プロジェクトの技術文書

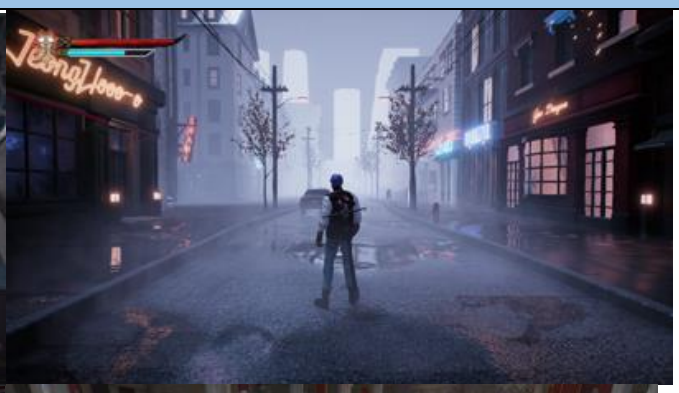
プロジェクト名	Evilness			
プロジェクト期間	プロジェクト人数			
2017-03月 ~ 11月 (約 8ヶ月間)	プログラマー	企画	グラフィック	合計
	2	2	11	15
開発道具	Unity 3D, C#			

詳細内容

*** プロジェクト紹介**

- EvilnessはUnity3Dエンジンで製作され、卒業作品のゲームにクールな戦闘と派手なエフェクトがある3Dアクションゲームで、敵を倒し、最後のボスを倒したらゲームをクリアすることができます。2017年に卒業作品として行われたプロジェクトです。3月初め、学期が始まる起点で総15人の開発者がプロジェクトに参加をしました。他のチームと比較したときのグラフィックスと、プログラマーが最も多いチームであり、最高のクオリティを示したと考えています。

