v

C++프로그래밍및실습

프로젝트 제목

진척 보고서 #03

제출일자:2024-12-01

제출자명:최민규

제출자학번:233935

1. 프로젝트 목표 (16 pt)

1) 배경

프로젝트의 주제로 콘솔 그래픽을 기반으로 한 게임을 만드는 것이 좋겠다고 생각했습니다. 이 주제로 적합한 장르의 게임을 생각하던 중 예전에 플레이했던 게임인 CAPCOM사의 록맨 시리즈가 생각이 나서, 이 게임의 스테이지 중 하나인 에어맨 스테이지를 구현하려 합니다.

2) 프로젝트 목표

액션 플랫포머 게임 록맨 2의 에어맨 스테이지 구현

3) 차별점

콘솔 그래픽으로 액션 플랫포머 게임을 구현

2. 기능 계획

1) 주인공

- 좌, 우, 점프, 공격의 유저 조작

- 점프가 입력되는 시간에 따라 점프 높이 조절 가능

- 체력이 존재하고 체력이 0이하가 되면 게임오버

- 목숨이 2개 존재하며 2개의 목숨이 모두 사라지면 세이브 포인트를 무시하고 처음부터 다시 시작

2) 적

- 스테이지 진행 시 나오는 6개의 적과 스테이지 보스

- 보스를 제외한 모든 적은 처치 시 일정 확률로 소형 및 대형 체력 회복 아이템을 드랍

(1) 플랫폼 형 적

- 근접하면 페이드 인 되며 나타남

- 등장하고 나서 몇 초 후 적의 좌 우에서 번갈아가며 소형 적이 등장

- 이 적의 머리 위를 플랫폼처럼 이용 가능

(2) 소형 적

- 플랫폼 형 적에게서 등장

- 유저를 향해 수직, 수평 방향으로 돌진(일직선으로)

(3) 구름 발판을 타고 있는 적

- 구름 발판 위의 적을 공격해야 타격 가능

- 적을 처치하고 남은 구름 발판을 유저가 이용할 수 있음

- 이 적은 투사체를 포물선으로 던지는 공격을 할 수 있음

(4) 새

- 빠른 속도로 수평으로 이동하며 유저가 아래에 있으면 알을 떨궈서 공격

(5) 회오리

- 유저가 근접하면 유저 쪽으로 세 번 회오리를 던짐

- 회오리는 포물선 형태로 나타나며 유저가 공격해 부술 수 있음

(6) 강풍기

- 등장하면 유저를 강풍기가 바라보는 방향으로 밈

(7) 보스

- 3번 유저의 투사체를 막는 탄환을 발사하고 2번 점프하며 반대편으로 이동

- 탄환을 발사할 때 유저를 보스가 바라보는 방향으로 밈

3) 스테이지와 스테이지 스크롤

- 스테이지에는 정해진 적과 플랫폼이 있으며 유저가 이동하면 그에 맞춰서 스테이지가 스크롤 되고, 그에 맞춰서 적이 등장함.

- 일정 진행도를 만족하면 사망해도 이어서 진행할 수 있는 지점(세이브포인트) 존재

- 스테이지의 끝에는 보스와 맞붙을 수 있는 보스 스테이지로 입장하는 문이 있음

- 주인공이 플랫폼이 없는 구멍으로 빠지게 되면 체력이 얼마나 남아있든 게임 오버

참고 사항

조작법은 a, d 좌우 움직임, j 액션, k 점프

Windows api를 이용하여 Windows에서만 작동 가능

3. 진척사항

1) 기능 구현

클래스 Engine

```h
#include <time.h>
#include <iostream>
#include <Windows.h>
#include "rockman.h"

class Engine
{
    private:
        Controls* buttons;
        Rockman* rockman;
        Map* map;

        HANDLE CIN = 0;
        HANDLE COUT = 0;

        bool Key_input();
    public:
        double FPS;// = 30.0;

        bool Render();
        void Shutdown();
        void E_Init();
        void gotoxy(int x, int y);

        //void renderRoom1();
        //void renderRoom2();
        //void renderRoom3();
        //void renderBossRoom();

        void renderTestRoom(Map* map);
        void updateScreenOffset(Map* map, int xPos);

        void renderRockman(Rockman* rockman, Map* map);
};
```

(1) 일정한 주사율로 출력되는 화면

- 입력: 없음, 출력: true

- 설명: 코드의 시작 시간과 끝나는 시간의 차를 계산해 일정한 딜레이를 주어 원하는 속도의 주사율로 출력할 수 있게 한다.

- 적용된 배운 내용 (반복문, 조건문, 클래스, 함수)

- 코드 스크린샷

```cpp
bool Engine::Render()
{
    static int Frames = 0;
    bool Return  = true;
    while(Return){
        time_t start, end;
        start = clock();
        end = clock();
        while((double)(end - start) < (1000.0 / FPS)){
            if(!Key_input()){
                Return = false;
            }
            end = clock();
        }

        rockman->R_Update(buttons, map);

        updateScreenOffset(map, rockman->getXPos());
        renderTestRoom(map);

        renderRockman(rockman, map);

        gotoxy(15, 18);
        //printf("xPos: %d, yPos: %d, xVel: %f, yVel: %f, prevXpos: %d, sc

        end = clock();
        std::cout << "FPS: " << Frames << " : " << end - start << "ms" ;
        Frames++;
    }

    return true;
}
```

(2) 키 입력

- 입력: 없음, 출력: true

- 설명: 키들의 입력을 받아 구조체 Controls(함수 내 변수명으로는 buttons)의 flag를 변경한다.

- 적용된 배운 내용 (함수, 포인터)

- 코드 스크린샷

```
bool Engine::Key_input()
{
    switchControls('W', buttons->upDown);
    switchControls('S', buttons->downDown);
    switchControls('A', buttons->leftDown);
    switchControls('D', buttons->rightDown);

    switchControls('J', buttons->actionPressed);
    switchControls('K', buttons->jumpPressed);

    return true;
}
```

```
void switchControls(int VirtualKey, bool &key){
    if (GetAsyncKeyState(VirtualKey) < 0 && key == false)
    {
        key = true;
    }
    if (GetAsyncKeyState(VirtualKey) == 0 && key == true)
    {
        key = false;
    }
}
```

(3) 스테이지 출력 및 화면 스크롤링

- 입력: 클래스 Map 포인터, xPos, 출력: 없음

- 설명: 클래스 Map과 록맨의 x좌표 xPos를 받아 현재 록맨의 좌표에 맞는 화면을 출력한다. (현재는 테스트를 하기 위한 테스트 맵으로 출력하도록 함) 맵의 끝에서는 화면 스크롤링이 되지 않는다.

