**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**

공간 분할(Space Division)의 종류 중 하나인 Octree를 이용하여, Octree가 적용된 지형에서 정적 및 동적 개체의 이동처리 및 충돌을 구현한다.

Octree 뿐만 아니라 Quad Tree가 적용된 지형에서 정적 및 동적 개체의 이동처리 및 충돌을 구현하고, 다형성을 위해 추상 클래스를 이용하여 조건문을 최소화

1. **프로젝트 주요 기술**

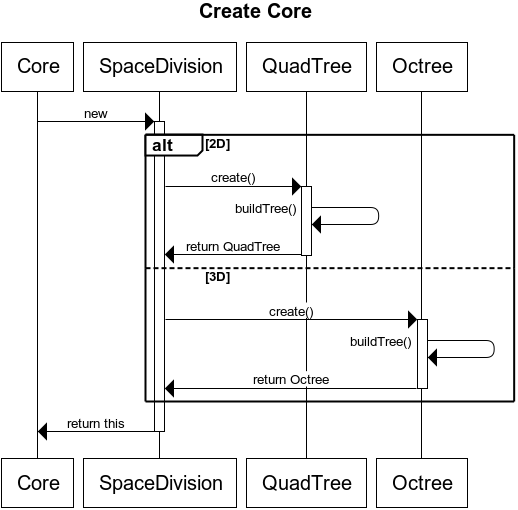
* **Space Division**

Quad Tree를 구현하여 2차원 평면에서의 정적 개체와 동적 개체의 이동 및 충돌을 구현하고, Octree를 구현하여 3차원 평면에서의 정적 개체와 동적 개체의 이동 및 충돌을 구현 함.

* **Design Pattern**

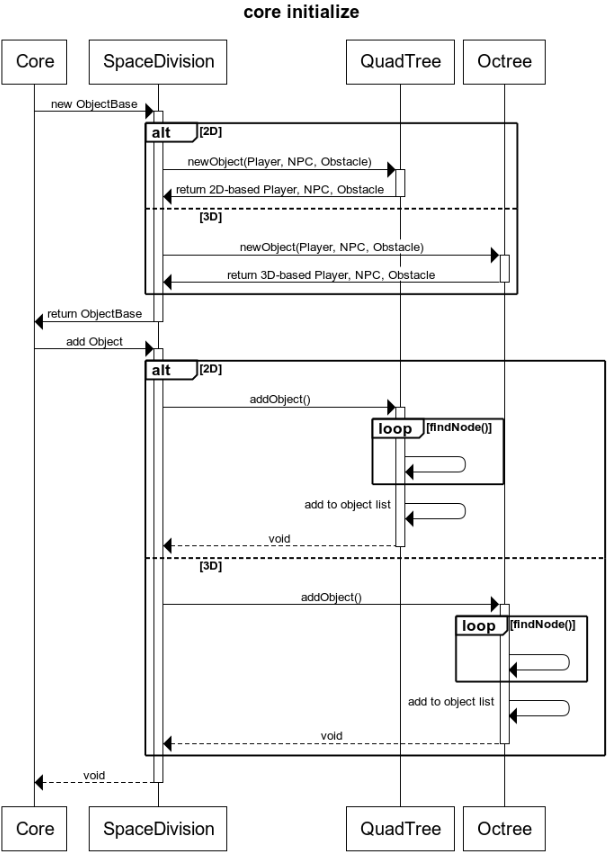
해당 프로젝트에서는 instance를 하나만 가지게 하고 접근성을 높이기 위해 Engine 클래스를 Singleton Pattern을 이용하여 구현하였으며, Abstract Factory Pattern을 이용하여 2D Core 혹은 3D Core를 생성하게 하여 서로 간섭이 불가능하게 하며 Quad Tree 혹은 Octree를 생성하여 별개의 동작을 하도록 구현하였다.

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

****

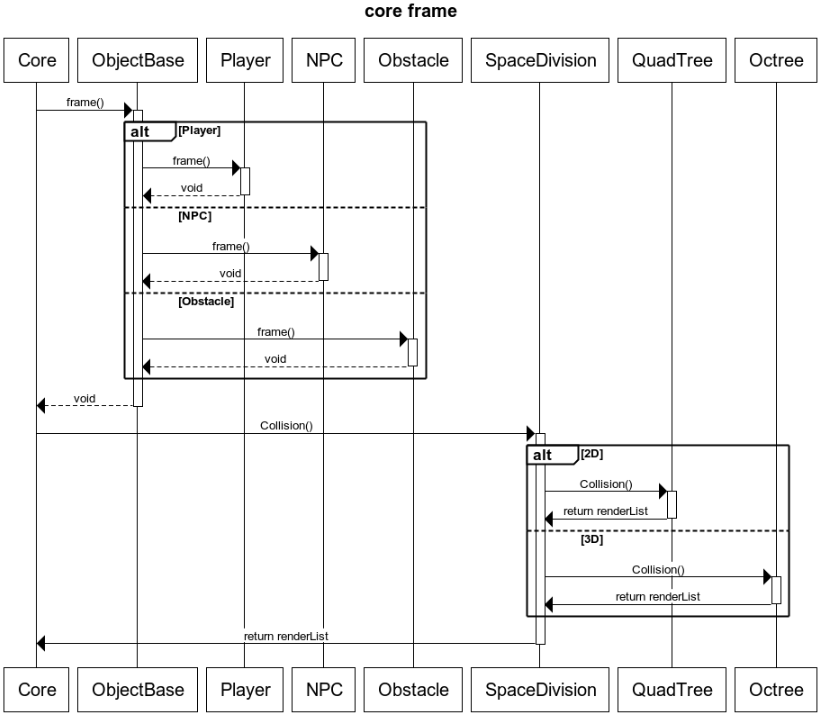
[그림 1-1] Create Core 시퀀스 다이어그램(Sequence Diagram)

