02. 루시의 일기

게임명 : 루시의 일기
제작 인원 : 4인
플랫폼 : Window API
제작 도구 : VisualStudio2019
제작 기한 : 2주
게임 장르 : 로그라이크 탄막슈팅

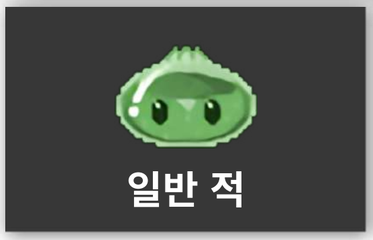
역할 선정 이유

단순한 움직임 뿐만 아니라 루시의 일기 게임의 특성상 다양한 탄막 요소가 포함 되어 있어 논리사고를 증가시키기 위해 루시의 일기에서는 적, 보스, 이펙트 를 맡아 진행 했습니다. 또한 원활한 협업을 위해 클래스 구조에 대해 중점을 두며 구현했습니다.

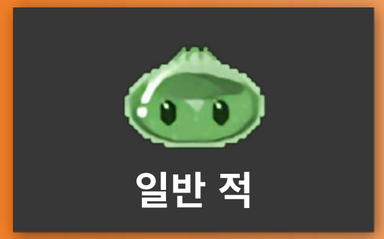
Used Skill

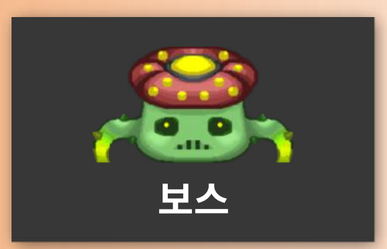
동영상 클릭

Name 김유민
Moblie 010 7480 0851
E-mail rladbals0129@naver.com



자연스러운 AI와 탄막패턴, 협업을 위한 클래스 구조에 중점을 두며 구현





협업과 클래스 구조

개인 프로젝트를 진행 할 때보다 실력도 늘어났고 혼자 작업 하는것이 아닌 협업 이기 때문에 클래스의 구조에 신경을 많이 썼습니다.

AI

맡은 역할이 '적' 이다 보니 자연스러운 AI에 신경을 많이 썼습니다. 플레이어와의 거리를 계산하고 애니메이션의 프레임에 맞게 사이클을 조절해 주었습니다.

탄막패턴

로그라이크 탄막 슈터라는 게임 장르의 특성상 다양한 탄막패턴을 구현하기 위해 stl과 funciont과 람다, 삼각함수를 적극적으로 활용

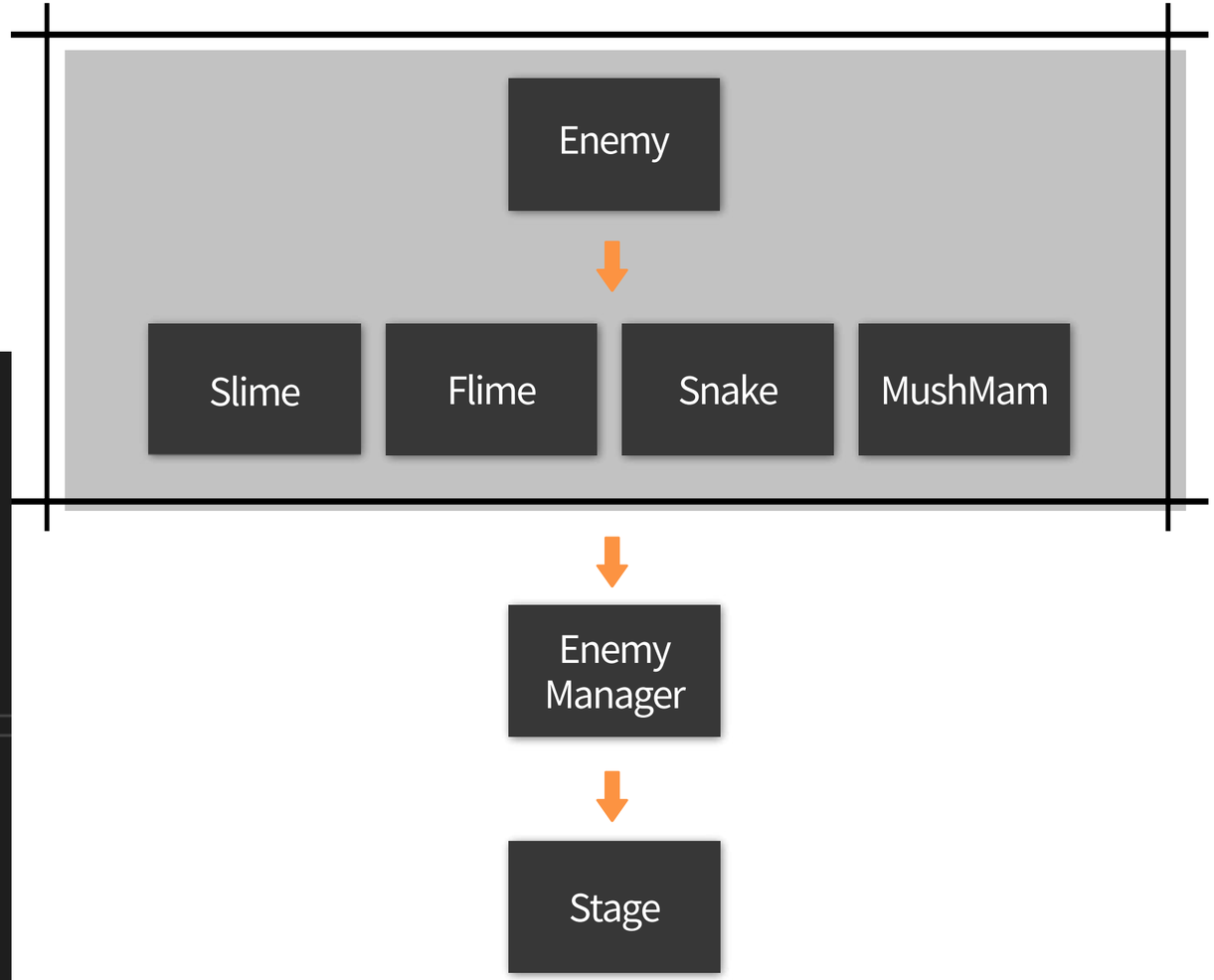
이펙트처리

렌더링 처리를 GDI기반으로 하고 있기 때문에 회전 및 이미지 자체 투명도로 인해 퀄리티 저하가 발생하기 때문에 이를 해결하기 위해 D2D1을 사용 하였습니다.

