




국민대학교  
소프트웨어융합대  
학  
소프트웨어학부

# C++프로그래밍 프로젝트

프로젝트 명	Snake-Game
팀 명	팀1-6
문서 제목	결과보고서

Version	1.1
Date	2020-JuN-26

팀원	박 건후 (팀장)
	민 대인

 <b>국민대학교 컴퓨터공학부 Snake-Game I</b>	<b>결과보고서</b>		
	<b>프로젝트 명</b>	Snake-Game	
	<b>팀 명</b>	팀1-6	
	<b>문서 정보</b>	Version 1.1	2020-JUN-26


#### 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


<b>Filename</b>	최종보고서-snake-game.doc
<b>원안작성자</b>	박건후, 민대인
<b>수정작업자</b>	박건후, 민대인

수정날짜	대표수정자	Revision	추가/수정 항목	내 용
2020-06-15	민대인	1.0	최초 작성	
2020-06-26	민대인	1.1	추가 작성	issue에 올라온 건후가 쓴 보고서 내용, 5단계, 6단계 내용, 자기평가, 결과물 목록, 사용자 메뉴얼

 <b>국민대학교</b> <b>컴퓨터공학부</b> <b>Snake-Game I</b>	<b>결과보고서</b>		
	<b>프로젝트 명</b>	Snake-Game	
	<b>팀 명</b>	팀1-6	
	<b>문서 정보</b>	Version 1.1	2020-JUN-26

## 목 차

1	개요	4
2	개발 내용 및 결과물	7
2.1	목표	7
2.2	개발 내용 및 결과물	9
2.2.1	개발 내용	9
2.2.2	시스템 구조 및 설계도	11
2.2.3	활용/개발된 기술	23
2.2.4	현실적 제한 요소 및 그 해결 방안	23
2.2.5	결과물 목록	25
3	자기평가	26
4	참고 문헌	31
5	부록	32
5.1	사용자 매뉴얼	32
5.2	설치 방법	33

 <b>국민대학교 컴퓨터공학부 Snake-Game I</b>	결과보고서		
	프로젝트 명	Snake-Game	
	팀 명	팀1-6	
문서 정보	Version 1.1		2020-JUN-26

## 1 개요

이번 프로젝트에 사용한 외부 라이브러리는 ncurses 한 가지입니다. 개발환경인 리눅스 우분투를 기준으로 획득/설치 방법은 `sudo apt-get install libncurses5-dev libncursesw5-dev` 입력하여 설치했습니다. 그 후 코드 내에서 `#include <ncurses.h>`를 기입하여 개발했습니다. 또한 make 유틸리티를 이용하여 컴파일을 다음과 같이 진행하고 실행합니다.

all:

```
g++ src/GameScene.cpp src/GameOverScene.cpp src/Snake.cpp src/Stage.cpp src/myFunction.cpp
src/ItemManager.cpp src/WaitingScene.cpp src/Item.cpp src/main.cpp -lncurses -o /tmp/a.out && /tmp/a.out
```

개발 방법으로는, git flow 방식과 github template을 이용하여 Pull Request 를 하였습니다. 저희팀의 협업방식은 feature/기능 이란 브랜치를 만들어 commit을 원격 브랜치로 푸시하고, Pull Request를 했습니다. Pull Request를 할 때, 자신이 작업한 브랜치에서는 어떤 점이 개선되었고, 버그가 있다면 어떤 버그가 있는지 간략하게 요약해 첨부하였습니다.

저희팀의 snake-game 구조는 Scene으로 구성되어 있습니다. 현재 사용자가 어떤 Scene에 있는지에 따라 다른 화면을 렌더링하는 구조를 택했습니다. 예를 들어 프로그램에서 대기 화면을 출력하고자 했을 때, 화면이 Update 되는 부분, Render 하는 부분이 반복될 것입니다. Update 부분은 사용자의 입력을 받거나, Scene이 바뀌어야 하는 조건들이 있습니다. Render 부분은 화면에 출력해야 하는 오브젝트를 출력하는 기능을 담고 있습니다. 따라서 myFunction.cpp 부분은 myFunction.h 를 include 하였습니다. 그리고 이것을 main.cpp에 include를 함으로써 현재의 Scene을 Update와 Rendering 합니다. Scene이 바뀌려면 myFunction.h에 있는 ChangeScene이라는 기능을 이용하여 이전의 Scene을 delete하고, nowScene에 바꾸려고 하는 Scene으로 초기화 해줍니다.

### myFunction.cpp

```
#include "myFunction.h"
#include "WaitingScene.h"

IScene *nowScene;
bool lkey[256], rkey[256];


int currentWidth;
int currentHeight;

std::chrono::steady_clock::time_point startTime;

void Init(){
    startTime = std::chrono::steady_clock::now();
    nowScene = new WaitingScene();
}

void Update(float eTime){
    nowScene->Update(eTime);
}

void Render(){
    nowScene->Render();
}
```

 국민대학교 컴퓨터공학부 Snake-Game I	결과보고서		
	프로젝트 명	Snake-Game	
	팀 명	팀1-6	
	문서 정보	Version 1.1	2020-JUN-26

```

void Destroy(){
    delete nowScene;
}

float GetElapsedTime(){
    auto endTime = std::chrono::steady_clock::now();
    std::chrono::duration<float> elapsed_seconds = endTime-startTime;
    float eTime = (float)elapsed_seconds.count();
    return eTime;
}

```

```

void ChangeScene(IScene *p,bool nowSceneErase){
    if(nowSceneErase) delete nowScene;
    nowScene = p;
}

```

myFunction.cpp 에서 GetElapsedTime() 함수는 프로그램이 실행된 시각에서 현재의 시각을 빼준 값입니다. 그 값을 실수로 처리하여 21.21312 등의 형식으로 main에서 계산되어 모든 Scene과 Object에 Update 매개변수로 넣어줍니다. 이것은 아이템을 생성할 때, 생성 후 시간이 몇 초간 지났는지 알기 위해서 만들었습니다.

### IScene.cpp

```

#pragma once
#include "Stage.h"
class IScene
{
public:
    IScene() {}
    virtual ~IScene() {}
    virtual void Render() = 0;
    virtual void Update(float eTime) = 0;
};

```

### IObjct.cpp

```

#pragma once
class IObjct
{
public:
    IObjct() {}
    virtual ~IObjct() {}
    virtual void Update(float eTime) = 0;
    virtual void Render() = 0;
};


```

### main.cpp

```

int main()
{
    Init();
    do
    {

```

 <b>국민대학교 컴퓨터공학부 Snake-Game I</b>	<b>결과보고서</b>		
	<b>프로젝트 명</b>	Snake-Game	
	<b>팀 명</b>	팀1-6	
	<b>문서 정보</b>	Version 1.1	2020-JUN-26

```

        Update(GetElapsedTime());
        Render();
    } while (true);
    Destroy();
    return 0;
}

```

게임에서의 핵심 Object는 Player(사용자), Snake(뱀 캐릭터) 입니다. Player는 사용자의 점수를 관리하기 위해 사용됩니다. Snake는 뱀이 움직일 때마다 각 요소(몸통, 꼬리, 머리)의 데이터를 저장하기 위해 사용됩니다. 그래서 이 오브젝트를 GameScene 이라고 하는 파일에서 관리합니다. GameScene에서 뱀이 죽거나, 시간이 초과되거나 등의 이벤트가 발생한다면 ChangeScene을 통해 Scene을 바꿔줍니다.

게임에서 Snake를 추상화하면, (몸통, 꼬리, 머리)라는 데이터를 저장해야하는 vector와 독 아이템을 먹었을 때 몸집이 줄어드는 Shrink 함수, 과일을 먹었을 때 몸집이 커지는 Grow 함수, 일정시간이 지나면 움직이는 Update 함수, Snake의 데이터를 출력하는 Render 함수로 추상화할 수 있습니다.


## 2 개발 내용 및 결과물

### 2.1 목표

적용단계	내용	적용 여부
1단계	Map의 구현	적용/미적용
2단계	Snake 표현 및 조작	적용/미적용
3단계	Item 요소의 구현	적용/미적용
4단계	Gate 요소의 구현	적용/미적용
5단계	점수 요소의 구현	적용/미적용

#### 1단계. Map의 구현

- Stage를 3단계로 구분하고 Stage 별로 형태가 다른 Map을 출력합니다. Map의 세로길이는 32, 가로길이는 62으로 설정합니다. MapManager의 char data 배열의 값 중 0은 뱀이 움직일 수 있는 빈칸, 1은 Snake의 머리와 충돌할 경우의 죽는 벽, 2는 gate가 생성될 수 없는 벽, 3은 Snake의 머리, 4는 Snake의 몸통, 5는 Snake의 크기가 늘어나는 아이템, 6은 Snake의 크기가 줄어드는 아이템, 7은 Gate, 8은 빈칸입니다.

