CSED231-01

객체지향프로그래밍

Team Project Report

Final Team Project

[POSTECH Survival]

Team 8

엄태강 (tkeom0114)

김희연 (heeyeona)

김정원 (jeongwon96)

1. Introduction

객체 지향 프로그래밍(object-oriented programming)이란 컴퓨터 프로그램을 '명령어의 나열'로 바라보는 시각에서 벗어나, '객체'라는 여러 개의 독립된 단위가 모여있는 형태로 구현하고자 하는 컴퓨터 프로그래밍 패러다임이다. 각각의 객체들이 서로 유기적으로 상호작용하면서 하나의 프로그램을 구성하는 이러한 형태는 '게임'이라는 장르를 구현하기에 아주 적합한 체계이다. 간단한 RPG(role playing game) 게임을 예로 들어보면, 게임 내부에 존재하는 모든 것들(지형, 아이템, 몬스터, NPC 등)을 개별적인 객체로 취급하여 하나의 독립적인 대상으로 다룰 수 있으며 그들이 가진 개별적인 특성이나 다른 객체와의 상호작용을 구현 해 줌으로써 게임이 진행되는 거대한 시스템을 만들 수 있다. 여기에, 사용자(user)를 캐릭터(character)라는 새로운 형태의 객체로 해당 시스템 내에 구현 해 놓는다면, 프로그램과 사용자가 서로 상호작용 할 수 있는 형태의 프로그램이 만들어지게 되는 것이다.

우리는 이러한 점에서 착안하여, 객체 지향 프로그래밍을 활용한 새로운 형태의 턴(turn) 제 서바이벌 게임을 구상하였다. 매 턴 플레이어는 자신이 수행할 행동을 결정하여 체력이 0이 되지 않도록 해야하며, 그렇게 100턴간 살아남는 것을 게임의 목표로 한다. 특히 단순한 서바이벌이 아니라 '포스텍'에서의 졸업을 목표로 한다는 컨셉을 게임 전반에 적용하여, 플레이어가 느끼는 재미나 익숙함을 끌어올림과 동시에 성공적인 졸업을 위한 계몽적인 의미를 담을 수 있도록 노력하였다.

2. Program Design

(1) Class 구성 및 전역변수

본 프로그램의 구현을 위한 class hierarchy는 **Figure 1**과 같다.

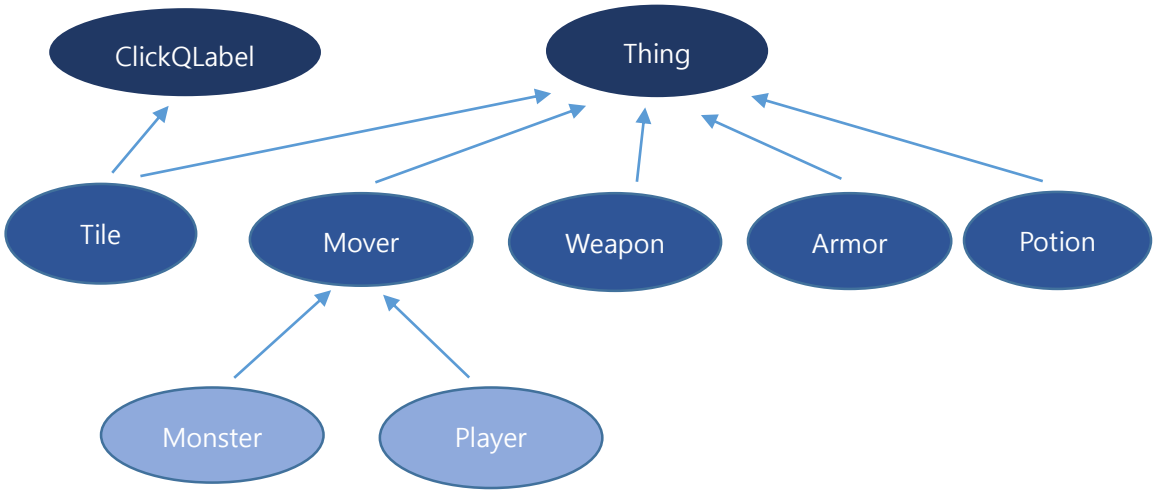


Figure 1. Class Hierarchy for POSTECH Survival

Thing은 본 프로젝트의 base class로써, 좌표를 가지는 모든 object(타일, 몬스터, 플레이어, 무기, 방어구, 포션)를 포괄하는 class이다. Thing은 아래와 같이 구현되어 있다.

```
class Thing
{
public:
    Thing(int _name,int _x, int _y){name=_name;x=_x;y=_y;} //생성될 칸을 정해주거나 플레이어가 대상을 소지하는 경우 사용하는 constructor
    Thing(int); //랜덤한 위치에 생성되는 경우 사용하는 constructor
    virtual ~Thing();
    int& getx();
    int& gety();
    int& getName();
    static bitset<64>& getOccupied();
    static int& getOccupiedNum();
private:
    static bitset<64> occupied; //(x,y)에 Thing Object 가 있는 경우 x+8*y 번째 자리에 1 을 씀
    static int occupiedNum; //화면에 생성되어 있는 Thing Object 의 개수
    int name; //enum 형태의 변수 사용
    int x; //x 좌표(-1 의 경우 플레이어가 소지하고 있는 물품을 의미)
    int y; //y 좌표
};
```

