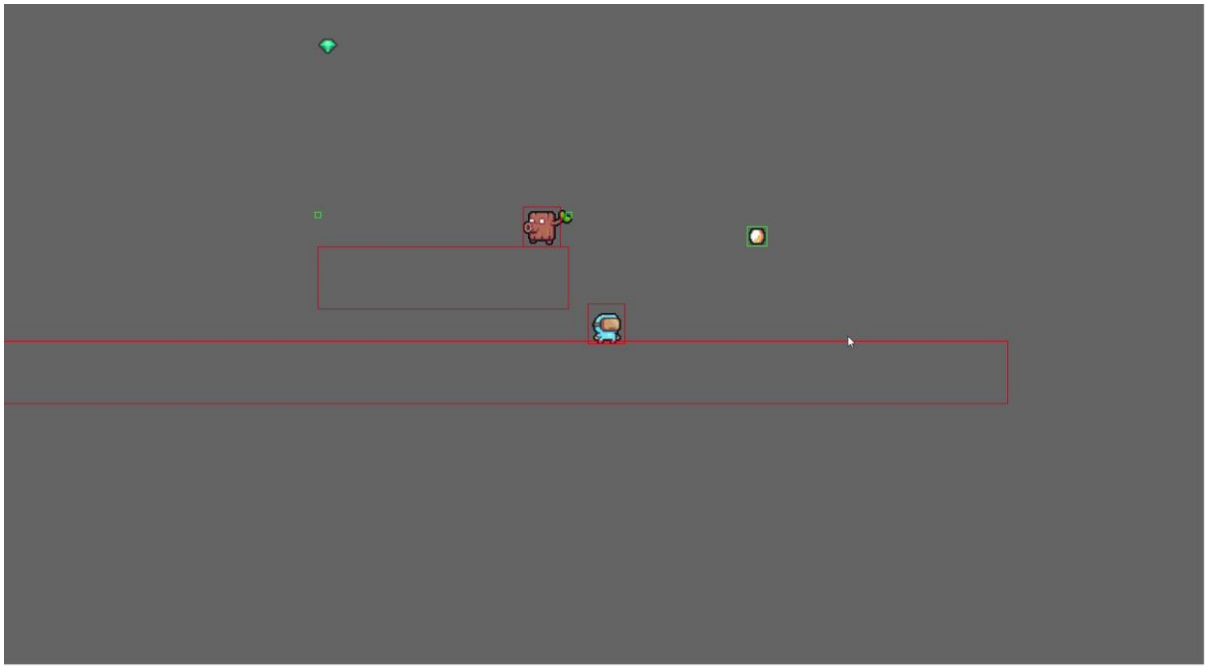
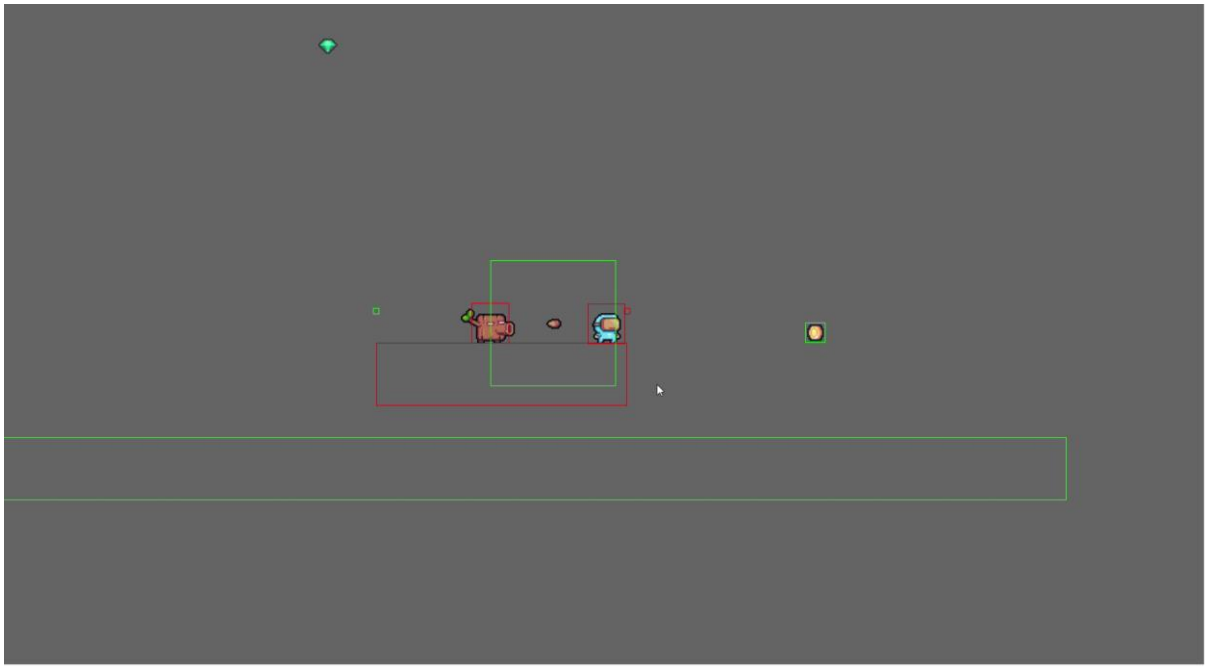
2주차_06/24 ~ 06/30

