**게임개발자전문가과정**

**과 목 명 : DirectX 2D Game Project**

**능력단위 : DirectX 2D**

**제출일자 : 2022년 10월 07일**

**포트폴리오 : Top Gun!**

**작 성 자 : 신승빈**

**<제출내역>**

1. **DirectX 2D를 이용하여 구현한 2D Game**
2. **2D Game 구현 분석 및 세부 문서**

|  |
| --- |
|  |

**< Contents >**

1. **프로젝트 소개 및 개요**
2. 프로젝트 소개
3. 프로젝트 주요 기술
4. **프로젝트 설계 및 다이어그램**
5. 시퀀스 다이어그램
6. 클래스 다이어그램

2-1) 프로젝트에서 사용된 변수 역할

2-2) 프로젝트에서 사용된 함수 역할

1. **최종 결과 및 추가 내용**
2. 범용성/유연성/확장성/간결성 고려하여 추가된 내용
   1. 현재 프로젝트 설계(구현 내용)
   2. 향후 개발 내용
3. 최종 결과
4. **프로젝트 소개 및 개요**
5. **프로젝트 소개**

지금까지 배운 DirectX의 2D 출력 기술을 이용하여 2D 게임을 직접 구현해보았다.

1. **프로젝트 주요 기술**

* **Model-View-Controller**

DirectX는 출력을 담당하는 라이브러리로 엄밀하게 따지면 게임의 메인 루프와는 관련이 없는 라이브러리이다. 객체 지향적 관점에서 DirectX와 게임 메인 루프를 분리하여 객체의 응집성을 높이고, 충돌 등의 작업을 수행하기 위해 지난 과정에서 구현한 QuadTree를 재사용하여 작업 효율성을 높일 수 있도록 MVC 패턴을 적용하여 전체적인 프로젝트를 설계하였다.

* **Memory Pool**

Tile, Effect, Cloud같이 즉발적으로 생성되지만 총 수요를 짐작하기 어렵고, 매 수요때마다 생성-해제를 하기에 아까운 객체를 재사용하기 위해 Memory Pool 기법을 이용하여 메모리 관리를 구현하였다. Memory Pool생성 초기에 미리 예측한 크기의 객체를 생성해두지만, 만약 In Game중에 요구하는 객체의 수요가 Memory Pool이 보유한 객체의 수량보다 많다면 기존에 미리 생성한 수량에 더해 미리 예약해둔 예측 수량만큼 객체를 한번에 생성한다.

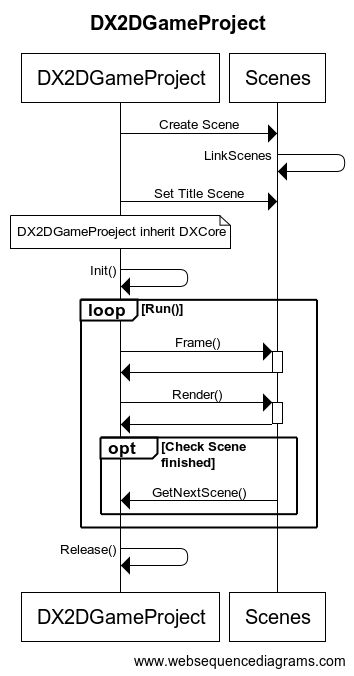
* **Terrain**

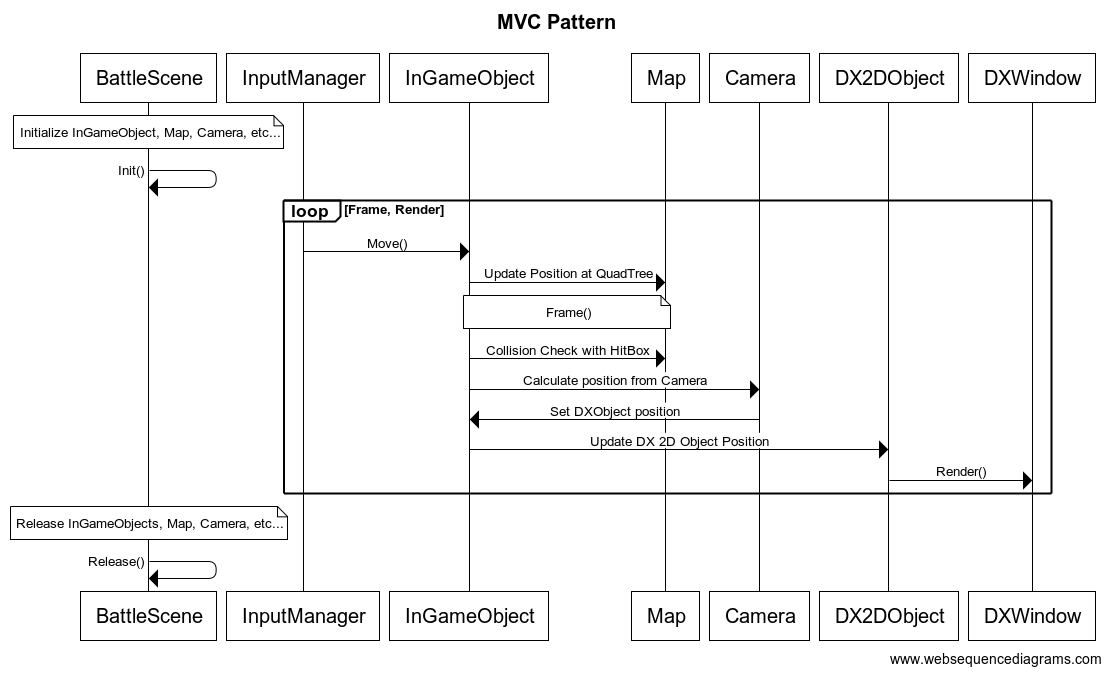
지속적으로 움직이는 전투기를 표현한 게임의 특성상 배경의 크기는 전투기의 이동 범위에 비례하여 커질 수 밖에 없다. 이러한 광대한 넓이의 대지를 미리 Draw하는 것은 연산 자원의 낭비가 심할 수 밖에 없고, 만약 Texture의 Scroll 방식으로 이동하는 연출을 수행한다 하더라도 반복적인 Texture의 출력은 User에게 쉽게 지루함을 느끼게 할 수 있어 긍정적이지 못한 경험을 줄 가능성이 높다. 이를 방지하기 위해 Terrain을 구현하여 Camera가 촬영하고 있는 범위의 Terrain에만 요구되는 수량만큼의 Tile을 배치하여 출력하였고, Memory Pool에서 출력을 위해 반환된 Tile들은 매번 임의의 Texture로 초기화 되어 전투기가 이동하더라도 불규칙한 지형을 출력하도록 구현하여 User가 긍정적인 경험을 할 수 있도록 구현하였다.