ClickQLabel은 Label을 클릭할 수 있도록 하기 위해 만든 class이며, 타일이나 버튼 등을 구현하는데 사용되었다. ClickQLabel은 아래와 같이 구현되어 있다.

```
class ClickQLabel: public QLabel
{
    Q_OBJECT
public:
    ClickQLabel(QWidget *parent);
    virtual ~ClickQLabel();
    void mousePressEvent(QMouseEvent *event); //Object 를 클릭하면 mousePressed 라는 signal 을 emit 한다.
    void enterEvent(QEvent *event); //Object 에 마우스를 올리면 mouseEntered 라는 signal 을 emit 한다.
    void leaveEvent(QEvent *event); //Object 에서 마우스를 떼면 mouseLeaved 라는 signal 을 emit 한다.

private slots:
```

```
signals:
    void mousePressed();
    void mouseEntered();
    void mouseLeaved();
};
```

Mover는 몬스터나 플레이어와 같이 움직이는 것들을 나타내는 class이다. Mover는 아래와 같이 구현되어 있다.

```
class Mover: public Thing
{
public:
    Mover(int);
    int &getHP();
    virtual int &getDeal();
    virtual int &getDefense();
    void move(int,int); //정해준 칸으로 이동
    void move(); //인접한 칸 중 랜덤하게 이동 또는 플레이어를 따라가도록 이동함
private:
    int hp; //체력
    int deal; //공격력
    int defense; //방어력
};
```

Weapon은 무기를 나타내는 class이다. Weapon은 아래와 같이 구현되어 있다.

```
class Weapon: public Thing
{
public:
    Weapon(int,int,int);
    virtual int getDeal();
private:
    int deal; //weapon 의 공격력
};
```

Armor는 방어구를 나타내는 class이다. Armor는 아래와 같이 구현되어 있다.

```
class Armor: public Thing
{
public:
    Armor(int,int,int);
```

```
    virtual int getDefense();
private:
    int defense; //armor 의 방어력
};
```

Potion은 소비 아이템을 나타내는 class이다. Potion은 아래와 같이 구현되어 있다.

```
class Potion: public Thing
{
public:
    Potion(int,int,int);
    int getRecovery();
private:
    int recovery; //회복량
};
```

Monster는 Mover의 subclass로써, 몬스터를 나타내는 class이다. Monster는 아래와 같이 구현되어 있다.

```
class Monster: public Mover
{
public:
    Monster(int);
    int getReward();
private:
    int reward; //몬스터 처치 시 드랍되는 아이템의 enum
};
```

Player 또한 Mover의 subclass로써, 플레이어가 조작하는 캐릭터를 나타내는 class이다. Player는 아래와 같이 구현되어 있다.

```
class Player: public Mover
{
public:
    Player(int);
    Weapon* &getWeapon();
    Armor* &getArmor();
    Potion* &getPotion(int);
    int &getLife();
private:
    Weapon *weapon; //착용 무기
    Armor *armor; //착용 방어구
```

```
    Potion *potions[4]; //소지중인 포션, 슬롯 4 개
    int life; //무은재 캐릭터의 경우 체력이 0 이어도 5 턴간 생존가능, 따라서 life variable 을 두어 상황을 관리하고자 함
};
```

게임의 구현을 위한 모든 object의 이름은 아래의 enum type으로 정의되어 있다.

```
enum{
    NONE=0,
    MATH,
    BIO,
    ELECTIC,
    COMPUTER,
    NONE_MONSTER,
    MATH_MONSTER,
    BIO_MONSTER,
    ELEC_MONSTER,
    COM_MONSTER,
    ALCOHOL,
    EARTHQUAKE,
    EXAM,
    WEAPON1,
    WEAPON2,
    WEAPON3,
    ARMOR1,
    ARMOR2,
    ARMOR3,
    DROP,
    CARRY,
    BREAK_LIMIT,
    POTION1,
    POTION2,
    POTION3,
    HP_FULL
};
```

mainwindow.cpp에서 아래와 같은 anonymous namespace를 사용해 해당 파일 내에서만 사용

할 수 있는 전역변수들을 만들었다.

```
namespace {
    int turn=100; //한 턴당 1 씩 감소하며 0 이되면 게임이 끝남
    Tile *tile[8][8] = { { nullptr } }; //게임이 진행될 보드를 구성하는 타일
}
```

또한, game이라는 namespace를 두어 다른 파일에서도 사용할 수 있는 전역변수들을 만들었다.

```
namespace game {
    list<Mover*> allMovers; //모든 몬스터와 플레이어들의 list
    list<Thing*> allItems; //보드위에 드랍된 모든 아이템들의 list
    Player *player; //현재 게임중인 플레이어
}
```

(2) Algorithm

본 프로그램은 다음의 알고리즘을 따라 작동한다.

1) main(main.c)

① chooseDialog(캐릭터 선택창)을 띄운다.

② connect를 통해 "gameStart" signal이 emit 되면, mainwindow.cpp의 "setGame"이 호출되도록 한다.

2) 캐릭터 선택(choosedialog.cpp)

① connect를 통해 game start라는 Label을 클릭하면 캐릭터 선택창이 뜨도록 한다.

