객체 지향 프로그래밍

학기 프로젝트 최종 발표

제작팀 · 낭만 팀장 양준서
발표자 · 김영인

목차

Ⅰ. 프로젝트 소개

〔 콘솔 서바이벌 〕

Console Survival

프로젝트 소개

1인 생존 게임으로, 높은 점수를 기록하여 다른 플레이어와 경쟁하는 게임입니다.
플레이어는 정해진 움직임 횟수를 잘 활용하여 전략적으로 몬스터의 공격에서 살아남아야 합니다.

프로젝트 목표

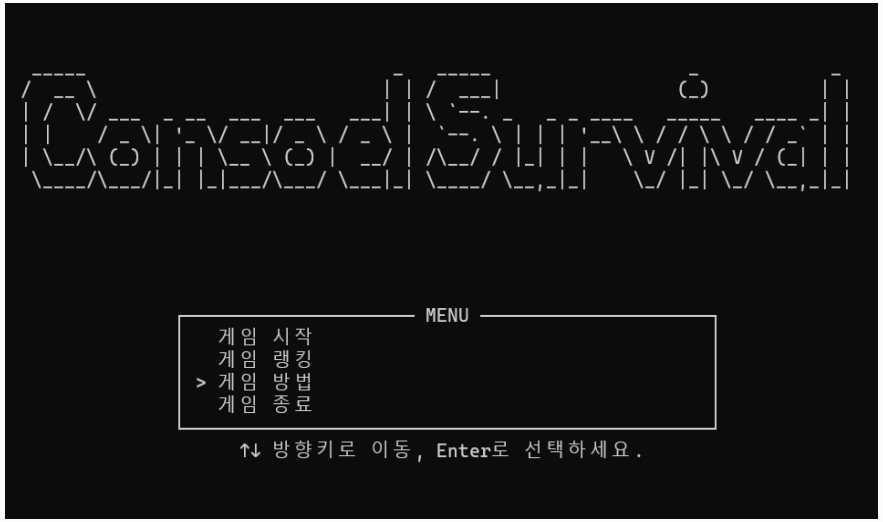
객체 지향 언어 개념을 학습하고, 팀원과 협업하며 하나의 프로젝트 결과물을 제작,
이 과정을 통해 개념 학습을 넘어 객체 지향 언어를 앞으로의 프로젝트에도 직접 활용할 수 있습니다.