スクリーンショット



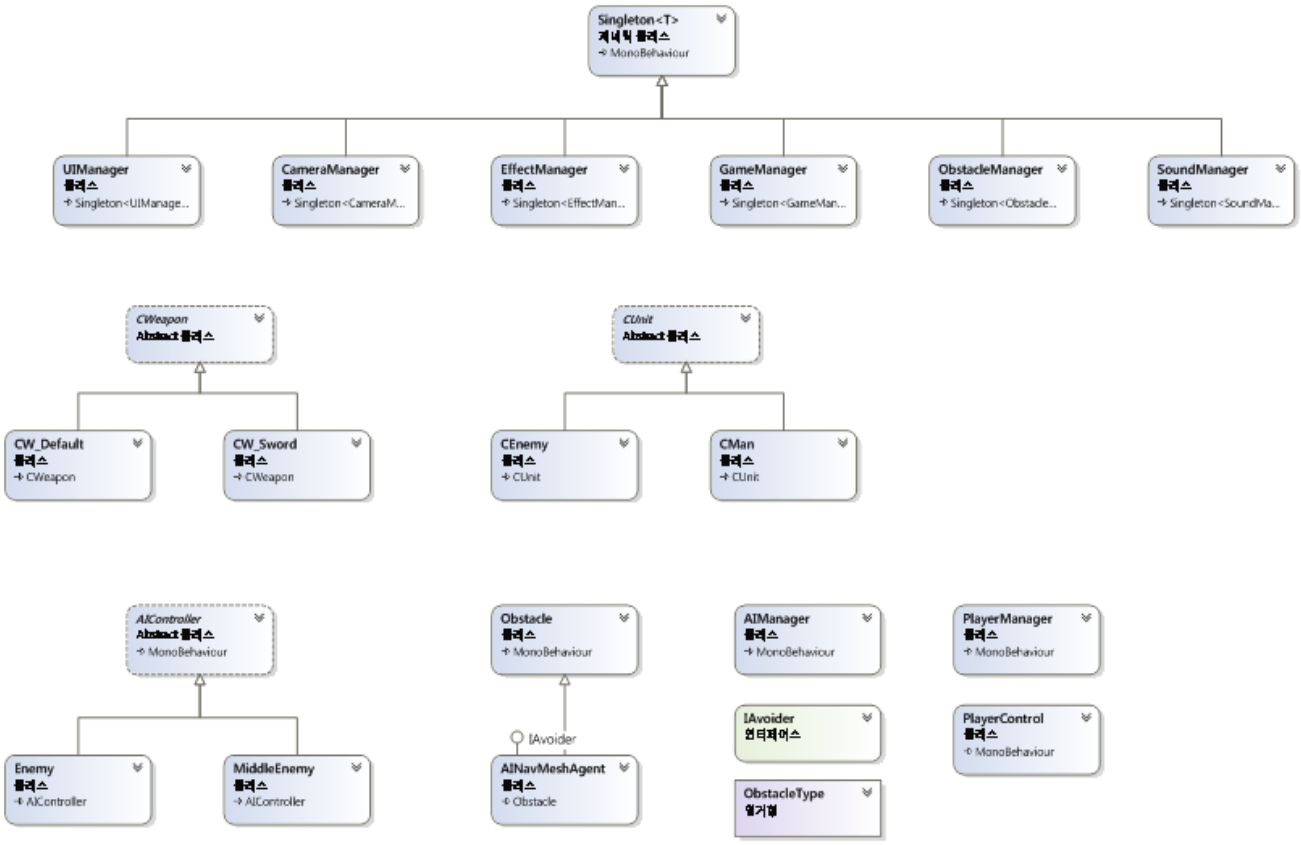
*** プロジェクトで引き受けた役割**

- Evilnessプロジェクトのメインプログラマを務めており、システム企画者と相談して、ゲームのロジックを設計しました。AIを主にしており、AIのFSMを作り、ボスモンスターのパターンを作り、Unity3DのNav Mesh Agentを使用せずにNavMeshのBakeされたMapの情報を持って、新しいAI Nav Mesh Agentを作って使用しました。また、Game Managerを使用してゲームにルールを製作しており、キャラクター、スクリプトのベースとキャラクターのスキルを作成しました。

*** プロジェクトの進行時にあったトラブルシューティングと進行過程**

- AIをどのようにすれば、お互いが固まらず、お互いが避けながらプレイヤーを攻撃するように具現化したかったです。最初はUnityにNav Mesh Agentを使用したが、目的地に到達した場合、お互いが押し出す現象があり、Obstacleスクリプトを作って、このスクリプトを持っているオブジェクトとは一定距離になると、避けながら行くように作りました。キャラクターが歩道ブロックに上ることも問題があったが、レイキャストを使って歩道ブロックが検出さになると、軽くキャラクターを浮かべて上るようになりました。また、アクションゲームであるため、何よりも自然なキャラクタアニメーションが重要でした。よりリアルに製作しようとしました。

*** YouTubeのリンク ： https://youtu.be/ITK0-riGupo**



<クラスダイアグラム>

マネージャーをよく使用するので、シングルトンクラスを作ってマネージャーに継承をさせて使用しています。戦略的な設計パターンでUnitの種類を管理して、武器の種類も管理しています。AIControllではモンスターの種類に応じて行動パターンだけ異なって実装することができ、継承受けて製作し、すべての動きはPlayerManager、AIManagerで管理をしています。

```
public enum ObstacleType { Point, Line, Sphere };

public class Obstacle : MonoBehaviour {
    // 움직이는 물체가 밀지 않고 피해가는 계수
    public float repulsion = 1.0f;
    // 장애물 타입
    public ObstacleType obstacleType = ObstacleType.Point;

    // 긴 물체일 경우 앞쪽과 뒷쪽을 포인트로 나눠준다
    [HideInInspector]
    public Transform pointA;

    [HideInInspector]
    public Transform pointB;

    // 장애물의 범위
    [HideInInspector]
    public float radius = 1.0f;
}
```

```
public class ObstacleManager : Singleton<ObstacleManager> {

    // 모든 장애물 찾기
    public static List<Obstacle> Obstacles = new List<Obstacle>();
    public int ObstaclesCount;
    // Use this for initialization
    void Start () {
        Obstacles = new List<Obstacle>();
        // 장애물 스크립트를 가지고 있는 애들을 찾아서 담아준다.
        Obstacle[] ObstacleArray = FindObjectsOfType<Obstacle>();
        for(int i = 0;i<ObstacleArray.Length; i++)
        {
            Obstacle newObst = ObstacleArray[i];
            // 스크립트가 켜져있고 게임오브젝트가 켜져있을 경우만 리스트에 담아준다.
            if (newObst.isActiveAndEnabled && newObst.gameObject.activeInHierarchy)
                Obstacles.Add(newObst);
        }
    }
```