* 1. **Core-Initialize**

****

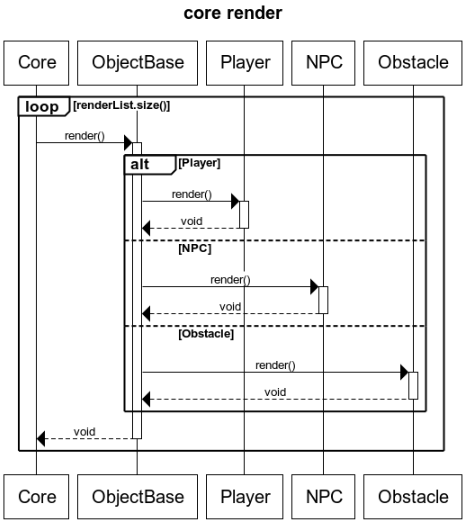
[그림 1-2] Core-Initialize 시퀀스 다이어그램(Sequence Diagram)

* 1. **Core-Frame**

****

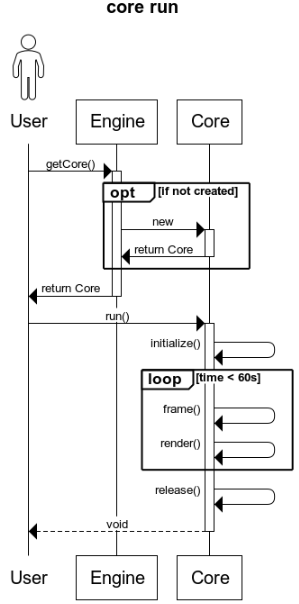
[그림 1-3] Core-Frame 시퀀스 다이어그램(Sequence Diagram)

* 1. **Core-Render**

****

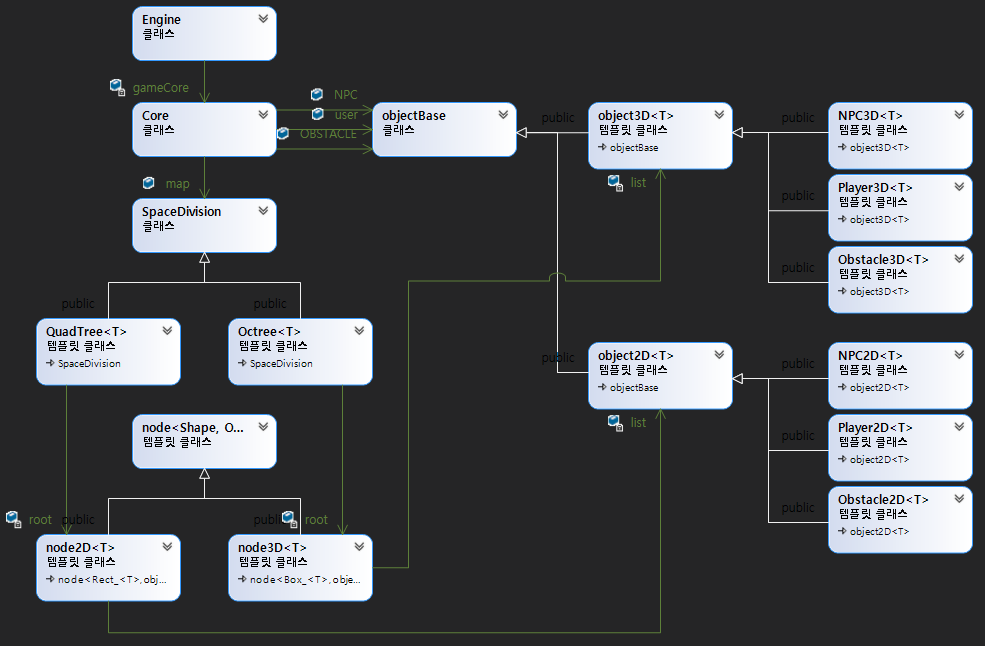
[그림 1-4] Core-Render 시퀀스 다이어그램(Sequence Diagram)

