음 게임 시작할 때 뱀을 소환한다.
public	void	CutTail( )	뱀의 꼬리가 잘렸을때의 상태를 저장한다.
public	void	EatItem(bool fruit, bool poison)	아이템 먹은 여부에 대해 저장한다.

2.2.2.11 Item 클래스

- 개발자 : 윤형준

- 소스파일 : Item.h, Item.cpp

- 아이템 객체 하나를 위한 클래스. 다른 게임 요소들과 마찬가지로 BaseObject를 상속받는다. 각 아이템은 필요한 위치 정보와 생성 시각, 종류를 멤버 변수로 갖는다. 아이템은 위치 정보만을 가지고 있으며 게이트에 대한 총괄은 아이템 핸들러가 담당한다.

- method

access modifier	return	identifier	description
public		Item( )	생성자

public		~Item( )	소멸자
public	void	Update(float tic)	item을 맵에 새로 반영한다
public	void	Render( )	item을 화면에 렌더링한다.
public	Position	GetPosition( )	아이템이 떨어지는 위치를 반환한다.
public	bool	ComparePosition(int x, int y)	아이템의 좌표를 표시한다.
public	char	GetType( )	아이템의 종류를 반환한다.
public	float	GetDropType( )	아이템이 나타나는 시간을 반환한다.
public	void	SetPosition(Position position)	아이템이 떨어지는 위치를 저장한다.
public	void	SetType(char type)	아이템의 종류를 저장한다.
public	void	SetDropType(float drop_time)	아이템이 나타는 시간을 저장한다.

2.2.2.12 ItemHandler 클래스

- 개발자 : 윤형준

- 소스파일 : ItemHandler.h, ItemHandler.cpp

- 모든 아이템들을 관리하는 핸들러 클래스. 다른 게임 요소들과 마찬가지로 BaseObject를 상속받는다. 아이템은 필수 구현 요구사항인 아이템 최대 개수(3개)를 초과할 수 없으므로 아이템 벡터를 이용해 관리한다. 또한 아이템 벡터와 마지막으로 생성한 아이템의 생성 시간을 멤버 변수로 갖는다. 아이템은 최대 3개 밖에 동시에 존재할 수 없으므로 벡터로 관리하더라도 성능상 큰 문제가 없을 것이라고 생각했다. 아이템 생성을 위한 랜덤한 위치 생성, 아이템 생성 및 삭제, 아이템 만료시간 등을 모두 관리한다.

- method

access modifier	return	identifier	description
public		ItemHandler( )	생성자
public		~ItemHandler( )	소멸자
public	void	Update(float tic)	item_list들의 상태를 업데이트한다.
public	void	Render( )	아이템을 화면에 렌더링한다.
public	Position	GetRandomPosition( )	아이템을 생성할 랜덤한 공간을 찾는다.
public	void	AddItem(char item_type, float tic)	item_list에 아이템을 추가한다.

public	void	DeleteExpiredItems(float tic)	시간이 만료된 아이템을 맵과 item_list에서 삭제한다.
public	void	DeleteBlock(int y, int x)	아이템을 item_list에서 삭제한다.
public	void	ApplyBlock( )	아이템을 맵의 블록에 적용한다.

2.2.2.13 Gate 소스코드

- 개발자 : 박성영

- 소스파일 : Gate.h, Gate.cpp

- 게이트 객체 하나를 위한 클래스. 다른 게임 요소들과 마찬가지로 BaseObject를 상속받는다. 게이트는 위치 정보만을 가지고 있으며 게이트에 대한 총괄은 게이트 핸들러가 담당한다.

- method

access modifier	return	identifier	description
public		Gate( )	생성자
public		~Gate( )	소멸자
public	void	Update(float tic)	게이트 생성 시간을 업데이트한다.
public	void	Render( )	게이트를 화면에 렌더링한다.
public	Positon	GetPosition( )	게이트의 위치를 반환한다.
public	void	SetPosition(Position position)	게이트 생성 위치를 저장한다.
public	void	SetPosition(int x, int y)	게이트 생성 좌표를 저장한다,
public	bool	ComparePosition(int x, int y)	게이트 페어의 위치를 확인한다.