01 클래스 플로우

협업을 통해 함께 작업 하는 것 이기에 클래스 구조에 신경을 많이 썼습니다.
최종적으로 EnemyManager를 통해 모든 일반 적을 생성 시킬 수 있고 충돌이나 거리감지와 같은 플레이어와 상호작용을 이뤄야 하는 작업들을 이 클래스에서 해주었습니다.

```
_em = new EnemyManager;
_em->init();
_em->setPlayerMemoryAddress(_player);
int rndSpownNum = RND->getFromIntTo(3,6);
for (int i = 0; i < rndSpownNum; i++)
{
    switch (RND->getInt(4))
    {
    case 0:
        _em->setSnake(RND->getFromIntTo(WINSIZE_X/2-200,WINSIZE_X/2+200),
            RND->getFromIntTo(WINSIZE_Y / 2 -150, WINSIZE_Y / 2 + 150));
        break;
    case 1:
        _em->setMushMam(RND->getFromIntTo(WINSIZE_X / 2 - 200, WINSIZE_X / 2 + 200),
            RND->getFromIntTo(WINSIZE_Y / 2 - 150, WINSIZE_Y / 2 + 150));
        break;
    case 2:
        _em->setSlime(RND->getFromIntTo(WINSIZE_X / 2 - 200, WINSIZE_X / 2 + 200),
            RND->getFromIntTo(WINSIZE_Y / 2 - 150, WINSIZE_Y / 2 + 150));
        break;
    case 3:
        _em->setFlime(RND->getFromIntTo(WINSIZE_X / 2 - 200, WINSIZE_X / 2 + 200),
            RND->getFromIntTo(WINSIZE_Y / 2 - 150, WINSIZE_Y / 2 + 150));
        break;
    default:
        break;
    }
}
```



- 다른 팀원이 EnemyManager를 활용해 적을 불러오는 모습

Enemy.h 추상화, 다형성

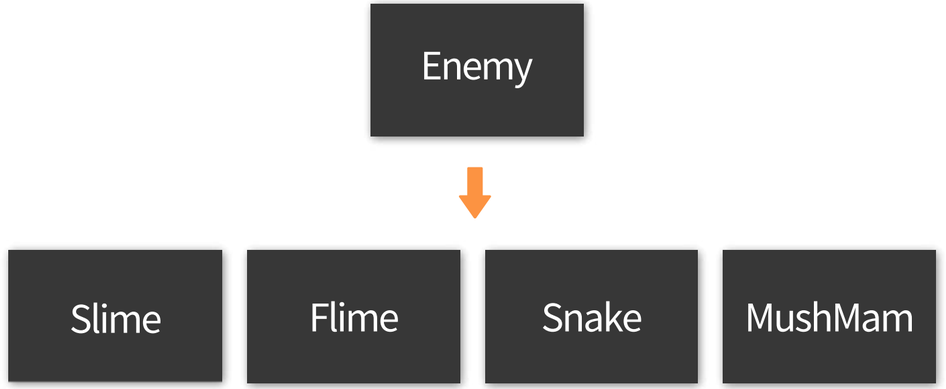
```
using namespace enemyAnimDirection;
using namespace enemyState;
class Enemy : public GameNode
{
public:

    virtual HRESULT init();
    virtual HRESULT init(const char* imageName, POINT position);

    virtual void release(void);
    virtual void update(void);
    virtual void render(void);
    virtual void d2drender(void);

    virtual void stateUpdate(void);
    virtual void draw(void);
    virtual void animation(void);
    virtual bool bulletCountFire(void);
```

- 공통되는 적의 특성은 Enemy클래스에서 정의
- 가상함수화 시켜 세부적인 내용은 하위클래스에서 정의
- 다형성, 추상화를 생각하며 설계

자식 클래스 예시

```
#pragma once
#include "Enemy.h"
class Slime : public Enemy
{
private:

    bool _slimeAttackSound;

public:

    HRESULT init(const char* imageName, POINT position);
    void release(void);
    void update(void);
    void render(void);
    void draw(void);
    void d2drender(void);
    void stateUpdate();
    void animation();

    bool bulletCountFire();

    Slime(State snakeState = IDLE) {}
    ~Slime() {}
};
```

- 예시로 Slime클래스에선 슬라임만의 특성을 정의
- 다음부턴 overide 키워드도 붙혀야겠다!



EnemyManager

```
class EnemyManager : public GameNode
{
private:
    typedef vector<Enemy*> vEnemy;
    typedef vector<Enemy*>::iterator viEnemy;
private:
    vEnemy _vSnake;
    viEnemy _viSnake;
    vEnemy _vSlime;
    viEnemy _viSlime;
    vEnemy _vFlime;
    viEnemy _viFlime;
    vEnemy _vMushMam;
    viEnemy _viMushMam;
    /*
    적 객체 추가가능..
    */
```

- 최종적으로 EnemyManager에서 Enemy타입을 만듬
- 다운캐스팅을 통해 자식클래스를 각각 관리해 줄 수 있는 구조로 구현

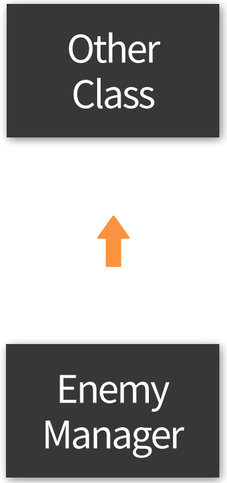
다른 팀원이 사용할 적 추가 코드

```
void EnemyManager::setSlime(int x, int y)
{
    Enemy* slime = new Slime(State::IDLE);
    slime->init("슬라임IDLE", PointMake(x, y));

    _vSlime.push_back(slime);
}
```

- 위치값을 입력받으면 그 위치로 적 생성

```
_em = new EnemyManager;
_em->init();
_em->setPlayerMemoryAddress(_player);
int rndSpownNum = RND->getFromIntTo(3,6);
for (int i = 0; i < rndSpownNum; i++)
{
    switch (RND->getInt(4))
    {
    case 0:
        _em->setSnake(RND->getFromIntTo(WINSIZE_X/2-200,WINSIZE_X/2+200),
            RND->getFromIntTo(WINSIZE_Y / 2 -150, WINSIZE_Y / 2 + 150));
        break;
    case 1:
        _em->setMushMam(RND->getFromIntTo(WINSIZE_X / 2 - 200, WINSIZE_X / 2 + 200),
            RND->getFromIntTo(WINSIZE_Y / 2 - 150, WINSIZE_Y / 2 + 150));
        break;
    case 2:
        _em->setSlime(RND->getFromIntTo(WINSIZE_X / 2 - 200, WINSIZE_X / 2 + 200),
            RND->getFromIntTo(WINSIZE_Y / 2 - 150, WINSIZE_Y / 2 + 150));
        break;
    case 3:
        _em->setFlime(RND->getFromIntTo(WINSIZE_X / 2 - 200, WINSIZE_X / 2 + 200),
            RND->getFromIntTo(WINSIZE_Y / 2 - 150, WINSIZE_Y / 2 + 150));
        break;
    default:
        break;
    }
}
```

- 다른 팀원이 EnemyManager를 활용해 적을 불러오는 모습



Utility

```
namespace MY_UTIL
{
    float getDistance(float startX, float startY, float endX, float endY)
    {
        float x = endX - startX;
        float y = endY - startY;

        return sqrt(x * x + y * y);
    }


    float getAngle(float startX, float startY, float endX, float endY)
    {
        float x = endX - startX;
        float y = endY - startY;
        float d = sqrt(x * x + y * y);

        float angle = acos(x / d);

        if (y > 0) angle = PI_2 - angle;

        return angle;
    }
```

- 거리와 각도에 대한 부분은 namespace로 관리해 주었습니다.

고민했던 부분과 어려웠던 부분

효율적인 설계를 함에 있어 정말 많이 고민했습니다. 혼자 작업하는 것이 아닌 협업을 통해 프로젝트를 제작하는 것 이였기 때문에 객체지향 설계의 중요성을 몸소 체감할 수 있는 시간이였습니다. 데이터 전달 방식에 대해 계획이 필요했고 소통과 설계가 얼마나 중요한지 알 수 있게 되는 뜻깊은 경험을 했습니다. 또한 다른 팀원이 작성한 코드를 분석하고 이해하는 코드 리딩 능력도 요구되어 다양한 스타일과 접근 방식을 경험할 수 있던 시간이였습니다.

