1.class 구성 및 전역변수

Thing은 이 프로젝트의 base class이고, 좌표를 가진 모든 것(타일, 아이템, 몬스터, 플레이어)들을 포괄하는 class이다.

base class인 Thing는 아래와 같이 구성되어 있다.

class **Thing**//좌표를 가진 모든 것들의 base class(player, 무기, 방어구, 포션)

{

public:

**Thing**(int \_name,int \_x, int \_y){name=\_name;x=\_x;y=\_y;}//생성될 칸을 정해주거나 플레이어가 소유하는 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 Object가 있는 경우 x+8\*y번째 자리에 1을 씀

static int occupiedNum;//화면에 생성되어 있는 Thing Object의 개수

int name;//enum형태의 변수 사용

int x;//x좌표(-1의 경우 플레이어가 소지하고 있는 물품을 의미)

int y;//y좌표

};

ClickQLabel은 Label을 클릭할 수 있도록 만든 class이며 타일, 버튼 등을 구현하는데 사용되었다. ClickQLabel은 아래와 같다.

class **ClickQLabel** : public QLabel

{

Q\_OBJECT

public:

**ClickQLabel**(QWidget \*parent);

virtual ~***ClickQLabel***();

void ***mousePressEvent***(QMouseEvent \*event);//Object를 클릭하면 mousePressed라는 signal을 emit한다.

void ***enterEvent***(QEvent \*event); //Object에 마우스를 올리면 mouseEntered라는 signal을 emit한다.

void ***leaveEvent***(QEvent \*event);//Object에서 마우스를 떼면 mouseLeaved라는 signal을 emit한다.

private slots:

signals:

void **mousePressed**();

void **mouseEntered**();

void **mouseLeaved**();

};

Tile은 게임이 이루어지는 보드이고, 이 게임에서는 8\*8개의 tile이 사용되었다. 각 타일마다 좌표를 가지고 있고, 타일을 클릭해서 플레이어를 이동시켜야 하므로 ClickQLabel과 Thing을 상속받아 만들었다.

Tile은 아래와 같이 구성되어 있다.

class **Tile**: public ClickQLabel, public Thing

{

Q\_OBJECT

public:

**Tile**(QWidget\*,int,int,int);

void **tileDisplay**();//칸에 따른 타일의 색을 정해줌

void ***mousePressEvent***(QMouseEvent \*event);//타일을 클릭했을 때, 플레이어가 이동 가능한 칸인 경우 플레이어를 그 칸으로 이동시키고, proceedNextTurn이라는 signal을 emit해 다음 턴으로 넘어간다(이때 slot으로는 MainWindow::nextTurn을 사용한다)

private slots

signals:

void **proceedNextTurn**();

};

Mover는 움직이는 것들(몬스터, 플레이어)를 나타내는 class이다.

Mover는 아래와 같이 구성되어 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 구성되어 있다.

class **Weapon**:public Thing

{

public:

**Weapon**(int,int,int);

virtual int ***getDeal***();

private:

int deal;//무기의 공격력

};

Armor는 방어구를 나타내는 class이다.

Armor는 아래와 같이 구성되어 있다.

class **Armor**:public Thing

{

public:

**Armor**(int,int,int);

virtual int ***getDefense***();

private:

int defense;//방어구의 방어력

};

Potion은 소비 아이템을 나타내는 class이다.

Potion은 아래와 같이 구성되어 있다.

class **Potion**:public Thing

{

public:

**Potion**(int,int,int);

int **getRecovery**();

private:

int recovery;//회복량

};

Monster는 몬스터를 나타내는 class이며 Mover class를 상속받았다.

Monster는 아래와 같이 구성되어 있다.

class **Monster**:public Mover

{

public:

**Monster**(int);

int **getReward**();

private:

int reward;//드랍 아이템의 enum

};

Player는 플레이어를 나타내는 class이며 Mover class를 상속받았다.