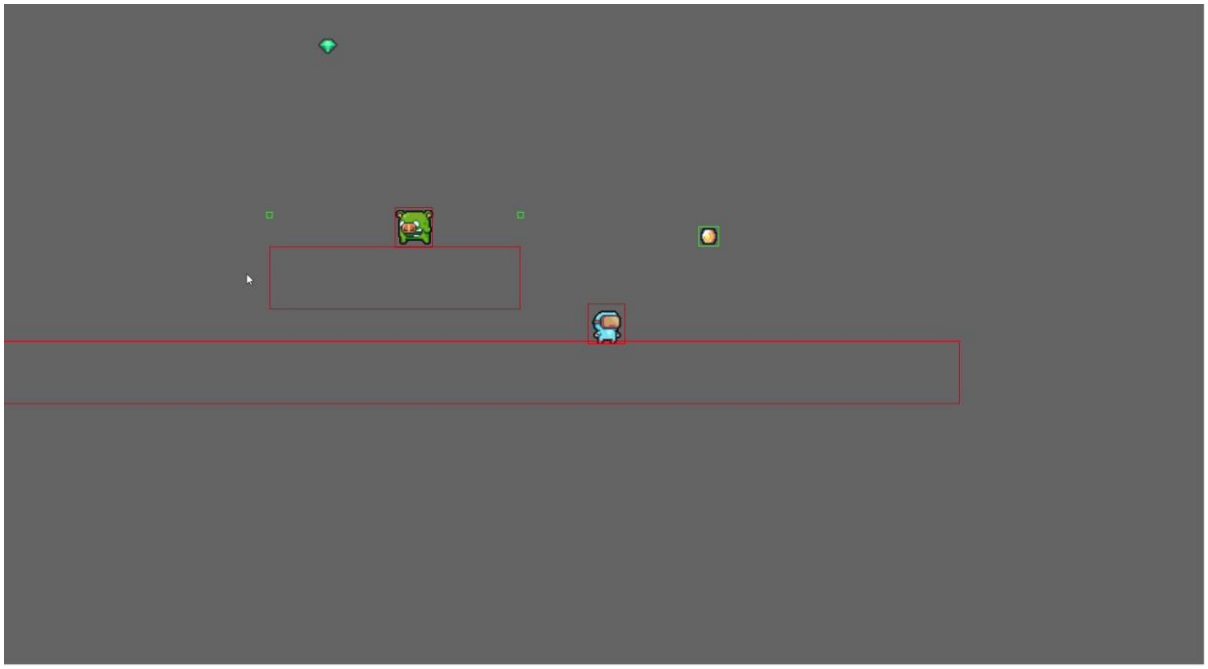
- 06/24

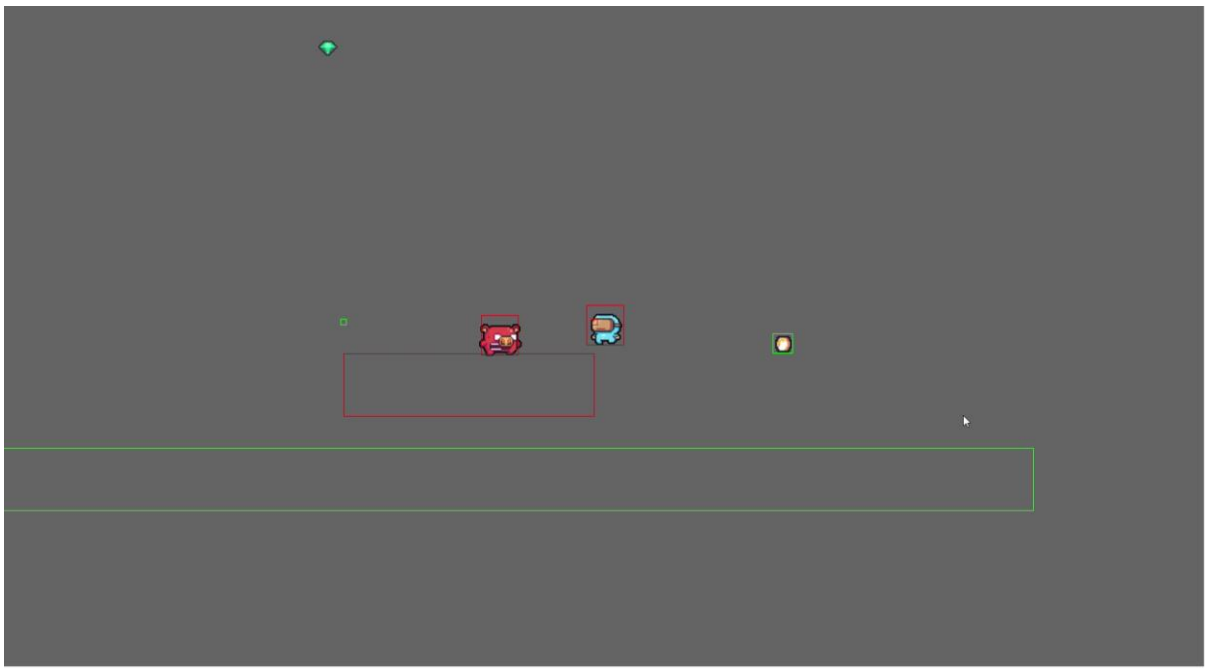




총 3종류의 몬스터를 구현하였다.

플레이어한테 반응을 하지 않는 패트롤 몬스터 / 플레이어를 감지하면 추격하는 몬스터 / 플레이어를 감지하면 원거리 공격을 하는 몬스터





몬스터는 플레이어와 같은 y축에 있을때 감지한다.

```
// Monster
DeathEventStrategy* deathEvent = new ItemDropEvent();
CMonster* monster1 = new CProjectileMonster(3,150.0f,300.0f,deathEvent);
monster1->SetScale(64.f, 64.f);
monster1->SetName(L"Monster");
monster1->SetPos(750.f,100.f);
AddObject(monster1, LAYER_TYPE::MONSTER);
```

각 몬스터가 죽으면서 발생할 deathEvent는 전략 패턴을 사용하여 몬스터의 종류와 별개로 관리할 수 있다. 이렇게 제작함으로써 몬스터의 파괴부분을 보다

유연하게 확장할 수 있게 되었다. 아이템을 떨어트리는 몬스터 외에 다양한 파괴이벤트를 구현한다면 몬스터와 이를 적절히 조합하여 다양한 몬스터를 생성할 수 있는 모듈식 설계가 가능해진다.