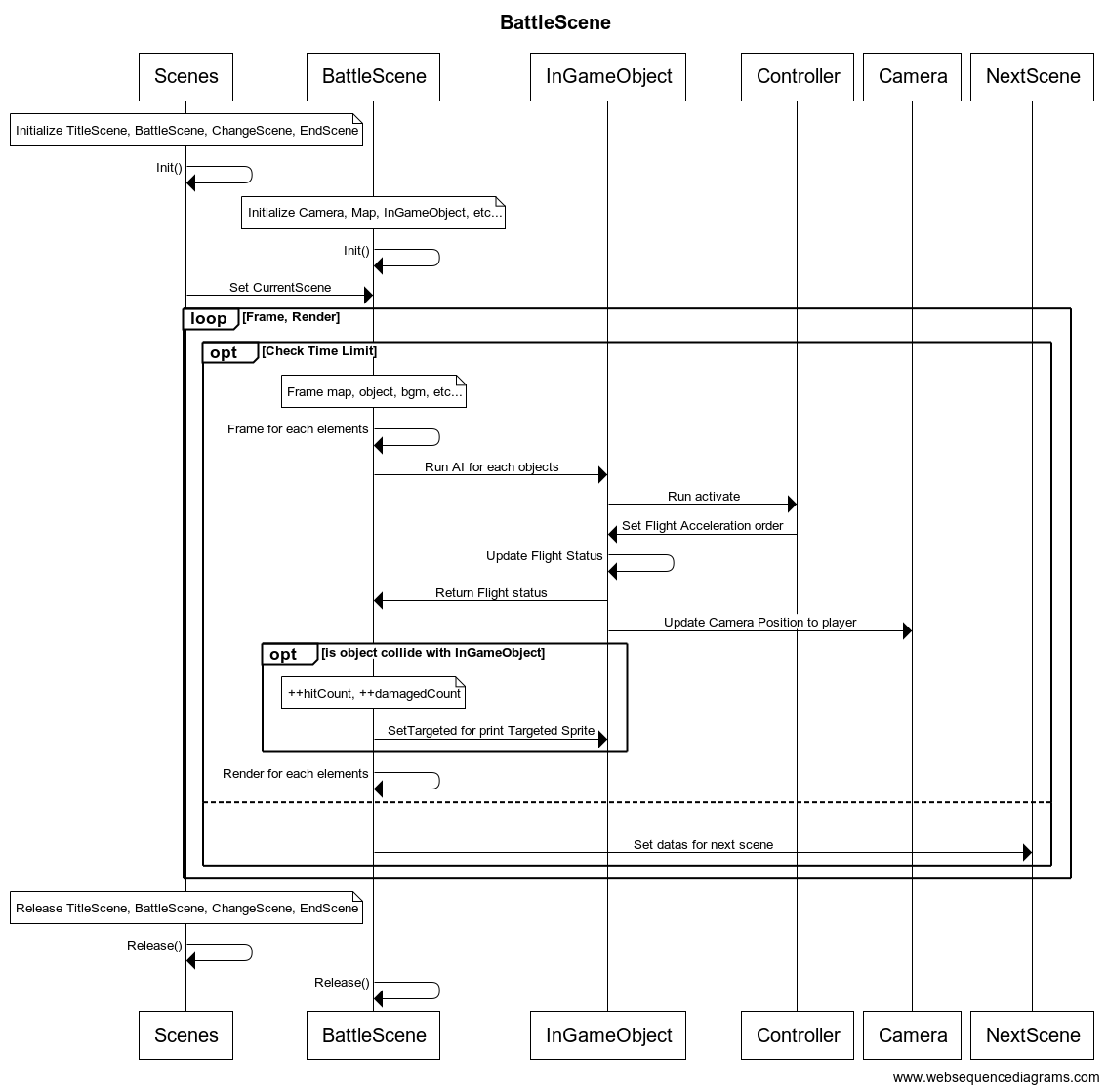
* **Text UI**

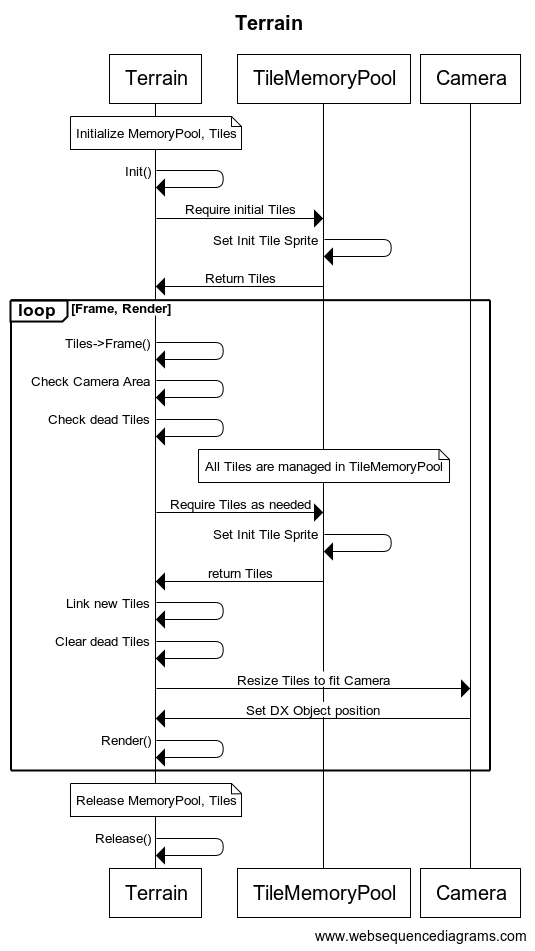
DirectX의 dwrite.h를 통한 Text출력은 많은 Device에서 font등 다양한 연산을 수행해야 하기 때문에 성능의 저하가 발생할 수 밖에 없다. 이를 막기 위해 font가 미리 저장된 Texture를 이용해 원하는 위치에 원하는 Text를 작성하여 쉽게 출력할 수 있도록 하여 Sprite나 Animation과 동일한 비용으로 쉽게 Text를 출력할 수 있도록 구현하였다.