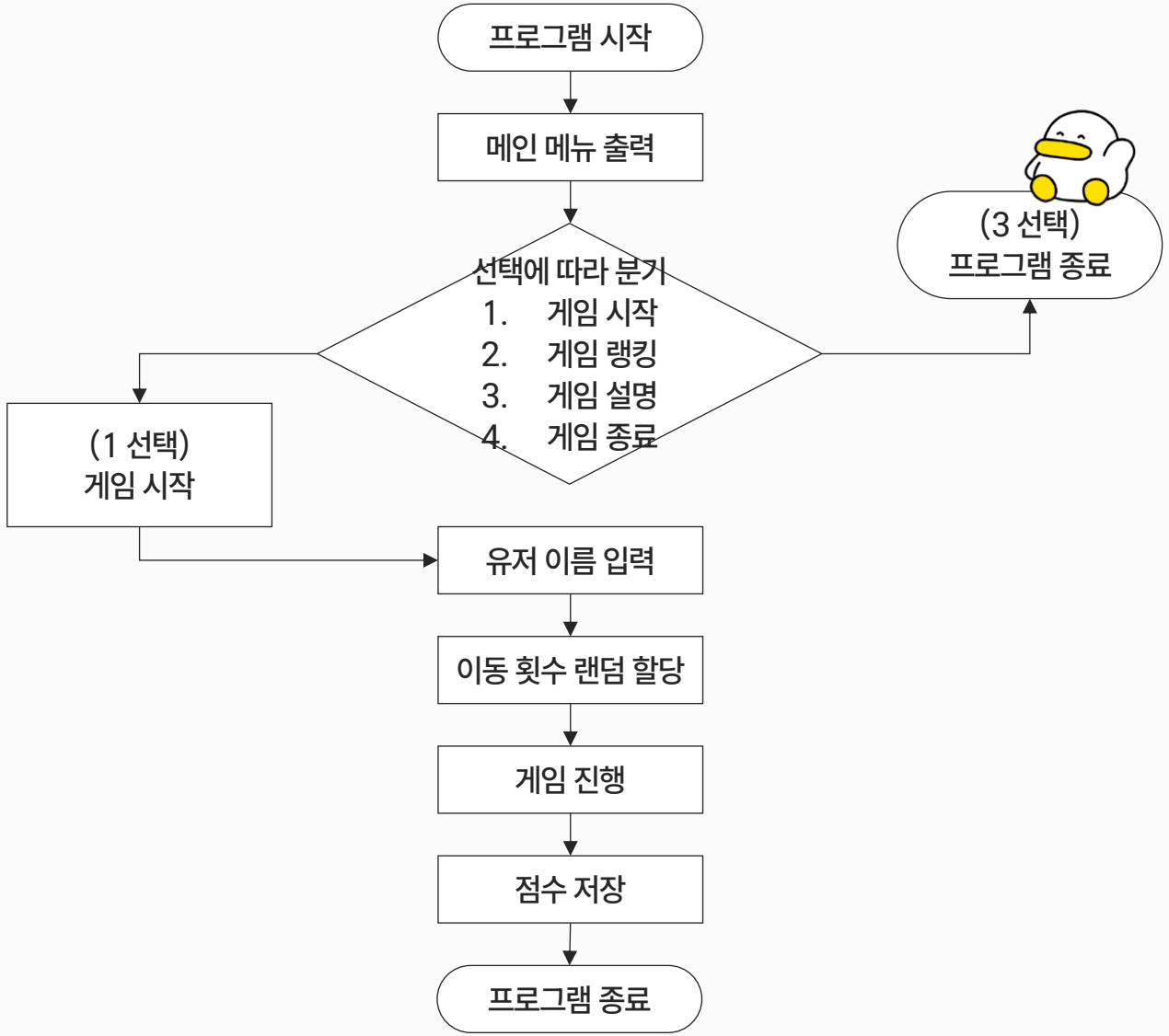
Ⅰ. 프로젝트 소개

프로그램 순서도

Flowchart

게임 플레이와 직접적으로 이어지는 부분만 간단하게 **순서도**로 정리하여 나타냈습니다.





Ⅱ . 주요 기능 설명

플레이어

Player

class Player {} 선언

플레이어 클래스 선언 후 클래스에
플레이어 **이동, 공격, 체력 등의 로직**을 모두 담아 관리합니다.

```
class Player {
protected:
    int x, y;           // 플레이어 위치
    int hp;             // 체력
    int moveCount;      // 움직임 횟수
    Direction lastDir;  // 마지막 이동 방향

public:
    int score;          // 점수
    int randNumber;     // 이동 횟수 랜덤 할당
    string name;        // 이름 입력

    Player();
    virtual ~Player() {}

    virtual void draw();
    virtual void remove();
    virtual void move(std::vector<Monster>& monsters);
    virtual void attack(std::vector<Monster>& monsters);

    virtual void decreaseHP();
    virtual int getHP();

    int getX() const;
    int getY() const;

    int getScore() const;
};
```

Ⅱ . 주요 기능 설명

플레이어

Player

class Player {} ‘ 플레이어 이동 ’

플레이어가 **마지막으로 입력한 방향키를 입력 받아** 해당 방향으로 자동 공격합니다.

이동 가능 횟수 랜덤 할당 후 **방향키 입력 횟수로 플레이어 이동 횟수를 처리합니다.**

```
void Player::move(std::vector<Monster>& monsters) {
    char in;
    draw();
    while (gamerun) {
        if (_kbhit()) {
            in = _getch();
            if (in == 0 || in == -32 || in == 224) {
                in = _getch();
                remove();
                bool moved = false;
                switch (in) {
                case 72: if (y > 6) { y--; moved = true; lastDir = UP; } break;
                case 80: if (y < 17) { y++; moved = true; lastDir = DOWN; } break;
                case 75: if (x > 13) { x--; moved = true; lastDir = LEFT; } break;
                case 77: if (x < 25) { x++; moved = true; lastDir = RIGHT; } break;
                }
                if (moved) {
                    moveCount++;
                    {
                        std::lock_guard<std::mutex> lock(output_mutex);
                        gotoxy(0, 3);
                        cout << "남은 이동 횟수: " << moveCount << "   ";
                    }
```

```
                    if (moveCount == randNumber) {
                        attack(monsters);
                        moveCount = 0;

                        // 새로운 랜덤 이동 횟수 재할당
                        randNumber = rand() % 3 + 3;
                        {
                            std::lock_guard<std::mutex> lock(output_mutex);
                            gotoxy(0, 1);
                            cout << "이동 횟수: " << randNumber << "   ";
                        }
```