 <b>국민대학교 컴퓨터공학부 Snake-Game I</b>	<b>결과보고서</b>		
	<b>프로젝트 명</b>	Snake-Game	
	<b>팀 명</b>	팀1-6	
	<b>문서 정보</b>	Version 1.1	2020-JUN-26

## 2단계. Snake 표현 및 조작

- Snake의 y좌표, x좌표를 묶어 vector에 저장합니다. CharPosition는 Object의 좌표 x,y를 효율적으로 나타내기 위해서 사용했습니다. CharPosition test; 와 같은 코드를 타이핑했다면 test.x, test.y 형식으로 접근할 수 있습니다. 이는 Snake의 몸을 표현할 때도 쓰이고, 아이템, 게이트의 위치를 표현할 때 사용됩니다. Map에서 Snake의 Head는 3, Body는 4로 구분합니다. 실행화면에서 머리는 H, 몸통은 B로 표시합니다. 사용자의 입력에 따라 스네이크의 위치가 변합니다. 사용자 입력이 없을 시 진행방향으로 이동하며 사용자의 입력이 진행방향과 같으면 그 진행방향 그대로 유지합니다. 반면 진행방향과 반대방향이면 게임오버입니다.

## 3단계. Item 요소의 구현


- Map에서 Fruit Item과 Poison Item은 각각 5, 6으로 구분하여 생성합니다. 5는 Snake의 크기가 늘어나는 아이템으로 G로 출력됩니다. 6은 Snake의 크기가 줄어드는 아이템으로 P로 출력됩니다. 모든 아이템을 관리하는 것은 ItemManager 클래스입니다. 이 클래스는 아이템을 일정 시간이 지나면 생성해줍니다. 또, 아이템이 생성된 뒤, 모든 아이템을 isExceedTime 함수로 일정 시간이 지났는지 검사합니다. 참값을 반환한다면 일정 시간이 지난 것이고 그 아이템은 vector로 생성된 data 에서 사라집니다.

## 4단계. Gate 요소의 구현

- Map에서 Gate는 7로 구분합니다. Gate는 실행 시, ?로 표시합니다. Gate 한 쌍이 겹치지 않고 벽에만 생성 될 수 있게 합니다. Gate 생성 후 10초가 지나면 사라지게 하였지만 Snake가 Gate를 통과 중일 땐 Gate가 사라지지 않고 다른 위치에서 Gate가 생성되지 않게 합니다. 다음 게이트의 위치에 따라서 Snake가 나오는 위치가 달라지니 게이트가 위치한 곳을 MapManager에서 data 배열을 가져와 동서남북 검사하여 어느 쪽으로 나갈지 구분합니다. 동서남북 검사한다는 말의 의미는 snake가 나갈 수 있는 쪽을 탐색한다는 것입니다. 예를 들어 게이트의 동쪽 좌표에 '0' 이 있다면 Snake가 나갈 수 있습니다.
- 따라서 possibleRight=true; 로 만들어주어 if 문으로 다음 게이트 좌표에서 나갈 수 있는 좌표를 반환합니다. Gate가 나타나는 벽이 Map의 가장자리일 때와 Map의 가운데에 있을 경우를 구분하고 벽이 Map 가장자리에 있을 경우 항상 Map의 안쪽 방향으로 진출하도록 하고 벽이 Map 가운데 있을 경우 진출하는 방향의 순서를 진입방향과 일치, 진입방향의 시계방향, 진입방향의 역시계방향, 진입방향의 반대방향으로 정합니다. 생성 조건은 Snake의 크기가 4이상일 때입니다.
- 현재 마스터 브런치에는 gate가 정상적으로 작동하는지 판단하기 위해서 Gate를 사용하지 않았으면 생성하지 않는 것으로 설정했습니다.

## 5단계. 점수 요소의 구현

- GameScene 우측 상단에 ScoreBoard 및 Mission 화면을 구현합니다. ScoreBoard에는 Snake의 size와 Fruit item, Poison Item을 먹은 갯수, Gate를 통과한 횟수를 나타내도록 하고 Mission은 Stage마다 다른 조건이 존재합니다.
- 현재 마스터 브런치에는 미션 조건을 모든 스테이지 동일하게 설정해놓았습니다.

 <b>국민대학교</b> <b>컴퓨터공학부</b> <b>Snake-Game I</b>	<b>결과보고서</b>		
	<b>프로젝트 명</b>	Snake-Game	
	<b>팀 명</b>	팀1-6	
	<b>문서 정보</b>	Version 1.1	2020-JUN-26

## 6단계. 메인 화면 구현 및 스테이지 번호 구현

- 프로그램 실행 시, WaitingScene이 nowScene으로 설정됩니다. WaitingScene에는 Snake game 대기 화면이 출력되며 입력한 스테이지 번호에 해당하는 스테이지로 실행될 수 있도록 Stage 클래스의 nowStage를 설정합니다

## 2.2 개발 내용 및 결과물

### 2.2.1 개발 내용


#### 1단계: map의 구현

Map은 2차원 배열로 구성되어 있습니다. 첫 번째 index엔 Map의 세로 길이를, 두 번째 index에는 가로 길이를 저장합니다. 각 stage별로 map의 모양을 바꾸어 난이도 조절을 하였습니다. Map의 구성 요소인 빈 공간, Wall등을 switch문과 ncurses함수 mvwaddch()을 활용하여 각각 ‘,’등으로 나타냅니다. MapManager의 char data 배열의 값 중 0은 뱀이 움직일 수 있는 빈칸, 1은 Snake의 머리와 충돌할 경우의 죽는 벽, 2는 gate가 생성될 수 없는 벽, 3은 Snake의 머리, 4는 Snake의 몸통, 5는 Snake의 크기가 늘어나는 아이템, 6은 Snake의 크기가 줄어드는 아이템, 7은 Gate, 8은 빈칸입니다.

#### 2단계: Snake 표현 및 조작

- Snake의 표현
  - Map에서 Snake의 Head는 3, Body는 4로 구분합니다. 실행화면에서 머리는 H, 몸통은 B로 표시합니다. 사용자의 입력에 따라 스네이크의 위치가 변합니다. 사용자 입력이 없을 시 진행방향으로 이동하며 사용자의 입력이 진행방향과 같으면 그 진행방향 그대로 유지합니다. 반면 진행방향과 반대방향이면 게임오버입니다.
- Snake의 조작
  - 화살표 입력을 받아 그 방향으로 Snake의 좌표를 1씩 움직이고 이를 MapManager를 이용하여 data Patching을 수행합니다. 이동 방향은 변수로 저장하였고, 함수 usleep()을 사용하여 1.5초간 멈추고, 사용자의 방향키 입력을 대기하고 있습니다.
  - GameScene에서 snake->isCollision()이 참값을 반환한다면 MapManager를 활용해 Snake 머리 부분 좌표를 확인하여 Snake가 진행할 방향의 좌표에 벽이 있거나, Poison 이 있어서 snake 의 데이터 수가 3이하가 되면 게임오버입니다. 또한 입력한 방향이 진행방향과 반대방향이면 게임오버입니다.



 <b>국민대학교 컴퓨터공학부 Snake-Game I</b>	<b>결과보고서</b>		
	<b>프로젝트 명</b>	Snake-Game	
	<b>팀 명</b>	팀1-6	
	<b>문서 정보</b>	Version 1.1	2020-JUN-26

### 3단계: Item의 구현

Item은 Item의 좌표, 아이템의 개수, 종류, 모양을 서로 다른 변수로 지정하였습니다. 아이템의 모양은 종류에 따라 Growth라면 출력은 'G'로 Map에는 5로, Poison이라면 출력은 'P'로 Map에는 6으로 나타냅니다. Item이 생성될 때 lastDropTime을 프로젝트 구조 내 float eTime으로 받아와 저장하고 현재 시각과 비교하여 5초가 지나면 Item을 재 생성하도록 함수를 만들었습니다. 이때 좌표는 getRandPoision 함수를 통해 Snake가 움직일 수 있는 '0' 좌표에만 생성 가능하게 해놓았습니다. Item의 종류는 Item 클래스의 string type으로 구분하며, "poison" type 아이템과 충돌하면 Snake의 Size를 1씩 줄입니다. "fruit" type 아이템과 충돌하면 Snake의 Size를 1씩 늘립니다.