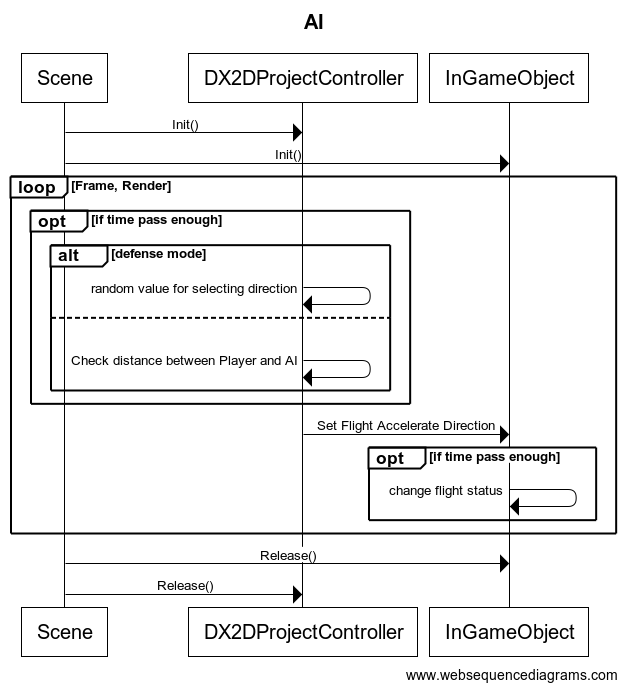
1. **프로젝트 설계 및 다이어그램**
2. **시퀀스 다이어그램(Sequence Diagram)**