- 적용된 배운 내용 (함수, 포인터, 클래스, 조건문, 반복문)

- 코드 스크린샷

```
void Engine::renderTestRoom(Map* map) {
    for (int y = 0; y < map->ScreenHeight; ++y) {
        for (int x = 0; x < map->ScreenWidth; ++x) {
            int mapX = x + map->screenOffsetX;
            if (mapX < map->ScreenWidth * map->test_room_numberofScreen) {
                gotoxy(x, y);  // 커서 위치 설정
                switch (map->test_room[y][mapX]) {
                    case 0: printf(" "); break;
                    case 1: printf("="); break;
                }
            }
        }
    }
}

void Engine::updateScreenOffset(Map* map, int xPos) {
    int dx = xPos - rockman->previousXpos;
    int mapWidth = map->ScreenWidth * map->test_room_numberofScreen;

    if (xPos < map->ScreenWidth / 2) {
        map->screenOffsetX = 0;
    } else if (xPos > mapWidth - map->ScreenWidth / 2) {
        map->screenOffsetX = mapWidth - map->ScreenWidth;
    } else {
        map->screenOffsetX += dx;
    }

    rockman->previousXpos = xPos;
}
```

(4) 록맨 출력

- 입력: 클래스 Rockman 포인터, 클래스 Map포인터 출력: 없음

- 설명: 콘솔 화면에 록맨을 그려준다. 맵의 끝에서는 화면 스크롤링이 되지 않으므로, 현재 좌표에 맞는 위치에 록맨을 출력한다.

- 적용된 배운 내용 (함수, 포인터, 클래스, 조건문)

- 코드 스크린샷

```
void Engine::renderRockman(Rockman* rockman, Map* map) {
    int mapWidth = map->ScreenWidth * map->test_room_numberofScreen;
    int renderX = rockman->getXPos() - map->screenOffsetX;

    if (rockman->getXPos() <= map->ScreenWidth / 2) {
        renderX = rockman->getXPos();
    } else if(rockman->getXPos() >= mapWidth - map->ScreenWidth / 2){
        renderX = rockman->getXPos() - (mapWidth - map->ScreenWidth);
    }

    gotoxy(renderX, rockman->getYPos());
    if (rockman->getfacingRight()) {
        printf("[\\");
    } else {
        printf("/]");
    }
}
```

클래스 Rockman

```h
#include <iostream>
#include <math.h>
#include <algorithm>
#include "map.h"
#include "controls.h"

class Rockman {
private:
    int xPos;
    int yPos;

    float xVel, yVel;
    float const acc = 0.5f;
    float const xMaxVel = 1.0f, yMaxVel = 1.0f;
    float const gravity = 0.5f;
    float const jumpforce = 1.0f;

    bool onGround;
    bool facingRight;
    bool isInvincible;
public:
    int previousXpos;

    void R_Init();
    void R_Update(Controls* control, Map* map);

    bool XCollision(Map* map, Controls* control);
    bool YCollision(Map* map, Controls* control);

    void setXPos(int x);
    void setYPos(int y);

    int getXPos();
    int getYPos();

    float getXVel();
    float getYVel();

    bool getfacingRight();
};
```

(5) 록맨 상태 업데이트(수정)

- 입력: 구조체 Controls 포인터, 클래스 Map 포인터 출력: 없음

- 설명: 키 입력 flag를 모아둔 구조체 Controls를 이용해 키 입력에 따른 록맨의 움직임

을 처리한다. (수정) 충돌 감지 및 위치 변화 코드의 순서를 변경하여 더욱 정확히 좌표 계산을 할 수 있도록 했다.

- 적용된 배운 내용 (함수, 포인터, 클래스, 조건문)

- 코드 스크린샷

```
    if(xVel != 0 && !control->leftDown && !control->rightDown){
        if (xVel > 0){
            xVel = std::max<float>(xVel - acc, 0);
        } else {
            xVel = std::min<float>(xVel + acc, 0);
        }
    } else {
        if(control->leftDown && xVel > -xMaxVel){
            xVel = std::max<float>(xVel - acc, -xMaxVel);
            facingRight = false;
        }

        if(control->rightDown && xVel < xMaxVel) {
            xVel = std::min<float>(xVel + acc, xMaxVel);
            facingRight = true;
        }
    }

    if(xVel != 0 && XCollision(map, control)){ // 충돌 시 xVel = 0 설정
        xVel = 0;
    }

    xPos += xVel;

    if(control->jumpPressed && onGround){
        yVel -= jumpforce;
    }else if (yVel < yMaxVel){
        yVel = std::min<float>(yVel + gravity, yMaxVel);
    }

    if(!control->jumpPressed && yVel < 0){
        yVel = 0;
    }

    if(yVel > 0) {
        onGround = YCollisionDown(map);
        if(onGround) {
            yVel = 0;
        }
    }else if (yVel < 0) {
        if(YCollisionUp(map)){
            yVel = 0;
        }
```

```
    onGround = false;
}

yPos += yVel;
```

(6) x, y 충돌 확인 (수정)

- 입력: 구조체 Controls 포인터, 클래스 Map 포인터 출력: true or false

- 설명: 각 맵에서의 x, y 충돌을 키 입력 flag와 록맨의 x, y 좌표를 이용해 처리한다. 충돌하면 true, 그렇지 않으면 false를 반환한다. Y축의 충돌을 위, 아래 두가지 경우로 나누어서 정확히 충돌을 감지하도록 했다.

- 적용된 배운 내용 (함수, 포인터, 클래스, 조건문)

- 코드 스크린샷

```
bool Rockman::XCollision(Map* map, Controls* control){
    if (!map || !control) { return false;} // 유효하지 않으면 충돌로 간주하지 않음

    if(map->In_testroom){
        if(  control->leftDown
        && ((map->test_room[yPos][xPos - 1] == 1)
        ||   xPos - 1 <= 0)){
            return true;
        }
        else if(control->rightDown
            &&((map->test_room[yPos][xPos + 2] == 1)
            ||  (xPos + 2) >= (map->ScreenWidth * map->test_room_numberofScreen))){
            return true;
        }
    }

    return false;
}

bool Rockman::YCollisionUp(Map* map){
    if(map->In_testroom){
        if((map->test_room[yPos - 1][xPos] == 1) || map->test_room[yPos - 1][xPos + 1] == 1){
            return true;
        }
    }

    return false;
}
bool Rockman::YCollisionDown(Map* map){
    if(map->In_testroom){
        if((map->test_room[yPos + 1][xPos] == 1) || map->test_room[yPos + 1][xPos + 1] == 1){
            return true;
        }
    }

    return false;
}
```

(7) 화면상의 x좌표 계산

- 입력: 클래스 Map 포인터 출력: int

- 설명: 화면 스크롤으로 인해 화면 상의 x좌표와 map 배열 상의 x좌표가 다르다. 이를 보완하기 위해 화면상의 x좌표를 계산해 반환한다.

- 적용된 배운 내용 (함수, 포인터, 클래스, 조건문)

(8) 액션 버튼(R_Update 함수의 추가사항)

- 입력: 구조체 Controls 포인터, 클래스 Map 포인터 출력: 없음

- 설명: 액션 버튼을 누르면 총알을 발사한다. 총알은 바라보고 있는 방향에 따라 진행하는 방향이 다르며, 화면상의 총알은 최대 3개가 존재할 수 있고, 총알을 발사하는 것에 지연시간이 존재한다.