### 4단계: Gate의 구현


벽을 표현하는 '2'에만 Gate가 생성될 수 있도록 합니다. 두 개의 Gate 좌표를 random하게 정합니다. Gate는 Snake의 Size가 4보다 클 때 생성됩니다. Gate가 생성될 때 lastDropTime을 프로젝트 구조 내 float eTime으로 받아와 저장하고 현재 시각과 비교하여 5초가 지나면 Gate를 재생성하도록 함수를 만들었습니다. Snake가 통과 중에는 사라지지 않아야 하므로 꼬리의 위치가 다음 게이트의 위치의 좌표에 도달한 경우 사라지게 했습니다. Gate 진입 시, 다른 Gate로 나올 Snake의 위치는 프로젝트 명세서에 따르어 계산된 Gate의 위치입니다.

### 5단계: 점수 요소의 구현

점수요소의 구현은 입력을 담당하는 Player와 출력을 담당하는 Format으로 나타낼 수 있습니다. Player는 GameScene에서 각각의 Object가 Collision했는지를 판단해, 획득한 Item과 Object들을 저장하는 객체입니다. 점수 요소의 출력은 Score와 Mission으로 나눌 수 있습니다. Score의 초기 값은 100점으로 Snake가 Growth Item 획득, Gate 통과를 하면 10점 상승합니다. Poison Item을 획득 시에는 5점 감소합니다. Mission은 현재 snake의 길이와 목표 길이가 있는 Length, 획득한 Growth Item의 개수와 목표 개수가 있는 Growth, 획득한 Poison Item의 개수와 목표 개수가 있는 Poison, 그리고 통과한 Gate의 개수와 목표 개수가 있는 Gate가 있습니다. 각 stage에서 각각의 요소들의 목표치를 시간(60초) 안에 충족해야 다음 stage로 넘어갈 수 있게 됩니다.

### 6단계: 메인 화면 구현 및 스테이지 번호 구현

화면은 Scene들과 Format으로 구성되어 있습니다. 우선 게임을 실행했을 때 출력되는 초기화면은 WaitingScene으로 1을 입력하면 게임이 시작하게 됩니다. 게임의 시작은 GameScene에서 stage 변수를 1로 맞춰 놓고 시작을 합니다. 게임이 플레이 될 때 화면은 GameScene과 Format으로 구성되어 있습니다. GameScene은 snake가 Item을 먹고 게임이 진행되는 화면이고, Format은 게임 시간, Score, Mission 등을 출력하는 부분입니다. 게임이 종료됐을 때는 GameOverScene을 출력하고, snake가 미션을 성공했을 경우 GameScene의 Update부분에서 isClear로 판단해 stage 2로 현재 stage를 설정하고 GameScene을 다시 생성합니다.

 국민대학교 컴퓨터공학부 Snake-Game I	결과보고서		
	프로젝트 명	Snake-Game	
	팀 명	팀1-6	
	문서 정보	Version 1.1	2020-JUN-26

## 2.2.2 시스템 구조 및 설계도

### 1단계. Map의 구현

```
#pragma once
```

```
struct CharPosition
```

```
{
    int32 x, y;
    CharPosition(int col, int row) : x(col), y(row) {}
    CharPosition() : x(0), y(0) {}
};
```

CharPosition는 Object의 좌표 x,y를 효율적으로 나타내기 위해서 사용했습니다. CharPosition test; 와 같은 코드를 타이핑 했다면 test.x, test.y 형식으로 접근할 수 있습니다. 이는 Snake의 몸을 표현할 때도 쓰이고, 아이템, 게이트의 위치를 표현할 때 사용됩니다.

MapManager라는 클래스는 data라는 char 배열을 담고 있습니다. 처음 GameScene이 초기화될 때, MapManager의 Load라는 함수를 호출합니다. Load 함수는 텍스트파일로 저장되어있는 맵 파일을 ifstream을 사용하여 한 줄씩 끝까지 읽습니다. 읽을 때마다 data배열에 값을 넣어줍니다.

data 배열에 접근하여 뱀이 움직여 위치가 바뀐다든지, 아이템이 spawn된다든지 등 이벤트가 발생하여 생성된 object의 위치에 해당 object를 기입하여 줍니다. 예를 들어 (3,4) 에서 아이템이 spawn 됐다면, data[3] [4] 위치에 아이템을 넣어줍니다. 그래서 ItemManager, GateManager, Snake에서는 MapManger의 PatchData라는 함수를 이용해 Map을 업데이트해줍니다.

```
#include "MapManager.h"
```

```
#include "myFunction.h"
```

```
#include <vector>
```

```
#include <fstream>
```

```
extern Stage *stage;
```

```
MapManager::MapManager() // load txt file. named stageName. ex) 1.txt 2.txt 3.txt 4.txt
{
}
```

```
void MapManager::Load(){
```

```
    std::ifstream readfile;
    string src = "map/map"+std::to_string(stage->getNowStage()+1) + ".txt";
```

```
    readfile.open(src);
```


```
    int height = 0;
```

```
    while (!readfile.eof()){
```

```
        char temp[120];
```

```
        readfile.getline(temp, 120);
```

```
        for(int i=0;i<WIDTH;i++){
```

 <div> <b>국민대학교</b>  <b>컴퓨터공학부</b>  <b>Snake-Game I</b> </div>	결과보고서		
	프로젝트 명	Snake-Game	
	팀 명	팀1-6	
	문서 정보	Version 1.1	2020-JUN-26

```

        data[height][i]=temp[i];
    }

    height++;
}

/* data가 올바르게 들어갔는지 확인하기 위한 검증 코드*/
// std::string Path = "test.txt";

// std::ofstream writeFile(Path);
// if (writeFile.is_open())
// {
//     for(int i=0;i<HEIGHT;i++){
//         for(int j = 0; j < WIDTH; j++){
//             writeFile <<data[i][j];
//             // cout<<data[i][j];
//         }
//         writeFile << "\n";
//     }
// }

}

void * MapManager::GetData(){
    return data;
}

void MapManager::PatchData(int y, int x, char patchData){
    data[y][x]=patchData;
}

MapManager::~MapManager()
{
}

void MapManager::Render()
{
}

void MapManager::Update(float eTime)
{
}

```


GameScene에서 관리하는 MapManager 클래스를 이용해 GameScene::Render() 에서 MapManager 안의 데이터를 출력합니다. MapManager 안의 data 배열은 0~8 문자가 들어가 있습니다. 0은 뱀이 움직일 수 있는 빈칸, 1은 Snake의 머리와 충돌할 경우의 죽는 벽, 2는 gate가 생성될 수 없는 벽, 3은 Snake의 머리, 4는 Snake의 몸통, 5는 Snake의 크기가 늘어나는 아이템, 6은 Snake의 크기가 줄어드는 아이템, 7은 Gate, 8은 빈칸입니다. 다시 말해, GameScene::Render() 부분은 Map을 출력하는 역할을 합니다.

```

void GameScene::Render()
{
    format->Render();

    for (int i = 0; i < HEIGHT; i++)
    {

```

 <b>국민대학교 컴퓨터공학부 Snake-Game I</b>	결과보고서		
	프로젝트 명	Snake-Game	
	팀 명	팀1-6	
	문서 정보	Version 1.1	2020-JUN-26

```


for (int j = 0; j < WIDTH; j++)
{
    switch (mapManager->data[i][j])
    {
        case '0':
            mvaddch(i, j, ' ');
            break;
        case '1':
            mvaddch(i, j, '-');
            break;
        case '2':
            mvaddch(i, j, 'X');
            break;
        case '3':
            mvaddch(i, j, 'H');
            break;
        case '4':
            mvaddch(i, j, 'B');
            break;
        case '5':
            mvaddch(i, j, 'G');
            break;
        case '6':
            mvaddch(i, j, 'P');
            break;
        case '7':
            mvaddch(i, j, '?');
            break;
        case '8':
            mvaddch(i, j, ' ');
    }
}
refresh();
}

```

## 2단계. Snake 표현 및 조작

Snake가 처음 생성될 때 entire vector에 30, 31, 32, 33, 34 위치에 CharPosition을 넣어줍니다. Snake::Update 부분은 뱀이 움직이는 부분입니다. 예를 들어 왼쪽 방향키를 눌렀다면, 이전 direction 'r'인 경우 반대되는 방향을 입력한 것이므로 뱀이 죽습니다. 이전 direction이 'r'이 아닐 경우 direction을 'l'로 초기화해줍니다.

그 뒤로 이전 head의 위치를 불러와 direction에 맞게 CharPosition을 만들어줍니다. 그리고 그것을 entire vector 시작 부분(첫번째 배열)에 insert를 해줍니다. 예를 들어 왼쪽 방향키를 눌러 현재의 direction이 'l'이면 entire.insert(entire.begin(), CharPosition(entire[0].x - 1, entire[0].y)); 를 수행합니다.