02 AI

자연스러운 AI 구현을 위해 플레이어의 위치 값을 필요한 타이밍에 갱신받고 IDLE, ATTACK등 애니메이션의 한 사이클이 순환할 때 마다 확인을 했습니다. 정해준 애니메이션 사이클이 끝나면 다음 사이클로 전환하는 방식입니다.

순찰

근처에 플레이어가 없는 경우 랜덤한 방향으로 주변을 순찰합니다. 보는 방향의 갱신 타이밍은 애니메이션 사이클이 끝날때마다 체크해주었습니다.

공격

예시로 스네이크 같은 경우에는 초기 공격모션을 취하는 타이밍에 플레이어의 위치를 갱신받아 정해진 Bullet을 소모합니다.

터렛

각도를 실시간으로 갱신받아 플레이어를 추격하는 유도탄을 생성합니다.

최대한 원작과 비슷하고 자연스러운 AI 구현을 하는데 중점을 두었습니다.

EnemyManager - 캡슐화

```
private:
    void enemyBulletFire(vector<Enemy*> vEnemyName,
        vector<Enemy*>::iterator viEnemyName);
    void enemyDirectionTracking(vector<Enemy*> vEnemyName,
        vector<Enemy*>::iterator viEnemyName);
    void enemyPositionTracking(vector<Enemy*> vEnemyName,
        vector<Enemy*>::iterator viEnemyName);
```

- EnemyManager에서 방향추적 위치추적 공격 각도등의 정의를 **캡슐화 시켜 정의**

최종적으로 외부로 노출된 함수

```
void EnemyManager::rangeInPlayer(vector<Enemy*> vEnemyName, vector<Enemy*>::iterator viEnemyName)
{
    for (viEnemyName = vEnemyName.begin(); viEnemyName != vEnemyName.end(); ++viEnemyName)
    {
        float playerX = _player->getRect().left + (_player->getRect().right - _player->getRect().left) /2;
        float playerY = _player->getRect().top + (_player->getRect().bottom - _player->getRect().top) / 2;

        if (getDistance((*viEnemyName)->getCenterPos().x, (*viEnemyName)->getCenterPos().y,
            playerX,
            playerY) < 400.f)
        {
            float angle = getAngle((*viEnemyName)->getCenterPos().x, (*viEnemyName)->getCenterPos().y,
                playerX,
                playerY);
            (*viEnemyName)->setAngle(angle);
            enemyPositionTracking(vEnemyName, viEnemyName);
            enemyDirectionTracking(vEnemyName, viEnemyName);
            enemyBulletFire(vEnemyName, viEnemyName);
        }
    }
}
```

- 플레이어가 거리 내에 있으면 앞선 함수를 작동시켜 위치를 갱신을 받음
- 거리 밖에 있을땐 랜덤한 위치로 주변을 패트롤

```
void EnemyManager::enemyDirectionTracking(vector<Enemy*> vEnemyName,
    vector<Enemy*>::iterator viEnemyName)
{
        (*viEnemyName)->setAnimDirectionAngle(getAngle((*viEnemyName)->getCenterPos().x,
            (*viEnemyName)->getCenterPos().y,
            _player->getRect().left + (_player->getRect().right -
                _player->getRect().left) / 2,
            _player->getRect().top + (_player->getRect().bottom -
                _player->getRect().top) / 2) - 0.785f);
        if ((*viEnemyName)->getState() == MOVE || (*viEnemyName)->getState() == IDLE)
        {
            if ((*viEnemyName)->getAnimDirectionAngle() >= 0.f &&
                (*viEnemyName)->getAnimDirectionAngle() < 1.543f)
            {
                (*viEnemyName)->setAnimDirection(UP);
            }
            if ((*viEnemyName)->getAnimDirectionAngle() >= 1.543f &&
                (*viEnemyName)->getAnimDirectionAngle() < 3.113f)
            {
                (*viEnemyName)->setAnimDirection(LEFT);
            }
            if ((*viEnemyName)->getAnimDirectionAngle() >= 3.113f &&
                (*viEnemyName)->getAnimDirectionAngle() < 4.656f)
            {
                (*viEnemyName)->setAnimDirection(DOWN);
            }
            if ((*viEnemyName)->getAnimDirectionAngle() >= 4.656f &&
                (*viEnemyName)->getAnimDirectionAngle() < 6.28f ||
                (*viEnemyName)->getAnimDirectionAngle() < 0 &&
                (*viEnemyName)->getAnimDirectionAngle() > -0.785f)
            {
                (*viEnemyName)->setAnimDirection(RIGHT);
            }
        }
}
```

- 방향 애니메이션 갱신

```
void EnemyManager::enemyBulletFire(vector<Enemy*> vEnemyName,
    vector<Enemy*>::iterator viEnemyName)
{
    (*viEnemyName)->setCanAttack(true);
    if (!(*viEnemyName)->getFireAngleInit())
    {
        float startX = (*viEnemyName)->getCenterPos().x;
        float startY = (*viEnemyName)->getCenterPos().y;
        float endX = _player->getPosition().x;
        float endY = _player->getPosition().y;

        (*viEnemyName)->setThornPos(PointMake(_player->getPosition().x,
            _player->getPosition().y));

        (*viEnemyName)->setFireAngle(getAngle(startX, startY, endX, endY));

        (*viEnemyName)->setFireAngleInit(true);

    }
}
```

- 플레이어의 위치를 받아 공격할 각도를 갱신

```
void EnemyManager::enemyPositionTracking(vector<Enemy*> vEnemyName,
    vector<Enemy*>::iterator viEnemyName)
{
        if (getDistance((*viEnemyName)->getRect().left +
            ((*viEnemyName)->getRect().right -
                (*viEnemyName)->getRect().left) / 2,
            (*viEnemyName)->getRect().top +
            ((*viEnemyName)->getRect().bottom -
                (*viEnemyName)->getRect().top) / 2,
            _player->getRect().left +
            (_player->getRect().right - _player->getRect().left) / 2,
            _player->getRect().top +
            (_player->getRect().bottom - _player->getRect().top) / 2) < 50.f)
        {
            (*viEnemyName)->setSafeDistance(false);
        }
        else if (getDistance((*viEnemyName)->getRect().left +
            ((*viEnemyName)->getRect().right - (*viEnemyName)->getRect().left) / 2,
            (*viEnemyName)->getRect().top + ((*viEnemyName)->getRect().bottom -
                (*viEnemyName)->getRect().top) / 2,
            _player->getRect().left + (_player->getRect().right -
                _player->getRect().left) / 2,
            _player->getRect().top + (_player->getRect().bottom -
                _player->getRect().top) / 2) > 50.f)
        {
            (*viEnemyName)->setSafeDistance(true);
        }
}
```

- 플레이어 위치추적 갱신

Flime

```
int tempSpeed;
switch (_state)
{
case IDLE:
    if (_idleFrameCount < 3)
    {

        tempSpeed = 500;
        if (tempSpeed + _frameCount <= GetTickCount64())
        {
            _frameCount = GetTickCount64();
            _currentFrameX++;

            if (_image->getMaxFrameX() < _currentFrameX)
            {
                _currentFrameX = 0;
                _idleFrameCount++;
            }
            _image->setFrameX(_currentFrameX);
        }
    }
```