- 적용된 배운 내용 (함수, 포인터, 클래스, 조건문)

- 코드 스크린샷

```
int screenXpos = calcScreenXpos(map);
if(control->actionPressed && bulletDelayCount == 0){
    if(bulletCount < maxBullet){
        if(facingRight){
            r_bullet[currentBullet++]->initBullet(screenXpos+2, yPos, facingRight);
        }else{
            r_bullet[currentBullet++]->initBullet(screenXpos-1, yPos, facingRight);
        }
        bulletCount++;
        bulletDelayCount = bulletDelay;
    }

    if(currentBullet > 2){
        currentBullet = 0;
    }
}else{
    if(tick % 30){
        bulletCount = 0;
    }
}

for(int i = 0; i < 3; i++){
    r_bullet[i]->B_Update(map, &bulletCount, xPos);
}

if(bulletDelayCount > 0) bulletDelayCount--;
```

클래스 Bullet

```
#pragma once

#ifndef MAP_H
#define MAP_H

#include "map.h"

#endif

class Bullet {
protected:
    int b_posX;
    int b_posY;
    int tick;

    bool direction;         //direction == true 이면 오른쪽을 향하고 false이면 왼쪽을 향한다
    bool b_isActive;
public:
    Bullet();
    void Deactivation();
    void initBullet(int x, int y, bool facing);
    bool f_isActive();
    void B_Update(Map* map, int* BulletCount, int r_xPos);
    bool checkCollision(Map* map);

    int getPosX();
    int getPosY();
};
```

(9) 총알 상태 업데이트

- 입력: 구조체 Map 포인터, int 포인터, 록맨의 int x좌표 출력: 없음

- 설명: 총알이 록맨이 바라보고 있는 방향에 따라 진행하며, 충돌하면 비활성화하고, rockman 객체의 bulletCount를 1 감소시킨다.

- 적용된 배운 내용 (함수, 포인터, 클래스, 조건문)

- 코드 스크린샷

```
Bullet::Bullet() : b_posX(0), b_posY(0), tick(0), b_isActive(false) {    }

void Bullet::B_Update(Map* map, int* bulletCount, int r_xPos){      //directi
    if(b_isActive){
        if(direction){
            b_posX += 1;
        }else{
            b_posX -= 1;
        }

        if(checkCollision(map)){ // Screen Width
            Deactivation();
            *bulletCount -= 1;
        }
    }
}
```

(10) 총알 충돌 감지

- 입력: 구조체 Map 포인터 출력: bool

- 설명: 총알이 화면 밖으로 나가면 true 반환, 구조물에 부딪히면 true 반환, 그렇지 않으면 false 반환

- 적용된 배운 내용 (함수, 포인터, 클래스, 조건문)

- 코드 스크린샷

```
bool Bullet::checkCollision(Map* map){
    int b_posX_inMap = b_posX + map->screenOffsetX;
    if(b_posX <= -1 || b_posX >= 32){
        return true;
    }
    if(map->test_room[b_posY][b_posX_inMap] != 0){
        return true;
    }
    return false;
}
```

(11) 총알 비활성화

- 입력: 없음 출력: 없음

- 설명: 총알 비활성화

- 적용된 배운 내용 (함수, 클래스)

- 코드 스크린샷

```
void Bullet::Deactivation(){
    b_isActive = false;
}
```

(12) 총알 초기화

- 입력: x, y 좌표, 방향 bool 출력: 없음

- 설명: 총알의 시작 좌표, 방향 지정 및 활성화

- 적용된 배운 내용 (함수, 클래스)

- 코드 스크린샷

```
void Bullet::initBullet(int x, int y, bool facing){
    b_posX = x;
    b_posY = y;
    direction = facing;
    b_isActive = true;
}
```

(13) 투사체 출력

- 입력: Rockman 포인터 출력: 없음

- 설명: 총알의 좌표에 따라 화면에 출력

- 적용된 배운 내용 (함수, 클래스, 포인터)

- 코드 스크린샷

```
void Engine::renderProjectiles(Rockman* rockman){                    //현재는 록맨의 투사
    for(int i = 0; i < rockman->maxBullet; i++){
        if(rockman->r_bullet[i]->f_isActive()){
            gotoxy(rockman->r_bullet[i]->getPosX(), rockman->r_bullet[i]->getPosY());
            printf("o");
        }
    }
}
```

클래스 Enemy 및 서브클래스(수정)

```
class Enemy {
protected:
    int e_hp;           // -1 이면 무적
    int e_width, e_height;
    int e_posX, e_posY; //맵 상의 위치

    int e_contactDamage;

    bool isActive;

    int tick = 0;
public:
    Enemy();
    bool CheckCrash(int b_X, int b_Y, int screenOffset);
    void GetDamage();
    void setXY(int x, int y);
    int getX();
    int getY();
    bool getIsActive();
    bool outOfScreen(int screenOffset);

    virtual void Update(Map* map, int r_posX, int r_posY, int screenOffset);
    virtual void Render(Map* map);
};

class Gremlin : public Enemy{
private:

public:
    bool previousState;

    Gremlin();
    void Init(int e_posX, int e_posY);
    void Update(Map* map, int r_posX, int r_posY, int screenOffset) override;
    void Render(Map* map) override;
    void setTicks(int t);
};
```

```
class AirTikki : public Enemy{
private:
    int e_attackDamage;

    static const int maxGremlin = 3;
    Gremlin* gremlins[maxGremlin];
    int gremlinCount;           //현재 활성화 되어있는 Gremlin의 수
    int totalGremlins;          //현재까지 나온 Gremlin의 수
    int currentGremlin;         //gremlins index

    bool isEncountered;
    bool isHornActivate;
public:
    AirTikki(int x, int y);
    void Update(Map* map, int r_posX, int r_posY, int screenOffset) override;
    void ActivateHorns();
    void ActivateGremlins(int r_posX, int r_posY, int screenOffset);
    void Render(Map* map) override;
};


class Pipi : public Enemy{
private:
    int e_attackDamage;
    PipiBullet* egg;
public:
    Pipi(int x, int y);
    void Update(Map* map, int r_posX, int r_posY, int screenOffset) override;
    PipiBullet* getEgg();
    void Render(Map* map);
};
```

```
class LightningLord : public Enemy{
private:
    int e_attackDamage;
    const int e_attackDelay = 50;
    Lightning* lightning;
    bool isAlive;

    int radius;
    double angle;
    int centerpoint_x, centerpoint_y;
    double double_x, double_y;
public:
    LightningLord(int x, int y);
    void Update(Map* map, int r_posX, int r_posY, int screenOffset);
    void throwLightning(int r_posX, int r_posY);
    Lightning* getLightning();
    void Render(Map* map);
};

class Scworm : public Enemy{
private:
    int e_attackDamage;
public:
    Scworm(int x, int y);
    void Update(Map* map, int r_posX, int r_posY, int screenOffset);
    void Render(Map* map);
};


class FanFriend : public Enemy{
private:

public:
    FanFriend(int x, int y);
    void Update(Map* map, int r_posX, int r_posY, int screenOffset);
    void Render(Map* map);
};
```

```
class Airman : public Enemy{
private:
    int e_attackDamage;
    bool isInvincible;
public:
    Airman(int x, int y);
    void Update(Map* map, int r_posX, int r_posY, int screenOffset);
    void Render(Map* map);
};
```

(14) Enemy 클래스 :: outOfScreen()

- 입력: int screenOffset 출력: bool

- 설명: 적들이 화면 밖으로 나가면 true 반환

- 적용된 배운 내용 (함수, 클래스)

- 코드 스크린샷

```
bool Enemy::outOfScreen(int screenOffset){
    int e_posX_inScreen = e_posX - screenOffset;
    if(e_posX_inScreen + e_width - 1 <= -1 || e_posX_inScreen >= 32){
        return true;
    }

    return false;
}
```

(15) Gremlin :: 상태 업데이트(수정)

- 입력: int 록맨의 x, y좌표, screenoffset 출력: 없음

- 설명: 록맨에게 서서히 접근함. Y축 우선 이동.