 <b>국민대학교 컴퓨터공학부 Snake-Game I</b>	결과보고서		
	프로젝트 명	Snake-Game	
	팀 명	팀1-6	
문서 정보	Version 1.1		2020-JUN-26

IsCollision 부분은 뱀의 머리가 어떤 오브젝트랑 충돌했는지 판단해주는 함수입니다. 현재 뱀이 움직이려고 하는 위치와 MapManager를 이용하여 맵 배열 값이 '0' 인 경우에는 문제가 없지만, '6', '7' 등 게이트, 아이템인 경우 충돌 판정을 내립니다.

Snake가 아이템과 충돌하여 GameScene에서 snake->Grow() 또는 snake->Shrink() 를 호출했다면, 각각 isGrow와 isShrink를 true로 설정합니다. entire vector의 첫번째에 갱신된 head의 좌표가 insertion되니 매번 꼬리를 잘라줘야합니다. 꼬리를 자르지 않을 경우는 과일을 먹어 Snake의 몸집의 커져야 할 경우입니다. 그래서 isGrow가 true인 경우 CutTail을 하지 않습니다. isShrink가 true인 경우 CutTail을 최종적으로 두 번 호출하게끔 구성했습니다. 그 이유는 게임이 진행되면서 기본적으로 잘려야 하는 꼬리 하나, 독 아이템을 먹어서 몸집이 줄어들어야 하니 잘려야 하는 꼬리 하나 때문에 두 번 호출하게끔 구성했습니다.


그 뒤로 Snake의 entire vector를 맵 데이터에 patching 해줘야하기 때문에, PushData 함수를 호출합니다. entire vector의 첫번째 요소는 Snake의 head이기 때문에  $i=0$  일 때 `mapManager->PatchData(entire[i].y, entire[i].x, '3');` 을 실행하고, 그 외에는 `mapManager->PatchData(entire[i].y, entire[i].x, '4');` 를 실행합니다.

### 3단계. Item 요소의 구현

모든 아이템을 관리하는 것은 ItemManager 클래스입니다. 이 클래스는 아이템을 일정 시간이 지나면 생성해줍니다. 또, 아이템이 생성된 뒤, 모든 아이템을 isExceedTime 함수로 일정 시간이 지났는지 검사합니다. 참값을 반환한다면 일정 시간이 지난 것이고 그 아이템은 vector로 생성된 data 에서 사라집니다. 아이템이 생성될 때는 PositionItem 함수를 호출합니다. 이 함수는 data vector에 Item을 생성하여 push\_back 해줍니다. 아이템이 생성될 때는 x, y 좌표를 랜덤 생성하는데 아이템이 알맞은 위치에 있어야 합니다. 따라서 Item 클래스에서는 아이템이 놓여질 수 있는 부분에만 놓아지도록 생성된 x, y좌표와 MapManager의 map data배열을 비교합니다. 생성된 x,y 좌표에 '0' 이 놓여져 있으면 뱀이 움직일 수 있는 좌표이므로, 아이템이 생성되는 위치로 적절합니다. 아이템이 생성될 때 poison인지 fruit인지 구별하기 위해서 클래스 안에 string type을 넣어놨습니다. 또한 DeleteCollisionData를 통해 data vector에서 충돌한 데이터를 찾고 그것을 지워줍니다.

### 4단계. Gate 요소의 구현

게이트를 관리하는 것은 GateManager 클래스입니다. getRandPosition 함수로 게이트가 적절하게 위치하도록 합니다. 게이트가 스네이크의 머리와 충돌했을 때는 getNextGate 함수로 Snake의 몸이 어디로 가야할지 좌표를 알려줍니다. 부딪힌 게이트가 어떤 게이트인지 구분하기 위해서 Snake의 머리의 위치를 가져오고, 게이트의 위치를 담고있는 data vector를 검사합니다. 그래서 다음 게이트가 무엇인지 알 수 있습니다. 다음 게이트의 위치에 따라서 Snake가 나오는 위치가 달라지니 게이트가 위치한 곳을 MapManager에서 data 배열을 가져와 동서남북 검사하여 어느 쪽으로 나갈지 구분합니다. 동서남북 검사한다는 말의 의미는 snake가 나갈 수 있는 쪽을 탐색한다는 것입니다. 예를 들어 게이트의 동쪽 좌표에 '0' 이 있다면 Snake가 나갈 수 있습니다. 따라서 possibleRight=true; 로 만들어주어 if 문으로 다음 게이트 좌표에서 나갈 수 있는 좌표를 반환합니다. 게이트 위치 우선순위 판단 기준은 e-campus에 업로드하신 프로젝트 명세 pdf와 같습니다. 또한 DeleteCollisionData를 통해 data vector에서 충돌한 데이터를 찾고 그것을 지워줍니다. Snake가 통과 중에는 사라지지 않아야 하므로 꼬리의 위치가 다음 게이트의 위치의 좌표에 도달한 경우 사라지게 했습니다.

 <b>국민대학교 컴퓨터공학부 Snake-Game I</b>	결과보고서		
	프로젝트 명	Snake-Game	
	팀 명	팀1-6	
	문서 정보	Version 1.1	2020-JUN-26

### 3,4 단계 통합 구현

GameScene의 Update 함수 구조는 다음과 같습니다. 먼저 미션이 완료되었는지 체크합니다. 완료했다면 Stage 클래스의 nowStage를 한 단계 올립니다. 그리고 새로운 GameScene으로 초기화해줍니다. Player 객체에는 현재 점수를 기록하기 위해 `player->SetLengthScore(snake->entire.size());` `player->SetTotalScore(stage->nowStage);` 를 수행합니다. Snake를 Update(키 입력을 받아 움직이게 하는 함수) 하고 snake의 head 좌표가 업데이트 되니 머리가 흑여나 어떤 것과 충돌하지 않았는지 판단합니다. `snake->isCollision()` 을 통해 충돌이라고 참값이 반환되면 ProcessCollision 함수를 수행합니다.


```
void GameScene::ProcessCollision()
{
    int y = snake->GetHead().y;
    int x = snake->GetHead().x;

    char temp = mapManager->data[y][x];

    if (temp == '1')
    {
        snake->isDied = true;
    }
    else if (temp == '5')
    {
        itemManager->DeleteCollisionData(y, x);
        player->growScore += 1;
        snake->Grow();
    }
    else if (temp == '6')
    {
        itemManager->DeleteCollisionData(y, x);
        player->poisonScore += 1;
        snake->Shrink();
    }
    else if (temp == '7')
    {
        CharPosition nextGate = gateManager->GetNextGate();
        player->gateScore += 1;
        gateManager->isUsed = true;
        snake->SetHeadPos(nextGate.y, nextGate.x);
    }
}
```

GameScene에서 충돌 판정 함수는 ProcessCollision 함수입니다. 이 함수에서는 Snake 머리의 현재 위치를 받아와 MapManager의 map data 배열과 비교합니다. 맵 배열 값이 '0' 인 경우에는 문제가 없지만, '6', '7' 등 게이트, 아이템인 경우 충돌 판정을 내리고 Snake를 커지게 하거나 줄어들게 하거나 다음 게이트의 위치로 보내주는 등의 작업이 이뤄져야 합니다.

ProcessCollision을 거치고 나면 Snake가 죽었는지 안 죽었는지 판단하기 위해 `snake->isDied` 를 체크하고 참값이라면 게임오버라고 판정합니다. 게임오버가 아니라면 Snake의 데이터를 MapManger에 Push 하기 위해 `snake->pushData()` 를 수행합니다. 또한 format으로 업데이트된 사용자의 점수와 시간을 표시하기 위해 `format->Update(eTime);` 를 수행합니다. ItemManager 클래스, GateManager 클래스 역시 게임에서 이벤트(뱀이 아이템을 먹었다던지, 게이트와 충돌했다던지)가 발생했을 수도 있으니 그것을 판정하고자 `itemManager->Update(eTime);`

 <b>국민대학교 컴퓨터공학부 Snake-Game I</b>	결과보고서		
	프로젝트 명	Snake-Game	
	팀 명	팀1-6	
	문서 정보	Version 1.1	2020-JUN-26

gateManager->Update(eTime); 를 수행합니다. 이 모든 것을 수행했다면 usleep(150000); 이라는 함수로 일정시간 멈춥니다. 이는 Snake의 움직임을 1.5초마다 움직이게 하기 위한 것입니다.