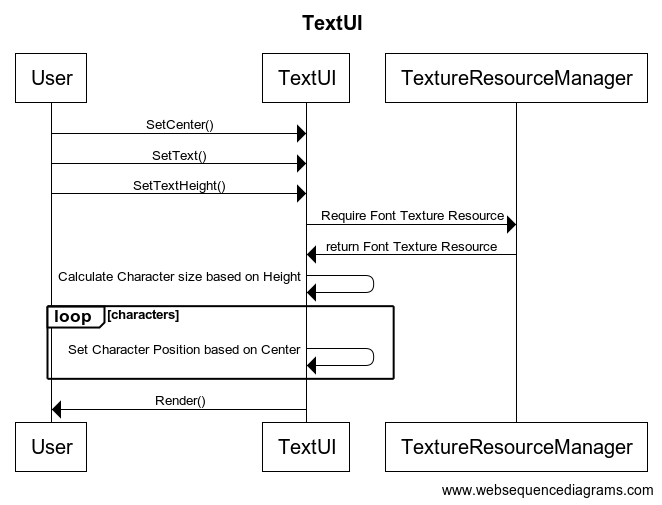
****

****

****

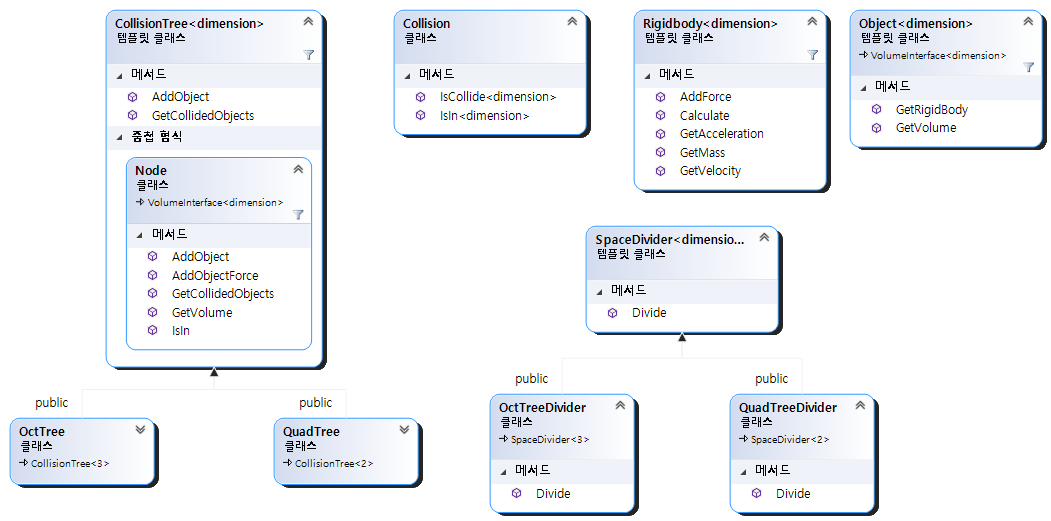
****

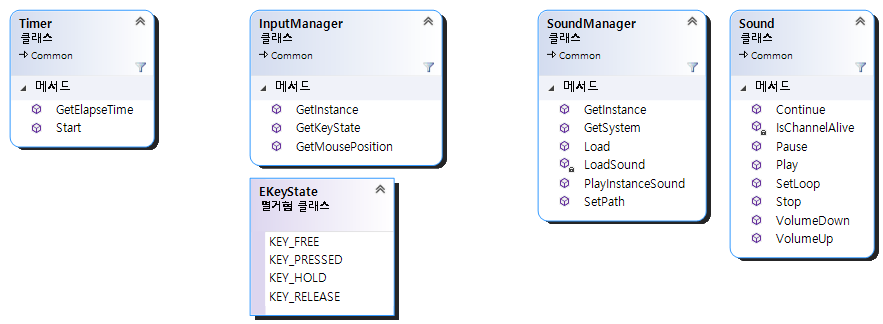
****

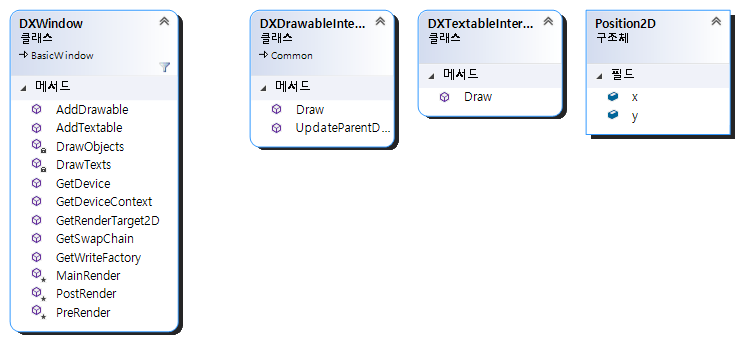
****

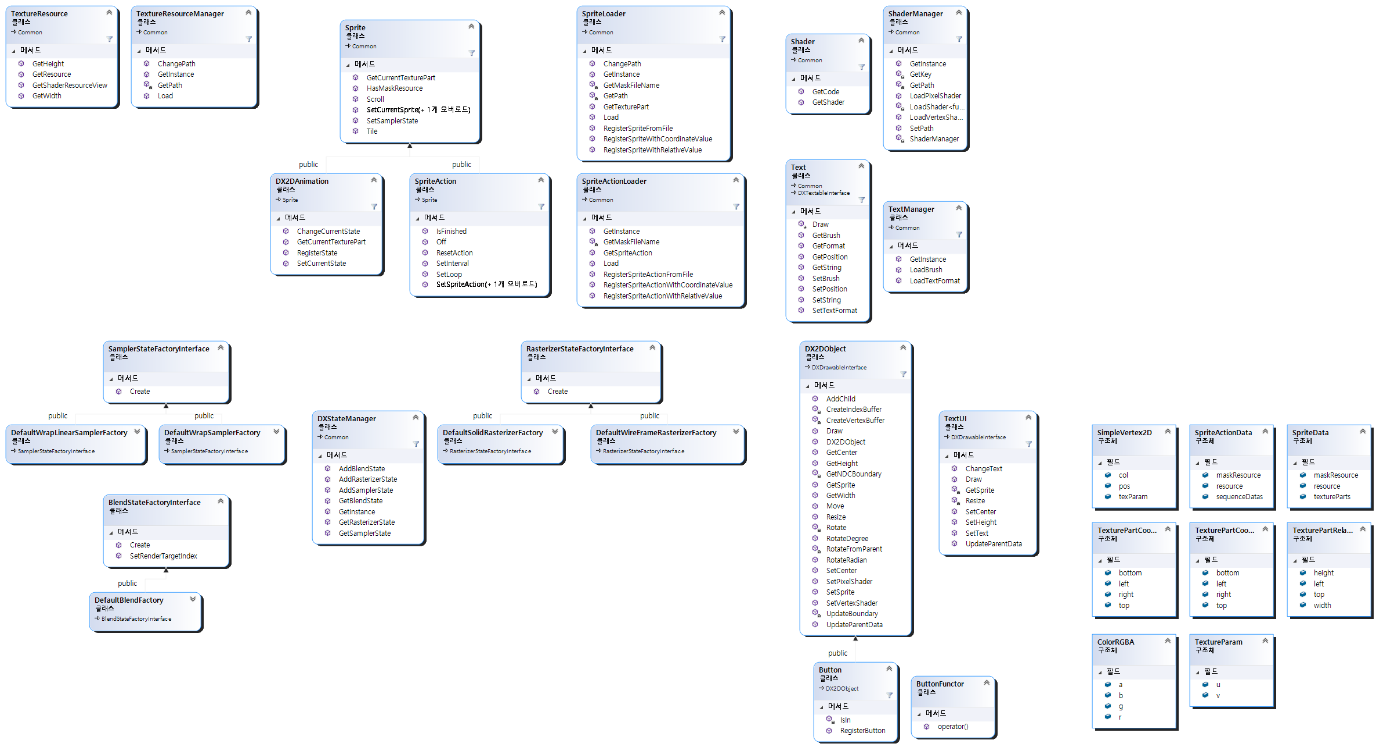
[그림 1-1] 시퀀스 다이어그램(Sequence Diagram)

