**게임개발자전문가과정**

**과 목 명 : 게임일반프로그래밍**

**능력단위 : 게임알고리즘**

**제출일자 : 2022년 08월 30일**

**포트폴리오 : Octree가 적용된 지형에서 개체 충돌이 구현된 프로그램**

**작 성 자 : 김동완**

**<제출내역>**

1. **Octree가 적용된 지형에서 개체 충돌 구현 프로젝트**
2. **Octree가 적용된 지형에서 개체 충돌 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
7. 클래스 Docs
8. **최종 결과 및 추가 내용**
9. 범용성/유연성/확장성/간결성 고려하여 추가된 내용
   1. 현재 프로젝트 설계(구현 내용)
   2. 향후 개발 내용
10. 최종 결과
11. **프로젝트 소개 및 개요**
12. **프로젝트 소개**

게임에는 수많은 유저, NPC, 장애물 등의 오브젝트 및 지형이 존재한다. 이러한 오브젝트들을 움직이기 위해서는 각 오브젝트 간의 간섭, 충돌을 먼저 확인할 필요성이 있다. 하지만 충돌 검사를 시행할 때 지형에 있는 모든 오브젝트를 검사하는 것은 시간도 오래 걸리며 비효율 적이다. 이러한 문제를 해결하기 위해 공간을 분할하여 트리구조로 만든 것이 공간 분할(Space Division) 알고리즘이다.

이 프로젝트에서는 공간 분할 알고리즘을 공부하기 위해 2차원 지형과 3차원 지형으로 나누어 2차원 지형은 Quad Tree, 3차원 지형은 Octree로 공간 분할을 적용하여, 공간 분할이 적용된 지형에서 정적 및 동적 오브젝트의 이동처리 및 충돌을 구현한다.

1. **프로젝트 주요 기술**

* **Space Division**

공간 분할 알고리즘은 공간을 분할하여 오브젝트가 속해 있는 노드 혹은 Frustum만 탐색하여 전체 공간을 탐색하는 것에 비해 탐색 속도가 빠르며 효율적인 탐색이 가능하다. 공간 분할의 종류로는 BSP, Quad Tree, Octree, K-D Tree 등이 있으며, 필요한 지형에 따라 다른 알고리즘을 적용하여 사용한다.

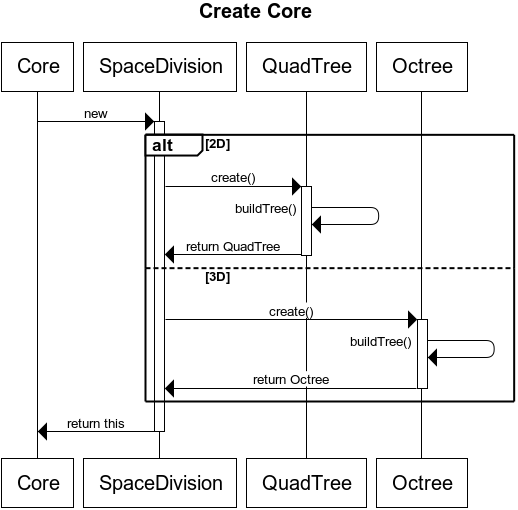
이 프로젝트에서는 2차원 지형에서는 Quad Tree를 3차원 지형에서는 Octree를 사용하여, 정적 개체와 동적 개체의 이동 및 충돌을 구현한다.

* **Design Pattern**

디자인 패턴은 프로그램 제작 중 발생하는 문제를 해결하기 위한 기법 등의 노하우를 명시적으로 작성해 놓은 것으로, 구조, 생성 및 알고리즘 등에 다양하게 사용된다.