## 5단계. 점수 요소의 구현

점수 요소의 구현은 점수 요소를 저장하는 Player와 형식을 맞춰 출력해주는 Format으로 나눌 수 있습니다. 우선, Player를 보면 int 자료형의 길이점수를 저장하는 lengthScore, 획득 Growth Item을 저장하는 growScore, 획득 Posion Item을 저장하는 poisonScore, 지나간 Gate 수를 저장하는 gateScore, 마지막으로 총점을 나타내는 totalScore라는 변수들과 이들을 설정하는 setter들로 구성되어 있습니다.

```
#pragma once

class Player
{
public:
    int lengthScore;
    int growScore;
    int poisonScore;
    int gateScore;
    int totalScore;

    Player()
    {
        lengthScore = 0;
        growScore = 0;
        poisonScore = 0;
        gateScore = 0;
        totalScore = 0;
    }
    ~Player() {}
    void SetLengthScore(int value)
    {
        lengthScore = value;
    }
    void SetGrowScore(int value)
    {
        growScore = value;
    }
    void SetPoisonScore(int value)
    {
        poisonScore = value;
    }
    void SetGateScore(int value)
    {
        gateScore = value;
    }
}
```

 <b>국민대학교 컴퓨터공학부 Snake-Game I</b>	<b>결과보고서</b>		
	<b>프로젝트 명</b>	Snake-Game	
	<b>팀 명</b>	팀1-6	
	<b>문서 정보</b>	Version 1.1	2020-JUN-26

```

}
void SetTotalScore(int value)
{
    totalScore = growScore * 10 - poisonScore * 5 + gateScore * 10 + value * 50 + 100;
}
};

```

형식을 갖춰 출력을 담당하는 Format은 Update(eTime)에서는 시간을 출력하는 DrawTime(eTime)이라는 method를 사용하고, Render()에서는 Score와 Mission을 출력하는 DrawScore(), DrawMission()을 사용해 출력합니다.

```

void Format::Update(float eTime)
{
    DrawTime(eTime);
}

void Format::Render()
{
    DrawScore();
    DrawMission();
}

```

Score의 구현은 다음과 같습니다. Score의 구현은 DrawScore()라는 method를 만들어 주어, 작게 점수 표시가 되는 것이 아니라, Format.h에서 생성한 score 배열에 있는 숫자 모양을 불러와서 화면에 player->totalScore를 출력합니다.

```


void Format::DrawScore()
{
    move(7, maxwidth / 5 * 4 + 4);
    printw("< S C O R E >");

    for (int i = 0; i < 26; i++)
    {
        move(8, maxwidth / 5 * 4 - 3 + i);
        addch('-');
    }

    int digit = 100, totalScore = player->totalScore;

```



 <b>국민대학교 컴퓨터공학부 Snake-Game I</b>	결과보고서		
	프로젝트 명	Snake-Game	
	팀 명	팀1-6	
	문서 정보	Version 1.1	2020-JUN-26

```

for (int i = 0; i < 3; i++)
{
    int digitScore;
    std::string s = "00000";

    digitScore = totalScore / digit;
    totalScore %= digit;

    for (int j = 0; j < 5; j++)
    {
        move(11 + j, maxwidth / 5 * 4 - 2 + 4 + i * 6);
        printw("%s", score[digitScore][j]);
    }
    digit /= 10;
}

for (int i = 0; i < 26; i++)
{
    move(18, maxwidth / 5 * 4 - 3 + i);
    addch('-');
}
}

```

Mission을 출력하는 DrawMission()의 구현은 아래와 같습니다. 우선, stage객체를 통해 현재 stage의 미션을 찾아 가져와 nowMission에 저장합니다. 이후, 항목들을 출력할 때, player객체에서 각각의 항목들을 가져와 출력을 하고, 마지막에 Complete(player->아이템)이라는 method를 실행시켜 미션을 달성했으면 'V'를 출력합니다.

```


void Format::DrawMission()
{
    int *nowMission = stage->getNowMission();

    move(maxheight / 2, maxwidth / 5 * 4 + 1);
    printw("< M I S S I O N >");

    for (int i = 0; i < 26; i++)
    {
        move(maxheight / 2 + 1, maxwidth / 5 * 4 - 3 + i);
        addch('-');
    }

    move(22, maxwidth / 5 * 4 + 4);
    printw("Length : %d/%d (%c)", player->lengthScore, nowMission[0],
    Complete(player->lengthScore, nowMission[0]));
}

```

 <b>국민대학교 컴퓨터공학부 Snake-Game I</b>	결과보고서		
	프로젝트 명	Snake-Game	
	팀 명	팀1-6	
	문서 정보	Version 1.1	2020-JUN-26

```

move(24, maxwidth / 5 * 4 + 4);
printw("Fruit : %d/%d (%c)", player->growScore, nowMission[1],
Complete(player->growScore, nowMission[1]));

move(26, maxwidth / 5 * 4 + 4);
printw("Poison : %d/%d (%c)", player->poisonScore, nowMission[2],
Complete(player->poisonScore, nowMission[2]));

move(28, maxwidth / 5 * 4 + 4);
printw("Gate : %d/%d (%c)", player->gateScore, nowMission[3],
Complete(player->gateScore, nowMission[3]));

for (int i = 0; i < 26; i++)
{
    move(30, maxwidth / 5 * 4 - 3 + i);
    addch('-');
}
}

```

아래는 남은 Time을 출력하는 DrawTime(float eTime)이라는 method의 구현입니다. 우선, eTime을 통해 60초에서 현재 지난 시간을 빼서 남은 게임 시간을 digitTime에 저장합니다. 만약, digitTime이 0보다 작거나 같을 경우 snake의 isDied를 true로 해 Game Over시킵니다. 이후, score 배열에 저장된 큰 숫자들을 digitTime에 맞게 출력합니다.


```

void Format::DrawTime(float eTime)
{
    int digit = 10;
    digitTime = (int)(60 - eTime);
    if (digitTime <= 0)
        snake->isDied = true;

    for (int j = 0; j < 5; j++)
    {
        move(1 + j, maxwidth / 5 * 4 - 2 + 2);
        printw("%s", score[0][j]);
    }

    for (int i = 0; i < 26; i++)
    {
        move(6, maxwidth / 5 * 4 - 3 + i);
        addch('-');
        move(0, maxwidth / 5 * 4 - 3 + i);
        addch('-');
    }
}

```

 국민대학교 컴퓨터공학부 Snake-Game I	결과보고서		
	프로젝트 명	Snake-Game	
	팀 명	팀1-6	
	문서 정보	Version 1.1	2020-JUN-26

```

move(2, maxwidth / 5 * 4 - 2 + 8);
addch(char(219));

move(4, maxwidth / 5 * 4 - 2 + 8);
addch(char(219));

for (int i = 0; i < 2; i++)
{
    for (int j = 0; j < 5; j++)
    {
        move(1 + j, maxwidth / 5 * 4 - 2 + 4 + (i + 1) * 6);
        printf("%s", score[digitTime / digit][j]);
    }
    digitTime = digitTime % digit;
    digit /= 10;
}
}

```

### 6단계. 메인 화면 구현 및 스테이지 번호 구현


화면의 구성은 다음과 같습니다. 게임 시작 전 화면인 WaitingScene, 게임 진행 화면인 GameScene, 게임 오버했을 때 화면 GameOverScene이 있습니다.

WaitingScene은 크게 입력을 받는 UserInput()과 scene폴더 안에 있는 WaitingScene.txt를 읽어와 출력을 해주는 Load(), UserInput()의 값을 리턴하고 제작자 이름이 적혀있는 IsUserReady(), 마지막으로 1이 입력 되었을 경우 stage를 1로 설정하고 게임을 시작하는 Update()라는 method로 구성되어있습니다.

GameScene은 게임이 플레이 되는 직접적인 곳입니다. 우선 아이템들의 달성치를 저장하는 player, 각 stage별로 알맞은 map을 출력해주는 mapManager, 주인공 snake, Growth item과 Poison Item을 랜덤하게 맵에 출력하는 itemManager, Gate를 랜덤하게 맵에 출력해주는 gateManager, Score, Mission, Time등 다채로운 출력을 해주는 format 등 다양한 객체를 생성하고 관리합니다. 이들의 Update(eTime)와, Render() 메서드를 GameScene의 Update(eTime)와 Render()에서 실행시킴으로써 게임이 플레이되는 것입니다.

Stage는 GameScene에서 isClear()라는 method를 통해서 Mission을 모두 달성했는지를 확인합니다. Mission을 모두 달성했을 경우, stage를 1단계 올리고 새로운 GameScene을 생성합니다.