- EnemyManager를 통해 바뀐 변수들은 각각의 적에서 이러한 구조로 동작을 변환해 주어 부드러운 AI를 구현

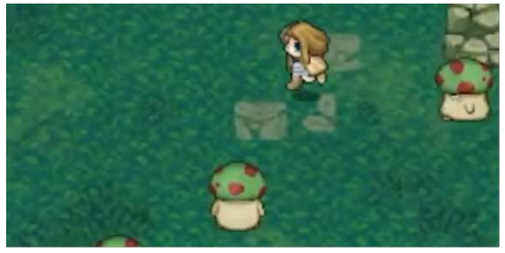
```
else
{
    resetIdleCount();
    resetRndIdleFrame();

    _fireAngleInit = false;
    if (_attackCount == 2)
    {
        setState(ATTACK);
    }
    else if (_attackCount == 1)
    {
        setState(ATTACK2);
    }
    else
    {
        setState(ATTACK);
        _attackCount = 2;
    }
}

break;
```

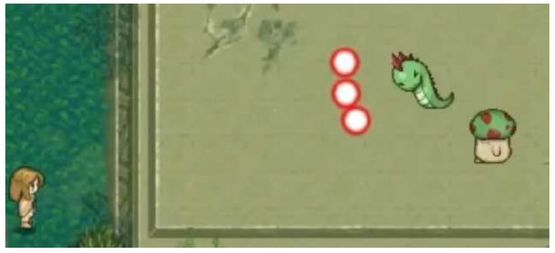
- 예를들어 Flime의 경우엔 정해진 애니메이션 프레임이 다 돌면 그 시점에 공격 각도를 갱신
- 현재의 상태에 따라 몇번째 공격패턴을 시전할 것인지 결정

```
case ATTACK:
    if (_attackCount == 2)
    {
        tempSpeed = 300;
        if (tempSpeed + _frameCount <= GetTickCount64())
        {
            _frameCount = GetTickCount64();
            _currentFrameX++;
            if (_image->getMaxFrameX() < _currentFrameX)
            {

                _attackCount--;
                float angleStep = PI / 5;
                SOUNDMANAGER->play("플라임독발사", 0.1f);
                for (int i = 0; i < 10; ++i)
                {
                    float fireangle = i * angleStep;

                    _poisonBullet->fire(_x, _y, fireangle, 1.f);


                }
                _currentFrameX = 0;
                _image->setFrameX(_currentFrameX);
            }

        }
    }
    else
    {
        setState(IDLE);
    }
    break;
```

- 애니메이션이 끝나는 타이밍에 총알을 생성해줌
- 각각의 공격 횟수가 끝나면 상태를 다시 IDLE로 돌려 AI가 반복할 수 있게 구조를 작성

결과 화면





고민했던 부분과 어려웠던 부분

초기 설계한 구조에서 코드를 한줄 한줄 추가할 때 마다 어느 부분에 추가를 해야 가장 효율적일까? 에 대한 고민을 가장 많이했습니다. 결과적으로 어떻게든 작동하는 구조는 많았지만 가장 효율적인 구조를 찾기 위해 고민했고 현재 구조가 가장 효율적이라고 판단해 이러한 구조로 작성했습니다. 또한 자연스러운 AI를 구현하기 위한 고민도 많이 했었는데 각각의 애니메이션 프레임이 끝나는 시점에 행동을 주어 자연스러운 AI를 구현 했습니다.

03 탄막 패턴 - 수학적 개념 코드적용

로그라이크 탄막 슈터라는 게임 장르의 특성상 다양한 탄막패턴을 구현하기 위해 std::function과 람다식, stl과 삼각함수등을 적극적으로 활용했습니다.
Bullet은 클래스를 만들어 vector로 관리해 주었으며 총알이 발사되는 타이밍에 push_back되며 최대 사정거리를 벗어나면 지워주었습니다.

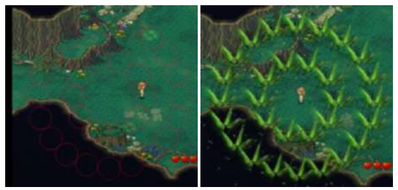
보스주요 패턴

킹슬라임의 탄막 패턴

Bubble이 사라지는 타이밍에 탄막이 원형으로 생성되어야 하는 킹슬라임의 패턴을 구현하기 위해 std::function과 람다식을 활용해 callback함수를 만들어 구현하였습니다.

메가 플라임의 탄막 패턴1

가시의 간격, 각도, 갈래 등을 이중 반복문으로 처리했습니다. 자세한 구현 내용은 아래 코드와 함께 말씀드리겠습니다.



메가 플라임의 탄막 패턴2

플레이어의 위치를 갱신받는 가시가 총 3번 생성됩니다. 안전한 자리의 위치는 각 가시 생성마다 달라집니다.

머쉬맘 패턴

직접 공격하는 것이 아닌 지속적으로 탄막을 발사하는 터렛을 생성합니다. 그 특성에 맞게 보스는 플레이어의 거리가 가깝다면 멀어지고 너무 멀다면 다시 가까워지려 하는 특성이 있습니다.

탄막 관리

```h
class EnemyBullet : public GameNode
{
private:
    vector<tagEnemyBullet> _vBullet;
    vector<tagEnemyBullet>::iterator _viBullet;

    string _imageName;
    int _bulletMax;
    std::function<void(float,float)> _bulletRemovedCallback;

    float _range;

    int _moveType;

public:
    HRESULT init(string imageName, int bulletMax, float range,int moveType);
    void release(void);
    void update(void);
    void render(void);
    void d2drender(void);

    void fire(float x, float y, float angle, float speed);

    void draw(void);
    void move(void);
    void move2(void);
    void removeBullet(int arrNum);
    vector<tagEnemyBullet> getBullet(void) { return _vBullet; }
    void setBulletRemovedCallback(std::function<void(float,float)> callback);
    EnemyBullet(void) {}
    virtual ~EnemyBullet() {}
};
```

- 구조체를 이용해 공통되는 bullet 속성을 정의 후 vector로 관리
- 총알의 특성상 언제 소멸할지 모르고 다형성을 위해 소멸자를 가상화

```h
void EnemyBullet::fire(float x, float y, float angle, float speed)
{
    if (_bulletMax <= _vBullet.size()) return;

    tagEnemyBullet bullet;
    ZeroMemory(&bullet, sizeof(tagEnemyBullet));
    bullet.img = D2DMANAGER->findImage(_imageName);
    bullet.angle = angle;
    bullet.count = 0;

    bullet.initialAngle = angle;
    bullet.currentAngle = 0;

    bullet.speed = speed;
    bullet.x = bullet.fireX = x;
    bullet.y = bullet.fireY = y;
    bullet.rc = RectMakeCenter(bullet.x, bullet.y,
        D2DMANAGER->findImage(_imageName)->width,
        D2DMANAGER->findImage(_imageName)->height);

    _vBullet.push_back(bullet);
}
```

- fire함수가 호출 되는 시점에 vector에 요소 추가
- 이후 move함수를 통해 각각에 탄막 특성에 맞게 움직임을 구현

킹슬라임의 탄막 패턴

EnemyBullet

```
class EnemyBullet : public GameNode
{
private:
    vector<tagEnemyBullet> _vBullet;
    vector<tagEnemyBullet>::iterator _viBullet;

    string _imageName;
    int _bulletMax;
    //bool _rangeOver;

    std::function<void(float,float)> _bulletRemovedCallback;

    float _range;
```

```
void EnemyBullet::setBulletRemovedCallback(std::function<void(float,float)> callback)
{
    _bulletRemovedCallback = callback;
}
```