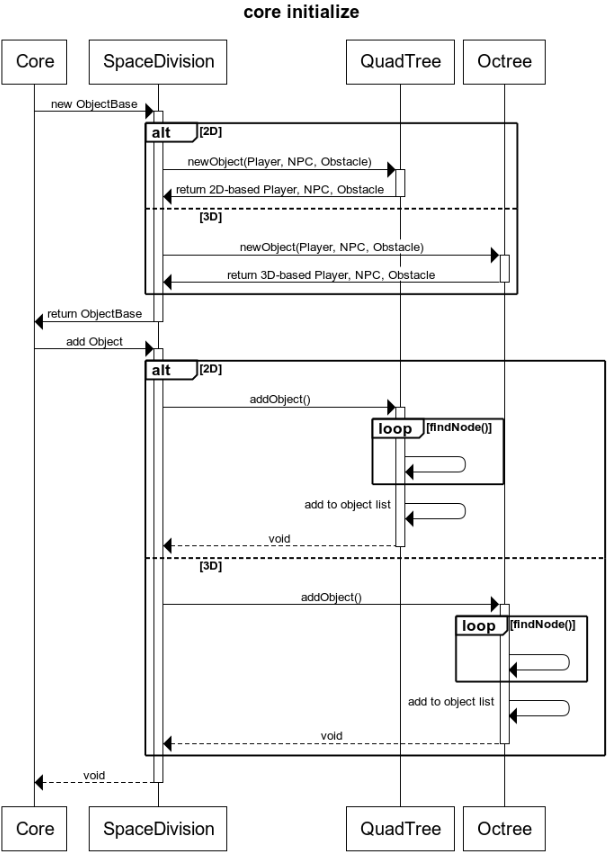
이 프로젝트에서는 기본적으로 Abstract Factory Pattern과 유사한 구조를 사용하며, 다형성을 위하여 Factory Method Pattern이 사용되었다. 또한 향후 자주 사용하게 될 Singleton Pattern을 시험적으로 적용시켜 학습 경험을 늘리고자 한다.

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

****

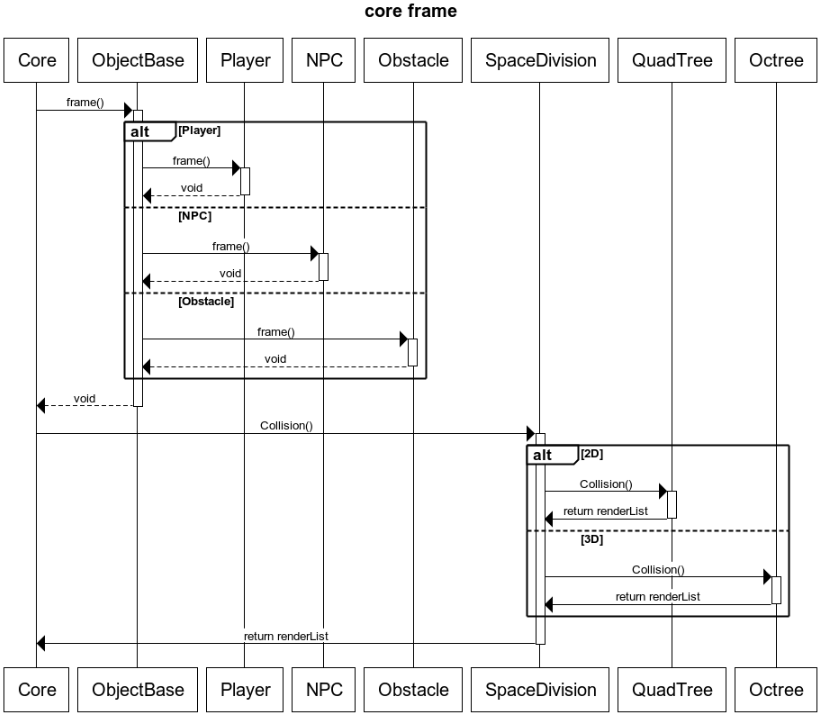
[그림 1-1] Create Core 시퀀스 다이어그램(Sequence Diagram)

* 1. **Core-Initialize**

****

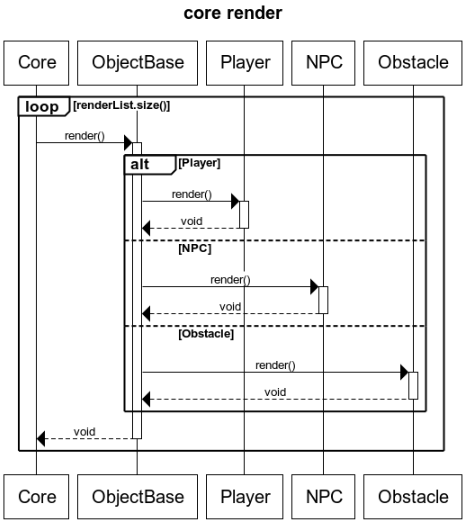
[그림 1-2] Core-Initialize 시퀀스 다이어그램(Sequence Diagram)