<Obstacle 障害物のチェック>

障害物クラスを作ってタイプを分けて、そのタイプに合わせて、どれだけの範囲を避けていくのかを計算しています。ObstacleManagerでは、すべて障害物のスクリプトを持っているオブジェクトを保存し、常に比較をしてくれてAIが障害物のスクリプトを持っているすべてのオブジェクトを自分で避けて行けるように作ってくれました。

```
//////////////////////////////////////////////////////////////////
RaycastHit hit;
Transform CameraTr = Camera.main.transform;
Debug.DrawRay(target.position, CameraTr.position - target.position, Color.red);
if (Physics.Raycast(target.position, CameraTr.position - target.position, out hit, distance))
{
    if (hit.transform.CompareTag("Maps") || hit.transform.CompareTag("Ground"))
    {
        //                Pivot.position = Vector3.Lerp(Pivot.position, hit.point, 1f);
        CameraTr.position = new Vector3(Mathf.Lerp(CameraTr.position.x, hit.point.x, Time.deltaTime * LerpSpeed),
            Mathf.Lerp(CameraTr.position.y, hit.point.y, Time.deltaTime * LerpSpeed),
            Mathf.Lerp(CameraTr.position.z, hit.point.z, Time.deltaTime * LerpSpeed));
    }
}
else
{
    CameraTr.localPosition = new Vector3(Mathf.Lerp(CameraTr.localPosition.x, 0, Time.deltaTime * LerpSpeed),
            Mathf.Lerp(CameraTr.localPosition.y, 0, Time.deltaTime * LerpSpeed),
            Mathf.Lerp(CameraTr.localPosition.z, 0, Time.deltaTime * LerpSpeed));
    //            Pivot.localPosition = Vector3.zero;
}

//////////////////////////////////////////////////////////////////
```

<カメラの後ろに障害物あるとき拡大縮小>

プレイヤーに追いかけてカメラが動くようになると、プレイヤーが障害物の近くに行った場合、障害物の中に入ってキャラクターを覆う現象がありました。他のゲームをしてみて、そのゲームは、どのような形式で、カメラを処理したかをたくさん見てみました。プレイヤーでRay Castを使用して、カメラまでRayを使用してチェックをして、RayにHitがされた情報からTagに「Maps（すべての障害物）」または「Ground（底）」の場合に、カメラの位置をRayが届いたHit PointまでLerpに徐々に動くました。また、Rayがタグ「Maps」または「Ground」に届かない場合は、再度元のポジションに戻るように製作しました。

```
public abstract class CUnit
{

    public float m_HP;          // 매니저에서 받아올 HP.
    public float m_HPMax;       // 매니저에서 받아올 최대HP.
    public float m_DamageMin;               // 플레이어의 기본 데미지 (무기 X).
    public float m_DamageMax;               // 기본 데미지 맥스.
    public Image m_HPBar;                   // 받아온 HPUI게임오브젝트에서 HPUIBar를 찾아서 넣기.

    protected CWeapon m_Weapon;             // 사용중인 무기.
    protected float m_UnitDamageMin;            // 혹시 변동될 수도 있으니깐.
    protected float m_UnitDamageMax;
    protected float m_TempUnitDamageMax;        // 기본데미지 저장.
    protected float m_TempUnitDamageMin;        // 기본데미지 저장.
    protected float m_PropertyDamage;

    public abstract void HPdisplay();       // 플레이어의 HP상태를 나타낸다.
    public abstract void Init();            // 최초 초기화.
    public void FromWeapon() { m_Weapon.WeaponDisplay(); }      // 현재 무기가 어떤것인지 알려준다.
    public float OnAttack()...
    public float DurationOnAttack()...
    public void FromOnDamage(float damage)      // 받은 데미지 정산...
    public void setWeapon(CWeapon weapon)...        // 플레이어의 무기를 설정한다.
    public float Health       // HP...
    public float HealthMax        // MaxHP...
    public float PropertyDamage...
}
```

```
[Serializable]
public class CMan : CUnit
{
    public float m_Skill_Use_Gauge;     // 스킬을 사용하는 게이지의 수
    public int m_Combo;     // 매니저에서 받아올 콤보 증가 수.
    public GameObject m_Combo_UI;     // 콤보 UI를 가져온다.
    public float m_GaugeMax;    // 스킬 게이지 최대치
    public float m_Gauge;       // 현재 게이지의 수치
    public float AnimationSpeed;
    private float m_ComboUITime = 3;
    private float m_ComboUITimeMax = 3;
    public override void Init()...
    public override void HPdisplay()...

    // ************** 플레이어용 **********************.
    public void Combodisplay()    // 콤보 화인...
    public void Gaugedisplay()          // 게이지 보여주게 하기...
    public bool UseGaugeSkill()     // 게이지 만큼 스킬 사용하기...
    public void CancelCombo()               // 콤보 캔슬...
    public void TimeCancelCombo()           // 콤보가 캔슬 되는 시간...
    // ************** 플레이어용 **********************.
}
```