② 캐릭터를 클릭하면 해당 캐릭터(Player)가 보드 상의 랜덤한 위치에 생성되면서 "gameStart" signal을 emit 한다. 이후의 동작도 connect를 기반으로 작동하게 한다.

3) 게임 세팅(void MainWindow::setGame())

① 배경화면과 Tile을 출력한다.

② Player의 속성과 동일한 Monster 10마리를 생성한다.

③ printAll()을 호출하여 Player에 관한 정보, 보드 위의 Player 및 Monster들을 출력한다.

④ connect를 통해 포션 창을 클릭하면 소지한 Potion을 사용할 수 있게 만든다.

⑤ connect를 통해 Tile을 클릭해 Player를 이동시킬 수 있게 한다.

4) 게임 진행 (void Tile::mousePressEvent (QMouseEvent *event), void MainWindow::nextTurn())

① user가 클릭한 Tile이 현재 Player의 바로 옆이고, Monster가 없는 Tile인 경우 이동하면서 "nextTurn"을 호출한다.

② Monster들이 Player와 동일한 메커니즘으로 한 칸씩 이동한다.

③ Player가 현재 위치의 상하좌우 Tile에 위치한 Monster에게 공격을 가하고, 이후 Monster들도 공격 범위의 Player를 공격한다. (allMovers를 순회하면서 fight(Monster*)를 호출한다)

④ allMover를 순회하는 과정에서 체력이 0인 Monster들은 보드에서 제거되고 아이템이 드랍된다. (allMover를 순회하면서 dropItem(Monster*, bool)을 호출한다)

⑤ 남은 턴이 30턴일 경우 기말고사(보스 Monster)를 소환한다.

⑥ 진행된 턴이 10턴 이하일 경우 Player의 속성에 맞는 Monster 만을, 그리고 진행된 턴이 40턴 이상일 경우 지진(특수 Monster)을 포함하도록 하여 0~3마리의 Monster를 랜덤하게 생성한다.

⑦ takeItem()을 호출하여 Player가 위치한 칸의 아이템을 줍는다.

⑧ Player의 체력, life, turn을 체크해 게임의 종료 여부를 확인한다.

⑨ 남은 turn이 0이 되었는데 기말고사의 체력이 절반 이상 남아있는 경우, 체력을 1 감소시킨다.

⑩ 게임이 끝나야 하는 경우 체력에 따라 엔딩 창(EndingWidget)을 출력한다.

⑪ 게임이 끝나지 않은 경우 Player가 "BIO(생명과)"라면 체력을 0.1 상승시키고, "NONE(무은재)"인 경우 체력이 0이 아니라면 life를 5로 재설정한다.

⑫ printAll()을 호출하여 변경된 모든 정보를 화면에 출력한다.

5) 화면 출력(printAll() (mainwindow.cpp))

① 주변 Monster information을 띄우는 창을 전부 지운다.

② 모든 Tile 위의 그림을 지운다. (tile[i][j]->clear())

③ QString strings[]에 strings[name]이 각 enum에 해당하는 이름이 들어가도록 저장한다. (ex, string[MATH]="수학과")

④ 남은 턴 수, Player의 속성, 체력, 공격력, 방어력, 소지 아이템(Weapon, Armor, Potion)을 맵 왼쪽에 모두 출력한다.

⑤ allMover를 순회하면서 모든 Monster와 Player를 해당 위치에 출력한다.

⑥ allItems를 순회하면서 모든 아이템을 해당 위치에 출력한다.

⑦ allMover를 순회하면서 Player 주변 Monster의 정보를 맵 오른쪽에 출력한다.

⑧ this->show()를 통해 mainwindow 전체가 보이도록 한다.

(그림 출력의 경우 printLabel(QLabel *parent, int type)을 사용하는데, 해당 Label에 enum type의 name(class Thing의 element)을 입력 받아 object의 모습을 출력한다)

6) 전투(void fight(Monster* monster) (mainwindow.cpp))

① player->getDeal() – monster->getDefense() 로 Monster가 실제로 입는 피해량을 구한다.

② monster->getDeal() – player->getDegense() 로 Player가 실제로 입는 피해량을 구한다.

③ Player와 Monster가 같은 속성인 경우(monster->getName() – player->getName() == 5) Monster가 입는 피해량에 1.5를 곱하고, Player가 입는 피해량에 0.5를 곱한다.

④ 0보다 작은 값인 경우 0으로 바꾼다.

⑤ Player의 공격이 먼저 처리되며, 공격 이후 Monster의 체력이 0보다 작으면 Player는 피해를 입지 않는다. Monster의 체력이 0보다 큰 경우에만 Player가 피해를 입는다.

7) 아이템 드랍(void dropItem(Monster* monster, bool drop) (mainwindow.cpp))

① drop == false 이거나 Monster가 드롭 아이템을 가지고 있지 않은 경우(monster->reward == -1), Monster가 있던 위치의 Thing::occupied bit를 0으로 만들고 Thing::occupiedNum을 1 감소시킨다.

② 보상이 존재하는 경우, monster->getReward()를 통해 보상의 enum type을 받아와 해당 item에 대한 메모리를 할당한다.

③ 새로이 생성된 item을 allItems에 추가한다.