* 1. **Core-Frame**

****

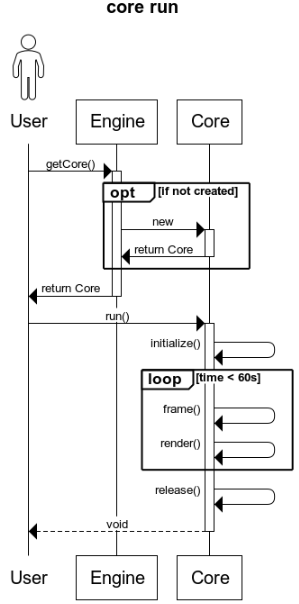
[그림 1-3] Core-Frame 시퀀스 다이어그램(Sequence Diagram)

* 1. **Core-Render**

****

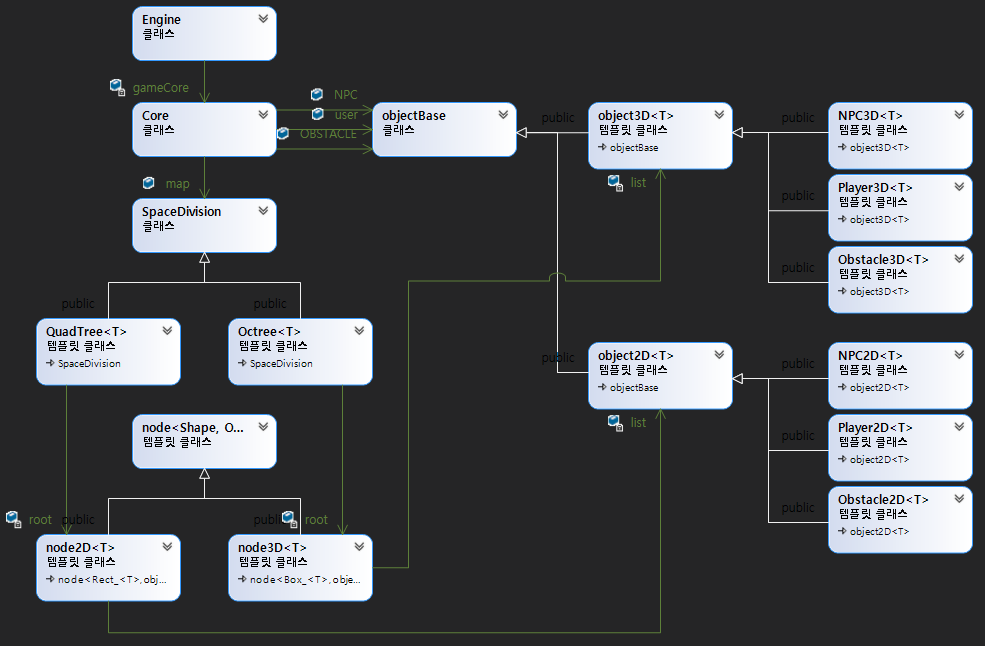
[그림 1-4] Core-Render 시퀀스 다이어그램(Sequence Diagram)

* 1. **Core-Run**



[그림 1-5] Core-Run 시퀀스 다이어그램(Sequence Diagram)

1. **클래스 다이어그램(Class Diagram)**



[그림 1-7] 클래스 다이어그램(Class Diagram)

1. **클래스 Docs**

**Engine Parameter**

|  |  |
| --- | --- |
| 이름 | 설명 |
| static Engine\* engine | SingleTone 생성을 위한 Engine 변수. |
| Core gameCore | 게임 코어 변수. |
| Node<T>\* next | 다음 노드를 위한 포인터 변수 |

**Engine Function**

|  |  |
| --- | --- |
| 이름 | 설명 |
| static Engine\* getEngine(CORE\_TYPE \_type) | \_type: 2D 혹은 3D 선택.  최초 1회에 한해 engine 생성 후 engine 반환. |
| Core\* getCore() | gameCore 반환. |

**Core Parameter**

|  |  |
| --- | --- |
| 이름 | 설명 |
| SpaceDivision\* map | Quad Tree 혹은 Octree를 가질 수 있는 변수. |
| objectBase\* user | User 캐릭터 오브젝트 변수. |
| std::vector<objectBase\*> NPCList | NPC 오브젝트 리스트. |
| std::vector<objectBase\*> obstacleList | Obstacle 오브젝트 리스트. |
| std::vector<objectBase\*> renderList | Render할 오브젝트 리스트. |