GameOverScene은 snake의 bool 자료형인 isDied가 true일 때, 발생합니다. isDied가 true인 경우는 snake의 length가 3보다 작거나 같을 때, 벽에 부딪혔을 때, 자신의 body에 head가 부딪혔을 때, 시간이 초과 되었을 때 등이 있습니다. 이와 같은 경우들에서 is Died가 true로 변하고 GameOverScene이 실행됩니다. 실행되면, Load()를 통해 scene폴더 안에 있는 GameOverScene.txt파일을 불러와 출력합니다. 출력 후, 다시 게임을 진행할 것인지 여부를 묻습니다. n이 입력되는 경우는 게임을

 <b>국민대학교</b> <b>컴퓨터공학부</b> <b>Snake-Game I</b>	<b>결과보고서</b>		
	<b>프로젝트명</b>	Snake-Game	
	<b>팀명</b>	팀1-6	
	<b>문서 정보</b>	Version 1.1	2020-JUN-26

종료하고, y가 입력 되면 다시 GameScene을 생성하고 ChangeScene을 통해 GameScene으로 이동합니다.

### 2.2.3 활용/개발된 기술

이 프로젝트에서 사용한 라이브러리는 ncurses와 vector가 있습니다. 가변적인 Snake의 길이, 데이터가 앞에서 삽입되는 것의 자료구조로서 vector가 어울린다고 생각했습니다. 프로젝트 구조를 짜는 데 있어서 예전에 DirectX를 이용한 게임을 만들어 본 적이 있어 예전의 코드를 가져왔습니다. Scene 을 만들어 nowScene에 새로운 Scene을 할당하는 구조는 이번 Snake Game에 적합하다고 생각했습니다.


Game을 Update하고 Rendering 하는 데에 ncurses를 사용하였고, mvaddch, printw 등을 이용하여 cli 환경에서 게임에서 오브젝트의 결과를 출력해보았습니다. 또한 ncurses의 usleep 함수를 이용해 게임 속도를 조절했으며, 1.5초마다 움직이게끔 구성했습니다.

새롭게 고안한 알고리즘은 게이트 출입 시, 어느 좌표로 Snake를 보내줘야 하는지 판단하는 알고리즘입니다. 게이트가 Snake의 머리와 충돌했을 때는 getNextGate 함수로 Snake의 몸이 어디로 가야할지 좌표를 알려줍니다. 부딪힌 게이트가 어떤 게이트인지 구분하기 위해서 Snake의 머리의 위치를 가져오고, 게이트의 위치를 담고있는 data vector를 검사합니다. 그래서 다음 게이트가 무엇인지 알 수 있습니다. 다음 게이트의 위치에 따라서 Snake가 나오는 위치가 달라지니 게이트가 위치한 곳을 MapManager에서 data 배열을 가져와 동서남북 검사하여 어느 쪽으로 나갈지 구분합니다. 동서남북 검사한다는 말의 의미는 snake가 나갈 수 있는 쪽을 탐색한다는 것입니다. 예를 들어 게이트의 동쪽 좌표에 '0' 이 있다면 Snake가 나갈 수 있습니다. 따라서 possibleRight=true; 로 만들어주어 if 문으로 다음 게이트 좌표에서 나갈 수 있는 좌표를 반환합니다.

### 2.2.4 현실적 제한 요소 및 그 해결 방안


3단계에서 Item을 어떻게 구분하고 vector에 넣을 수 있는지 고민을 많이 해봤습니다. 초기 version에도 ItemManager라는 것이 있었지만, 그것은 지금처럼 Item vector로 관리한 것이 아니었습니다. 아이템을 나누어 Fruit, Poison 클래스를 만들었고 각각 vector를 만들어주었습니다. 이렇게 하니 통합성이라는 측면에서 굉장히 비효율적이었습니다. 그래서 생각한 것이 Item이라는 클래스를 하나 만들어 통합 관리를 하고 Item 클래스에 있는 string type을 이용하여 아이템을 구분지어주기로 했습니다. 그 뒤로 ItemManager에서 통합 관리하여 기능을 개발할 때 불필요한 코드를 줄일 수 있었습니다.

3단계에서는 Snake가 item을 먹었을 경우 Snake의 Body를 어떻게 줄이고 늘리는지에 대한 어려움이 있었습니다. 이는 MapManager의 data vector에서 Snake의 꼬리 부분을 '0'으로 바꿔주고, Snake에서는 entire.pop\_back()을 이용하여 해결하였습니다. Snake가 아이템과 충돌하여 GameScene에서 snake->Grow() 또는 snake->Shrink() 를 호출했다면, 각각 isGrow와 isShrink를 true로 설정합니다. entire vector의 첫번째에 갱신된 head의 좌표가 insertion되니 매번 꼬리를 잘라줘야합니다. 꼬리를 자르지 않을 경우는 과일을 먹어 Snake의 몸집의 커져야 할 경우입니다. 그래서 isGrow가 true인 경우 CutTail을 하지 않습니다. isShrink가 true인 경우 CutTail을 최종적으로 두 번 호출하게끔 구성했습니다. 그 이유는 게임이 진행되면서 기본적으로

 <b>국민대학교 컴퓨터공학부 Snake-Game I</b>	<b>결과보고서</b>		
	<b>프로젝트 명</b>	Snake-Game	
	<b>팀 명</b>	팀1-6	
	<b>문서 정보</b>	Version 1.1	2020-JUN-26

잘려야 하는 꼬리 하나, 독 아이템을 먹어서 몸집이 줄어들어야 하니 잘려야 하는 꼬리 하나 때문에 두 번 호출하게끔 구성했습니다.


4단계에서는 Gate가 생성되는 방식에서 개선점을 찾았습니다. Gate는 Wall에서만 생성되어야만 합니다. Gate의 생성 방식은 랜덤으로 뽑은 좌표가 '1'(벽)일 경우 Gate가 생성 가능하다고 판단했습니다. 따라서 몇 백번의 랜덤 좌표 연산이 이루어질 수 있습니다. Gate를 생성하는 좌표를 찾는데 시간이 오래 걸릴 수 밖에 없어 Gate를 생성할 때 Wall의 좌표를 따로 모으기 위해 WallManager를 만들까 고민했습니다. 이 설계는 WallManager의 데이터 vector에 wall 좌표 데이터들을 추가하여, data의 size 내에서 무작위 숫자를 뽑아 무작위 숫자에 해당하는 데이터의 요소를 불러오는 설계입니다. 그렇게 진행된다면 gate generation 때, 많은 연산을 진행하지 않아도 된다고 생각합니다. 다만 아직까지 눈에 띄는 게임 렉 현상은 없어 지금과 같은 구조를 유지하고 있습니다. 추후 feature 브랜치로 작업할 예정입니다.

 <b>국민대학교</b> <b>컴퓨터공학부</b> <b>Snake-Game I</b>	<b>결과보고서</b>		
	<b>프로젝트 명</b>	Snake-Game	
	<b>팀 명</b>	팀1-6	
	<b>문서 정보</b>	Version 1.1	2020-JUN-26

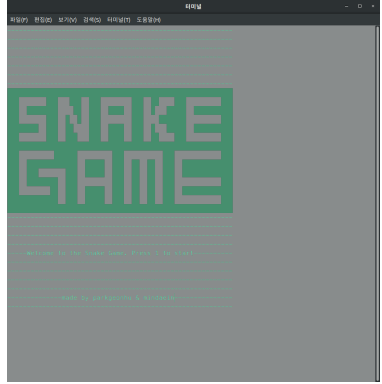


## 2.2.5 결과물 목록

### 1) 게임 구조

—	Makefile
—	map
—	map1.txt
—	map2.txt
—	map3.txt
—	scene
—	GameOverScene.txt
—	WaitingScene.txt
—	src
—	CharPosition.h
—	Format.cpp
—	Format.h
—	Fruit.h
—	GameOverScene.cpp
—	GameOverScene.h
—	GameScene.cpp
—	GameScene.h
—	GateManager.cpp
—	GateManager.h
—	IObject.h
—	IScene.h
—	Item.cpp
—	Item.h
—	ItemManager.cpp
—	ItemManager.h
—	MapManager.cpp
—	MapManager.h
—	Player.h
—	Poison.h
—	Snake.cpp
—	Snake.h
—	Stage.cpp
—	Stage.h
—	WaitingScene.cpp
—	WaitingScene.h
—	Wall.cpp
—	Wall.h
—	main.cpp
—	myFunction.cpp
—	myFunction.h

 <b>국민대학교</b> <b>컴퓨터공학부</b> <b>Snake-Game I</b>	<b>결과보고서</b>		
	<b>프로젝트 명</b>	Snake-Game	
	<b>팀 명</b>	팀1-6	
	<b>문서 정보</b>	Version 1.1	2020-JUN-26

## 2) 전체적인 게임 진행 화면

WaitingScene(게임 시작 전)	GameScene(게임 시작)	GameOverScene(게임 종료)
		

## 3 자기평가


박건후 : 기여도(70%) : @parkgeonhu

### 역할

Project Manager를 담당했으며, 협업을 위한 git flow 방식을 적용했습니다. git flow 방식과 github template을 이용하여 Pull Request 를 하는 방법을 팀원에게 알려주었습니다. 저희의 협업방식은 feature/기능 이란 브랜치를 만들어 commit을 원격 브랜치로 푸시하고, Pull Request를 했습니다. Pull Request를 할 때, 자신이 작업한 브랜치에서는 어떤 점이 개선되었고, 버그가 있다면 어떤 버그가 있는지 간략하게 요약해 첨부하였습니다. 또한 그 브랜치를 develop 브랜치에 merge 하기 전 모두의 동의가 필요하다고 생각해서 모든 이가 코드리뷰를 하고 별다른 의견이 없다면 merge 하는 방식으로 했습니다. 그래서 이번 프로젝트의 제 역할은 이러한 소통의 규격을 만들었고, 프로젝트의 구조를 담당했습니다. Scene을 바꾼다는 개념을 코드화해보았고, 이를 응용한 ItemManager, GateManager 클래스 작성과 snake의 충돌판정을 통해 snake의 변화가 제대로 적용될 수 있도록 클래스를 세분화하고 구현해보았습니다.