Monster는 아래와 같이 구성되어 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

int life;//무은재의 경우 체력이 0이어도 5턴간 생존가능, 따라서 life를 둬서 상황을 관리함

};

모든 아이템의 이름은 아래의 enum type이다.

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 사용할 수 있는 전역변수를 만들었다.

namespace {

int turn=100;//한 턴당 1씩 감소하며 0이되면 게임이 끝남

Tile \*tile[8][8] = { { nullptr } };//게임이 진행될 보드를 구성하는 타일

}

그리고 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.알고리즘

1) main(main.c)

①chooseDialog(캐릭터 선택창)을 띄운다.

②connect를 통해 gameStart라는 signal이 emit되면 mainwindow.cpp의 setGame이 호출되도록 한다.

2)캐릭터 선택(choosedialog.cpp)

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

②캐릭터를 클릭하면 해당 학과의 플레이어가 랜덤한 위치에 생성되면서 gameStart를 emit한다. 캐릭터를 클릭한 이후의 동작도 connect를 통해 작동하게 한다.

3)게임 세팅(void MainWindow::setGame())

①배경화면과 타일들을 띄운다.

②자신의 속성에 맞는 몬스터 10마리를 생성한다.

③printAll()을 호출해 플레이어에 관한 정보, 맵 위의 플레이어, 몬스터 들을 출력한다.

④connect를 통해 포션창을 클릭하면 포션을 사용할 수 있게 만든다.

⑤connect를 통해 타일을 클릭해 플레이어를 이동시킬 수 있게 한다.

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

①클릭된 칸이 현재 플레이어의 바로 옆이고, 몬스터가 없는 칸인 경우 이동하면서 nextTurn을 호출한다.

②몬스터들이 한 칸씩 이동하면서, 플레이어의 상하좌우에 위치한 몬스터들에게 공격을 가하고, 몬스터들도 플레이어를 공격한다(allMovers를 순회하면서 fight(Monster\*)를 호출한다.)

③ allMovers를 순회하면서 체력이 0인 몬스터들을 맵에서 제거하고 아이템을 드랍한다. (allMovers를 순회하면서 dropItem(Monster\*, bool)호출한다.

④30턴 남은 경우 기말고사를 소환한다.

⑤3마리 이하의 몬스터를 랜덤으로 소환하고, 10턴 이하로 진행된 경우는 플레이어의 속성에 맞는 몬스터만 소환하고, 40턴 이상 진행된 경우는 지진 몬스터까지 소환할 수 있게 한다.

⑥takeItem()을 호출해 플레이어가 있는 칸의 아이템을 줍는다.

⑦플레이어의 체력, life, turn을 검사해 게임이 끝나야 하는지 확인한다.

⑧턴이 다되었고, 기말고사의 체력을 절반 이상 줄이지 않은 경우 학점을 1점 감점한다.

⑨게임이 끝나야 하는 경우 학점에 따라 엔딩창(EndingWidget)을 띄운다.

⑩게임이 끝나지 않은 경우 플레이어가 생명과면 학점 0.1을 회복하고, 무은재인 경우 학점이 0점이 아니면 life를 5로 되돌린다.

⑪printAll()를 호출해 변경된 모든 정보들을 화면에 나타낸다.

5)화면 출력(printAll()(mainwindow.cpp))

①주변 몬스터 정보를 띄우는 창을 전부 지운다.

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

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

④남은 턴 수, 플레이어의 학과, 체력, 공격력, 방어력, 소지 아이템(무기, 방어구, 포션)을 맵 왼쪽에 모두 출력한다.

⑤allMovers를 순회하면서 모든 몬스터와 플레이이들을 해당 위치에 출력한다.

⑥allItems를 순회하면서 모든 아이템을 해당 위치에 출력한다.

⑦allMovers를 순회하면서 플레이어 주변 몬스터의 정보를 맵 오른쪽에 출력한다.