- EnemyBullet 클래스에 bubble이 사라지는 시점을 기억하기 위해 매개변수로 std::fundtion을 이용해 callback함수를 생성

EnemyBullet.cpp

```
void EnemyBullet::move(void)
{
    for (_viBullet = _vBullet.begin(); _viBullet != _vBullet.end();)
    {
        _viBullet->x += cos(_viBullet->angle) * _viBullet->speed;
        _viBullet->y += -sinf(_viBullet->angle) * _viBullet->speed;

        _viBullet->rc = RectMakeCenter(_viBullet->x, _viBullet->y,
            _viBullet->img->width,
            _viBullet->img->height);

        if (_range <= getDistance(_viBullet->fireX, _viBullet->fireY,
            _viBullet->x, _viBullet->y))
        {
            float deleteX = _viBullet->x;
            float deleteY = _viBullet->y;
            _viBullet = _vBullet.erase(_viBullet);
            if (_bulletRemovedCallback)
            {
                _bulletRemovedCallback(deleteX, deleteY);
            }

        }
        else
        {
            ++_viBullet;

        }
    }
}
```

- EnemyBullet의 move함수
- 버블이 최대 거리를 이동하면 사라지며 사라진 위치 값을 콜백함수에 전달

KingSlime의 init함수

```
_bubble = new EnemyBullet;
_bubble->init("버블", 100, 270,0);
_bubbleBullet = new BubbleBullet;
_bubbleBullet->init("큰일반탄", 500, 130);

//이곳은 init함수 내부

function<void(float, float)> callback = [this](float X,float Y)
{
    float angleStep = PI / 5;
    for (int i = 0; i < 10; ++i)
    {
        float fireangle = i * angleStep;
        this->_bubbleBullet->fire(X, Y, fireangle, 2);
    }
};
this->_bubble->setBulletRemovedCallback(callback);

return S_OK;
```

- KingSlime의 초기화 함수 내부에서 콜백함수를 람다식으로 생성
- 매개변수로 X Y값을 받아 그 위치에서 fire함수 작동
- setBulletRemovedCallback에 람다식을 전달해 사라진 위치의 X Y 값을 전달 받아 총알이 생성될 수 있게 구현

BubbleBullet

```
void BubbleBullet::move(void)
{
    float rotateSpeed = 0.03f;
    for (_viBullet = _vBullet.begin(); _viBullet != _vBullet.end();)
    {
        if (!_viBullet->reverse)
        {
            _viBullet->angle += rotateSpeed;
            _viBullet->x += cos(_viBullet->angle) * _viBullet->speed;
            _viBullet->y += -sinf(_viBullet->angle) * _viBullet->speed;
        }
        else
        {
            _viBullet->angle -= rotateSpeed;
            _viBullet->x -= cos(_viBullet->angle) * _viBullet->speed;
            _viBullet->y -= -sinf(_viBullet->angle) * _viBullet->speed;
        }
        _viBullet->rc = RectMakeCenter(_viBullet->x, _viBullet->y,
            _viBullet->img->width,
            _viBullet->img->height);
        if (_range < getDistance(_viBullet->fireX, _viBullet->fireY,
            _viBullet->x, _viBullet->y))
        {
            _viBullet->reverse = true;

        }
        if( 10 > getDistance(_viBullet->fireX, _viBullet->fireY,
            _viBullet->x, _viBullet->y) &&
            _viBullet->reverse)
        {
            _viBullet = _vBullet.erase(_viBullet);
        }
        else
        {
            ++_viBullet;
        }
    }
}
```

- 버블로 인해 생성된 총알은 생성돼서 반지름이 커지다 다시 작아지는 로직을 작성

최종 결과 화면

고민했던 부분과 어려웠던 부분

Bubllet이 사라지는 시점을 보다 효율적으로 관리할 수 있는 부분이 무엇이있을까에 대해 고민을 했습니다. 단순히 bool변수를 이용해서 구현하는 방법도 있었지만 좀 더 괜찮은 방법이 없을까 고민하다 익명함수와 function을 활용해서 한번 해보자 해서 이러한 방법으로 구현하였습니다. 결과적으로 확장성 있는 효율적인 코드를 작성한 것 같아 배운점이 많았습니다.

메가플라임 탄막 패턴

```
_bigBullet = new EnemyBullet;
_bigBullet->init("큰일반탄", 300, 300,1);
_thorn = new Thorn;
_thorn->init("가시", 500, 500);
_poisonBullet = new PoisonBullet;
_poisonBullet->init("독탄", 50, 300);
_smallBullet = new EnemyBullet;
_smallBullet->init("큰일반탄", 300, 300, 0);
```

- MegaFlime이 사용할 Bullet의 종류를 초기화

```
void Thorn::draw(void)
{
    for (_viBullet = _vBullet.begin(); _viBullet != _vBullet.end(); ++_viBullet)
    {
        _viBullet->previewCnt++;
        if (_viBullet->previewCnt < 30)
        {
            D2DMANAGER->render("가시미리보기", _viBullet->x - 32, _viBullet->y- 32);
        }
        else
        {
            D2DMANAGER->frameAlphaRender(_imageName, _viBullet->rc.left, _viBullet->rc.top,
                _viBullet->frameX, _viBullet->frameY, _viBullet->alpha);

            _viBullet->count++;


            if (_viBullet->count % 15 == 0)
            {
                _viBullet->frameX++;
                if (_viBullet->frameX >= D2DMANAGER->findImage(_imageName)->maxFrameX)
                {
                    //SD: 가시 생성 사운드
                    if(_viBullet->frameX == 4 && _viBullet->frameY == 0)
                        SOUNDMANAGER->play("플라임가시생성", 0.1f);

                    if (_viBullet->frameY >= D2DMANAGER->findImage(_imageName)->maxFrameY)
                    {
                        _viBullet->frameY = _viBullet->img->maxFrameY;
                        _viBullet->frameX = _viBullet->img->maxFrameX - 1;
                    }
                    else
                    {
                        _viBullet->frameY++;
                        _viBullet->frameX = 0;
                    }
                }
                _viBullet->count = 0;
            }
        }
    }
```

- 가시는 잠깐의 previeTime을 가진 후 생성됩니다.

첫번째 패턴

```
if (_thornCount == 3)
{
    SOUNDMANAGER->play("플라임가시생성", 0.1f);

    float angleStep = PI * 2 / 6;
    float radiusStep = 50;
    float rotationOffset = PI / 30;

    for (int i = 0; i < 6; ++i)
    {
        float baseAngle = i * angleStep;
        for (int j = 0; j < 7; ++j)
        {
            float fireangle = baseAngle + j * rotationOffset;
            float radiusG = (j + 2) * radiusStep;
            _thorn->fire(_x, _y, fireangle, radiusG, 530);
        }
    }
}
```

1. 회오리 처럼 생성되는 가시에 대한 코드
2. 가시간 반지름 간격, 6갈래 각도 , 회전 간격 (휜 정도)를 설정
3. 기본 각도 계산 (6갈래)
4. 회전 간격 적용 (7개의 가시)

결과 화면

첫번째 패턴

```
if (_thornCount == 2)
{
    float angleStep = PI * 2 / 6;
    float speed = 0.06f;
    for (int i = 0; i < 6; ++i)
    {
        float baseAngle = i * angleStep;
        _bigBullet->fire(_x, _y, baseAngle, speed);
    }
}
```