<CUnitで 共通ステータスを生成>

プレイヤーとモンスターは共通で持つステータスを継承受けて宣言を行うことができるように設計さており、プレイヤーとモンスターだけに持っている属性を別々に作ってくれて、コードを簡単に整理がしました。複数のモンスターが出てくることもあり、プレイヤーに様々な剣が出てくる可能性を持って戦略デザインパターンを使用して製作しました。

プロジェクト名	Count Slime			
プロジェクト期間	プロジェクト人数			
2016-08月 ~ 09月 約 1ヶ月	プログラマー	企画	グラフィック	合計
	1	2	4	7
開発道具	Unity 3D, C#			

詳細内容

*** プロジェクト紹介**

- Count Slimeプロジェクトは可愛らしくて可愛いスライムをオークから守りながらステージをクリアするディフェンスゲームです。ウェーブごとに様々なモンスターたちが出て、スライムは農場の中で活発に遊んで体力が取り付ければ空腹を感じて周囲の木の樹液を飲みながら体力を回復して再農場に戻って活発に遊べるかわいいゲームです。

スクリーンショット

*** プロジェクトで引き受けた役割**

- Count Slimeプロジェクトで、私は引き受けた分野は、プログラミングの分野を担当進行しました。全体的な構造を作ることから始め戦略デザインパターンを利用してキャラクター、モンスター、スライムを管理しました。キャラクターは武器交換があるので、簡単に変えるためにWeaponというクラスを作って管理しました。モンスターはUnity 3D Nav Mesh Agentを使用しました。