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

(그림 출력의 경우 printLabel(QLabel \*parent, int type)을 사용하는데 해당 라벨에 enum type의 name(Class thing의 element)을 입력받아 object의 모습을 출력한다.)

6)전투(void fight(Moster\* monster)(mainwindow.cpp))

① player->getDeal()-monster->getDefense()로 몬스터에게 가해지는 데미지를 구한다.

② monster->getDeal()-player->getDefense()로 플레이어가 입는 피해를 구한다.

③플레이어와 몬스터가 같은 속성인 경우(monster->getName()-player->getName()==5) 플레이어가 가하는 데미지에 1.5배를 곱하고, 입는 피해에 0.5배를 곱한다.

④0보다 작은 값인 경우 0으로 바꾼다.

⑤플레이어가 먼저 공격을 가해 몬스터의 체력이 0보다 큰 경우에만 플레이어가 피해를 입는다.

7)아이템 드랍 (void dropItem (Moster\* monster, bool drop) (mainwind ow.cpp))

①만약 drop==false이거나, 몬스터가 드롭 아이템을 가지고 있지 않은 경우(monster->reward==-1) monster가 있는 위치의 Thing::occupied bit를 0으로 만들고 Thing::occupiedNum을 1감소시킨다.

②보상이 존재하는 경우 monster->getReward()로 보상의 enum type을 받아와 item을 메모리 할당해준다.

③item을 allItems에 추가한다.

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

①allItem을 순회하면서 플레이어가 위치한 칸에 드랍된 아이템이 있는지 확인한다.

②해당 아이템이 무기나, 방어구인 경우, 수치가 더 높은 아이템인 경우 원래 플레이어가 소지한 아이템을 할당 해제하고, getWeapon()이나 getArmor()를 통해 드랍된 아이템을 추가한다.

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

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

①item==nullptr인 경우 바로 return한다.

②item->getRecovery()!=0이면 해당 회복량만큼 체력을 올리고 return한다.

③item->getName()==DROP이면 allMovers를 순회하며 모든 몬스터를 없애고, dropItem ((\*iterPos),false)를 호출해 아이템이 드랍되지 않게 한다.

④item->getName()==CARRY이면 플레이어 주위 1칸이내 (총 8칸)의 몬스터를 모두 없애고, 아이템이 정상적으로 드랍되게 한다.

⑤ item->getName()==BREAK\_LIMIT이면 플레이어 주위 2칸이내 (총 24칸)의 몬스터를 모두 없애고, 아이템이 정상적으로 드랍되게 한다.

⑥item을 할당해제한다.

⑦item을 nullptr로 만든다.

10)몬스터 움직임 (void Mover::move())

①플레이어가 전자과가 이고, 한번도 공격받지 않은 일반 몬스터(기말고사를 제외한 몬스터)는 다른 object와 겹치지 않고(Thing::occupied가 0인 bit), 원래 위치에서 인접한 위치로 이동하거나 그대로 있는다.

②그 외의 경우에는 플레이어를 쫓아가고 자세한 알고리즘은 다음과 같다.

③4개의 방향 중 원래 위치에 비해 플레이어와의 거리가 줄어들고, 이동할 수 있는 위치(Thing::occupied가 0인 bit)중 플레이어와의 거리가 최소가 되는 위치로 이동한다. 이동할 수 있는 위치를 검사하는 과정에서 가능한 위치는 vector에 저장한다.

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

⑤만약 이동할 수 있는 방향이 없는 경우에는 이동하지 않는다.

3. 그 외 특이사항

1)이 프로젝트에서 STL을 사용하였으며, 그 중 list, bitset, vector를 사용했다.

2)virtual destructor를 사용해 polymorphism을 구현했고, 이를 사용한 이유는 list<Mover\*> allMovers, list<Thing\*> allItems의 원소를 제거할 때, 메모리 해제도 같이 해야하고, 이 과정에서 destructor가 virtual이 아니면 destructor가 제대로 작동하지 않기 때문이다.