2.2.2.14 GateHandler 클래스

- 개발자 : 박성영

- 소스파일 : GameHandler.h GameHandler.cpp

- 게이트 쌍을 관리하는 핸들러 클래스. 다른 게임 요소들과 마찬가지로 BaseObject를 상속받는다. 아이템 핸들러와 마찬가지로 게이트 객체를 관리한다. 이때 게이트는 동시에 두 개밖에 존재할 수 없으므로 벡터로 관리할 필요가 없다. 따라서 멤버 변수로 두 개의 게이트만들 갖는다. 게이트 생성에 필요한 랜덤한 위치 정보와 게이트 생성/삭제, 만료 시간을 담당한다. 또한 한 게이

트에서 다른 게이트로 통과하는 기능을 위해 게이트 짝을 찾는 함수도 포함한다.

- method

access modifier	return	identifier	description
public		~GateHandler( )	소멸자
public	void	Update(float tic)	게이트 페어전체를 맵에 새로 반영한다.
public	void	Render( )	게이트를 화면에 렌더링한다.
public	void	GetRandomPosition( )	게이트를 생성할 랜덤한 벽을 찾는다.
public	Position	GetGatePair( )	게이트 짝을 찾아서 반환한다.
public	Gate	SpawnGate(float tic)	게이트 페어를 생성한다.
public	void	DeleteGate( )	게이트 페어를 삭제한다.
public	void	ApplyBlock( )	수동으로 맵 데이터에 적용한다.
public	void	IsExitGate( )	뱀 꼬리가 게이트를 빠져나오는지 확인한다.

2.2.2.15 Map 클래스

- 개발자 : 윤형준

- 소스파일 : Map.h, Map.cpp

모든 스테이지의 맵을 담당하는 클래스. 다른 게임 요소들과 마찬가지로 BaseObject를 상속받는다. 맵의 각 위치에 접근하여 블럭을 설정하거나 삭제하는 메소드를 담당한다.

- method

access modifier	return	identifier	description
public		Map( )	생성자
public		~Map( )	소멸자
public	void	Render( )	map을 화면에 렌더링한다.
public	void	Update(float tic)	map의 상태를 업데이트한다.
public	void	Load ( )	map 데이터를 파일에서 읽어온다.
public	void	SetBlock(int y, int x,	맵의 배경의 좌표를 저장한다.

		char e)	
public	void	SetBlock(Position position, char e)	맵의 블럭의 위치를 저장한다.
public	char	GetBlock(int x, int y)	맵의 블럭의 위치를 반환한다.
public	void	DeleteBlock(int y, int x)	맵의 블럭을 삭제한다.

2.2.2.16 Mission 클래스

- 개발자 : 박성영

- 소스파일 : Mission.h, Mission.cpp

- 각 스테이지의 미션을 담당하는 클래스. 다른 게임 요소들과 마찬가지로 BaseObject를 상속받는다. 각 점수들을 관리하며 현재 점수 및 최고 점수를 관리한다.

- method

access modifier	return	identifier	description
public		Mission( )	생성자
public		~Mission( )	소멸자
public	void	Update( )	뱀의 길이와 점수를 업데이트한다.
public	void	Render( )	뱀의 길이와 총점수를 화면에 렌더링한다.
public	void	CalcTotalScore(int stage)	총 점수를 계산한다.
public	bool	IsMissionClear(int stage)	미션을 클리어했는지 확인한다.
public	int	GetLengthScore(int stage)	뱀의 길이 점수를 반환한다.
public	int	GetFruitScore( )	과일 아이템 점수를 반환한다.
public	int	GetPositionScore( )	독 아이템 점수를 반환한다.
public	int	GetGateScore( )	게이트 점수를 반환한다.
public	int	GetTotalScore( )	총 점수를 반환한다.
public	void	SetLengthScore(int score)	뱀의 길이 점수를 저장한다.
public	void	SetFruitScore(int	과일 아이템 점수를 저장한다.

		score)	
public	void	SetPositionScore(int score)	독 아이템 점수를 반환한다.
public	void	SetGateScore(int score)	게이트 점수를 저장한다.

2.2.2.17 Stage 클래스

- 개발자 : 윤형준

- 소스파일 : Stage.h

- 각 스테이지를 담당하는 클래스. 다른 게임 요소들과 마찬가지로 BaseObject를 상속받는다. 현재 스테이지를 설정하거나 가져온다.

- method

access modifier	return	identifier	description
public		Stage( )	생성자
public		~Stage( )	소멸자
public	int	GetCurrentStage( )	현재 스테이지를 반환한다.
public	void	SetNowStage(int stage)	현재 스테이지를 저장한다.