Ⅱ. 주요 기능 설명

몬스터

Monster

class Monster {} 선언

플레이어 클래스와 같이 클래스 선언 후
몬스터 **생성, 활성화 여부, 체력 등의 로직**을 모두 담아 관리합니다.

```
class Monster {
public:
    int x, y;
    bool alive;
    int HP;
    string ch[4] = { "▲", "△", "■", "◆" };

    Monster(int startX = 12, int startY = 5, int hp = 3);  // 생성자

    void MonsterCreate(vector<Monster>& monsters);

    void PrintMonster(int x, int y, int select);
    void MonsterClear(int x[], int y[], int count, vector<Monster>& monsters);
};
```

Ⅱ . 주요 기능 설명

몬스터

Monster

class Monster {} ‘ 몬스터 생성 ’

몬스터는 **벡터로 위치 값을 입력** 받으며,
제한된 위치 내에서 자동으로 랜덤 생성됩니다.

```
while (gamerun) {
    int mcnt = rand() % 3 + 2;  // 2~4마리 생성

    for (int i = 0; i < mcnt; i++) {
        int dir = rand() % 4;
        int x, y;

        switch (dir) {
            // 위쪽
        case 0: y = MAP_TOP - 1; x = rand() % 20 + 25; break;
            // 아래쪽
        case 1: y = MAP_BOTTOM + 1; x = rand() % 20 + 25; break;
            // 왼쪽
        case 2: x = MAP_LEFT - 2; y = rand() % 10 + 5; break;
            // 오른쪽
        case 3: x = MAP_RIGHT + 5; y = rand() % 10 + 5; break;
        }
        xs[i] = x;
        ys[i] = y;
        dirs[i] = dir;

        bool positionTaken = false;
        for (auto& m : monsters) {
            if (m.alive && m.x == x && m.y == y) {
                positionTaken = true;
                break;
```

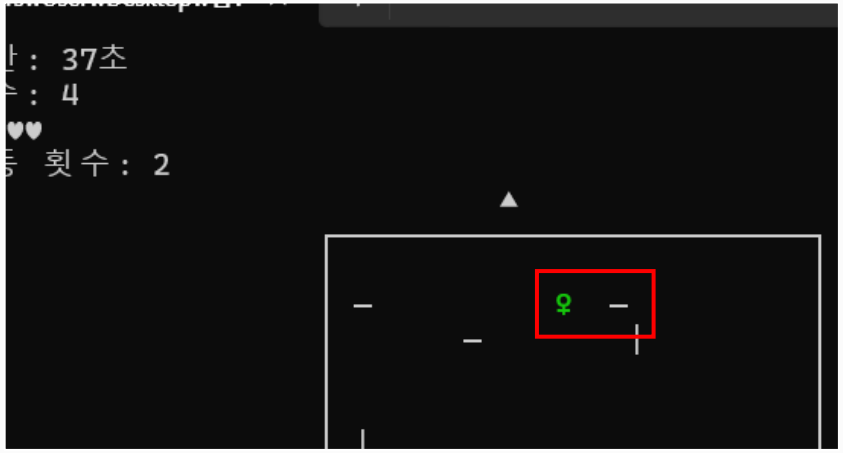
Ⅱ. 주요 기능 설명

몬스터

Monster

‘ 몬스터 공격 체크 ’