- 가시 생성 후 시간이 지나면 총알 생성
- 생성될 위치와 각도를 fire함수에 넘겨줌

```
void EnemyBullet::move2(void)
{
    float rotateSpeed = 0.03f;

    for (_viBullet = _vBullet.begin(); _viBullet != _vBullet.end();)
    {
        _viBullet->currentAngle += rotateSpeed;


        float radiusG = _viBullet->speed *
            (_viBullet->currentAngle / rotateSpeed);
        _viBullet->x += cos(_viBullet->initialAngle +
            _viBullet->currentAngle) * (radiusG * 1.2f);
        _viBullet->y -= sin(_viBullet->initialAngle +
            _viBullet->currentAngle + 0.3f) * (radiusG * 1.2f);
        _viBullet->rc = RectMakeCenter(_viBullet->x, _viBullet->y,
            _viBullet->img->width,
            _viBullet->img->height);

        if (_range <= getDistance(_viBullet->fireX, _viBullet->fireY,
            _viBullet->x, _viBullet->y))
        {
            _viBullet = _vBullet.erase(_viBullet);
        }
        else
        {
            ++_viBullet;
        }
    }
}
```

1. moveType를 1로 넘겨줘 move2 함수로 진입
2. 회전 속도를 정의해 현재 각도에 회전속도를 합함
3. 총알의 이동 경로 계산
4. 총알의 새로운 x,y좌표를 계산 후 위치 업데이트
5. cos, sin 함수를 이용해 원형 경로를 따라 이동

첫번째 패턴의 최종 결과 화면





두번째 패턴

```
if (_thornCount == 3)
{
    float angleStep = PI / 5;
    for (int i = 0; i < 10; ++i)
    {
        float fireangle = i * angleStep;
        _smallBullet->fire(_x, _y, fireangle, 2.f);
    }
}
else if (_thornCount == 2)
{
    float angleStep = PI / 5;
    for (int i = 0; i < 10; ++i)
    {
        float fireangle = i * angleStep;
        _smallBullet->fire(_x, _y, fireangle, 2.f);
    }


    for (int i = 0; i < 10; ++i)
    {
        float fireangle = i * angleStep;
        _poisonBullet->fire(_x, _y, fireangle, 1.f);
    }
}
else if (_thornCount == 1)
{
    float angleStep = PI / 5;
    for (int i = 0; i < 10; ++i)
    {
        float fireangle = i * angleStep;
        _smallBullet->fire(_x, _y, fireangle, 2.f);
    }
}
```

- 시간에 맞게 변화되는 _thornCount를 이용해 원형으로 각 패턴에 맞는 Bullet을 생성

최종 결과 화면

세번째 패턴

```
float radiusIncrement = 100;
for (int i = 1; i <= 2; ++i)
{
    float angleIncrement = (PI * 2) / (10 * i);
    float radius = radiusIncrement * i;

    for (float angle = 0; angle < PI * 2; angle += angleIncrement)
    {
        _thorn->fire(_thornPos.x + cos(angle) * radius,
            _thornPos.y + sin(angle) * radius, 64);
    }
}
```

```
float radiusIncrement = 150;
for (int i = 0; i <= 1; ++i)
{
    float radius = radiusIncrement * i;

    if (i == 0)
    {
        _thorn->fire(_thornPos.x, _thornPos.y, 64);
    }
    else
    {
        float angleIncrement = (PI * 2) / (20 * i);
        for (float angle = 0; angle < PI * 2; angle += angleIncrement)
        {
            _thorn->fire(_thornPos.x + cos(angle) * radius,
                _thornPos.y + sin(angle) * radius, 64);
        }
    }
}
```

- angleIncrement는 각 원에서 가시가 생성되는 각도의 증가량
- 반지름은 radiusIncrement에 i를 곱해 계산. 즉 i가 증가할수록 원의 반지름 증가 (패턴에 맞게 i값 조절, 총 3회)
- 두번째 for루프에서 각 원에서 가시를 방사형으로 생성
- angle은 현재 가시가 생성되는 각도를 나타내며 angleIncrement값 만큼 증가시켜줌

세번째 패턴의 최종 결과 화면

어려웠던점

탄막 패턴이 가장 많이 들어간 보스인 만큼 탄막 알고리즘을 작성하는 것에 고민을 하고 코드를 작성하다 보니 실제 게임과 다른 부분과 내가 의도한 것과는 다른 탄막 패턴이 구현되어 여러번의 수정 결과 원작과 가장 유사한 탄막 패턴을 구현할 수 있었습니다.

머쉬맘 탄막 패턴

Mushmam

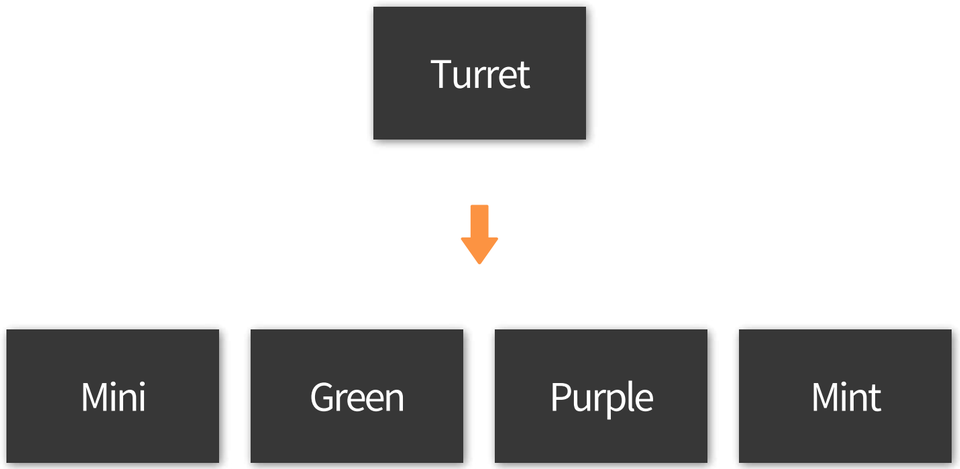
```
private:
    typedef vector<Turret*> vTurret;
    typedef vector<Turret*>::iterator viTurret;

private:
    vTurret _vMiniTurret;
    viTurret _viMiniTurret;
    vTurret _vGreenTurret;
    viTurret _viGreenTurret;
    vTurret _vPurpleTurret;
    viTurret _viPurpleTurret;
    vTurret _vMintTurret;
    viTurret _viMintTurret;
```

- 터렛 생성 준비

```
_espBullet = new EspBullet;
_espBullet->init("유도탄", 100, 150);
_greenEspBullet = new EspBullet;
_greenEspBullet->init("유도탄", 100, 50);
_bigBullet = new EnemyBullet;
_bigBullet->init("큰일반탄", 100, 400, 0);
_poisonBullet = new PoisonBullet;
_poisonBullet->init("독탄", 300, 800);
```

- 터렛의 총알 준비

Turret의 클래스 구조도



터렛

```
class Turret : public GameNode
{
protected:

    GImage* _image;
    RECT _rc;
    float _x, _y;
    POINT _center;

    float _fireTime;
    float _bulletFireCount;
    int _fireCount;
    int _hp;
    bool _lastFire;
    bool _die;


public:
    virtual HRESULT init(const char* imageName, POINT position);
    virtual void release(void);
    virtual void update(void);
    virtual void render(void);
    virtual void d2drender(void);
    virtual bool bulletCountFire();
    virtual void draw(void);
```