- 적용된 배운 내용 (함수, 클래스, 상속)

- 코드 스크린샷

```
void Gremlin::Update(Map* map, int r_posX, int r_posY, int screenOffset){
    if(isActive){
        static int startTick = tick;
        if(tick > startTick + 4){
            if(tick % 5 == 0){
                if(e_posY > r_posY){
                    e_posY--;
                }else if(e_posY < r_posY){
                    e_posY++;
                }else{
                    if(e_posX > r_posX){
                        e_posX--;
                    }else if(e_posX < r_posX){
                        e_posX++;
                    }
                }
            }
        }


        if(outOfScreen(screenOffset)){
            isActive = false;
        }
    }
}
```

(16) AirTikki :: 상태 업데이트(수정)

- 입력: int 록맨의 x, y좌표, screenoffset 출력: 없음

- 설명: 록맨이 일정 수준으로 접근하면 등장. 벽, 발판으로 사용 가능, 일정 시간마다 뿔이 솟아나고, 최대 3개의 Gremlin을 생성함

- 적용된 배운 내용 (함수, 클래스, 상속)

- 코드 스크린샷

```
void AirTikki::Update(Map* map, int r_posX, int r_posY, int screenOffset){
    if(e_posX - r_posX <= 7){
        isEncountered = true;
    }

    if(!outOfScreen(screenOffset) && isEncountered){
        isActive = true;
    }else{
        isActive = false;
    }

    if(isActive){
        if(r_posX >= e_posX && r_posX <= e_posX + e_width){
            ActivateGremlins(r_posX, r_posY, screenOffset);
        }
        ActivateHorns();
    }
    tick++;

    int count = 0;
    for(int i = 0; i < maxGremlin; i++){
        gremlins[i]->Update(map, r_posX, r_posY, screenOffset);
        gremlins[i]->setTicks(tick);

        if(gremlins[i]->getIsActive()){
            count++;
        }
    }
    gremlinCount = count;


}
```

(17) AirTikki :: 뿔 활성화(수정)

- 입력: 없음 출력: 없음

- 설명: 일정 시간마다 뿔을 생성

- 적용된 배운 내용 (함수, 클래스, 상속)

- 코드 스크린샷

```
void AirTikki::ActivateHorns(){
    static int startTick = tick;
    if(startTick + 20 <= tick){
        if(isHornActivate == false){
            isHornActivate = true;
        }else if(isHornActivate == true){
            isHornActivate = false;
        }
        startTick = tick;
    }
}
```

(18) AirTikki :: Gremlin 활성화

- 입력: 록맨의 x, y 좌표, screenOffSet (int) 출력: 없음
- 설명: 일정 시간마다 좌우로 번갈아 Gremlin들을 생성. 한번에 최대 3마리
- 적용된 배운 내용 (함수, 클래스, 상속)
- 코드 스크린샷

```
void AirTikki::ActivateGremlins(int r_posX, int r_posY, int screenOffset){
    static int startTick = tick;
    if(startTick + 15 <= tick){
        if(gremlinCount < maxGremlin){
            if(totalGremlins % 2 == 0){                         //번갈아
                gremlins[currentGremlin]->Init(e_posX - 1, e_posY + 2);
            }else{
                gremlins[currentGremlin]->Init(e_posX + e_width, e_posY + 2);
            }

            currentGremlin++;
            totalGremlins++;
        }
        startTick = tick;
    }

    if(currentGremlin > maxGremlin - 1){
        currentGremlin = 0;
    }

}
```

(19) Pipi :: 상태 업데이트

- 입력: Map 포인터 변수, int 록맨의 x, y좌표, screenoffset 출력: 없음

- 설명: 화면 내로 들어오면 왼쪽으로 진행, 록맨과 가까워지면 알 투척

- 적용된 배운 내용 (함수, 클래스, 상속)

- 코드 스크린샷

```
void Pipi::Update(Map* map, int r_posX, int r_posY, int screenOffset){
    if(!outOfScreen(screenOffset)){
        isActive = true;
        if(egg->getHolding()){      //알을 들고있을때만 알 활성화(안하면 비활성화가 안됨)
            egg->setActive(true);
        }
    }

    if(isActive){
        e_posX--;
        egg->B_Update(map, e_posX, e_posY);
        if(r_posX + 2 >= e_posX){
            egg->setHolding(false);
        }
    }

    if(outOfScreen(screenOffset)){
        isActive = false;
    }
}

PipiBullet* Pipi::getEgg(){
    return egg;
}
```

(20) Pipi :: 렌더

- 입력: Map 포인터 변수 출력: 없음

- 설명: Pipi의 위치에 맞게 화면에 그린다

- 적용된 배운 내용 (함수, 클래스, 상속)

- 코드 스크린샷

```
void Pipi::Render(Map* map){
    if(isActive){
        int renderX = e_posX - map->screenOffsetX;
        gotoxy(renderX, e_posY);
        printf("<}");
    }
}
```

(21) Pipi :: 알 업데이트(bullet 클래스)

- 입력: Map 포인터 변수, 적의 좌표 x, y출력: 없음

- 설명: 알의 좌표에 맞게 화면에 그린다

- 적용된 배운 내용 (함수, 클래스, 상속)

- 코드 스크린샷

```
void PipiBullet::B_Update(Map* map, int e_posX, int e_posY){
    if(b_isActive && holding){
        b_inMapPosX = e_posX;
    }

    b_posX = b_inMapPosX - map->screenOffsetX;

    if(!holding){
        b_posY++;
    }

    if(checkCollision(map)){
        Deactivation();
    }

}
```

(22) LightningLord :: 상태 업데이트

- 입력: Map 포인터 변수, int 록맨의 x, y좌표, screenoffset 출력: 없음

- 설명: 원 운동을 한다

- 적용된 배운 내용 (함수, 클래스, 상속)

- 코드 스크린샷

(23) LightningLord :: 렌더

- 입력: Map 포인터 변수 출력: 없음

- 설명: 록맨이 밟을 수 있는 발판과 적을 렌더

- 적용된 배운 내용 (함수, 클래스, 상속)

- 코드 스크린샷

```cpp
void LightningLord::Render(Map* map) {
    if (isActive) {
        // 현재 위치와 이전 위치를 저장
        static int prevPosX = -1, prevPosY = -1;
        static int prevTile_0, prevTile_1, prevTile_2, prevTile_3;

        if (prevPosX != -1 && prevPosY != -1) { // 초기 상태가 아닌 경우에만 복구 수행
            if (prevPosX != e_posX || prevPosY != e_posY) { // 이전 위치와 현재 위치가 다른 경우
                map->test_room[prevPosY][prevPosX + 0] = prevTile_0;
                map->test_room[prevPosY][prevPosX + 1] = prevTile_1;
                map->test_room[prevPosY][prevPosX + 2] = prevTile_2;
                map->test_room[prevPosY][prevPosX + 3] = prevTile_3;

                // 현재 위치의 타일 값을 저장
                prevTile_0 = map->test_room[e_posY][e_posX + 0];
                prevTile_1 = map->test_room[e_posY][e_posX + 1];
                prevTile_2 = map->test_room[e_posY][e_posX + 2];
                prevTile_3 = map->test_room[e_posY][e_posX + 3];
            }
        }

        prevPosX = e_posX;
        prevPosY = e_posY;

        // 현재 위치를 적의 타일로 표시
        map->test_room[e_posY][e_posX + 0] = 1;
        map->test_room[e_posY][e_posX + 1] = 1;
        map->test_room[e_posY][e_posX + 2] = 1;
        map->test_room[e_posY][e_posX + 3] = 1;

        // 적을 화면에 렌더링
        int renderX = e_posX - map->screenOffsetX;
        gotoxy(renderX, e_posY - 2);
        printf("/==\\");
        gotoxy(renderX, e_posY - 1);
        printf("[--]");
    }
}
```