**Core Function**.

|  |  |
| --- | --- |
| 이름 | 설명 |
| void initialize() | User, NPC 및 Obstacle 오브젝트 생성. |
| void frame(float \_dt); | \_dt: frame 시간.  동적 오브젝트의 이동 및 User 오브젝트의 충돌 체크. |
| void render() | 오브젝트 출력. |
| void release() | map 삭제 및 리스트 릴리즈. |
| void run() | 초기화 및 frame, render 수행. |

**SpaceDivision Virtual Function**

|  |  |
| --- | --- |
| 이름 | 설명 |
| virtual void create(void\* \_shape) | \_shape: node의 모양.  Quadtree, Octree 생성을 위한 가상 함수. |
| virtual void addObject(objectBase\* \_obj) | \_obj: 추가할 오브젝트  오브젝트 추가를 위한 가상 함수. |
| virtual bool Collision(objectBase\* \_src, std::vector<objectBase\*>\* \_dst, std::vector<void\*>\* \_dstSection) | \_src: 충돌 주체.  \_dst: 주체와 충돌한 오브젝트 리스트.  \_dstSection: 충돌 범위 리스트.  충돌 체크를 위한 가상 함수. |
| virtual void updateDynamicObject() | 오브젝트 위치 업데이트를 위한 가상 함수. |
| virtual bool checkBorder(objectBase\* \_target) | \_target: 오브젝트.  오브젝트가 맵 밖을 벗어나는지 확인하는 가상 함수. |

**Octree : SpaceDivision Parameter**

|  |  |
| --- | --- |
| 이름 | 설명 |
| node3D<T>\* root | 3D 데이터를 가지고 있는 뿌리 노드. |

**Octree : SpaceDivision Function**

|  |  |
| --- | --- |
| 이름 | 설명 |
| void create(Box\_<T> \_box) | \_box: 생성 할 box 영역.  Octree 생성. |
| node3D<T>\* findNode(node3D<T>\* \_parent, object3D<T>\* \_obj) | \_parent: 찾기 시작할 위치 노드.  \_obj: 찾을 오브젝트.  오브젝트가 포함된 노드 반환. |
| bool isHitMinX(node3D<T>\* \_border, object3D<T>\* \_target, bool \_move) | \_border: 제한 영역.  \_target: 주체.  \_move: 범위 안으로 이동.  오브젝트가 맵 범위 x축 최소값에 도달했는지 확인 및 이동과 방향전환 유도하는 함수. |
| bool isHitMaxX(node3D<T>\* \_border, object3D<T>\* \_target, bool \_move) | \_border: 제한 영역.  \_target: 주체.  \_move: 범위 안으로 이동.  오브젝트가 맵 범위 x축 최대값에 도달했는지 확인 및 이동과 방향전환 유도하는 함수. |
| bool isHitMinY(node3D<T>\* \_border, object3D<T>\* \_target, bool \_move) | \_border: 제한 영역.  \_target: 주체.  \_move: 범위 안으로 이동.  오브젝트가 맵 범위 y축 최소값에 도달했는지 확인 및 이동과 방향전환 유도하는 함수. |
| bool isHitMaxY(node3D<T>\* \_border, object3D<T>\* \_target, bool \_move) | \_border: 제한 영역.  \_target: 주체.  \_move: 범위 안으로 이동.  오브젝트가 맵 범위 y축 최대값에 도달했는지 확인 및 이동과 방향전환 유도하는 함수. |
| bool isHitMinZ(node3D<T>\* \_border, object3D<T>\* \_target, bool \_move) | \_border: 제한 영역.  \_target: 주체.  \_move: 범위 안으로 이동.  오브젝트가 맵 범위 z축 최소값에 도달했는지 확인 및 이동과 방향전환 유도하는 함수. |
| bool isHitMaxZ(node3D<T>\* \_border, object3D<T>\* \_target, bool \_move) | \_border: 제한 영역.  \_target: 주체.  \_move: 범위 안으로 이동.  오브젝트가 맵 범위 z축 최대값에 도달했는지 확인 및 이동과 방향전환 유도하는 함수. |