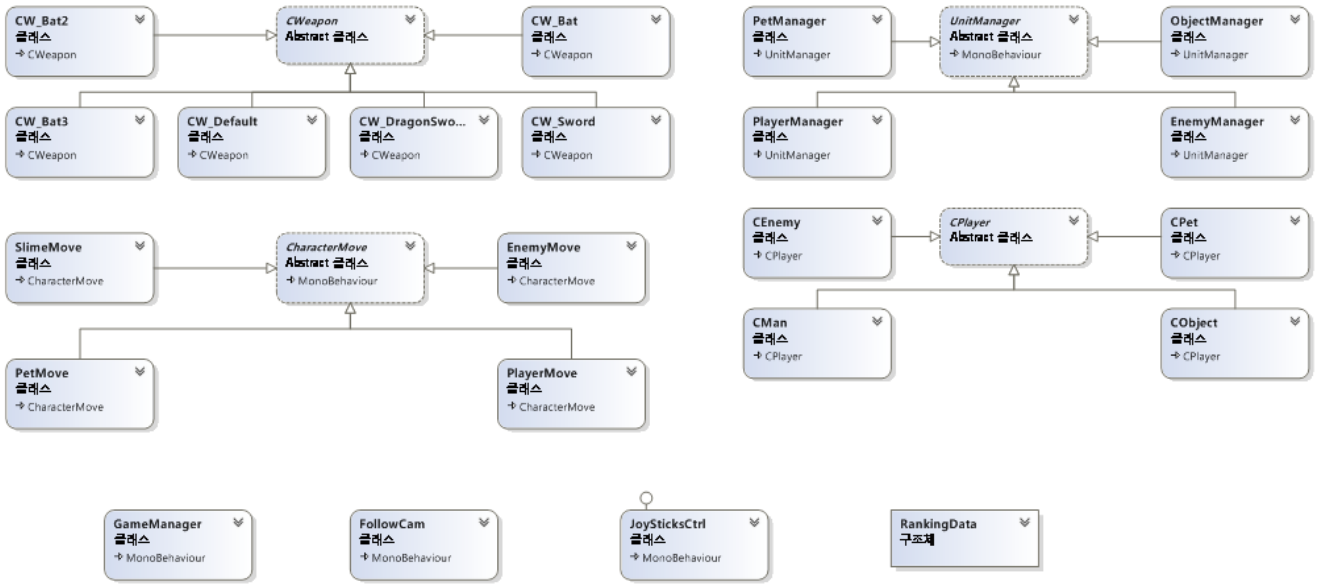
*** プロジェクトの進行時にあったトラブルシューティングと進行過程**

- プロジェクトを進行する際に、クラスの設計をどのようにするかを悩んでましたが、授業時間に学んだ戦略デザインパターンを使ってみよう！という気がして、全体的な枠組みは、戦略的な設計パターンを利用し、このプロジェクトを進行すると、最も大変だった部分は、スライムが農場内で自由に動くが、HPが50以下に落ちたときの木の樹液を食べ回復する部分で多くのエラーがあったが、Coroutineを誤って使用した部分を直して解決しました。その後Coroutineはどのような機能をしていることは少し見てみる機会ができました。

*** プロジェクトを仕上げながら感じた点**

- 短い開発期間に合わせて、他の人よりも創造性があることを見つけることが思ったよりも簡単ではないことを感じるようになりました。チームメンバーが一定に遅くなるやたら促すよりも、励ましてチームメイトを信じて待っていることが最も重要なものともう一度考えました。