플레이어 위치와 몬스터 공격 위치를 입력 받아 **bool로 확인 및 충돌 처리(플레이어 체력, 점수 등)** 합니다.




```cpp
void UpdateAttacks(Player* player)
{
    for (auto& atk : attacks) {
        if (!atk.active) continue;

        {
            std::lock_guard<std::mutex> lock(output_mutex);
            gotoxy(atk.x, atk.y);
            std::cout << "  ";
        }

        atk.x += atk.dx;
        atk.y += atk.dy;

        if (atk.x == player->getX() * 2 && atk.y == player->getY()) {
            atk.active = false;
            player->decreaseHP();
```

```cpp
// 공격 객체 구조체
struct AttackObj {
    int x, y;          // 위치
    int dx, dy;        // 이동 방향
    std::string symbol;     // 출력 기호 (' | ' 또는 '—')
    bool active;       // 공격이 살아있는지 여부
};
```

Ⅱ. 주요 기능 설명

타이머

Timer

‘ 타이머 출력 ’

타이머 **출력 시작 시간과, 전체 시간(약 60초 할당),**
경과된 시간으로 계산합니다.


```
bool gamerun = true;
mutex output_mutex;

void TimerThread(ULONGLONG start_time, int total) {
    {
        lock_guard<mutex> lock(output_mutex);
        gotoxy(0, 0);
        cout << "남은 시간: ";
    }

    while (gamerun) {
        int elapsed = (GetTickCount64() - start_time) / 1000;
        int time_left = total - elapsed;
        if (time_left <= 0) {
            gamerun = false;
            break;
        }

        {
            lock_guard<mutex> lock(output_mutex);
            gotoxy(11, 0);
            cout << time_left << "초 ";
        }

        Sleep(200);
    }
}
```

Ⅱ . 주요 기능 설명

동시 실행 처리

Thread

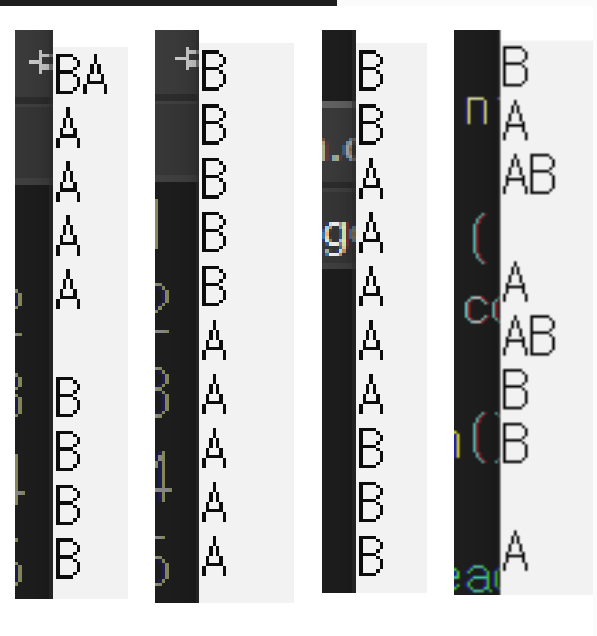
Thread

하나의 프로그램 안에서 동시에 실행

예를 들어 ,
메인 스레드인 플레이어의 공격 및 이동 입력을 출력하는 동안
보조 스레드인 타이머를 계속 실행하는 것처럼
여러 작업을 병렬로 수행할 수 있습니다.

```
void PrintA()
{
    for (int i = 0; i < 5; i++)
        cout << "A"<<endl;
}
void PrintB()
{
    for (int i = 0; i < 5; i++)
        cout << "B" << endl;
}
int main()
{
    thread t1(PrintA);
    thread t2(PrintB);

    t1.join();
    t2.join();
    return 0;
}
```