**QuadTree : SpaceDivision Parameter**

|  |  |
| --- | --- |
| 이름 | 설명 |
| node2D<T>\* root | 2D 데이터를 가지고 있는 뿌리 노드. |

**QuadTree : SpaceDivision Function**

|  |  |
| --- | --- |
| 이름 | 설명 |
| void create(Rect\_<T> \_rect) | QuadTree 생성. |
| node2D<T>\* findNode(node2D<T>\* \_parent, object2D<T>\* \_obj); | 오브젝트가 포함된 노드 반환. |
| bool isHitLeft(node2D<T>\* \_border, object2D<T>\* \_target, bool \_move) | \_border: 제한 영역.  \_target: 주체.  \_move: 범위 안으로 이동.  오브젝트가 맵 범위 x축 최소값에 도달했는지 확인 및 이동과 방향전환 유도하는 함수. |
| bool isHitRight(node2D<T>\* \_border, object2D<T>\* \_target, bool \_move) | \_border: 제한 영역.  \_target: 주체.  \_move: 범위 안으로 이동.  오브젝트가 맵 범위 x축 최대값에 도달했는지 확인 및 이동과 방향전환 유도하는 함수. |
| bool isHitTop(node2D<T>\* \_border, object2D<T>\* \_target, bool \_move) | \_border: 제한 영역.  \_target: 주체.  \_move: 범위 안으로 이동.  오브젝트가 맵 범위 y축 최소값에 도달했는지 확인 및 이동과 방향전환 유도하는 함수. |
| bool isHitBottom(node2D<T>\* \_border, object2D<T>\* \_target, bool \_move) | \_border: 제한 영역.  \_target: 주체.  \_move: 범위 안으로 이동.  오브젝트가 맵 범위 y축 최대값에 도달했는지 확인 및 이동과 방향전환 유도하는 함수. |

**objectBase Parameter**

|  |  |
| --- | --- |
| 이름 | 설명 |
| std::string name | 오브젝트 이름. |
| float mass | 오브젝트 질량. |
| OBJECT\_TYPE type = OBJECT\_TYPE::STATIC\_OBJECT | 오브젝트 타입. |

**objectBase Virtual Function**

|  |  |
| --- | --- |
| 이름 | 설명 |
| virtual void Random() | 오브젝트 랜덤 생성. |
| virtual void frame(float \_dt) | \_dt: frame 시간.  오브젝트 움직임 연산. |
| virtual void render() | 오브젝트 출력. |

**object2D : objectBase Parameter**

|  |  |
| --- | --- |
| 이름 | 설명 |
| Rect\_<T> shape | 오브젝트 모양. |
| Vector2D\_<T> force | 오브젝트에 가해진 벡터. |
| Vector2D\_<T> accel | 오브젝트의 가속도. |
| Vector2D\_<T> velocity | 오브젝트의 속도. |

**object3D : objectBase Parameter**

|  |  |
| --- | --- |
| 이름 | 설명 |
| Box\_<T> shape | 오브젝트 모양. |
| Vector3D\_<T> force | 오브젝트에 가해진 벡터. |
| Vector3D\_<T> accel | 오브젝트의 가속도. |
| Vector3D\_<T> velocity | 오브젝트의 속도. |