*** YouTubeのリンク ：** https://youtu.be/j7gS0oNLZec



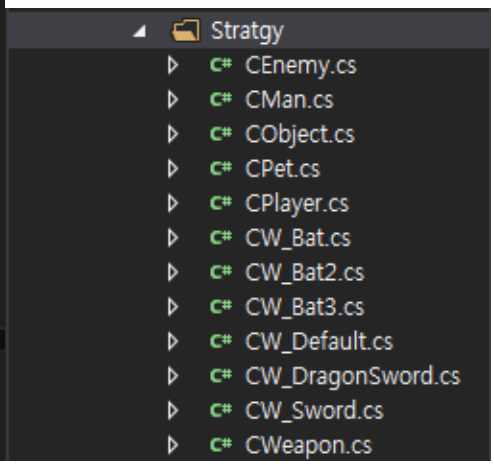
<クラスダイアグラム>

私は共通に入ることを考えて、継承を使用し、仮想クラスで作成し、財政で使用しました。そして、その中で独自の能力は、継承されたクラス内での機能を具現化し、これを具現化すると、管理することも容易であると判断しました。

```
public abstract class CPlayer {
    protected int HP;       // HP
    protected int HPMax;    // 최대 HP
    protected Image HPBar;  // HPBar
    //    protected int Damage;
    public Transform HeadHPBar; // HPBar부모객체
    private CWeapon m_Weapon;   // 무기
    private int DefaultDamage;  // 기본데미지
    public abstract void HPdisplay();   // HP가 변화를 보여준다.
    public abstract void display();     // 어떤 종족인지 알려준다.

    // 어떤 무기인지 알려준다.
    public void FromWeapon(){ m_Weapon.WeaponDisplay(); }
    // 무기의 사운드를 들려준다.
    public void FromAttack(){ m_Weapon.AttackDisplay(); }
    // 무기 내구도를 알려준다.
    public void FromStamina(string name) { m_Weapon.StaminaDisplay(name); }
    // 받은 공격력 만큼 HP를 변화시켜준다
    public int FromAttackDamage(string name, int damage) { Health -= damage;
        if (Health <= 0) Health = 0;

        HPdisplay();
        return damage;
    }
    public void setWeapon(CWeapon weapon) { m_Weapon = weapon; } // 무기 변경
```



<戦略デザインパターンを利用したすべてのオブジェクトの基本データベース管理>

各基本的に共通に入るHP、Damageのような場合ではCPlayerという共通のクラスを生成してくれました。

HPが変化すると武器ごとの攻撃音が異なって出なければならないのでCWeaponというクラスを作成した後、武器クラスを複数のを作って管理しました。

```
protected int TriggerSet<T>(Collider col, T Move, CPlayer Opponent) where T: CharacterMove
{
    int damage;
    int RandDamage;

    Move.SetHit();
    Opponent.FromAttack();
    if (Opponent.GetStamina() != -1) {
        RandDamage = Random.Range(((Opponent.Damage + Opponent.GetAttackDamage()) - Opponent.GetStaminaMax()),
            (Opponent.Damage + Opponent.GetAttackDamage()) + 1);
    } else {
        RandDamage = Opponent.Damage + Opponent.GetAttackDamage();
    }
    if (Move is EnemyMove) {
        PlayerMove Player = col.GetComponentInParent<PlayerMove>();
        if (Player.GetPlayerAni().GetBool("IsSkill")) {
            RandDamage += (int)(RandDamage * 10 / 100);
        }
    }
    damage = m_Player.FromAttackDamage(gameObject.tag, RandDamage);
    if (m_Player.Health <= 0)
    {
        Move.SetIsDie();
    }
    return damage;
}
```

<共通Trigger処理>

クラス構造がすべて継承に処理をしたので衝突処理も共通にしました。テンプレートを使用してどんなクラスが入って来るのか把握して作られてるので各自の状況に合わせてダメージを与えました。

```
protected void ItemSet(Collider col)
{
    switch (col.gameObject.tag)
    {   // 무기 습득시 태그를 찾아 변경
        case "Sword":
            m_Weapon = new CW_Sword();
            WeaponChange(col);
            break;
        case "DragonSword":
            m_Weapon = new CW_DragonSword();
            WeaponChange(col);
            break;
        case "Bat":
            m_Weapon = new CW_Bat();
            WeaponChange(col);
            break;
        case "Bat2":
            m_Weapon = new CW_Bat2();
            WeaponChange(col);
            break;
        case "Bat3":
            m_Weapon = new CW_Bat3();
            WeaponChange(col);
            break;
    }
```

```
public void WeaponChange(Collider col)
{
    m_Player.setWeapon(m_Weapon);
    m_Player.FromWeapon();  // 플레이어가 무슨 무기를 들었는지 알려준다.
    CopyComponent(col.gameObject, Weapon);
//    Debug.Log(col.gameObject.tag + " 로 무기가 변경 됐다.");
    m_Player.FromStamina(gameObject.tag);   // 무기 내구도 확인
    GameObject.Destroy(col.gameObject);
}
protected void CopyComponent(GameObject original, GameObject destination)
{
    Mesh mesh;
    Material material;

    mesh = original.GetComponent<MeshFilter>().mesh;
    material = original.GetComponent<MeshRenderer>().material;

    destination.GetComponent<MeshFilter>().mesh = mesh;
    destination.GetComponent<MeshCollider>().sharedMesh = mesh;
    destination.GetComponent<MeshRenderer>().material = material;

    destination.SetActive(false);
    destination.SetActive(true);
    destination.GetComponent<MeshCollider>().enabled = false;

}
```