8) 아이템 획득(void takeItem() (mainwindow.cpp))

① allItem을 순회하면서 Player가 위치한 Tile에 드랍된 아이템이 있는지 확인한다.

② 해당 아이템이 소지한 Weapon이나 Armor보다 공격력/방어력 수치가 높은 아이템인 경우, 원래 Player가 소지한 아이템을 할당 해제하고 getWeapon()이나 getArmor()를 통해 드랍된 아이템을 추가한다.

③ 아이템이 Potion인 경우, player->getPotion(i)를 통해 player->potions[i] 중 비어있는 칸에 해당 아이템을 추가하고 비어있는 칸이 없는 경우 드랍된 아이템을 할당 해제한다.

9) 아이템 사용(void usingItem(Potion* &item) (mainwindow.cpp))

① item == nullptr 인 경우, 바로 return 한다.

② item->getRevovery() != 0 이면, 해당하는 회복량만큼 Player의 체력을 증가시키고 return 한다.

③ item->getName() == DROP 이면 allMover를 순회하면서 모든 Monster를 없애고, dropItem((*iterPos),false)를 호출하여 아이템이 드랍되지 않도록 한다.

④ item->getName() == CARRY 이면 Player 주위 1칸 이내(총 8칸)의 Monster를 모두 없애고, 아이템이 정상적으로 드랍되게 한다.

⑤ item->getName() == BREAK_LIMIT 이면 Player 주위 2칸 이내(총 24칸)의 Monster를 모두 없애고, 아이템이 정상적으로 드랍되게 한다.

⑥ item을 할당해제한다.

⑦ item을 nullptr로 만든다.

10) 몬스터의 움직임(void Mover::move())

① Player가 "ELECTIC(전자과)"인 경우, 한번도 공격받지 않은 일반 Monster(기말고사 제외)는 다른 object와 겹치지 않고(Thing::occupied가 0인 bit) 원래 위치에서 인접한 위치로 1칸 이동하거나, 아예 이동하지 않는다.

② 그 외의 경우(!=ELECTIC), Monster는 Player를 쫓아간다. 그 알고리즘은 다음과 같다.

③ 4개의 방향 중 원래 위치에 비해 Player와의 거리가 줄어들면서 실제로 이동이 가능한 위

치를 선정해, Player와의 거리가 최소가 되는 위치로 이동한다. 이동할 수 있는 위치를 검사하는 과정에서, 가능한 위치는 vector에 저장된다.

④ Player와의 거리가 줄어드는 위치 중 이동할 수 있는 위치가 아무 곳도 없으면 이동 가능한 방향 중 아무 방향이나 랜덤하게 선택한다.

⑤ 이동할 수 있는 방향이 없는 경우에는 이동하지 않는다.

3. Program Details (Game Rule)

(1)개요

플레이어는 포스텍에 입학한 신입생으로써, 대학생활이 진행되는 100턴 간 학점이 0이 되지 않도록 살아남아 무사히 졸업하는 것이 게임의 목표이다. 매 턴마다 플레이어는 캐릭터를 기준으로 상하좌우의 비어있는 칸들 중 하나의 칸으로 이동할 수 있으며, 이동 후 주변 칸에 위치한 몬스터에게 공격을 가하게 된다. 플레이어의 학점에 영향을 주는 몬스터인 '과제' 역시 동일한 이동 및 공격패턴을 가지고 있다. 필드에는 랜덤하게 과제가 생성되며, 70턴 째에는 확정적으로 특별한 과제가 생성된다. 단 게임의 초반부(~10턴)에는 캐릭터와 동일한 속성의 과제만 생성되어 보다 쉽게 과제를 처리할 수 있다. 과제를 처치 할 경우 아이템이 드랍되며, 아이템이 존재하는 칸으로 이동하게 되면 해당 아이템을 획득할 수 있다. 게임 종료 시까지 특별한 과제의 체력을 절반이상 줄이지 못 한 경우 학점이 1점 감점된다.

(2) 필드

게임은 8*8 규격의 체크보드에서 진행된다. 플레이어 캐릭터는 랜덤한 위치에서 생성되어 게임을 시작한다.

(3) 캐릭터

총 5종류의 캐릭터 중 한가지를 선택할 수 있다. 각 캐릭터는 '학과' 라는 속성을 가지며, 그에 따라 고유한 능력을 가지고 있다. 플레이어의 학과가 과제와 일치하는 경우, 가하는 공격력은 1.5배 증가하고 받는 피해량은 0.5배로 감소한다. 각 캐릭터의 기본 학점 및 공격력, 방어력 수치는 모두 동일하다.

기본 수치

학점: 4.3

공격력: 0

방어력: 0

캐릭터 별 능력

무은재: 체력이 0이 되어도 5턴 동안 생존

수학과: 공격력 1.5배 증가

생명과: 매 턴마다 학점 0.1 회복

전자과: 몬스터들이 플레이어에게 먼저 접근하지 않음

컴공과: 방어력 1.5배 증가

Figure 2. Character Design

(4)아이템

아이템은 하나의 과제를 처치할 때 마다 하나씩 랜덤하게 드랍된다. 아이템이 있는 자리로 이동함으로써 아이템을 주울 수 있다. 아이템은 줍지 않는 이상 영원히 필드에 존재한다. 몬스터는 아이템이 존재하는 자리로 이동하지 못한다.