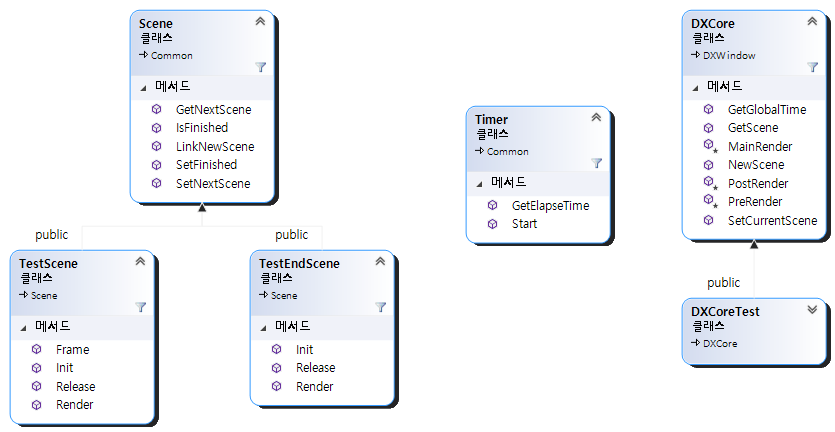
1. **클래스 다이어그램(Class Diagram)**

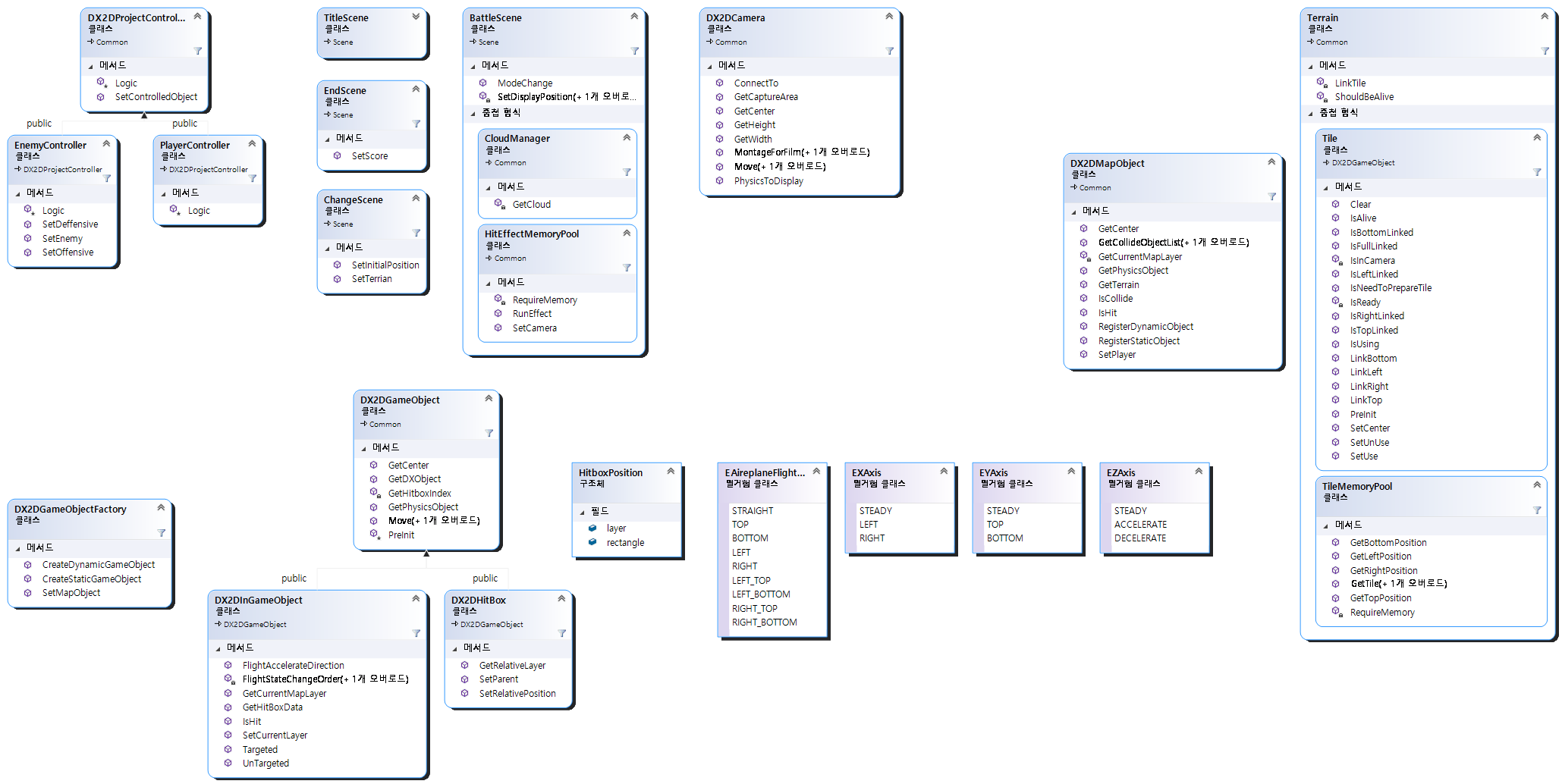
****

****

****

****

****

****

[그림 1-2] 클래스 다이어그램(Class Diagram)