**NPC2D : object2D<T>**

**Obstacle2D : object2D<T>**

**Player2D : object2D<T>**

**NPC3D : object3D<T>**

**Obstacle3D : object3D<T>**

**Player3D : object3D<T>**

**node Parameter**

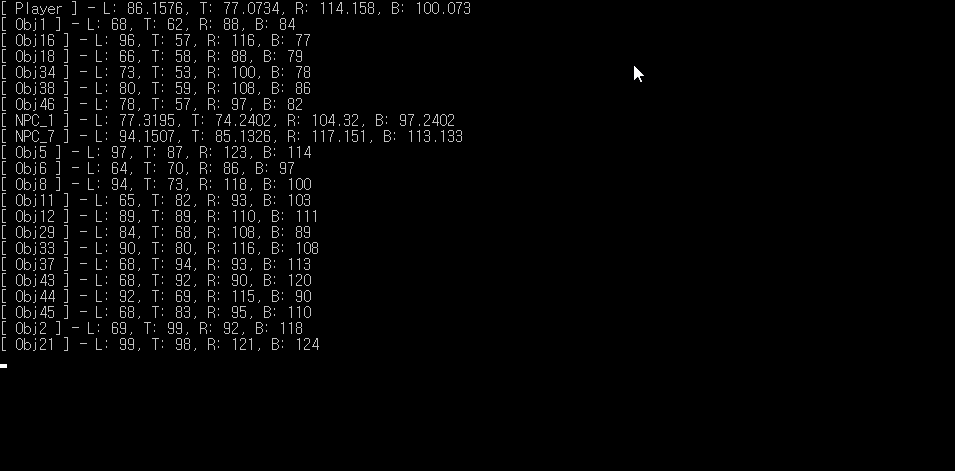
|  |  |
| --- | --- |
| 이름 | 설명 |
| node<Shape, ObjectDimension>\* parent | 부모 노드. |
| std::vector<node<Shape, ObjectDimension>\*> child | 자식 노드 리스트. |
| int depth = 0 | 노드의 깊이. |
| Shape shape | 노드의 모양. |
| std::vector<ObjectDimension\*> stObjList | 정적 오브젝트 리스트. |
| std::vector<ObjectDimension\*> dyObjList | 동적 오브젝트 리스트. |

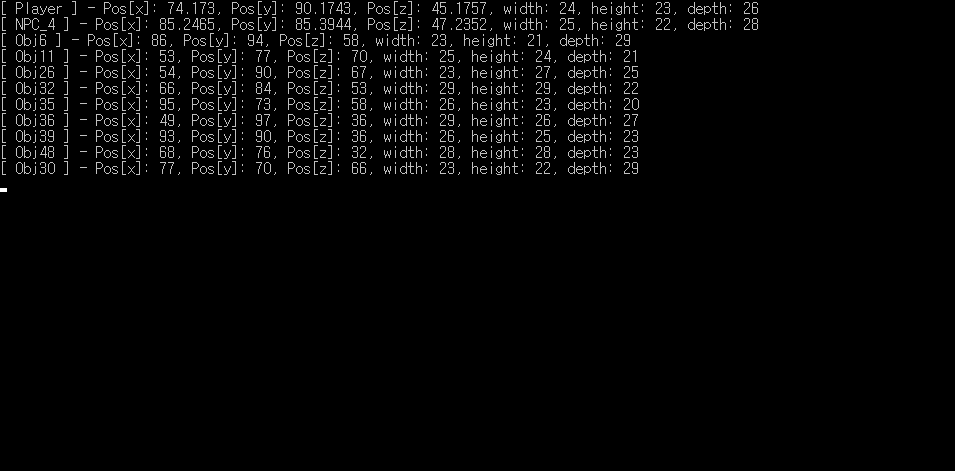
**node2D : node<Rect\_<T>, object2D<T>>**

**node3D : node<Box\_<T>, object3D<T>>**

1. **최종 결과 및 추가 내용**
2. **범용성/유연성/확장성/간결성 고려하여 추가된 내용**
   1. **현재 프로젝트 설계(구현 내용)**
3. Quad Tree와 Octree 구현, 정적 및 동적 개체의 충돌 기능 구현.
4. 확장성을 고려한 Design Pattern을 이용하여 설계.
5. Template Class와 상속을 적극 활용하여 다형성 확보.
   1. **향후 개발 내용**
6. 이 프로젝트에서는 Quad Tree와 Octree를 따로 구현해 놓은 후 하나로 합치는 리펙토링 작업을 거쳐 구현을 하였지만, 실제로 새로운 공간 분할 알고리즘을 추가하였을 때 기존 코드의 수정이 일어나지 않는지 확인하지 못하여 아쉬운 부분이 있어 BSP 등의 새로운 알고리즘 추가 후 테스트 필요함.
7. 그래픽 없이 콘솔로 출력하기 때문에 수작업으로 디버깅 하는 것 이외엔 정상적으로 동작을 하고 있는지에 대한 확인이 불가능하여 Direct X 적용하여 그래픽 출력 확인 예정.
8. 다형성과 재사용성 등을 위해 템플릿 클래스와 상속을 위주로 하여 클래스를 작성하였으나, 구조가 깔끔하지 않아 리펙토링 필요로 함.
9. 충돌 이후 충돌된 오브젝트들에 대한 처리가 이루어 지지 않아 아쉬운 부분이 있어 향후 충돌한 오브젝트의 파괴 및 이동 등의 기능 추가 예정.
10. **최종 결과**

**2-1) 2D - Quad Tree**



**2-2) 3D - Octree**