(24) Scworm :: 상태 업데이트 및 렌더

- 입력: Map 포인터 변수, int 록맨의 x, y좌표, screenoffset 출력: 없음

  입력: Map 포인터 변수 출력: 없음

- 설명: 활성화 / 비활성화 여부 및 위치에 맞게 렌더

- 적용된 배운 내용 (함수, 클래스, 상속)

- 코드 스크린샷

```
void Scworm::Update(Map* map, int r_posX, int r_posY, int screenOffset){
    isActive = !outOfScreen(screenOffset);
}

void Scworm::Render(Map* map){
    if(isActive){
        int renderX = e_posX - map->screenOffsetX;
        gotoxy(renderX, e_posY);
        printf("==");
    }
}
```

(25) FanFriend :: 상태 업데이트 및 렌더

- 입력: Map 포인터 변수, int 록맨의 x, y좌표, screenoffset 출력: 없음

  입력: Map 포인터 변수 출력: 없음

- 설명: 활성화 / 비활성화 여부 및 위치에 맞게 렌더

- 적용된 배운 내용 (함수, 클래스, 상속)

- 코드 스크린샷

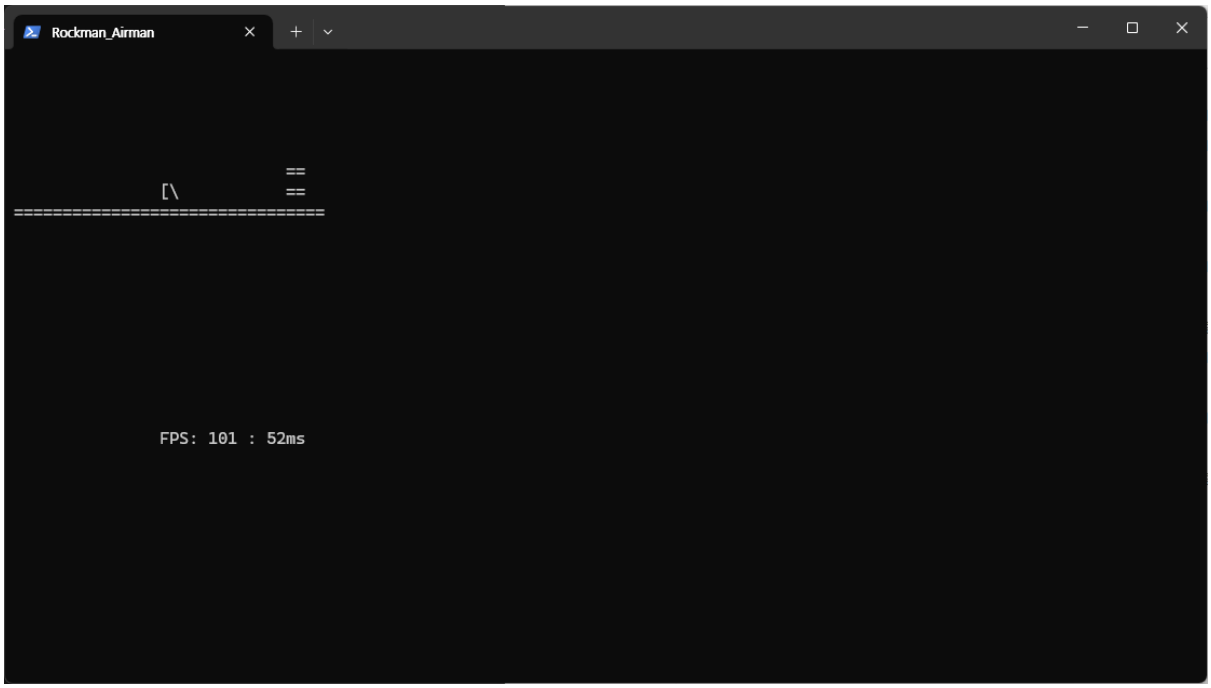
```
void FanFriend::Update(Map* map, int r_posX, int r_posY, int screenOffset){
    isActive = !outOfScreen(screenOffset);
}

void FanFriend::Render(Map* map){
    if(isActive){
        int renderX = e_posX - map->screenOffsetX;
        gotoxy(renderX, e_posY - 1);
        printf("<]-_");
        gotoxy(renderX, e_posY);
        printf("[__\\");
    }
}
```

2) 테스트 결과

(1) 화면 출력 및 일정한 주사율

- 일정한 주사율로 화면 출력

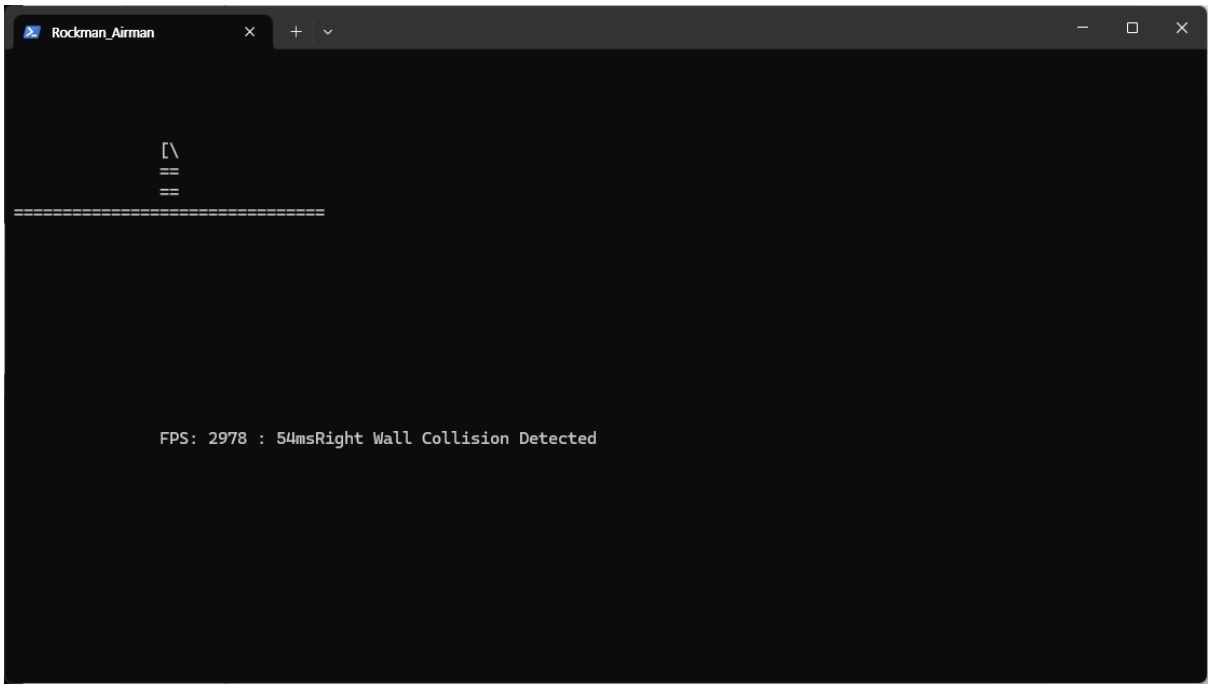
- 테스트 결과 스크린샷



(2) 키 입력 및 록맨의 이동

- 키 입력을 받으면 록맨이 이동한다.