**2-1) 프로젝트에서 사용된 변수 역할**

|  |  |  |
| --- | --- | --- |
| **클래스명** | **변수** | **내용** |
| Timer | DWORD \_startTime; | Process가 시작한 시간을 저장하여 클래스 시작 지점을 저장하는 변수 |
| DWORD \_currentTime; | 매 Frame마다 update하는 시간 |
| SoundManager | FMOD::System\* \_system = nullptr; | FMOD의 System을 저장하는 변수 |
| std::map<std::wstring, FMOD::Sound\*> \_soundList; | Fmod를 통해 읽어온 Sound를 저장하는 변수 |
| Sound | FMOD::Sound\* \_sound; | FMOD로부터 읽어온 Sound를 제어하는 변수 |
| FMOD::Channel\* \_channel = nullptr; | Sound가 출력되는 Channel에 대한 정보를 담은 변수 |
| InputManager | EKeyState \_keyState[256]; | 다양한 키의 상태를 저장하는 변수 |
| POINT \_mousePosition; | 마우스의 위치를 저장하는 변수 |
| Shader | ID3D11DeviceChild\* \_shader; | Device에서 사용하는 Shader정보를 저장한 변수 |
| ID3DBlob\* \_code; | Shader Code를 저장한 변수 |
| ShaderManager | std::map<std::wstring, Shader\*> \_vertexShaderData;  std::map<std::wstring, Shader\*> \_pixelShaderData; | Shader별 정보를 저장하는 변수 |
| DXStateManager | std::map<std::string, SamplerStateFactoryInterface\*> \_samplerStateFactory;  std::map<std::string, RasterizerStateFactoryInterface\*> \_rasterizerStateFactory;  std::map<std::string, BlendStateFactoryInterface\*> \_blendStateFactory; | Sampler, Rasterizer, Blend 등 Device의 State정보를 생성하는 Factory객체를 저장하는 변수 |
| std::map<std::string, ID3D11SamplerState\*> \_samplerStateList;  std::map<std::string, ID3D11RasterizerState\*> \_rasterizerList;  std::map<std::string, ID3D11BlendState\*> \_blendStateList; | Sampler, Rasterizer, Blend 등 Device의 State를 저장하는 변수 |
| const static std::string kDefaultWrapSample;  const static std::string kDefaultWrapLinearSample;  const static std::string kDefaultSolidRasterizer;  const static std::string kDefaultWireFrameRasterizer;  const static std::string kDefaultBlend; | 기본적으로 생성된 State를 쉽게 검색할 수 있도록 미리 저장된 변수 |
| Text | ID2D1SolidColorBrush\* \_brush; | Text를 출력하기 위한 Brush 정보를 저장한 변수 |
| IDWriteTextFormat\* \_textFormat; | Text의 Font정보를 저장한 변수 |
| TextManager | std::map<std::wstring, ID2D1SolidColorBrush\*> \_brushes;  std::map<std::wstring, IDWriteTextFormat\*> \_textFormats; | dwrite.h Text출력에 필요한 객체를 저장하는 변수 |
| TextureResource | ID3D11Resource\* \_textureResource;  D3D11\_TEXTURE2D\_DESC \_desc;  ID3D11ShaderResourceView\* \_textureResourceView; | Texture원본 정보를 저장하는 변수 |
| Sprite | TexturePartCoordinate \_currentTexturePart; | Texture원본에서 출력하고자 하는 부분을 가리키는 변수 |
| TextureParam \_dTexture;  TextureParam \_dTile; | Tile, Scroll기능을 위한 변수 |
| SpriteAction | std::vector<TexturePartCoordinate> \_actionSequence; | Animation을 위해 출력해야하는 Texture의 부분들을 저장한 변수 |
| float \_interval; | 현 Sprite에서 다음 Sprite로 넘어가기 위해 필요한 시간을 저장한 변수 |
| DX2DAnimation | std::map<std::string, SpriteAction\*> \_actionList; | DX Object에서 출력될 수 있는 Animation들의 정보를 저장한 변수 |
| DX2DObject | std::vector<SimpleVertex2D> \_vertexList; | 중심에 대한 상대적인 정점 위치 정보를 저장한 변수 |
| std::vector<DXDrawableInterface\*> \_childObjectList; | 자신에게 달려있는 자식 Object들의 List |
| ID3D11Buffer\* \_vertexBuffer; | Device에 생성된 정점 버퍼 |
| Button | std::map<std::string, Sprite\*> \_spriteSet; | Button Animation에 필요한 정보를 저장한 변수 |
| ButtonFunctor\* \_functor; | Button이 눌릴 경우 수행할 Action을 저장한 Functor 객체 |
| TextUI | std::wstring \_fontFileName; | Font 정보를 저장한 파일 이름 |
| float \_height = 30; | 단일 글자의 높이 |
| std::map<char, Sprite\*> \_fontSprites; | 각 문자별 Sprite 정보 |
| std::vector<DX2DObject\*> \_outputText; | 출력하고자하는 Text의 Sprite Sequence가 저장된 변수 |
| DXCore | std::map<std::string, Scene\*> \_scenes; | Core에서 수행될 Scene들의 정보가 저장된 변수 |
| Timer\* \_timer; | Core의 시작부터 측정될 Timer 변수 |
| Scene | std::map<std::string, Scene\*> \_linkedScenes; | 이 Scene으로부터 이어질 수 있는 다음 Scene들의 종류 |
| DX2DCamera | Object2D\* \_captureArea; | Camera가 촬영하는 크기 |
| DX2DGameObject | DX2DObject\* \_dxObject;  Object2D\* \_physicsObject; | MVC 패턴의 Controller를 위한 변수 |
| DX2DHitBox | HitboxPosition\* \_relativePosition; | HitBox의 부모위치에 대한 상대위치를 저장한 변수 |
| DX2DInGameObject | EAireplaneFlightState \_flightStateChangeOrder = EAireplaneFlightState::STRAIGHT; | 기체가 진행하고자하는 비행상태를 저장한 변수 |
| EAireplaneFlightState \_currentFlightState = EAireplaneFlightState::STRAIGHT; | 기체의 현재 비행 상태를 저장한 변수 |
| const DWORD \_stateTransitionRequiredTime = 300; | 기체가 비행상태를 변경하기위해 필요한 시간을 저장한 변수 |
| HitboxPosition\* \_hitBoxData[9][hitboxCount]; | 비행상태에 따른 Hitbox들의 상대위치를 저장한 변수 |
| EnemyController | const DWORD \_maxDecisionTime = 300; | AI가 다음 행동을 결정하기 위해 필요한 시간 |
| DX2DMapObject | std::vector<QuadTree\*> \_layer; | Layer별 QuadTree |
| std::map<Object2D\*, DX2DInGameObject\*> \_physicsToDX2DMatch; | 충돌 판정을 위해 물리 객체인 Object2D와 DXObject를 mapping하는 함수 |
| Tile | Tile\* \_left;  Tile\* \_top;  Tile\* \_right;  Tile\* \_bottom; | 인접한 Tile들을 가리키는 변수 |
| bool \_used; | Tile이 현재 사용 중인지 알려주는 변수 |
| TileMemoryPool | std::vector<Tile\*> \_tileList; | 생성된 Tile객체들을 저장하는 변수 |
| Terrain | std::map<std::pair<int, int>, Tile\*> \_tiles; | 현재 출력에 사용되는 Tile들을 지칭하는 변수 |