- Turret 클래스로 공통된 특성을 정의
- 공격속도, 자폭 등 터렛마다 다른 특성을 하위클래스에서 구체화

```
HRESULT PurpleTurret::init(const char* imageName, POINT position)
{
    Turret::init(imageName, position);

    _fireTime = 8.f;
    return S_OK;
}

bool PurpleTurret::bulletCountFire()
{
    if (_fireTime + _bulletFireCount <= TIMEMANAGER->getWorldTime())
    {
        _bulletFireCount = TIMEMANAGER->getWorldTime();
        _fireTime = 8.f;
        return true;
    }
    return false;
}
```

```
HRESULT MiniGreenTurret::init(const char* imageName, POINT position)
{
    Turret::init(imageName,position);
    _fireCount = 5;
    _fireTime = RND->getFromFloatTo(1.5f,3.5f);
    return S_OK;
}

bool MiniGreenTurret::bulletCountFire()
{
    if (_fireTime + _bulletFireCount <= TIMEMANAGER->getWorldTime())
    {
        _fireCount--;
        if (_fireCount == 1)
        {
            //SD: 미니버섯 터지기 전 사운드 (띵..띵..띵!)
            SOUNDMANAGER->play("미니버섯쪽발카운트", 1.0f);
        }

        if (_fireCount > 0)
        {
            _bulletFireCount = TIMEMANAGER->getWorldTime();
            _fireTime = RND->getFromFloatTo(1.5f, 3.5f);

            return true;
        }
        else
        {
            _lastFire = true;

        }

    }
    return false;
```

MushmamBoss.cpp

```
void MushMamBoss::setMiniTurret(int x, int y)
{
    Turret* miniGreen = new MiniGreenTurret;
    miniGreen->init("초록포탑", PointMake(x, y));
    _vMiniTurret.push_back(miniGreen);
}

void MushMamBoss::setGreenTurret(int x, int y)
{
    Turret* green = new GreenTurret;
    green->init("큰초록포탑", PointMake(x, y));
    _vGreenTurret.push_back(green);
}

void MushMamBoss::setPurpleTurret(int x, int y)
{
    Turret* purple = new PurpleTurret;
    purple->init("큰보라포탑", PointMake(x, y));
    _vPurpleTurret.push_back(purple);
}

void MushMamBoss::setMintTurret(int x, int y)
{
    Turret* mint = new MintTurret;
    mint->init("큰민트포탑", PointMake(x, y));
    _vMintTurret.push_back(mint);
}
```

MushmamBoss.cpp

```
void MushMamBoss::removeMiniTurret(int arrNum)
{
    SAFE_DELETE(_vMiniTurret[arrNum]);
    _vMiniTurret.erase(_vMiniTurret.begin() + arrNum);
}

void MushMamBoss::removeGreenTurret(int arrNum)
{
    SAFE_DELETE(_vGreenTurret[arrNum]);
    _vGreenTurret.erase(_vGreenTurret.begin() + arrNum);
}

void MushMamBoss::removePurpleTurret(int arrNum)
{
    SAFE_DELETE(_vPurpleTurret[arrNum]);
    _vPurpleTurret.erase(_vPurpleTurret.begin() + arrNum);
}

void MushMamBoss::removeMintTurret(int arrNum)
{
    SAFE_DELETE(_vMintTurret[arrNum]);
    _vMintTurret.erase(_vMintTurret.begin() + arrNum);
}
```

- 필요한 시점에 터렛을 소환하고 지속시간이 지나면 터렛이 삭제되기 함수로 묶어 vector로 관리

Mushmam의 TurretBulletFire

```
for (_viMiniTurret = _vMiniTurret.begin(); _viMiniTurret != _vMiniTurret.end(); ++_viMiniTurret)
{
    if ((*_viMiniTurret)->bulletCountFire())
    {
        RECT rc = (*_viMiniTurret)->getRect();
        _espBullet->fire(rc.left + (rc.right - rc.left) / 2,
            rc.bottom + (rc.top - rc.bottom) / 2,
            getAngle(rc.left + (rc.right - rc.left) / 2,
                rc.bottom + (rc.top - rc.bottom) / 2,
                _player->getPosition().x,
                _player->getPosition().y), 3.f);
    }

    if ((*_viMiniTurret)->getLastFire())
    {
        RECT rc = (*_viMiniTurret)->getRect();
        float angleStep = PI / 10;
        for (int i = 0; i < 20; ++i)
        {
            float fireangle = i * angleStep;
            _bigBullet->fire(rc.left + (rc.right + -rc.left) / 2,
                rc.bottom + (rc.top - rc.bottom) / 2 + 30, fireangle, 3.f);
        }

        (*_viMiniTurret)->setLastFire(false);
        (*_viMiniTurret)->setDie(true);

    }
}
```

- mini와 Green터렛은 유도탄을 생성해야 하기에 각도를 넘겨줌
- mini터렛은 정해진 공격 횟수가 지나면 자폭하기에 그에 대한 fire 함수에 위치와 각을 넘겨줌

ESP Bullet

```
void EspBullet::move(void)
{
    for (_viBullet = _vBullet.begin(); _viBullet != _vBullet.end();)
    {
        float destroyRange = _range + 350;
        if (_range < getDistance(_viBullet->fireX, _viBullet->fireY, _viBullet->x, _viBullet->y))
        {
            if (!_viBullet->isDelayOver)
            {
                _viBullet->isMoving = false;
                _viBullet->fireTime++;
                if (_viBullet->fireTime % 50 == 0)
                {
                    _viBullet->isMoving = true;
                    _viBullet->isDelayOver = true;
                }
            }
        }
        if (_viBullet->isMoving)
        {
            _viBullet->x += cos(_viBullet->angle) * _viBullet->speed;
            _viBullet->y += -sinf(_viBullet->angle) * _viBullet->speed;
        }

        _viBullet->rc = RectMakeCenter(_viBullet->x, _viBullet->y, _viBullet->img->width,
            _viBullet->img->height);
        if (destroyRange < getDistance(_viBullet->fireX, _viBullet->fireY, _viBullet->x, _viBullet->y))
        {
            _viBullet = _vBullet.erase(_viBullet);
        }
        else
        {

            ++_viBullet;
        }
    }
}
```

- 생성되면 플레이어 위치로 발사되다 DelayTime을 가짐
- DelayTime동안 플레이어의 위치 각도를 추적
- 각도를 갱신하는 동안 총알의 렌더링도 함께 회전

ESPBullet

```
void EspBullet::updateAngle(int playerX, int playerY)
{
    for (_viBullet = _vBullet.begin(); _viBullet != _vBullet.end();++_viBullet)
    {
        if (!_viBullet->isMoving)
        {
            _viBullet->angle = getAngle(_viBullet->x, _viBullet->y, playerX, playerY);
        }
    }
}
```

- 움직임이 멈추었을때 각도를 갱신

```
_espBullet->update();
_espBullet->updateAngle(_player->getPosition().x, _player->getPosition().y);
_greenEspBullet->update();
_greenEspBullet->updateAngle(_player->getPosition().x, _player->getPosition().y);
```

- 업데이트 되는 각도엔 Player의 위치를 입력

ESPBullet의 draw함수

```
void EspBullet::draw(void)
{
    for (_viBullet = _vBullet.begin(); _viBullet != _vBullet.end(); ++_viBullet)
    {

        float degrees = -(_viBullet->angle) * (180 / PI);

        D2DMANAGER->rotateRender(_imageName, _viBullet->rc.left, _viBullet->rc.top, degrees+ 90);

    }
}
```

- 업데이트 한 각도를 바탕으로 화면에 렌더링

결과 화면

ESPBullet이 플레이어의 위치를 추적합니다

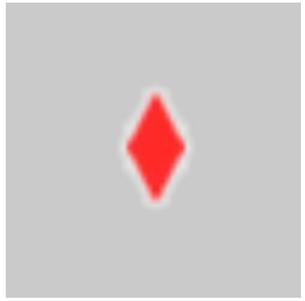
각각의 상황에 맞게 터렛을 소환하며 터렛이 가진 특성의 bullet으로 지속적으로 공격합니다.