2.2.2.18 Display 클래스

- 개발자 : 윤형준

- 소스파일: Display.h, Display.cpp

- 게임의 스코어 보드나 미션 보드의 출력을 담당하는 클래스. 다른 게임 요소들과 마찬가지로 BaseObject를 상속받는다. 게임 State 내부에서 주로 사용된다. 또한 모든 State에서 추가로 출력하고자 하는 메시지가 있는 경우 DisplayMessage 함수를 통해 해당 메시지를 추가적으로 출력할 수 있도록 구현하였다.

- method

access modifier	return	identifier	description
public		~Display( )	소멸자
public	void	Update(float tic)	점수, 미션, 특정 메시지를 맵에 새로 반영한다.
public	void	Render( )	점수와 미션을 화면에 렌더링한다.

public	void	DisplayScore( )	점수를 화면에 표시한다.
public	void	DisplayMission( )	미션을 화면에 표시한다.
public	void	DisPlayMessage(char * msg, int height, int width)	특정 메시지를 화면에 출력한다.
public	void	MissionClear(int score, int goal)	각 미션별 클리어 여부를 반환한다.

2.2.3 활용/개발된 기술

작성요령 (10 점)

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.

▼ 추상 클래스

- 전술하였듯이 게임 요소를 크게 State와 Object로 분리하였으며, 필요한 경우 핸들러를 이용하도록 하였다. 특히 모든 State는 BaseState를, 모든 Object는 BaseObject를 상속 받도록 구현되었다. 덕분에 실제 핸들러들은 Update나 Render 함수만 호출하는 방법으로 게임을 동작시킬 수 있다.

▼ 핸들러

- 핸들러가 필요한 경우 오브젝트들을 관리하는 핸들러를 구현하였다. 특히 모든 State들을 관리하는 StateHandler가 게임 전체를 관리한다. 게임 State 내부에서는 Item, Gate들을 관리하는 ItemHandler, GateHandler를 구현하여 관리하도록 했다.

▼ ncurses

- ncurses는 GNU에서 개발한 new curses 라이브러리이며, 1990년대 중반에 curses 라이브러리의 개발이 중단된 후 개발된 라이브러리이다. 이때, curses는 Cursor Optimization에서 유래했으며, 이를재미있게 발음한 것을 말한다. curses 라이브러리는 유닉스 계열 운영체제를 위한 제어 라이브러리 중하나이고 매우 유연하고 효율적인 Application Programing Interface를 제공한다. 또한, 커

서를 움직이거나, 윈도우를 생성하고, 색깔을 표시하거나, 마우스 관련 함수를 제공하는 등의 TUI 응용프로그램들의 구성을 가능하게 하는 라이브러리이다.
프로세스의 모든 프론트엔드는 ncurses의 도움을 받았다. 특히 Display 클래스의 메소드는 ncurses와 다른 함수를 혼합하여 사용한 경우가 많다.

▼ vector

- STL 라이브러리 중 vector를 사용하였다. 특히 뱀을 구현할 때 벡터가 가장 먼저 필요했는데, 연결 리스트 등의 컨테이너와 달리 어떤 원소에도 직접 접근이 가능해야하며, 머리나 꼬리 등에 자유롭게 접근이 가능해서 벡터가 필요했다. 또한 삭제, 추가가 용이하다는 점이 벡터의 가장 큰 이점이다.

뱀 뿐 아니라 아이템 핸들러의 멤버 변수도 벡터로 구현했는데, int와 같은 기본적인 변수가 아닌 Item, Position 등 그 어떤 변수 등 벡터로 구현할 수 있었기 때문에 벡터를 사용했다.

▼ 파일 입출력

- 맵 데이터, 스테이트, 최고 점수를 모두 파일에 저장하고 읽어온다. 처음에는 맵 데이터를 램에 올려서 사용했는데, 이는 속도 관점에서 이점이 있지만 반대로 메모리 사용량 관점에서 불필요한 요소이므로 파일을 사용했다.

또한 최고 점수는 게임 프로세스의 실행/종료 여부와 상관없이 최고 점수가 유지되어야 하며, 현재 점수가 최고 점수를 넘었을 경우 최고 점수를 동기화해야 한다. 따라서 파일에 점수를 기록하고 관리할 수 있도록 구현했다.