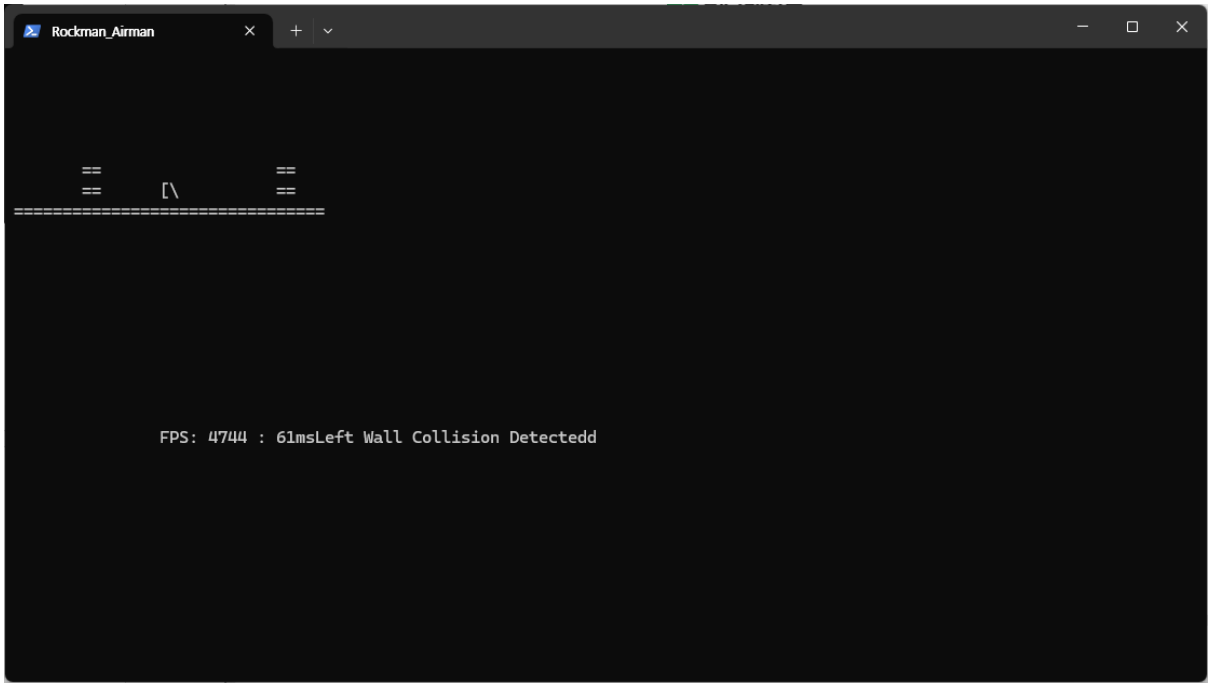
- 테스트 결과 스크린샷

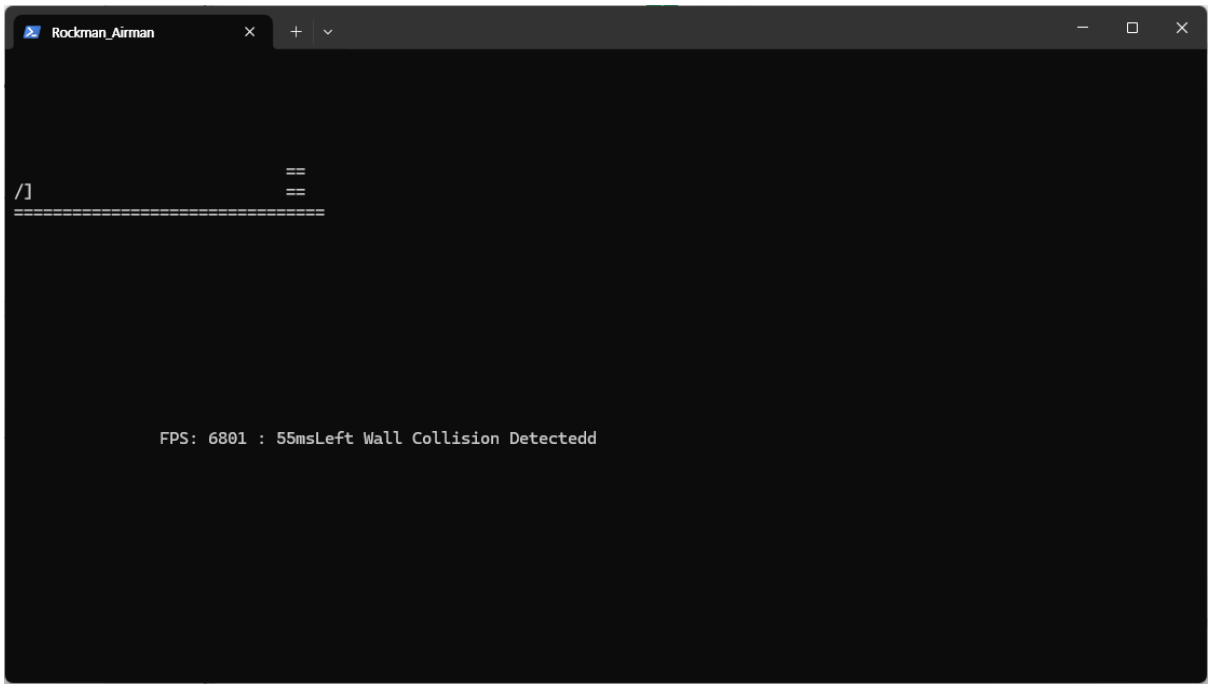


(3) 스크린 스크롤링

- 록맨의 위치에 따라서 화면이 움직인다.