<武器変更スクリプト>

プレイヤーが武器を習得する場合、ItemSetクラスでは、落ちた武器との競合になった場合、その武器の情報に変更をさせてくれるクラスです。WeaponChangeクラスは落ちた武器に変更させてくれるために、実際にイメージが変更された部分は、CopyComponentでプレイヤーが持っている武器と落ちた武器のMeshとMaterialを変更してくれて、イメージ的に変更をさせてくれました。

プロジェクト名	Dragon Flights	
プロジェクト期間	プロジェクト人数	
2015-11月 ~ 2015-12月 約 1ヶ月	プログラマー	合計
	1	1
開発道具	Visual Studio 2015, Win API, C++	

詳細内容

*** プロジェクト紹介**

- Dragon FlightsはWin APIで製作され、カカオゲームのドラゴンフライトに沿って制作しました。1年生2学期のWindowsプログラミングの授業時間に課題で製作したプロジェクトです。ドラゴンフライトを同様に製作しようとしたし、各種アイテムとゴールドを食べ、さらに強くなって進む種スクロールアクションゲームです。

スクリーンショット

*** プロジェクトで引き受けた役割**

- Dragon Flightsは1人の開発として、すべてのシステムを製作しました。ドラゴンフライトをたくさんプレイして企画をしたし、シングルトンゲームマネージャーを作成した状態に応じて、管理をしてくれました。状態に応じてレンダリング方式を変えて与え、状態は全6種類でIntro、Start、Shop、Help、Destroy、Game Overに分けました。