▼ 매크로

- 처음 작성한 코드에서는 매크로를 전혀 사용하지 않았다. 따라서 모든 구현 요구 사항을 구현한 뒤에, 추가 구현 사항을 구현할 때 많은 어려움이 있었다. 초기 목표를 모두 달성한 뒤에 수정이 어렵다는 것을 느낀 뒤에 가능한 변수들을 매크로로 정의하여 관리하였다.

2.2.4 현실적 제한 요소 및 그 해결 방안

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

○ 최적화

- 매 틱마다 백엔드에서 게임 요소 관련 정보를 업데이트한 뒤 프론트 엔드에서 이를 출력해여 한다. 따라서 속도/메모리 측면에서 가능한 모든 최적화를 수행할 필요가 있었다.

먼저 string 비교 연산 등을 피하기 위해 게임 요소를 매크로로 정의하여 int/char와 같은 변수로 변환했다. 또한 각 요소의 Update 함수에서 조건문을 통해 불필요한 경우엔 특정 basic block으로 jmp하지 않도록 구현했다. 또한 게이트 핸들러의 맴버 변수를 게이트 벡터에서 게이트 두 개로 제한하는 등 불필요한 메모리 사용이 없도록 수정했다.

2.2.5 결과물 목록

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

파일명	역할
StateHandler.cpp	StateHandler.h에서 정의한 메소드를 구현하는 소스코드
StateHandler.h	게임 메뉴나 게임이 끝났을 때의 상황 등 state들을 관리하는데 필요한 것을 정의한 소스코드
Display.cpp	Display.h에서 정의한 메소드를 구현하는 소스코드
Display.h	점수나 미션을 화면에 표시하는데 필요한 것을 정의한 소스코드
GameClearState.cpp	GameClearState.h에서 정의한 메소드를 구현하는 소스코드
GameClearState.h	게임을 모두 클리어 했을 때 화면을 표시하는데 필요한 것을 정의한 소스코드
GameMenuState.cpp	GameMenuState.h에서 정의한 메소드를 구현하는 소스코드
GameMenuState.h	게임 level을 선택하여 게임을 실행하는데 필요한 것을 정의한 소스코드
GameOverState.cpp	GameOVerState.h에서 정의한 메소드를 구현하는 소스코드
GameOVerState.h	게임이 끝났을 때의 상태를 다루기 위해 필요한 것을 정의한 소스코드
GameState.cpp	GameState.h에서 정의한 메소드를 구현하는 소스코드
GameState.h	전반적인 게임을 관리하기 위해 필요한 것을 정의한 소스코드
Gate.cpp	Gate.h에서 정의한 메소드를 구현하는 소스코드
Gate.h	Gate의 좌표를 표시하기 위해 필요한 것을 정의해놓은 소스코드
global.h	게임 위치나 아이템의 좌표를 표시하고 매크로를 사용하는데 필요한 것을 정의해놓은 소스코드
BaseObject.h	게임 내 아이템, 게이트, 뱀의 기능들을 계속해서 업데이트하는데 필요한 것을 정의해놓은 소스코드
BaseState.h	게임이 끝났을 때 와 메뉴를 선택할 때 필요한 기능들을 정의해놓은 소스코드

Item.cpp	Item.h에서 정의한 메소드를 구현하는 소스코드
Item.h	아이템의 위치, 타입, 생성 시기를 저장하기 위해 필요한 것을 정의한 소스코드
ItemHandler.cpp	ItemManager.h에서 정의한 메소드를 구현하는 소스코드
ItemHandler.h	아이템의 생성과 삭제를 다루기 위해 필요한 것을 정의한 소스코드
GateHandler.cpp	GateHandler.h에서 정의한 메소드를 구현하는 소스코드
GateHandler.h	게이트의 생성, 삭제를 관리하기 위해 필요한 것을 정의한 소스코드
main.cpp	게임 전체 실행을 하기 위해 필요한 것을 정의한 소스코드
Map.cpp	Map,h에서 정의한 메소드를 구현하는 소스코드
Map.h	게임의 맵의 좌표를 표시하기 위해 필요한 것을 정의해놓은 소스코드
Mission.h	뱀의 최대길이, 과일, 독, gate등 게임 내 mission을 관리하는데 필요한 것을 정의한 소스코드
Mission.cpp	Mission.h에서 정의한 메소드를 구현하는 소스코드
Snake.h	뱀의 움직임, 뱀과 아이템, 게이트와의 상호 작용을 구현하기 위해 필요한 것을 정의해놓은 소스코드
Snake.cpp	Snake.h에서 정의한 메소드를 구현하는 소스코드
Stage.cpp	Stage.h에서 정의한 메소드를 구현하는 소스코드
Stage.h	현재 게임의 단계를 불러오는 소스코드

3 자기평가

작성요령 (5 점)

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.