[표 1-1] 프로젝트의 클래스 변수

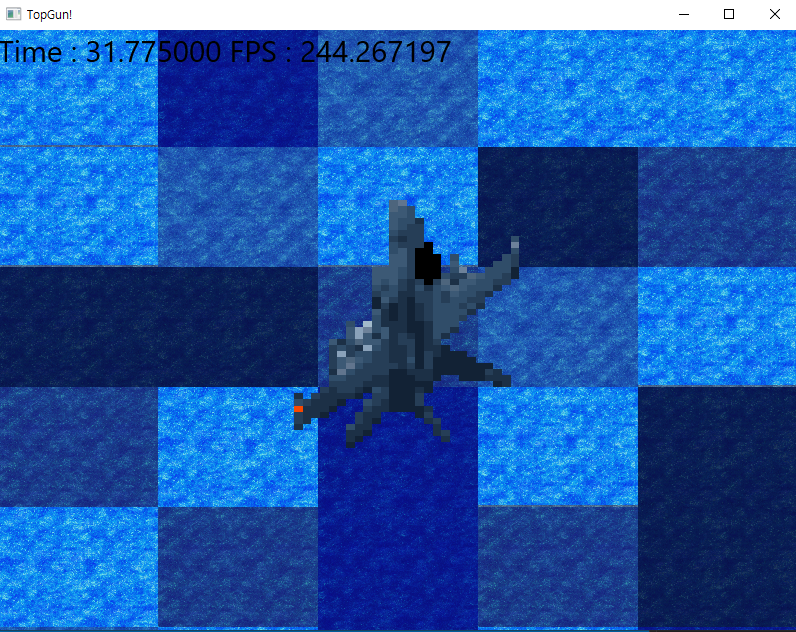
**2-2) 프로젝트에서 사용된 함수 역할**

|  |  |  |
| --- | --- | --- |
| **클래스명** | **함수** | **내용** |
| Timer | void Start(); | Timer가 시작한 시점의 Process의 시간을 저장하는 함수 |
| DWORD GetElapseTime(); | Timer가 시작한 시점부터 현재까지 Process가 진행한 시간까지의 사이값을 반환하는 함수 |
| SoundManager | Sound\* Load(std::wstring fileName); | Play, Stop, Pause 등 제어할 수 있는 Sound를 반환하는 함수 |
| void PlayInstanceSound(std::wstring fileName) | 제어할 필요가 없는 Sound를 출력하는 함수 |
| InputManager | EKeyState GetKeyState(DWORD key); | 원하는 Key의 상태를 반환하는 함수 |
| POINT GetMousePosition() { return \_mousePosition; } | Client Window에 대한 Mouse의 위치를 반환하는 함수 |
| DXDrawableInterface | virtual void Draw(ID3D11DeviceContext\* dc) = 0; | DirectX를 통해 Draw하는 기능을 구현하는 순수 가상 함수 |
| virtual void UpdateParentData(Position2D parentCenter, float parentRadian) = 0; | 객체들의 상대적인 위치에 따른 절대 위치를 계산할 수 있도록 부모의 정보를 Update하는 순수 가상 함수 |
| DXTextableInterface | virtual void Draw(ID2D1RenderTarget\* renderTarget) = 0; | dwrite.h를 통해 출력하는 Text를 Draw하는 기능을 구현하는 순수 가상 함수 |
| ShaderManager | template<typename func> Shader\* LoadShader(std::wstring fileName, std::string entryPoint, std::string target, std::map<std::wstring, Shader\*>& data, func); | 파일로부터 Shader를 읽어오는 함수 |
| Shader\* LoadVertexShader(std::wstring fileName, std::string entryPoint, std::string target);  Shader\* LoadPixelShader(std::wstring fileName, std::string entryPoint, std::string target); | Shader종류별로 가져오는 함수 |
| DXStateManager | void AddSamplerState(std::string name, SamplerStateFactoryInterface\* factory);  void AddRasterizerState(std::string name, RasterizerStateFactoryInterface\* factory);  void AddBlendState(std::string name, BlendStateFactoryInterface\* factory); | 원하는 State를 생성하는 Factory를 추가하는 함수 |
| ID3D11SamplerState\* GetSamplerState(std::string name);  ID3D11RasterizerState\* GetRasterizerState(std::string name);  ID3D11BlendState\* GetBlendState(std::string name); | 원하는 State를 반환받은 함수 |
| Text | void Draw(ID2D1RenderTarget\* renderTarget); | dwrite.h를 통해 Text를 출력하는 방법을 구현한 함수 |
| TextureResourceManager | TextureResource\* Load(std::wstring resourceFileName); | 파일로부터 Texture원본을 읽는 함수 |
| SpriteLoader | void RegisterSpriteFromFile(std::wstring resourceFileName, std::wstring infoFileName);  void RegisterSpriteWithRelativeValue(std::wstring resourceFileName, std::wstring spriteName, TexturePartRelative part);  void RegisterSpriteWithCoordinateValue(std::wstring resourceFileName, std::wstring spriteName, TexturePartCoordinate part); | Texture원본으로부터 필요한 Sprite범위 정보를 저장하기 위한 함수 |
| SpriteActionLoader | void RegisterSpriteActionFromFile(std::wstring resourceFileName, std::wstring infoFileName);  void RegisterSpriteActionWithRelativeValue(std::wstring resourceFileName, std::wstring actionName, std::vector<TexturePartRelative> sequenceRel);  void RegisterSpriteActionWithCoordinateValue(std::wstring resourceFileName, std::wstring actionName, std::vector<TexturePartCoordinate> sequenceCoord); | Texture원본으로부터 필요한 Animation 범위 정보를 저장하기 위한 함수 |
| DX2DAnimation | void RegisterState(std::string stateName, SpriteAction\* resource); | DX Object에서 출력될 수 있는 Animation을 저장하는 함수 |
| DX2DObject | void RotateDegree(float degree);  void RotateRadian(float radian); | DXObject를 회전시키는 함수 |
| void Draw(ID3D11DeviceContext\* dc) override; | DXObject를 출력하는 함수 |
| Button | void RegisterButton(std::string name, Sprite\* sprite); | Button Animation을 저장하는 함수 |
| TextUI | void Draw(ID3D11DeviceContext\* dc) override; | TextUI를 출력하는 함수 |
| bool Frame() override; | 이 함수에서 TextUI의 각 문자의 위치가 결정 |
| Scene | void LinkNewScene(std::string sceneName, Scene\* scene) { \_linkedScenes.insert(std::make\_pair(sceneName, scene)); } | 이 Scene으로부터 이어질 수 있는 다음 Scene을 추가하는 함수 |
| DX2DCamera | void MontageForFilm(DX2DGameObject\* object);  void MontageForFilm(DX2DInGameObject\* object); | Object와 Camera의 상대적인 위치에 따른 Object가 출력될 위치와 크기를 계산하는 함수 |
| void ConnectTo(DX2DGameObject\* object); | Camera를 특정 Object의 위치에 고정하는 함수 |
| Position2D PhysicsToDisplay(Vector2D absolute); | 물리좌표계에서 화면좌표계로 변환하는 함수 |
| DX2DGameObject | int GetHitboxIndex(EAireplaneFlightState flightState); | Flight상태 별 지정된 Hitbox정보가 저장된 Index값을 가져오는 함수 |
| DX2DHitBox | void SetRelativePosition(HitboxPosition\* position); | Hitbox를 Flight Status에 따른 지정된 상대위치로 이동시키는 함수 |
| DX2DInGameObject | void FlightStateChangeOrder(EXAxis xAxis, EYAxis yAxis, EZAxis zAxis);  void FlightStateChangeOrder(EAireplaneFlightState state); | 비행하고자 하는 상태를 지정하는 함수 |
| void FlightAccelerateDirection(EXAxis xAxis, EYAxis yAxis, EZAxis zAxis); | 비행하고자 하는 방향에 대한 정보를 비행하고자 하는 상태에 대한 정보로 변환하는 함수 |
| DX2DProjectController | virtual void Logic() = 0; | Controller가 수행할 logic을 구현하는 순수 가상 함수 |
| DX2DMapObject | bool IsHit(DX2DInGameObject\* attack, DX2DInGameObject\* defense); | 공격자의 HitBox와 방어자가 충돌하는지 점검하는 함수 |
| bool IsCollide(DX2DInGameObject\* object); | 특정 Object가 충돌하는지 점검하는 함수 |
| Tile | bool IsNeedToPrepareTile(); | 만약 Tile이 Camera의 범위에 걸쳐있다면 Tile과 인접한 Tile이 미리 준비되어야 한다 |
| TileMemoryPool | void RequireMemory(int newCount); | 미리 생성한 Tile들이 모두 사용중이라면 새 Tile생성을 요청하는 함수 |
| Terrain | bool ShouldBeAlive(Tile\* tile); | 특정 Tile이 Camera안에 있지도 않고, 인접 Tile 역시 Camera외부에 존재한다면 그 Tile은 필요가 없다 |