### 프로젝트 수행 시 어려운 점

ncurses 라이브러리를 사용해본 것은 처음이었습니다. 익숙하지 않은 라이브러리에 타 라이브러리에 비해 영어 문서가 많아 해석하는데 애를 먹기도 했습니다. 또한 리눅스라는 환경에서 make 유틸리티를 이용한 컴파일은 처음이었고, gdb로 디버깅해본 경험도 처음이어서 개발하는데 있어 진전속도가 빠르지 않았습니다. 이번 프로젝트는 구조에 신경을 굉장히 많이 썼습니다. 게임을 Scene이라는 개념을 이용하여 구현해보겠다는 일념 하에 일관성 있는 추상적 접근을 해봤습니다. 또한 그렇게 하나의 개념으로 구체화 하다보니 자연스럽게 팀원이 프로젝트 구조를 이해하는데 많은 도움이 된 것 같습니다.

 <b>국민대학교 컴퓨터공학부 Snake-Game I</b>	결과보고서		
	프로젝트 명	Snake-Game	
	팀 명	팀1-6	
	문서 정보	Version 1.1	2020-JUN-26

### 프로젝트 운영에 개선이 필요하다고 생각하는 점

역시나 변수명이나 함수명을 짓기 어렵습니다. 어쩌다보니 Snake 클래스의 데이터 vector의 이름을 다른 클래스와 달리 통합하지 않았습니다. 다른 클래스에서는 데이터를 저장하는 vector이름을 data라고 지었지만, Snake 클래스에서는 entire 이라고 지었습니다. 그 이유는 github에서 다른 사람의 코드를 참고하다가 몇 가지 부분을 굵었는데 프로젝트 규모가 엄청 커질지는 몰랐어서 entire라고 지어진 부분을 냅뒀습니다. Snake 클래스가 여러 파일에서 쓰이다보니 변수명을 한번 수정하려면 여러가지 파일을 손봐야했습니다. 그것이 엄두가 안나 아직까지 변수명을 고치지 못하고 있습니다. 다음에 이렇게 규격화하고 협업에 쓰일 프로젝트는 변수명을 짓는 데 있어서 고민을 많이 해보고 설계해야겠습니다.

### 팀원 평가

팀원이 훌륭하게 잘 따라왔습니다. 제 코드를 팀원이 볼 때 이해하지 못할 것 같다는 생각도 했는데, 제가 설명하는 구조를 잘 이해해주었고 그 구조에 맞게 코딩을 잘했습니다. 특히 팀원이 제가 만든 IObject 클래스를 이용해 Format 클래스를 짰 것을 보면 정말 기특하다는 생각을 많이합니다. 이번 프로젝트 덕분에 협업이란 무엇인가라는 것을 제대로 배웠고, git flow 방식은 앞으로 다른 프로젝트에서도 사용할 예정입니다. 재밌었습니다.

## **민대인 : 기여도(30%): @bamin0422**

### 역할

Project Manager(박건후)가 설계 해놓은 구조를 기반으로 코드를 작성하였습니다. 우선, github에서 feature/..의 형태의 branch들을 생성해 기능들을 발전시켰습니다. 이렇게 작성한 코드들은 바로 Pull Request를 활용해, Project Manager에게 향상된 기능과 코드를 작성할 때 발생했던 문제점들을 공유했습니다. 이후, 해결해야하는 문제점들이 없이 구현이 잘 될때, Project Manager와 상의를 통해 develop branch에 merge했습니다. 작성한 코드들은 게임 시작 시, 발생하는 WaitingScene 부분과, GameOver될 때 나오는 GameOverScene을 작성했습니다. 또한, 다음 스테이지로 넘어가기 위해 달성해야 하는 Mission, 최종 점수인 Score를 담아 적절한 모습으로 화면에 출력해주는 Format도 작성했습니다.


### 프로젝트 수행 시 어려운 점

프로젝트를 수행하면서, 어려웠던 점은 ncurses 라이브러리를 처음 사용해 봐서 모든게 신기하고 당황스러웠습니다. 하지만, 차근차근 이를 공부해 나가다 보니 결국 이렇게 snake-game을 완성할 수 있었습니다. 그리고 이렇게 github에서 Pull request와 여러 branch들을 활용하는 협업을 처음 진행해 열심히 3~4시간 걸려서 작성한 코드를 잘못 commit해 날려버려서 다시 짜는 일도 종종 있었습니다. 이러한 실수들을 겪으면서 github활용 능력도 올라가고 기능 branch를 따로 생성해 develop에 merge하는 방식을 사용하니 아주 깔끔하고 결국에는 이러한 협업 방식이 확실히 효율적이고 탁월하다는 생각을 하게 되었습니다.

### 프로젝트 운영에 개선이 필요하다고 생각하는 점

아무래도 C++이라는 언어에 대해서 좀 더 밀도 있게 탐구를 했었으면 더 프로젝트를 진행할 때 효율적인 코드를 작성할 수 있을 것 같았다고 생각했습니다. 또한, 프로그래밍을 하면서 다른 사람이 잘 이해할 수 있는 코드도 중요하다는 것을 느꼈습니다. 아무래도 주석을 다는 습관이 안



 <b>국민대학교 컴퓨터공학부 Snake-Game I</b>	결과보고서		
	프로젝트 명	Snake-Game	
	팀 명	팀1-6	
	문서 정보	Version 1.1	2020-JUN-26

잡혀 있다 보니, Project Manager가 코드를 읽는데 불편함을 느꼈을 것 같습니다. 이를 통해, 프로그래밍 언어를 좀 더 심도 있게 탐구하고, 프로그래밍을 할 때 다른 협업자가 이해하기 쉬운 변수명과 주석과 같은 간단한 설명들을 달아줘 원활한 협업을 진행해야겠다고 느꼈습니다.

## 팀원 평가

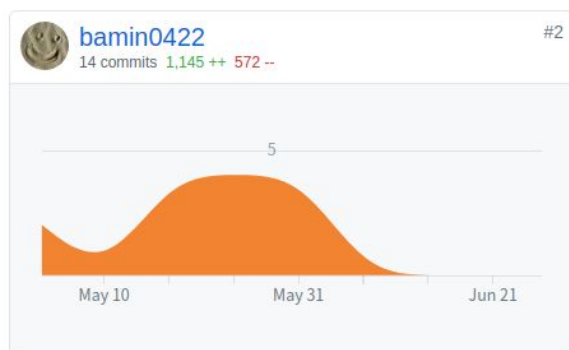
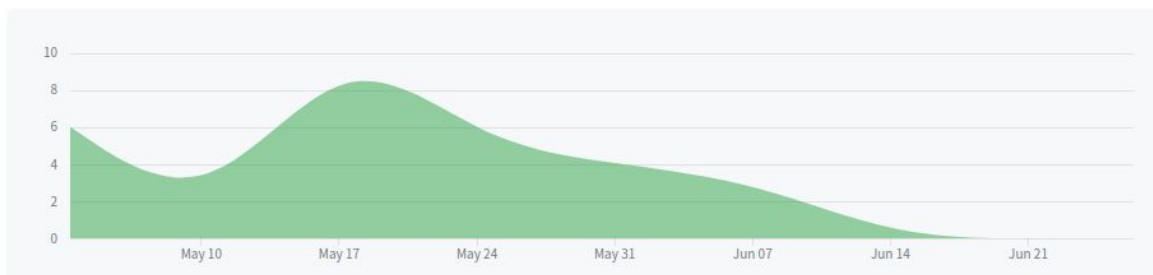
Project Manager가 처음 작성한 구조를 보고 감탄할 수 밖에 없었습니다. 이렇게 기능별로 자세하고 객체지향적으로 코드를 작성할 수 있구나하고 느꼈습니다. 또한, Pull Request를 작성했을 때, 빠른 feedback으로 코드를 작성할 때 도움이 많이 되었습니다. 마지막으로 Project Manager 덕분에 협업의 재미와 github로 효율적으로 협업하는 방법을 배울 수 있어서 너무 좋았습니다. 여태껏 팀 Project를 하면서 이렇게 재밌고, 열정적이게 했던 적이 없었는데 좋은 Project Manager를 만나서 snake-game을 완성할 수 있었습니다.