- 테스트 결과 스크린샷

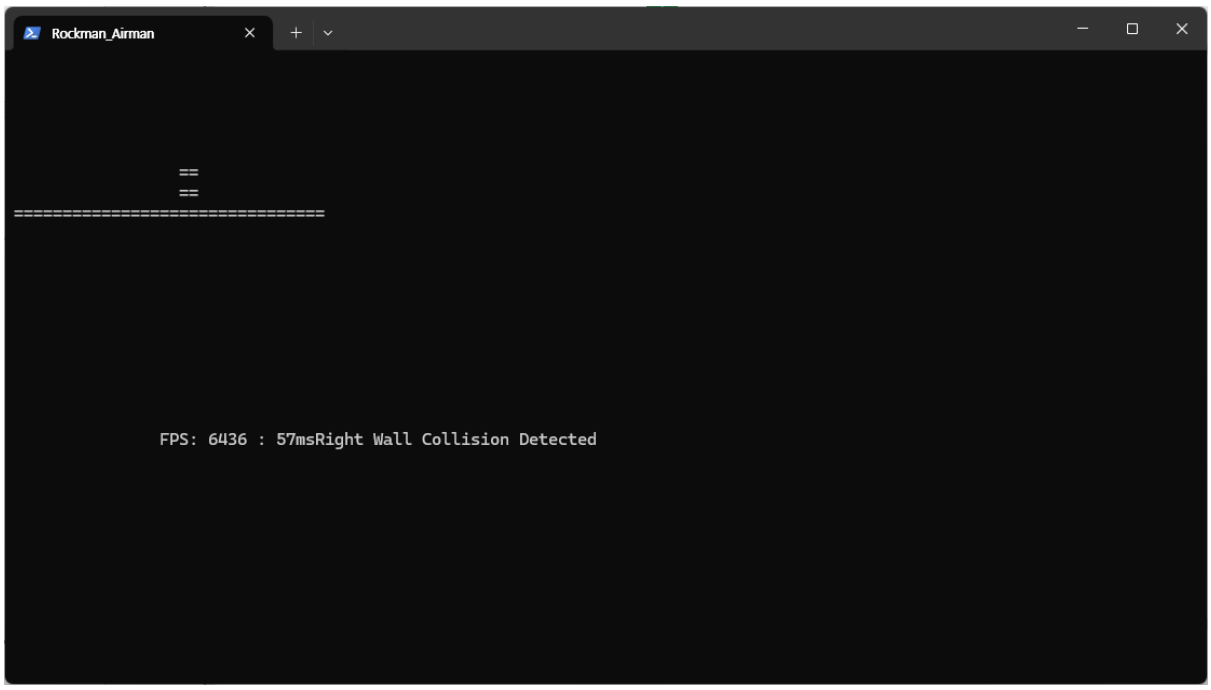




(4) 충돌 확인

- 충돌 확인

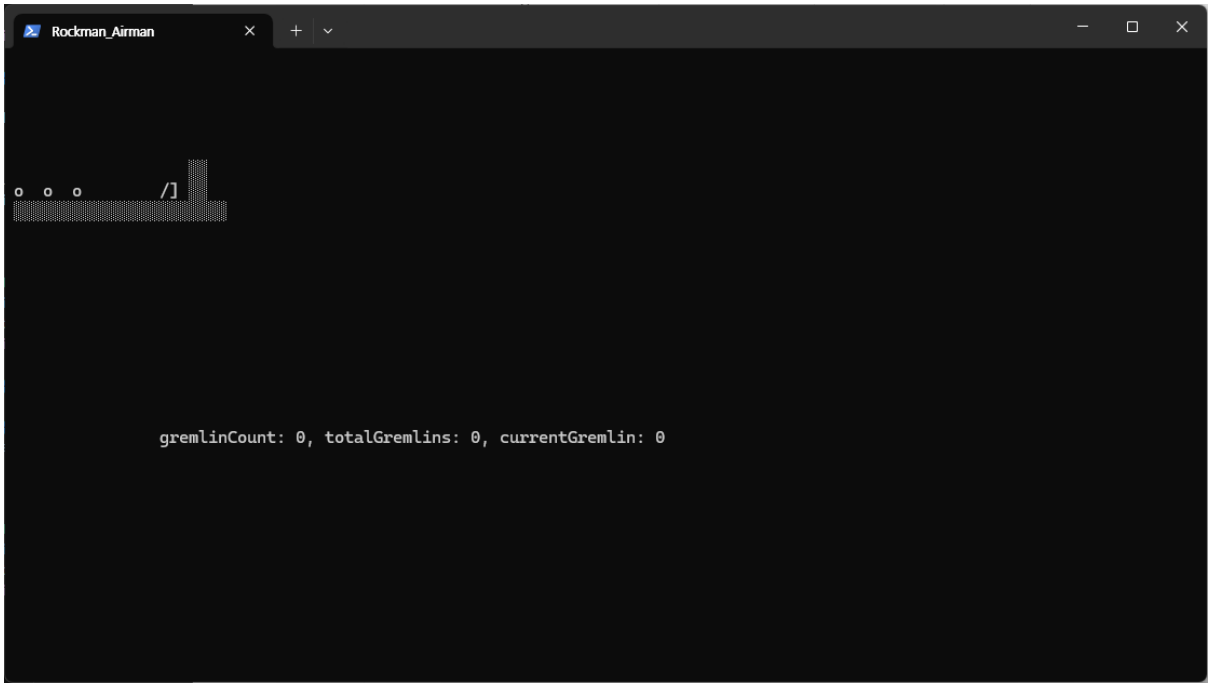
- 테스트 결과 스크린샷

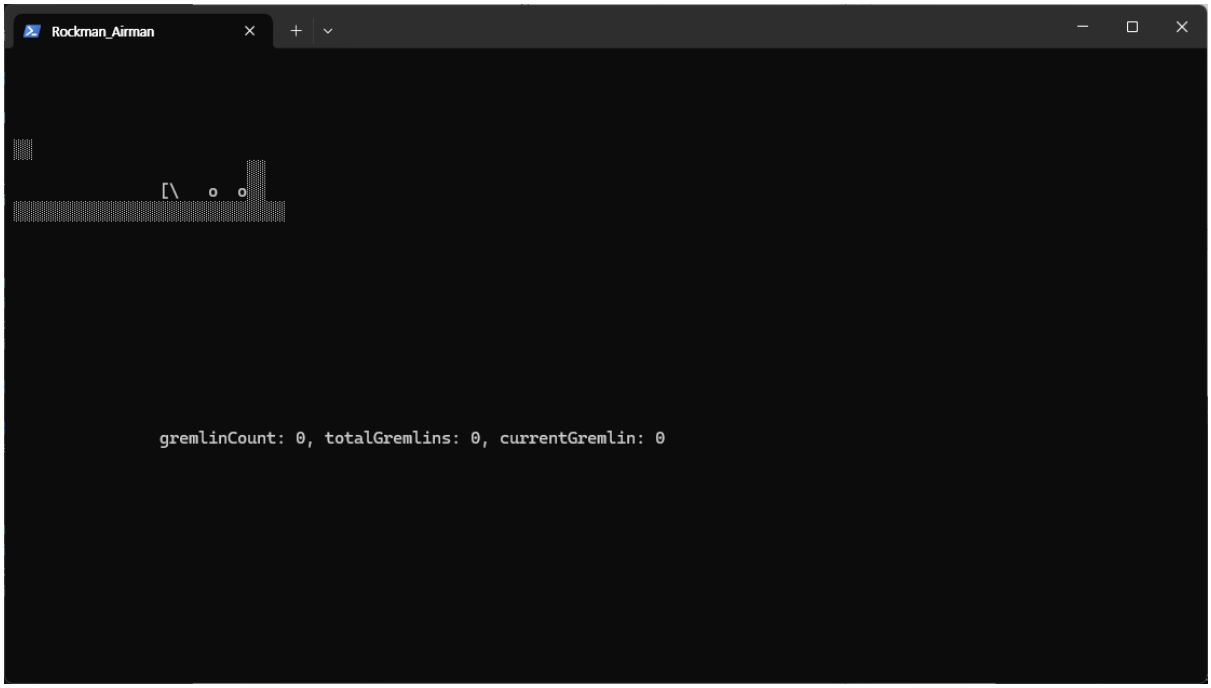


(4) 총알

- 총알 발사, 충돌, 최대 개수

- 테스트 결과 스크린샷

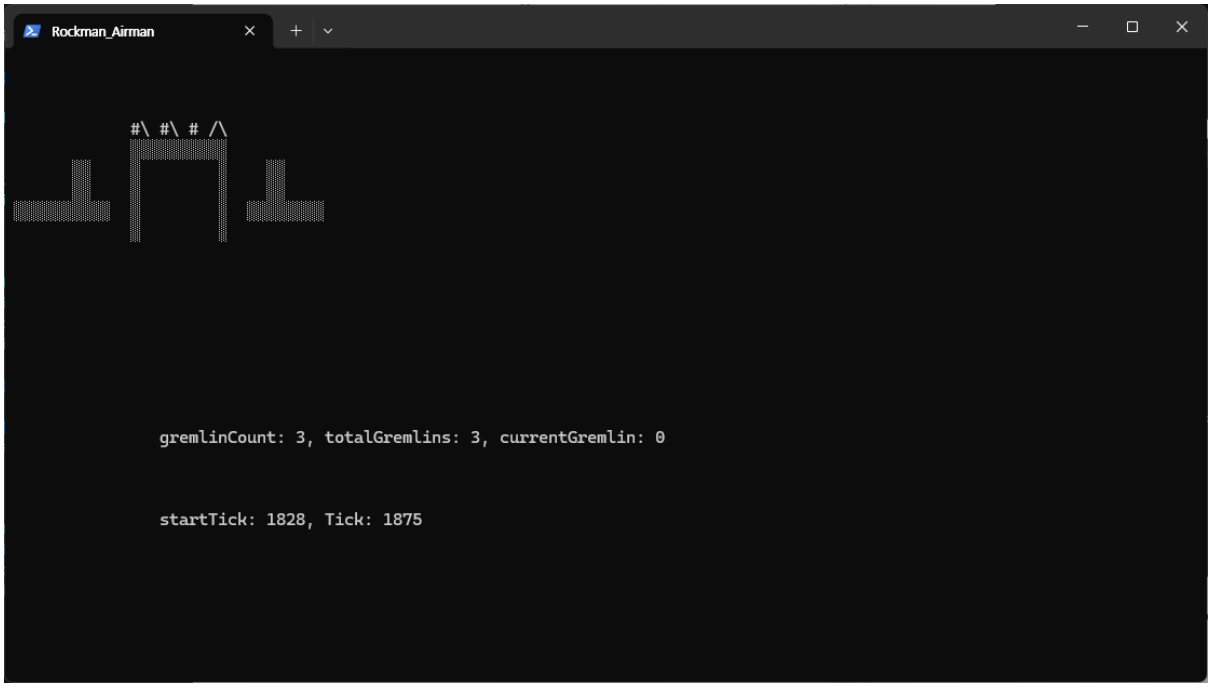




(5) Gremlin 및 Air Tikki

- Gremlin 및 Air Tikki

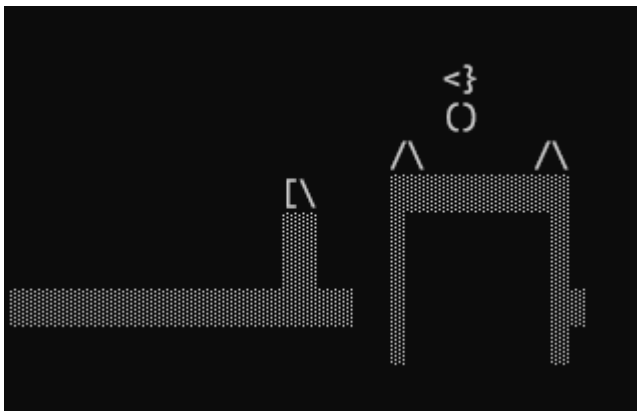
- 테스트 결과 스크린샷

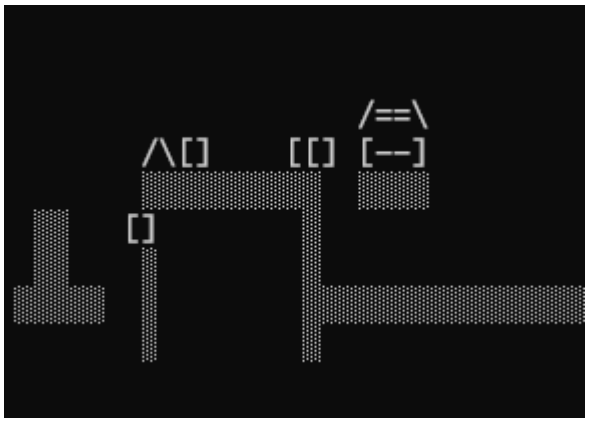


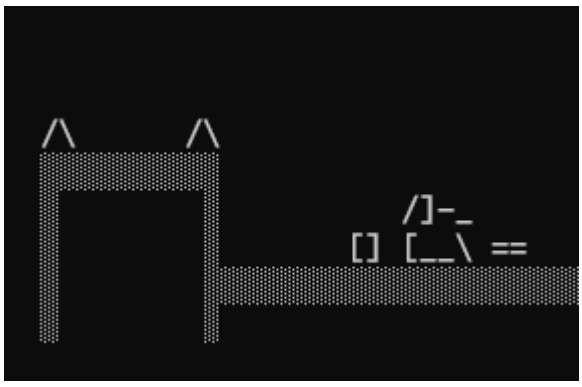
(6) Pipi, LightningLord, Scworm, FanFriend

- Pipi, LightningLord, Scworm, FanFriend

- 테스트 결과 스크린샷







4. 계획 대비 변경 사항

1) 일정 조정

- 스테이지의 완벽한 구현은 프로젝트 후반부로 옮겨야 할 것 같음

- 사유: 해당 스테이지가 적을 먼저 구현하지 않으면 진행할 수 없어서 적들을 먼저 구현한 후 스테이지를 구현해야 할 것 같음

- 록맨의 조작과 물리 조정을 조금 더 해야 할 것 같음