(4-1). 무기

캐릭터는 기본적으로 1단계 무기를 장착한 상태에서 시작한다.

1단계: 필기 노트, 공격력 수치 +40

2단계: SMP 정리자료, 공격력 수치 +80

3단계: 선배가 숨겨뒀던 소스, 공격력 수치 +120

3단계 무기는 '지진' 몬스터를 통해서만 획득할 수 있다.

(4-2). 방어구

캐릭터는 기본적으로 1단계 방어구를 장착한 상태에서 시작한다.

1단계: 분반 야구잠바, 방어력 수치 +0.2

2단계: 과 야구잠바, 방어력 수치 +0.4

3단계: 포스텍 야구잠바, 방어력 수치 +0.6

3단계 방어구는 '지진' 몬스터를 통해서만 획득할 수 있다.

무기와 방어구의 경우, 더 좋은 단계의 아이템을 획득했을 때 자동으로 기존의 아이템이 사라지고 새로운 아이템이 장착된다. 더 낮은 단계의 아이템은 획득 시 해당 칸에서 사라지게 된다.

(4-3). 소모품

소모품은 총 4개까지 소지할 수 있으며, 인터페이스에서 클릭하면 사용된다. 4개의 소모품이 꽉 찬 상태에서 새로운 소모품을 획득할 경우, 해당 소모품은 사라진다.

드랍(W): 맵 위의 모든 과제가 삭제된다. 이때, 아이템은 드랍되지 않는다.

친구의 캐리: 캐릭터 주위 8칸 내 모든 과제가 삭제되며 아이템을 드랍한다.

두뇌 각성: 캐릭터 주위 24칸 내 모든 과제가 삭제되며 아이템을 드랍한다.

핫식스: 학점이 0.5 회복된다.

휴강 공지: 학점이 1 회복된다.

용돈 입금: 학점이 1.5 회복된다.

종강: 학점이 최대 수치(4.3)로 회복된다.

종강 아이템은 '기말고사' 몬스터를 처치했을 때 확정적으로 드랍된다.

Figure 3. Items Design (Left: Weapons & Armor, Right: Potion)

(5) 과제

기말고사를 제외한 일반 과제들은 매 턴마다 0마리에서 3마리까지 랜덤하게, 랜덤한 위치에서 생성된다. 기말고사는 70턴 째에 고정적으로 생성된다. 지진 몬스터는 40턴째부터 생성되기 시작한다.

정리증명(수학): 공격력 0.8, 체력 170

실험 보고서(생명): 공격력 0.8, 체력 170

어싸인(컴공): 공격력 0.8, 체력 170

회로설계(전자): 공격력 0.8, 체력 170

수강신청(무은재): 공격력 0.8, 체력 170

술의 유혹(무속성): 공격력 0.8, 체력 170

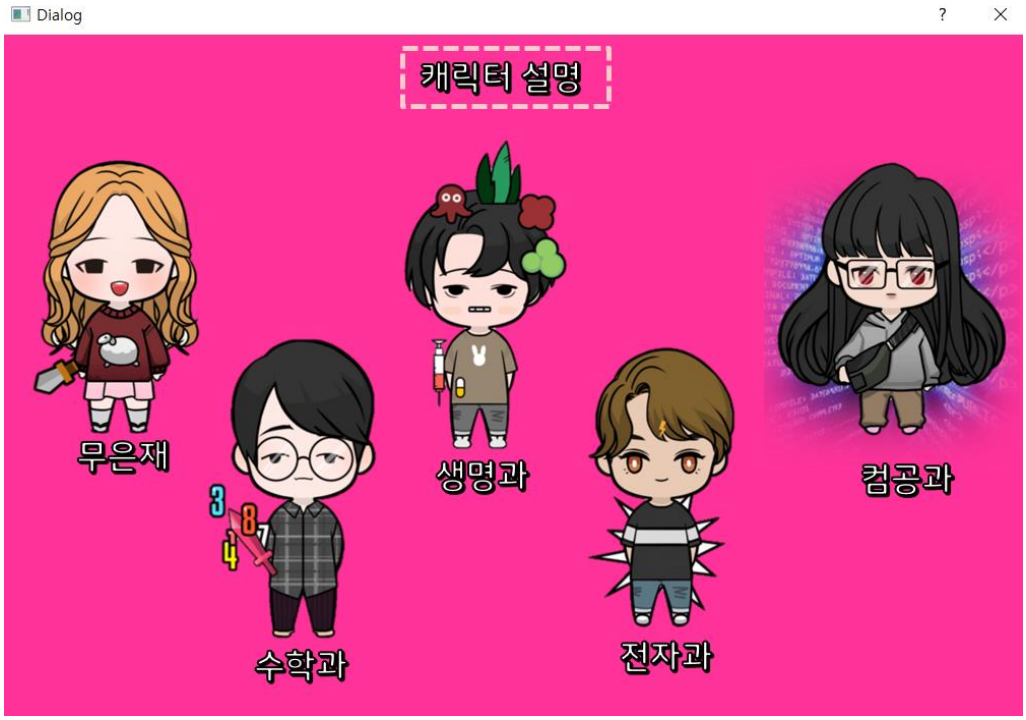
지진(무속성): 공격력 1, 체력 150, 상하좌우 2칸 내에 있을 시 공격받음(범위공격)