[표 1-2] 프로젝트의 클래스 함수

1. **최종 결과 및 추가 내용**
2. **범용성/유연성/확장성/간결성 고려하여 추가된 내용**
   1. **현재 프로젝트 설계(구현 내용)**
3. 지난 수업 과정에서 미리 구현한 DirectX와 QuadTree를 재사용할 수 있도록 MVC패턴 적용하기 위해 DX2DGameProject용 Controller를 제작하여 Game loop가 돌아가도록 구현하였다.
4. 하늘을 날아다니는 전투기의 특성 상 고도 차이는 중요한 요소일 수 밖에 없다. 이러한 고도 차이를 구현하기 위해 1개의 Map에 Multi Layer의 QuadTree를 생성하여 각 Layer단위로 충돌 검사를 하도록 진행하였다.
5. Multi Layer에서 AI의 이동을 위해 지정된 시간이 지나면 AI가 진행하고자 하는 방향을 Random하게 선택한 후 이동할 수 있도록 구현하여 User가 상투적이지 않은 경험을 할 수 있도록 구현하였다.
6. 또한 2D 게임의 특성 상 고도 차이를 느끼기 힘들 수 있다. 이 문제를 해결하고자 상대 고도 차이에 따른 출력 객체의 크기 비율을 변경하여 높이 있는 객체는 더 크게 출력하고, 낮게 있는 객체는 더 작게 출력하는 방식으로 직관성을 높이도록 구현하였다.
   1. **향후 개발 내용**
7. Memory의 잦은 해제는 Fragmentation 문제를 발생시킬 수 있으며, 많은 비용을 요구한다. 이를 막기위해 Memory Pool을 구현하여 생성된 객체를 재사용할 수 있도록 하였지만, 현재 구현 방식은 객체 생성 시 객체 단위로 메모리 할당을 요청하기 때문에 한번에 큰 메모리를 할당받은 후, Placement New방식으로 초기화 하는 방식에 비하면 여전히 높은 생성 비용을 요구한다. 생성 비용 감소를 위해 대규모 메모리 할당 후 Placement new를 이용한 방식을 구현할 예정이다.
8. 기존에 구현해놓은 QuadTree의 경우 고정된 장소에서 사용할 것을 가정하여 Static Object와 Dynamic Object를 분리하였고, 매 Frame마다 Dynamic Object만 위치를 재계산하는 방식으로 구현하였다. 하지만 이번 프로젝트의 경우 Player의 위치는 기존의 방식보다 극단적으로 움직일 가능성이 높아 지정된 QuadTree의 크기보다 더 먼 위치에 배치될 가능성이 높고, 이 경우 QuadTree의 최상단 노드에 저장되도록 하여 기능에는 문제가 없지만, 만약 다수의 Object가 생성된다면 QuadTree의 장점인 공간분할이 의도와 다르게 수행되지 않을 가능성이 높다. 이를 위해 QuadTree의 원점을 재배치할 수 있도록 수정할 수 있도록 할 예정이다.
9. Scene단위로는 문제가 없지만 여러 Scene을 교체하다보면 메모리가 누수되는 현상이 발생한다. 메모리 관리를 수정할 예정이다.
10. **최종 결과**

****

****

****