04 이펙트 처리 및 회전

공부를 진행하며 팀 프로젝트를 진행하던 도중 편하게 사용할 공통적인 렌더링 포맷이 필요했고 비교적 가져와서 사용하기 편리한 D2D1라이브러리를 사용하였습니다.
팀원들이 원활하게 사용할 수 있게 여러가지 기능을 지원하는 함수를 만들어 클래스를 매니저화 시켰습니다.

이펙트

팀원들과 함께 작업하는 환경에서 이펙트 이미지를 비교적 편리하게 사용하기 위해서 다른 렌더링 포맷이 필요했습니다.



회전이 필요한 이미지

팀원들과 함께 작업하는 환경에서 비교적 편리하게 회전하기 위해선 다른 렌더링 포맷이 필요했습니다.

D2D 매니저

```
D2DImageInfo* loadImageD2D(string key, LPCWSTR lstr, int Width, int Height);
D2DImageInfo* loadImageD2D(string key, LPCWSTR lstr, int Width, int Height, int maxFrameX, int maxFrameY);


void render(string key, int destX, int destY);
void render(string key, int destX, int destY,
    int resizeX, int resizeY);
void frameRender(string key, int destX, int destY,
    int currentFrameX, int currentFrameY);
void frameRender(string key, int destX, int destY,
    int currentFrameX, int currentFrameY, int resizeX, int resizeY);

void alphaRender(string key, int destX, int destY,float alpha);
void alphaRender(string key, int destX, int destY,
    int resizeX, int resizeY, float alpha);

void frameAlphaRender(string key, int destX, int destY,
    int currentFrameX, int currentFrameY,float alpha);
void frameAlphaRender(string key, int destX, int destY,
    int currentFrameX, int currentFrameY, int resizeX, int resizeY,float alpha);

void rotateRender(string key, int destX, int destY,
    float angle, float alpha = 1.0f);
void rotateRender(string key, int destX, int destY,
    float angle, int resizeX, int resizeY, float alpha = 1.0f);
void frameRotateRender(string key, int destX, int destY,
    int currentFrameX, int currentFrameY, float angle, float alpha = 1.0f);
void frameRotateRender(string key, int destX, int destY,
    int currentFrameX, int currentFrameY,  int resizeX, int resizeY,
    float angle, float alpha = 1.0f);
```

- 팀원들이 원활하게 사용할 수 있게 하기 위해 여러가지 기능을 지원하는 함수를 만들어 매니저화

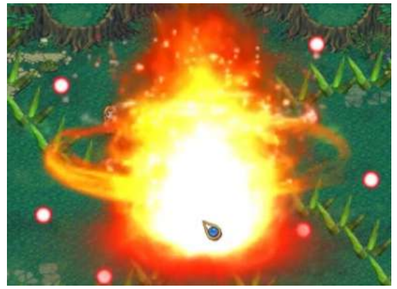
```
map<string, D2DImageInfo*> _mImage;
map<string, D2DImageInfo*>::iterator _miImage;
```

- 이미지는 map을 사용해 관리해 주었습니다.

map을 사용한 이유

팀 작업을 수행하며 효율성과 편의성을 위해 map을 사용했습니다. map은 내부적으로 레드블랙트리를 사용하여 데이터를 저장하므로 검색, 삽입, 삭제 연산을 O(log n) 시간 내에 수행할 수 있기 때문에 효율성을 챙기고 string을 key값으로 사용하여 직관적으로 사람이 알아볼 수 있기 때문에 다른 팀원들이 직관적으로 알 수 있게 하기 위해 map을 사용해 이미지를 관리해 주었습니다.

팀원들이 프레임워크를 기반해 렌더링한 이펙트 및 회전 결과



발생했던 문제점

기존 GDI기반으로 화면을 렌더링 하고 있던 상태에서 D2D1라이브러리를 가져와 사용하려 하니 **프레임 드랍**이 심하게 생겼던 문제가 있었습니다.

해결방안

```
void MainGame::render(void)
{
    PatBlt(getMemDC(), 0, 0, WINSIZE_X, WINSIZE_Y, WHITENESS);
    //=============================================
    SCENEMANAGER->render();
    D2DMANAGER->beginDraw();
    SCENEMANAGER->d2drender();
    D2DMANAGER->endDraw();
    SCENEMANAGER->UIrender();
    if(KEYMANAGER->isToggleKey(VK_F1)) TIMEMANAGER->render(getMemDC());
    //=============================================
    this->getBackBuffer()->render(getHDC());
}
```

최상위 GameCore의 관리자 클래스인 MainGame의 render함수에 D2D에 대한 렌더링 처리를 따로 진행해 주어 프레임 드랍 문제를 해결하였습니다.

느낀점

D2D1 라이브러리에 대한 심도있는 공부를 진행한 후 사용한 것이 아닌 가져와 사용한것 이지만 WindowAPI와 GDI기반 렌더링 사이의 연결점, D2D를 그냥 사용했을때 프레임 드랍이 생기는 문제 등을 해결하려 노력하면서 구성되어 있는 프레임워크에 대한 이해도가 증가할 수 있었습니다.

배운점

어느 문제를 해결하려 고민해 보는것이 코딩 실력 향상에 도움을 줄 수 있다는 사실을 몸소 배웠습니다. 이 후 팀 포트폴리오 작업을 진행하면서 사운드나 프로젝트가 갑자기 터지는 등 다른 팀원이 작성한 코드에서 어떠한 버그나 문제점이 생기면 결과적으로 제가 해결할 수 있는 능력을 얻을 수 있게 된 계기였습니다.

결과/회고

팀 프로젝트를 마치며

협업을 함에 있어 객체지향의 중요성과 소통능력이 얼마나 중요한지 알게되는 좋은 시간이였습니다.

1. 팀 프로젝트를 진행하면서 객체지향 설계의 중요성을 몸소 체감할 수 있었습니다. 개별적으로 코드를 작성하는 것이 아니라, 다른 팀원들과 공유해야 하는 코드를 작성하기 때문에, 데이터 전달 방식과 필요한 정보의 교환에 대한 계획이 필요했습니다. 이 과정에서 객체지향 원칙을 지켜 설계하는 것이 얼마나 중요한지 깨달았습니다. 또한 객체지향의 특성에 맞춰 코드를 설계하려 노력했습니다.

2. 팀 프로젝트에서는 혼자 포트폴리오를 만들 때와는 다르게 다른 팀원들이 작성한 코드를 분석하고 이해하는 능력도 필요했습니다. 다양한 스타일과 접근 방식을 경험하며 개발 역량을 확장할 수 있는 기회였습니다.

3. 마무리 하며 이런 협업 경험들이 앞으로 게임 개발자로 살아가는데 큰 도움이 될 것 같아 매우 의미있는 시간이였다고 생각합니다.

영상을 보고 싶으시면 여기를 클릭해주세요

Name 김유민
Moblie 010 7480 0851
E-mail rladbals0129@naver.com