기말고사(무속성, 보스): 공격력 1.2 체력 1000, 방어력 50

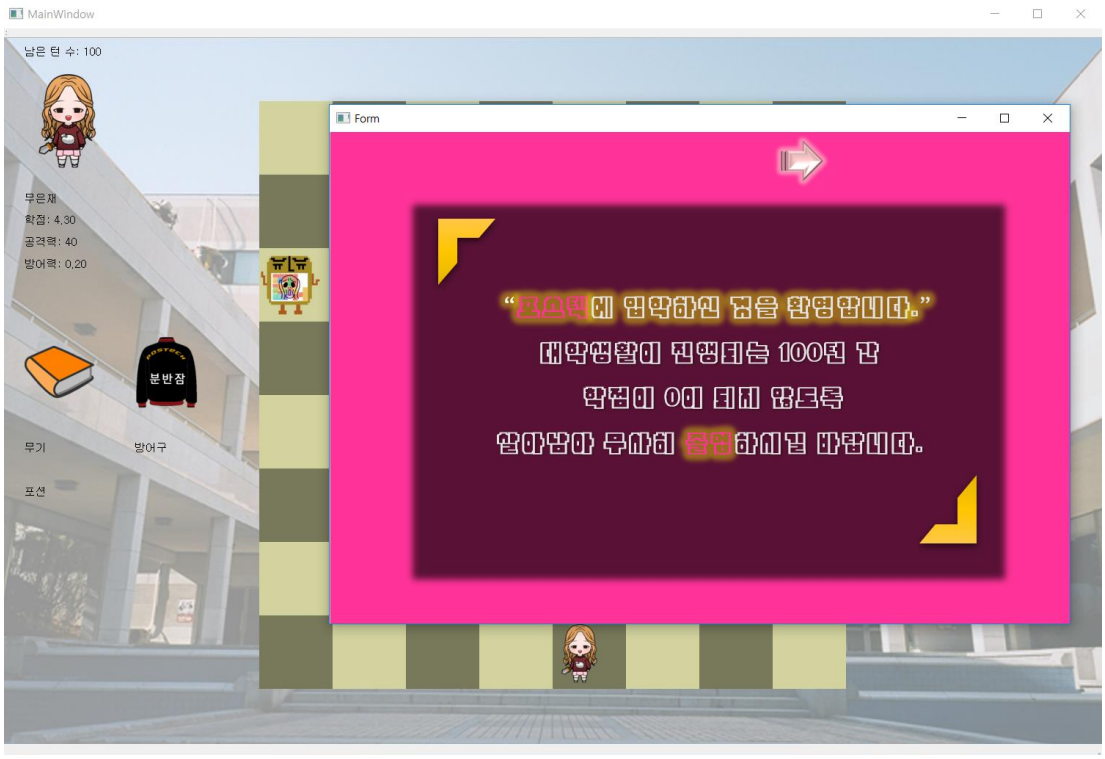
Figure 4. Monster Design

4. Program Running Examples

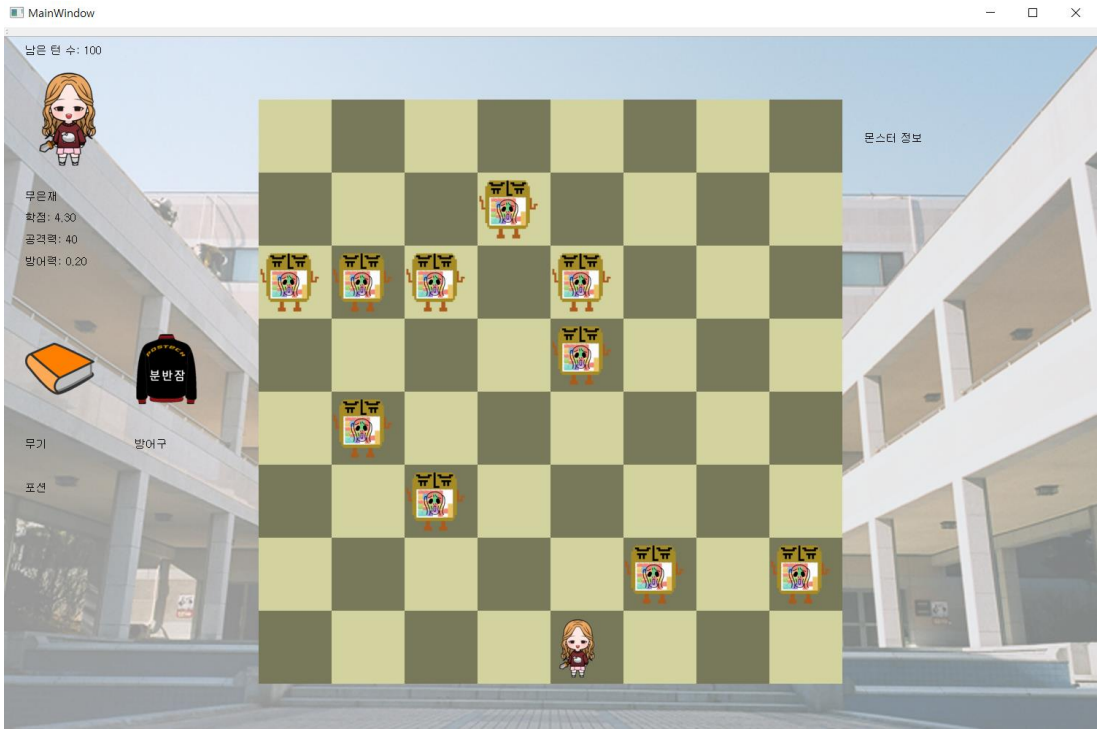
→ 게임 실행 시 시작 화면. GAME START 버튼을 누르면 캐릭터 선택창으로 이동.