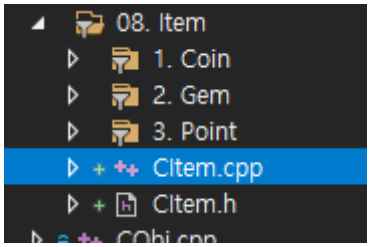
- 06/26

```
class CItem :
    public CObj

protected:
    CTexture* texture;
    CCollider* collider;

public:
    void SetTexture(CTexture* texture) { this->texture = texture; }

    virtual void Tick( ) override;
    virtual void Render( ) override;
    virtual void BeginOverlap(CCollider* ownCollider, CObj* otherObj, CCollider* otherCollider);

    virtual void GetItem( ) = 0;

    virtual CItem* Clone( ) = 0;
    CItem( );
    ~CItem( );
```

아이템 클래스를 제작하였다.

아이템이 기본적으로 가질 텍스처, 콜라이더와 각 아이템 종류별로 해당 아이템을 획득하였을때 발생할 함수를 GetItem이라는 순수가상함수로 선언하였다.

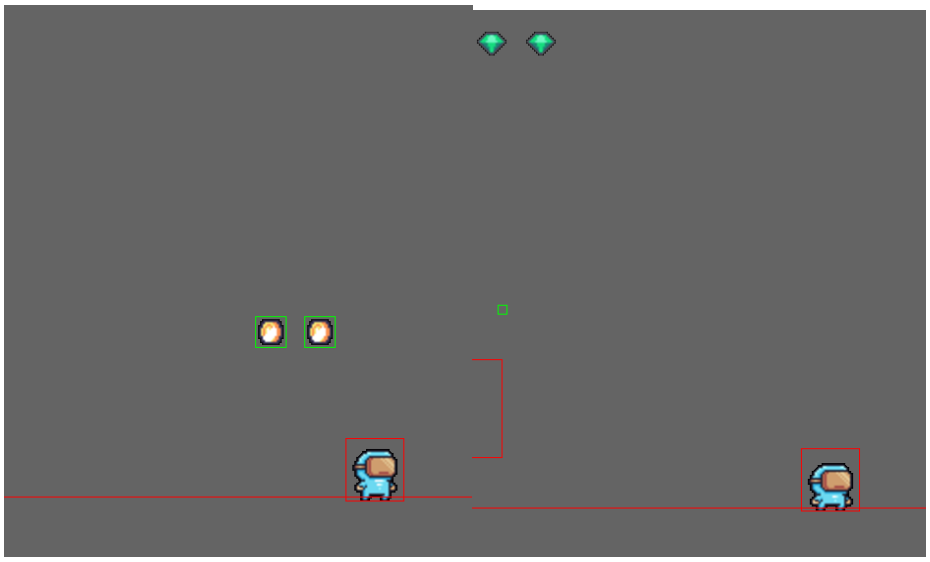
이 클래스를 상속받아 만들어질 다양한 아이템은 GetItem을 구현하여 아이템을 획득할 경우 발생할 이벤트를 만들 수 있다.



현재까지는 코인,젬,체크포인트 3개의 아이템만 존재하며 이 아이템은 게임매니저를 통해 관리될 예정이다.