윤형준	프로젝트 초기에 팀원과 공동으로 프로젝트 전체를 설계하였으며, OOP의 특징이 프로젝트에 잘 녹아들 수 있도록 노력하였다. 게임을 스테이트로 구분하였고 스테이트와 게임 요소들이 상속받을 수 있는 추상 클래스를 설계하였다. 전체 게임 스테이트를 담당해 설계하였으며, 필수 구현 사항 중 맵, 아이템, 미션(디스플레이)를 구현하였다. 추가 구현 사항 중 게임 오버 이유 출력, 최고 점수 출력, 게임 클리어 스테이트 출력을 담당하였다.
박성영	프로젝트 초기에 팀원과 공동으로 프로젝트 전체를 설계하였으며, 매크로 사용, 함수 호출 횟수 줄이기 등 게임의 최적화를 담당하였다. 전체 게임이 스테이트 단위로 동작할 수 있도록 스테이트 핸들러를 담당해 설계하였으며, 필수 구현 사항 중 뱀, 게이트, 미션(점수)를 구현하였다. 추가 구현 사항 중 새로운 규칙, 게임오버 시 점수 출력, 스테이지 선택 기능을 담당하였다.

4 참고 문헌

번호	종류	제목	출처	발행년도	저자	기타
1	책	visual C++ 6 완벽가이드	p157 ~ 170	2004	김용성	
2	웹사이트	vector – C++ Reference	https://www.cplusplus.com/reference/vector/vector/			
3	웹사이트	Input/output with files - C++	https://www.cplusplus.com/doc/tutorial/files/			

5 부록

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

5.1 사용자 매뉴얼

5.1.1 게임 규칙

- 사용자는 뱀을 상하좌우로 움직일 수 있으며, 벽에 닿으면 게임이 끝난다.
- 게이트로 들어가면 반대 게이트로 나올 수 있다.
- 벽이 아닌 곳에 아이템이 생성되며, 먹으면 뱀의 길이가 변한다.

5.1.2 뱀의 이동

- 뱀은 상하좌우로 움직일 수 있으며, 아래와 같이 방향키로 움직일 수 있다.
  - ⇧ : 상
  - ⇩ : 하
  - ⇦ : 좌
  - ⇨ : 우
- 뱀의 머리가 자신의 몸이나 벽으로 이동할 수 없다.

5.1.3 아이템

- Item은 Fruit Item 과 Poison Item으로 2 종류가 존재한다.
- F : Fruit Item
  - 이 아이템을 먹으면 뱀의 몸길이가 1길어진다.
  - 이 아이템 생성된 시점에서 18틱이후 없어진다.
  - 이 아이템이 없어진 시점에서 6틱이후 새로운 아이템이 생성된다.

- P : Poison Item
  - 이 아이템을 먹으면 뱀의 몸길이가 1 짧아진다.
  - 몸의 길이가 3보다 작아지면 gameover 된다.
  - 이 아이템 생성된 시점에서 18틱이후 없어진다.
  - 이 아이템 없어진 시점에서 6틱이후 새로운 아이템이 생성된다.

5.1.4 게이트

- 0 : 게이트
  - 게이트 입구로 들어가면 다른 게이트로 나온다.

- 게이트는 14틱이후 다시 생성된다.
- 같은 벽에 2개의 게이트는 생성되지 않는다.

5.2 설치 방법

1. g++ 컴파일러 설치/업그레이드(g++ 버전은 7.x 버전을 사용한다. ex. 7.5.0)

```
$ sudo apt-get install -y software-properties-common
$ sudo add-apt-repository ppa:ubuntu-toolchain-r/test
$ sudo apt update
$ sudo apt install g++-7 -y
$ sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-7 60 ₩
                        --slave /usr/bin/g++ g++ /usr/bin/g++-7
$ sudo update-alternatives --config gcc
```

2. 게임소스 코드 파일을 다운로드 한다.

- 제출한 zip 파일 압축을 해제한다.

3. make 명령어로 컴파일 한다.

```
$ make
```

4. 실행파일(게임)을 실행한다.

```
$ ./a.out
```