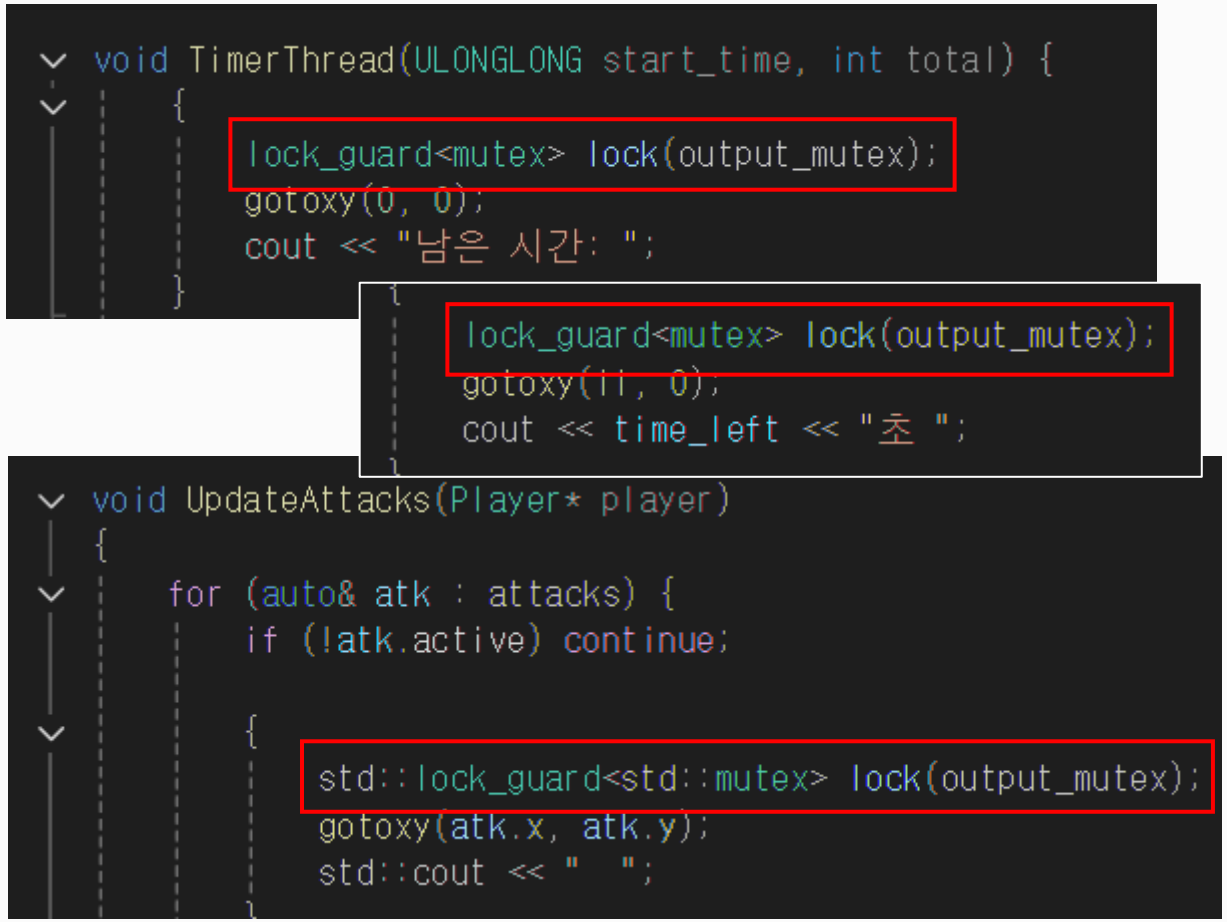
Ⅱ . 주요 기능 설명

출력 보호

mutex

여러 스레드에서 **cout을 동시에 출력할 때**
글자가 섞이는 등의 오류가 발생할 수 있습니다.

이때, mutex를 이용해 **한 번에 하나의 스레드에만 접근** 가능하도록
cout 코드를 잠시 잠가두는 기능을 수행합니다.



Ⅱ. 주요 기능 설명

게임 랭킹

Game ranking

Player 클래스에서 객체를 받아 데이터를 저장합니다.

scores.txt 파일에 데이터가 저장되며, 파일이 존재하지 않으면 파일을 새롭게 생성합니다.

점수가 높은 순서대로 랭킹이 출력되고, 동일한 점수라면 먼저 플레이 한 경우가 높은 랭킹으로 출력됩니다.

```
void saveScore(const Player& player) {
    ofstream file("scores.txt", ios::app);
    if (file.is_open()) {
        file << player.name << " " << player.score << "₩n";
        file.close();
    }
}
```

```
    string name;
    int score;
    while (file >> name >> score) {
        rankings.push_back({ name, score });
    }
    file.close();

    sort(rankings.begin(), rankings.end(), compareByScore);
```

```
> 게임 랭킹 <
------------------------------------------

1. unicorn - 5
2. KYI - 5
3. professor_Joe - 2
```

Ⅲ. 결과물 시연

결과물 시연

Demonstration