* 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**

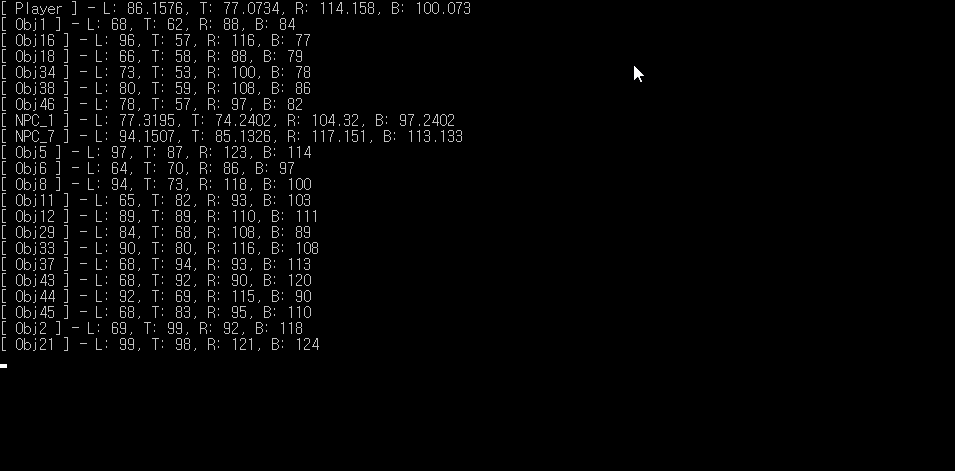
|  |  |
| --- | --- |
| 이름 | 설명 |
| node<Shape, ObjectDimension>\* parent | 부모 노드. |
| std::vector<node<Shape, ObjectDimension>\*> child | 자식 노드 리스트. |
| int depth = 0 | 노드의 깊이. |
| Shape shape | 노드의 모양. |
| std::vector<ObjectDimension\*> stObjList | 정적 오브젝트 리스트. |
| std::vector<ObjectDimension\*> dyObjList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

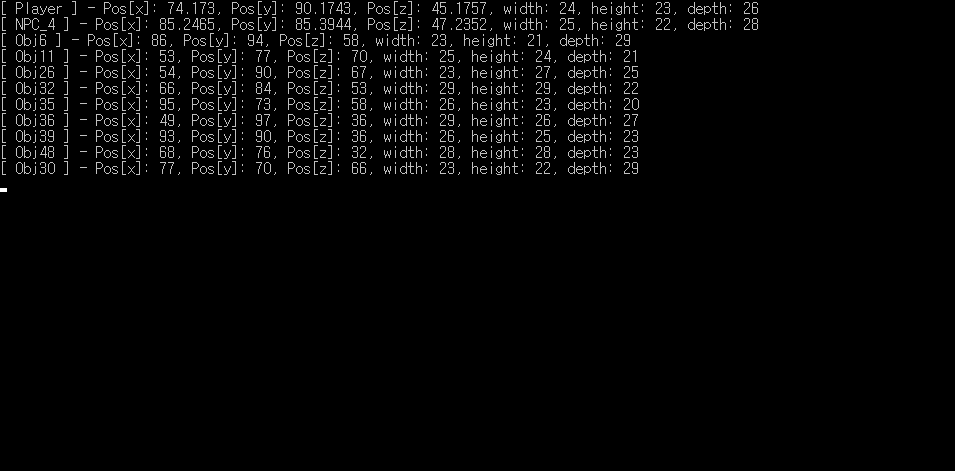
**node2D : node<Rect\_<T>, object2D<T>>**

**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. 범용성, 유연성, 확장성, 재 사용성 모두를 고려하여 Template Double Linked List와 iterator를 함께 구현하여 STL의 List처럼 사용 가능.
4. Double buffering의 기능만을 가진 console UI 클래스를 상속받아 사용하여 재 사용성 높임.
5. 파일 입출력 시 결과를 쉽게 보기 위해 csv포맷을 채용하여 간결성을 높여 구현.
   1. **향후 개발 내용**
6. 현 프로젝트에서는 블록 단위 입출력을 사용하지 않았지만, 다른 프로젝트에서의 재사용을 위해 블록 단위 입출력 기능 추가 및 별개의 File I/O Class로 제작하여 재 사용성 높일 예정.
7. 시간 분배 미흡으로 UI Class의 경우 가시성이 좋지 못하여 재 사용이 어려울 것으로 예상되고, Menu Class의 상속을 통한 다형성 확보가 되지 않아 아쉬운 부분이 있어, 상속을 통해 각 Menu Class를 생성하여 다형성과 간결성 확보하며 전체적인 UI Class의 리펙토링 예정.
8. 몇 가지의 warning과 return 값을 반환하지 않는 경우가 있어, 예외 처리 및 형 변환 추가 예정.
9. 이미 입력된 데이터 수정 기능이 없어 리펙토링 시 구현 예정.
10. **최종 결과**

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



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