## 기여도

May 3, 2020 – Jun 26, 2020

Contributions: Commits ▾

Contributions to master, excluding merge commits



 <b>국민대학교 컴퓨터공학부 Snake-Game I</b>	<b>결과보고서</b>		
	<b>프로젝트 명</b>	Snake-Game	
	<b>팀 명</b>	팀1-6	
	<b>문서 정보</b>	Version 1.1	2020-JUN-26

May 26, 2020 – June 26, 2020

Period: 1 month ▾

#### Overview

14 Active Pull Requests

3 Active Issues

14

Merged Pull Requests

0

Proposed Pull Requests

1

Closed Issue

2

New Issues

Excluding merges, **2 authors** have pushed **11 commits** to master and **35 commits** to all branches. On master, **0 files** have changed and there have been **0 additions** and **0 deletions**.



14 Pull requests merged by 2 people

Merged

#32 **Improve all** 8 days ago

Merged

#31 **Add scene for UI** 9 days ago

Merged

#27 **Fix Score and Mission in Format** 9 days ago

Merged

#24 **[Feature/scoreboard] Add scoreBoardView** 10 days ago

Merged

#23 **[Feature/gate-fix] Fix gate errors** 11 days ago

Merged

#22 **[Feature/gate] Fix errors** 12 days ago

Merged

#21 **[Feature/manager fix] Fix MapManager** 13 days ago

Merged

#20 **Add MapManager for TextFile Map** 13 days ago

Merged

#15 **Fix Item, ItemManager for integrated management** 16 days ago

Merged

#14 **Add GetElapsedTime to project structure** 16 days ago

Merged

#13 **Feature/format** 22 days ago

Merged

#11 **Fix Scene Logic** 25 days ago

Merged

#10 **Add Item.h, Fruit.h, Poison.h for ItemManager** 25 days ago

Merged

#8 **Feature/item manager** 29 days ago

1 Issue closed by 1 person

Closed


#16 **[TO-DO]** 9 days ago

Opened

#33 **[Report] 레포트 작성과 분담** 16 hours ago

Opened

#25 **[PR #24] Improvement, TO-DO** 11 days ago

 <b>국민대학교</b> <b>컴퓨터공학부</b> <b>Snake-Game I</b>	<b>결과보고서</b>		
	<b>프로젝트 명</b>	Snake-Game	
	<b>팀 명</b>	팀1-6	
	<b>문서 정보</b>	Version 1.1	2020-JUN-26

## 4 참고 문헌

<https://github.com/parkgeonhu/Where-is-Tipy> // 프로젝트 구조를 짤 때, 전에 direct X로 게임 만들던 구조가 생각나 그것을 베이스로 구조를 짰습니다.

<https://widian.tistory.com/58> //ncurses 라이브러리를 사용할 때 모르는 함수가 있으면 여기에서 예시를 확인하고 작업했습니다.

<https://www.cplusplus.com/> // c++ 코딩할 때 모르는 stl 라이브러리가 있으면 검색해보았습니다. 특히 vector를 활용할 때 좋았습니다.


<https://doitnow-man.tistory.com/98> // 세그먼테이션 오류가 날 때 gdb 디버깅 검색

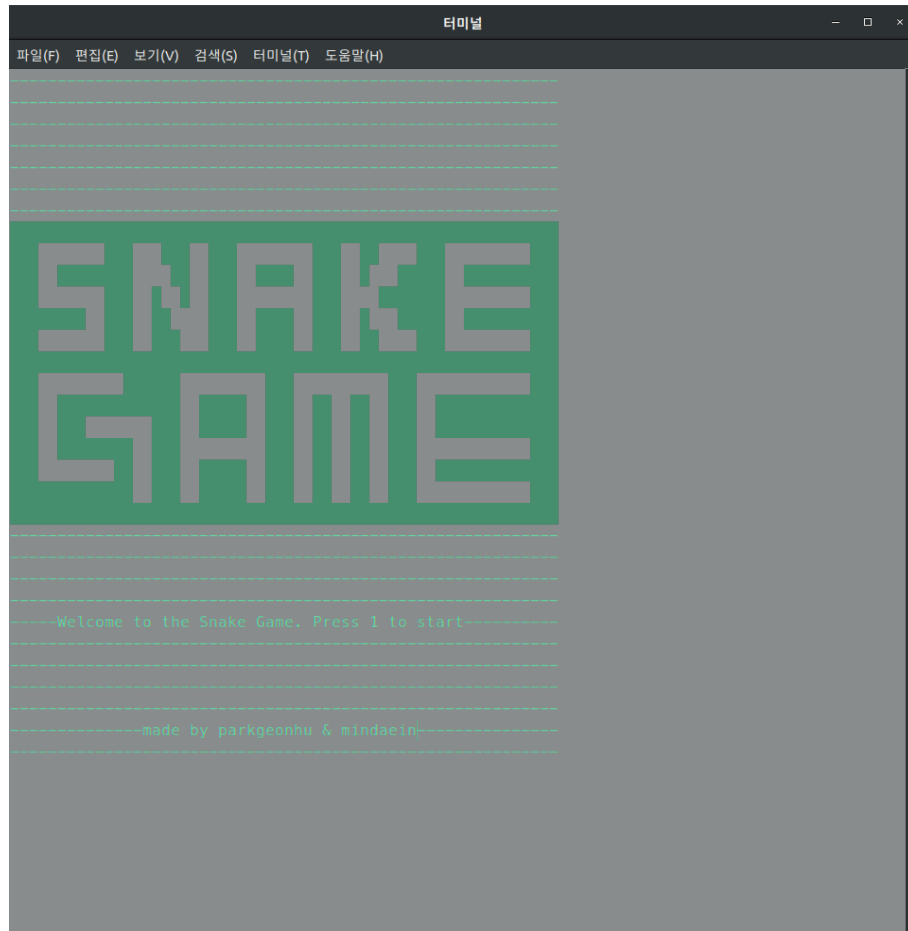
## 5 부록

### 5.1 사용자 매뉴얼

#### 실행

아래 5.2절을 따라서 먼저 설치를 끝냅니다. 설치가 끝나고 나서 make 명령어를 실행하면 snake-game이 아래와 같이 실행됩니다. 숫자 1을 누르면 게임이 실행됩니다.

 <b>국민대학교 컴퓨터공학부 Snake-Game I</b>	<b>결과보고서</b>		
	<b>프로젝트 명</b>	Snake-Game	
	<b>팀 명</b>	팀1-6	
	<b>문서 정보</b>	Version 1.1	2020-JUN-26



## 방법

- 1) G(Growth Item)에 H(snake-head)가 닿으면, G를 1개 획득하고 snake의 길이는 1 증가, Score는 10점 상승합니다.
- 2) P(Poison Item)에 H(snake-head)가 닿으면, P를 1개 획득하고 snake의 길이는 1 감소, Score는 5점 감소합니다.
- 3) ?(Gate)는 한 쌍으로 ?(Gate)에 H(snake-head)가 닿으면, 다른 ?(Gate)로 나옵니다.
- 4) 제한시간 안에 Mission에 있는 Growth, Poison, Length, Gate이 충족해야 다음 stage로 넘어갑니다. (충족하지 못할 경우 Game Over)
- 5) Game Over일 경우, y를 누르면 재시작할 수 있습니다.

## 5.2 설치 방법

<https://github.com/parkgeonhu/Where-is-Tipy> 로 이동하여 clone합니다.

프로젝트 최상단에서 make 명령어를 실행하면

all:

```
g++ src/GameScene.cpp src/GameOverScene.cpp src/Snake.cpp src/Stage.cpp src/myFunction.cpp
src/ItemManager.cpp src/WaitingScene.cpp src/Item.cpp src/main.cpp -lncurses -o /tmp/a.out && /tmp/a.out
```

이런 형식으로 실행됩니다.

 <b>국민대학교</b> <b>컴퓨터공학부</b> <b>Snake-Game I</b>	<b>결과보고서</b>		
	<b>프로젝트 명</b>	Snake-Game	
	<b>팀 명</b>	팀1-6	
	<b>문서 정보</b>	Version 1.1	2020-JUN-26