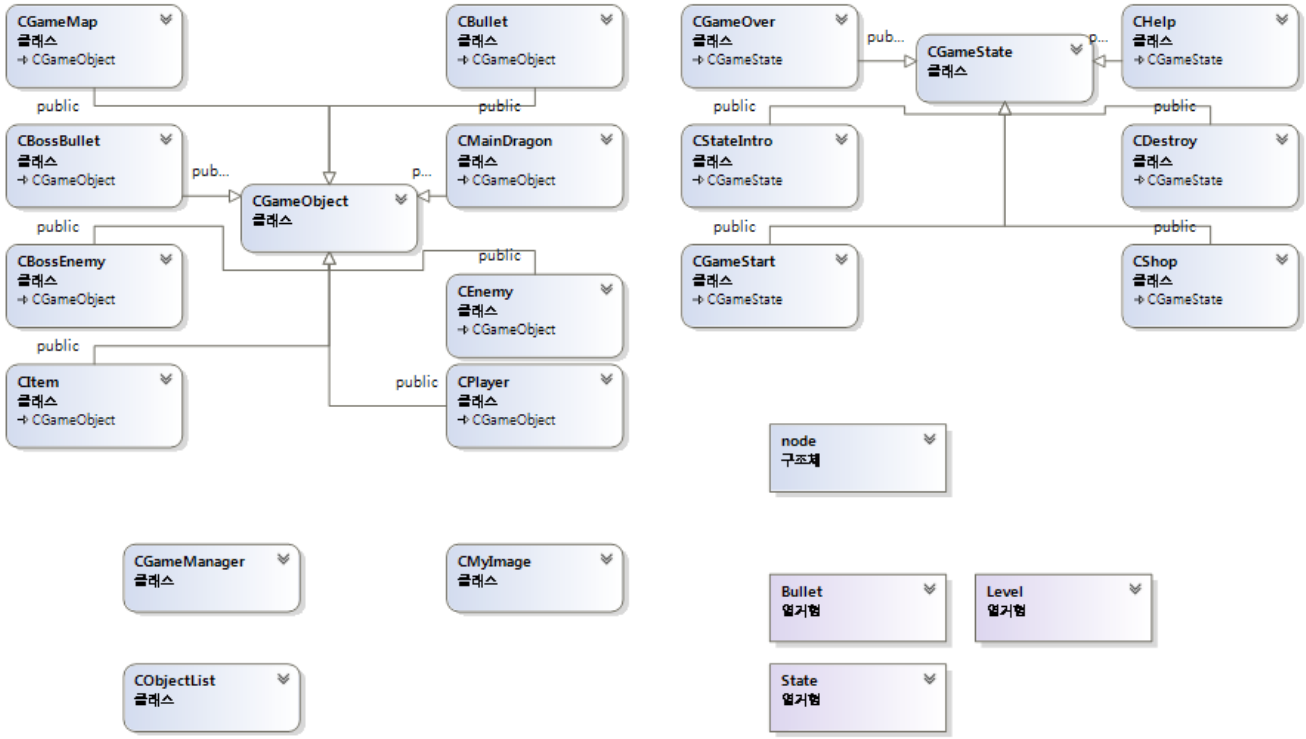
*** プロジェクトの進行時にあったトラブルシューティングと進行過程**

- 最初に、作業する前に、優先的にクラス設計をどのようにすれば良くできるかをずっと悩んでました。状態に応じて変更することができるように設計さをしたし、プログラミングについてはまだ未熟な部分が多かった時だったので、メッセージの値をどのような形式で受けて、処理すべきかを知らなく、上手な友達とGoogle検索を使用して多くの情報を獲得しました。難易度を追加し、一定距離ごとに敵の体力を増加させる方法。また、コインを集めて店から武器を強化することができ、デュアルショットアイテムを追加し、一定時間の間、武器が頭髪ずつ出るようにしました。衝突処理部分が最も大変だった、数値が正確ではない出来事であることを知って絵を描きながら正確に合わせました。また、オブジェクトが宣言にならない時があったが、その部分は、ヘッダーが重複呼び出しにより生じ重複最小化させて解決しました。

*** プロジェクトを仕上げながら感じた点**

- 確かに、コンソールウィンドウで製作するよりも、絵が見えるのがあって面白かったし、また、ヘッダ重複宣言を最小限に抑える必要があることを感じました。この期末課題で1位をして大変嬉しかったです。

*** YouTubeのリンク：** https://youtu.be/dXZrARfQd4Q



<クラスダイアグラム>

すべてのオブジェクトは、Game Objectから継承されて使用がされ、Game Stateですべての状態に継承を受けて再定義で作成しました。My Imageでは、すべてのリソースを受けてきて呼び出して使用することができるようにし、Object ListはLinked Listを作って武器を管理してくれました。

ObjectList.h | ObjectList.cpp | StateIntro.h | GameManager.cpp | ClassDiagram1.cd*

드래곤플라이트 — CObjectList

```h
#pragma once
#include "GameObject.h"
struct node
{
    CGameObject object;
    node *next;
};

class CObjectList
{
    node* m_head;
    int m_count;
public:
    CObjectList();
    ~CObjectList();

    void insertObject(int pos, CGameObject obj);
    void removeObject(int pos);
    CGameObject *getObject(int pos);
    int getCount() { return m_count; }
    void pushFront(CGameObject obj) { insertObject(0, obj); }
    // 0번째 리스트에만 집어넣어준다.

    bool isEmpty() { return m_head == nullptr; }
};
```

```cpp
void CObjectList::insertObject(int pos, CGameObject obj)
{
    node* curNode = m_head;
    if (pos < 0 || pos>m_count) return;

    node *newNode = new node;
    newNode->object = obj;

    if (pos == 0 || m_head == nullptr) {
        newNode->next = m_head;
        m_head = newNode;
    }
    else {
        for (int i = 0; i < pos - 1; curNode = curNode->next, i++)
            //          curNode = curNode->next;
            newNode->next = curNode->next;
        curNode->next = newNode;
    }
    m_count++;
}
```

<Linked List>

授業で学んだリストを使用して、プレイヤーの武器を動的に生成して、削除することができるよう作成しました。Insert Objectから生成してくれ、Remove Objectからの削除をしてくれています。この概念を学び、このプロジェクトに使用すると便利になるだろと思ったし、それを忘れてしまう前に復習するつもりで使用しました。