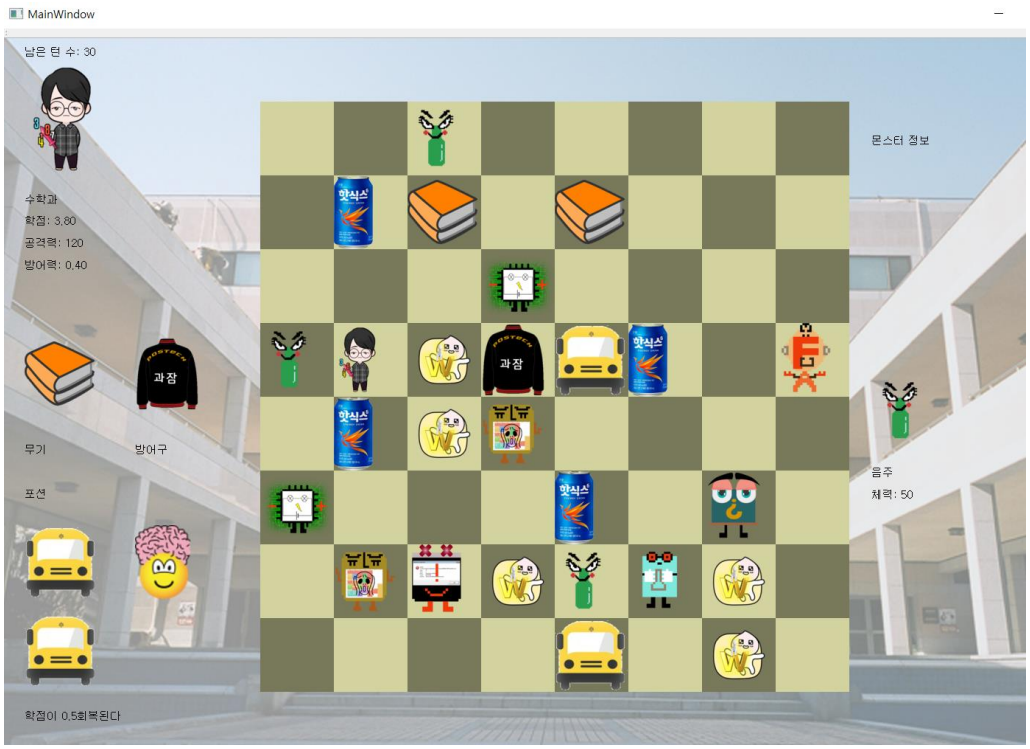
→ 캐릭터 선택창. 캐릭터 위에 마우스를 올리면 설명과 캐치프레이즈가 표시됨.



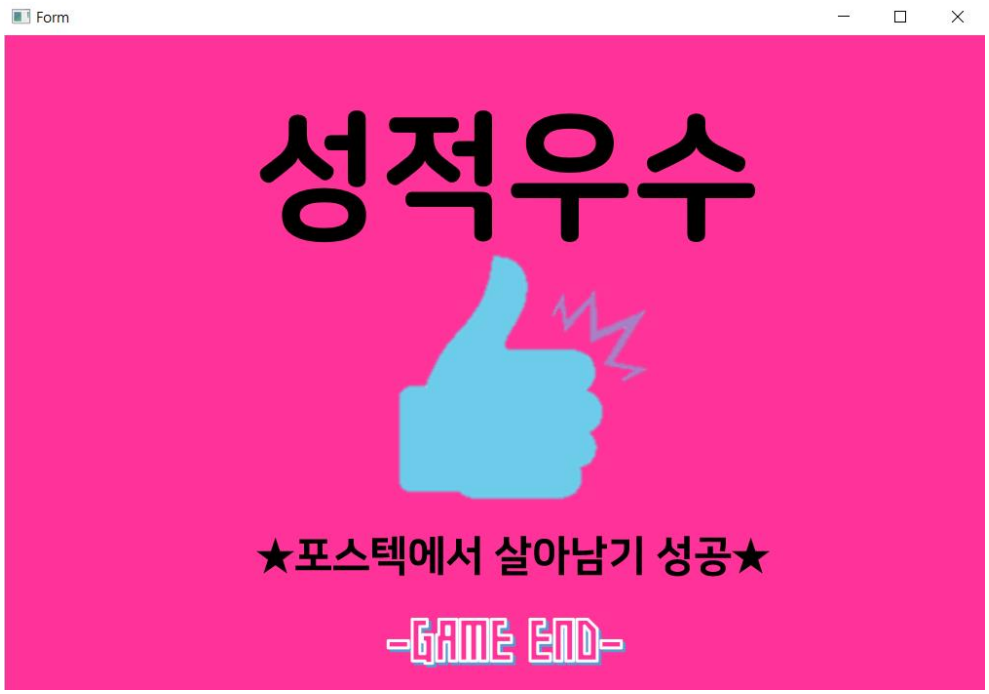
→ 게임 시작을 위한 필드 구성 후, 게임에 대한 간략한 설명이 표시되는 창이 출력됨.