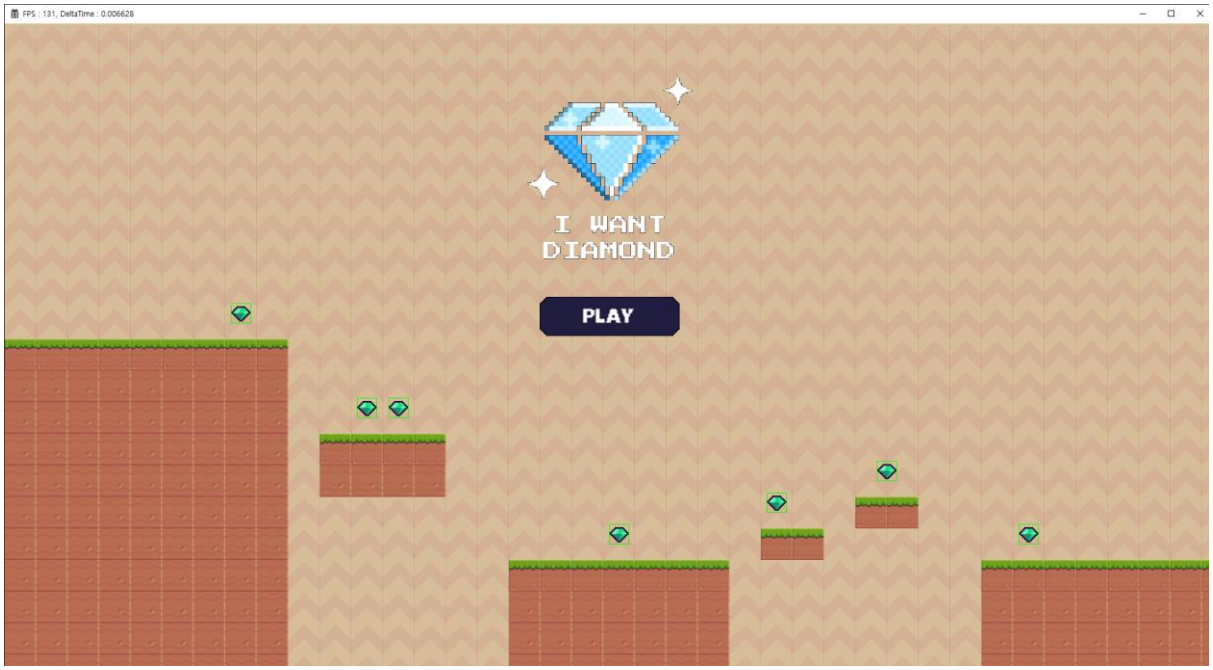
```
class CGameManager
{
    SINGLE(CGameManager);

private:
    int coins;        // 코인 개수
    int gems;         // 젬 개수
    const int gemOffset = 50.f; // 화면 상단에 나올 젬의 간격
    const int coinsPerGem = 5;  // 젬으로 교환하기 위한 코인개수

    vector<CImage*> gemArray; // 젬UI를 관리하기 위한 벡터

public:
    void Init();
    void Tick();

    void GetCoin(); // 코인 획득
    void GetGem(); // 코인 획득
    bool UseGem();  // 젬 사용
};
```

게임매니저 클래스를 통해 전반적인 게임의 진행,흐름을 제어할 수 있도록 하였다. 현재로서는 단순히 플레이어가 획득한 아이템을 관리하는 역할만 해주지만 추후에 게임이 확장되면서 이 게임매니저의 기능이 추가될 수 있다.



코인 아이템을 획득하여서 게임매니저의 GetCoin 함수가 호출되면 조건을 통해 게임매니저에서 관리하는 gemArray에 젬이 추가된다. 여기에 추가된 젬은 일정 오프셋(간격)만큼 떨어져서 상단에 표시되며 해당 젬을 사용하면 배열에서 삭제되면서 젬 UI도 같이 파괴된다.

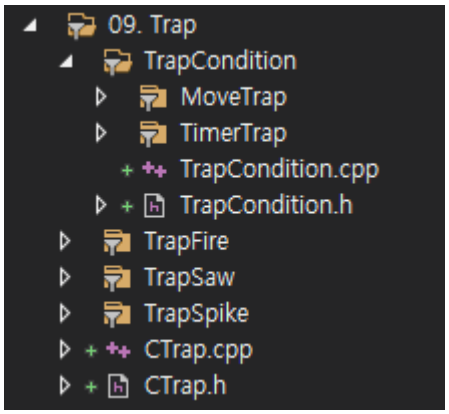
- 06/27



메인화면을 제작하였다. 게임로고는 AI로 생성하였으며 간단하게 밑에 플레이 버튼UI를 두었다.

맵 제작은 타일맵을 깔아두고 해당 타일맵을 저장한 다음, 각 레벨마다 불러야할 타일맵을 로드하는 방식으로 구현하였다.