- 사유: 화면 주사율과 내가 원하는 키 입력 정도를 구현하는 것에 시간을 너무 써서 액션 기능과 물리 구현을 끝내지 못했다.

2) 구현 실패

- 록맨의 피격, 적의 피격, LightningLord와 Scworm의 투사체, FanFriend의 기능, Airman의 구현, 스테이지의 구현 등을 구현 실패로 두어야 할 것 같음

- 사유: 벽과의 충돌, 화면 스크롤에 의해 나타난 각 요소들의 실제 좌표와 화면 상의 좌표의 동기화, 적의 패턴 구현과 같은 게임을 구성하는 것들이 내가 생각하던 것 보다 난이도가 높아 예상했던 것보다 구현에 시간이 더 들어가게 되었다. 또한 학업과 병행하여 진행하다 보니 프로젝트에 들일 수 있는 시간이 예상보다 적었고, 프로젝트의 남은 기간에는 기말고사에 의해 더는 프로젝트에 쏟을 수 있는 시간이 없을 것이라 생각해 구현해야 할 다른 요소들을 구현 실패로 남겨두고자 한다.

5. 프로젝트 일정

(진행한 작업과 진행 중인 작업 등을 표기)

업무		11/3	11/10	11/17	11/24
제안서 작성		완료			
주인공			진행 중		

스테이지				진행중	
중간 보고서 1				완료	
		12/1	12/8	12/15	12/22
적 (1), (2)		완료			
중간 보고서 2		완료			
적	(3), (4)	(3)완료, (4)진행중			
	(5), (6)		진행중		
중간 보고서 3				완료	
	(7)			진행중	
최종 보고서					---------->