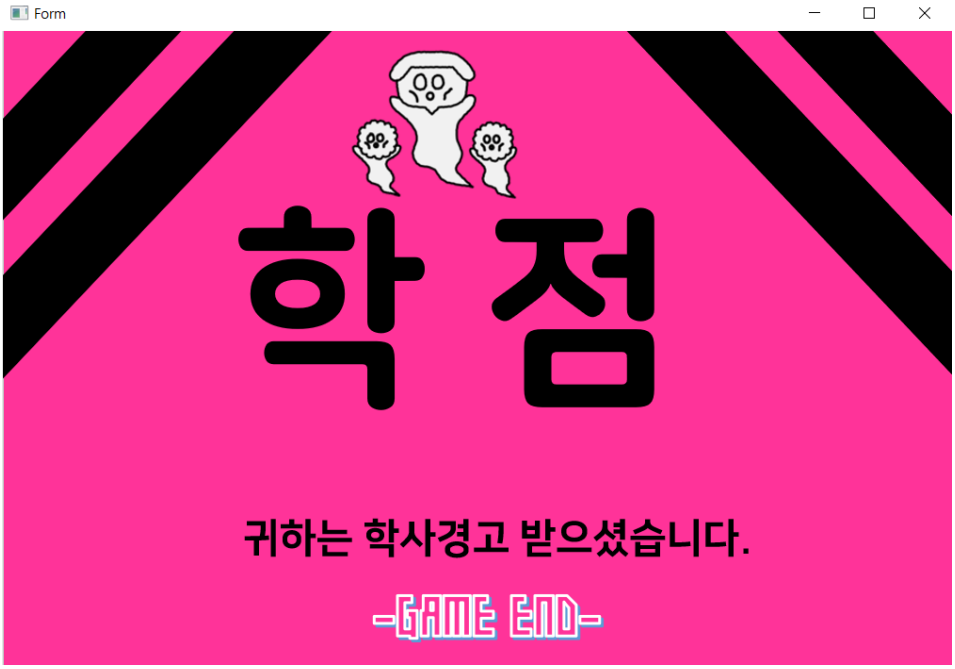
→ 게임 시작 직후. 캐릭터의 정보가 좌측에 표시되고 Player 주변(상하좌우)의 몬스터에 대한 정보가 우측에 표시됨.



→ 게임이 70턴 진행되었을 때(30턴 남았을 때), 기말고사 Monster(보드의 가장 우측)가 생성. 보유하고 있는 Potion의 현황이 좌측에 표시되며, 보드 위의 Potion에 마우스를 올리면 아이템에 대한 설명이 좌측 하단에 표시됨.



→ 남은 턴 수가 0이 될 때까지 살아남았을 때의 엔딩 화면



→ 학점이 0이 되었을 경우의 엔딩 화면

5. Discussion

본 프로젝트를 위한 프로그래밍 과정에서의 특이사항으로는 두가지를 꼽을 수 있는데, 하나는 STL의 list, bitset, vector를 사용했다는 점이고, 다른 한 가지는 virtual destructor를 사용하여 polymorphism을 구현했다는 점이다. 특히 destructor의 경우, list<Mover*> allMovers와 list<Thing*> allItems(존재하는 모든 Monster/Player 및 Weapon/Armor/Potion을 tracking하는데 쓰이는 list)의 원소를 제거할 때 메모리 할당 해제를 함께 해 주어야 하는데, virtual하게 destructor를 정의하지 않으면 destructor가 제대로 작동하지 않기 때문에 이와 같이 디자인하게 되었다.

처음에 전체적인 게임의 흐름에 대한 구상을 진행하던 단계에서는 별다른 고민 없이 필요한 object들의 class를 구성하고 적당히 이들간에 정보를 주고받도록 만들면 될 것이다라고 간단하게 생각했었지만, 실제 프로그래밍 단계에서는 생각만큼 쉽지 않았다. 프로그램 내에서 일어나는 모든 행동(Player/Monster/item 생성, Player/Monster 이동, 공격, item 사용, etc.)들에 대해서, 구상하는 대로의 결과가 이루어 질 수 있도록 선후 관계의 flow를 모두 고려해주어야 했음은 물론 수많은 예외 상황(bug)이 발생하지 않도록 하는 것 또한 매우 어려운 일임을 느낄 수 있었다. 특히 게임이라는 플랫폼의 특성상 user를 배려하는 장치(게임 설명 화면, 정보 표시, 결과창의 가독성 등)들에 대해서도 새롭게 고민해 볼 수 있었다는 점이 매우 긍정적이었다. 시간과 프로그램의 복잡성을 문제로 실제로 구현하지는 못하였지만 더욱 다양한 몬스터 및 아이템을 추가하거나, 혹은 점점 더 강력한 몬스터가 생성되는 상황에서 얼마나 오래 살아남는지를 경쟁하는 형태의 새로운 게임모드를 추가한다면 더욱 풍성하고 재미있는 게임이 될 것이라 기대한다.