- 06/28

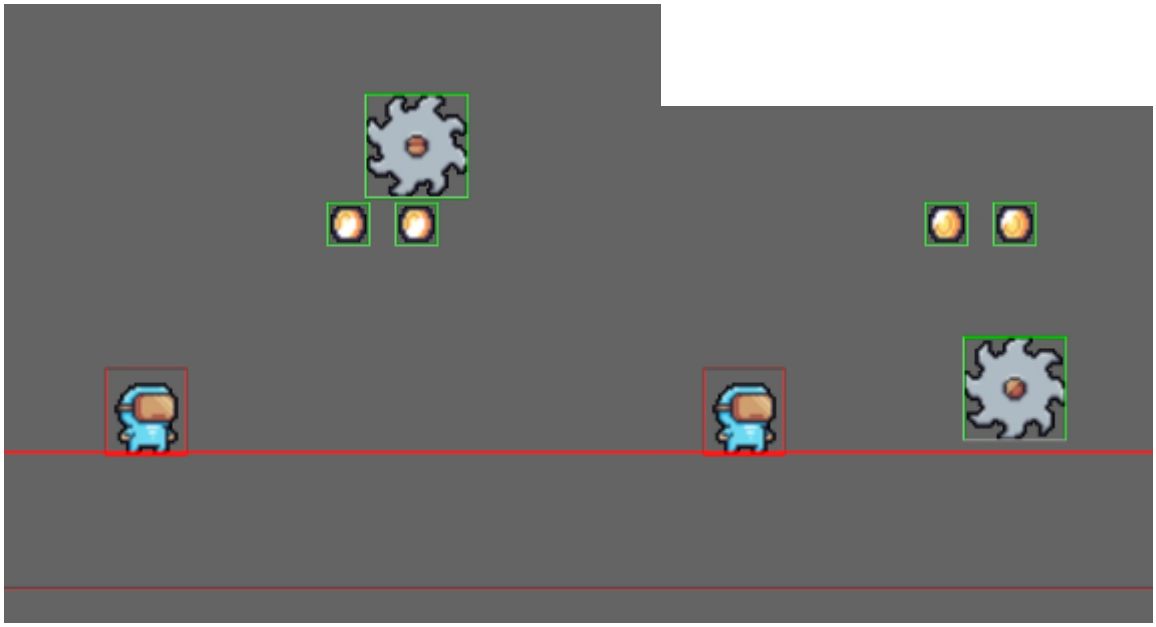


맵에 배치할 함정(Trap) 클래스를 설계하였다. 총 3종류가 있으며 각 트랩은 트랩의 활성화를 유연하게 가질 수 있도록 TrapCondition 이라는 것을 가지게 된다. 이를 통해 함정의 종류와 상관없이 움직이는 트랩, 나타났다 사라졌다 하는

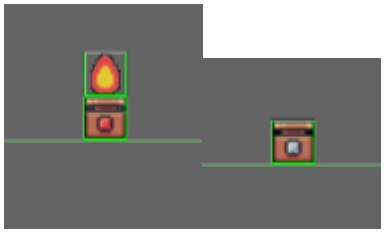
트랩 등 다양한 트랩을 조합할 수 있다. TrapCondition 내에서 구현된 부분을 따로 Trap에 붙이는 모듈식 설계이기 때문에 직접 각 트랩의 코드를 수정하지 않고도 확장할 수 있도록 하였다.

```
// trap
MoveTrapCondition* timerTrapCondition = new MoveTrapCondition(300.f,400.f);
CTrapSaw* saw = new CTrapSaw(timerTrapCondition);
saw->SetName(L"saw");
saw->SetPos(1250.f, 650.f);
saw->SetScale(32.f, 32.f);
AddObject(saw, LAYER_TYPE::TRAP);
```

예를 들어 TrapSaw 클래스의 트랩에 MoveTrapCondition을 붙여보았다.



위와 같이 Saw가 회전하는 애니메이션이 재생되면서 위아래로 왕복하는 함정이 완성되었다. 만약, 여기에 다른 트랩컨디션을 붙인다면 코드의 수정 하나없이 완전히 다른 트랩이 될 것이다.



일정 시간을 간격으로 불꽃이 생기면서 콜라이더가 활성화 되는 함정도 제작하였다. 위와 마찬가지로 만약에 이 함정에 Move컨디션을 붙이면 움직이는 함정이 될 것이다.