**게임 엔진 프레임워크 개발 보고서**

2020105618 백건희

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | **주제** | | 오브젝트 인식과 오브젝트 간 충돌 감지 및 처리 프레임워크 |  |  | | --- | | **개발 기간** | | 11월 26일 ~ 12월 10일 |  |  | | --- | | **구현 목적** | | 게임 프로그래밍 입문 강의를 들으면서 가장 인상깊고 기억에 남았던 강의인 Collision Detection과 RigidBody, 그리고 Dot Product의 범위 내 오브젝트 인식 부분을 바탕으로 오브젝트 간의 충돌 처리를 직접 구현하고 싶다는 마음이 동기가 되어 실제 유니티에서 사용되는 Transform 및 BoxCollider2D 컴포넌트를 모방하여 게임 화면에서 오브젝트 간의 충돌 감지 및 충돌 처리에 대한 구현을 시작했다. 최종 목표는 충돌 처리와 지정한 범위 내의 오브젝트를 인식하여 이 프레임워크를 사용하는 사용자에 의해 적절히 처리될 수 있도록 만드는 것이다. |  |  | | --- | | **구현 내용** | | 오브젝트의 위치, 회전, 스케일 정보를 담은 Transform 정보와, 충돌 감지 및 처리를 위한 CollisionDetection 기능을 가지는 Component 프레임워크를 개발한다. Component는 2D를 기반으로 동작하며, Transform 클래스와 그를 상속하는 CollisionDetection 클래스를 보유하고 있으며 이외에 추가적인 기능을 부여하기 위한 내장 함수와 특정 좌표 형태를 나타내기 위한 Vector2D 프레임워크를 추가적으로 구현한다. Vector2D는 일반적으로 크기와 방향을 가지는 기하적 관점의 벡터와 동일하나, 때에 따라선 단순히 각 성분에 따라 정보를 나누기 위해서도 활용되도록 구현한다. |  |  | | --- | | **주요 클래스** | | |  |  | | --- | --- | | **Transform** | | | 멤버 변수 | 메소드 | | position, velocity, acceleration, scale, angle, name, points | Translation, Move, Scaling, Rotation, transform |  |  |  | | --- | --- | | **CollisionDetection(Tranform)** | | | 멤버 변수 | 메소드 | | Transform에서 상속된 멤버 + colliderPosition, colliderScale, colliderAngle, collider, mass, kinfOfCollider, collisionTimer, latelyCollidedObject | setPosition, setScale, setAngle, getCollider, Move(오버라이딩), OBB, getAxes, overlapOnAxis, line\_segment, overlapCircleArea, overlapCircularArea |  |  |  | | --- | --- | | **BoxCollider** | | | 멤버 변수 | 메소드 | | CollisionDetection에서 상속된 멤버 + collder(부모 클래스의 Collider 지정 방식과 다름) | None |  |  |  | | --- | --- | | **Vector2D** | | | 멤버 변수 | 메소드 | | x, y | add, sub, scalarProduct, dotProduct, magnitude, normalize, toList, toTuple | |  |  | | --- | | **클래스 상호관계도** | | 텍스트, 도표, 평행, 라인이(가) 표시된 사진  자동 생성된 설명 |  |  |  | | --- | --- | | **구현 과정** | | | 개발 1회차 | **Vector2D와 Transform의 전반적인 형태를 구현한다**. 테스트용 메인함수에서는 Transform을 바탕으로 화면 위 적절한 위치에 객체가 생성되는지 확인한다. / **유효**( ~ 11.30) | | 개발 2회차 | **CollisionDetection을 구현한다.** Transform을 그대로 상속하여 객체의 Transform을 유지하며 collider의 정보를 Transform을 그대로 가져와 사용하되, collider의 관리는 부모 클래스와 독립적으로 작용하도록 개발한다. 테스트용 메인함수에서는 Transform과 collider가 독립적으로 동작하는지 확인한다. / **유효** (11.31 ~ 12.01) | | 개발 3회차 | CollsionDetection에 들어있는 collider정보를 바탕으로 **서로 다른 두 객체 간의 충돌 여부를 확인하는 기능을 구현한다.** 첫 시도는 OBB 알고리즘을 이용한 충돌 감지이며 테스트용 메인함수에서는 객체 간 충돌 감지가 일어나는 경우 콘솔창에 적절한 메시지를 띄운다. / **유효** (12.02 ~ 12.03 | | 개발 4회차 | **충돌 감지가 일어난 객체 간의 속도 변화를 구현한다.** 테스트용 메인함수에서는 속도를 가진 객체 간의 충돌 시 적절한 속도 변화를 만들어지는지 확인한다. / **무효** (12.03 ~12.06) | | 개발 5회차 | OBB 대신 **Raycast를 이용한 충돌 감지를 구현했다**. Raycast를 이용함으로써, 강의노트 RigidBody편에 있는 벽과 충돌했을 경우 속도 변화를 달리 줄 수 있는 메커니즘을 구현했다. 테스트용 메인함수에서는 4회차와 마찬가지로 충돌 시 적절한 속도 변화가 있는지 확인한다. / **유효** (12.06 ~ 12.08) | | 개발 6회차 | **지정된 범위 내에 감지하고자 하는 대상이 있는지 확인하는 기능을 구현한다**. 원형 범위와 부채꼴형 범위를 만들어 해당 범위 내에 감지하고자 하는 대상을 확인한다. 테스트용 메인코드에서는 지정 범위 내에 대상이 있는 경우 적절한 메시지를 띄운다. / **유효** (12.08 ~ 12.09) | | 개발 7회차 | **최적화 작업을 수행한다.** 특히 충돌 감지를 수행하는 경우 매 틱마다 수행하면 성능 저하가 극심하다. 따라서 원형 범위를 지정하여 충돌 확률이 1이 가까울 정도로 근접한 경우에 Raycast를 이용하여 충돌을 감지하고 그렇지 않은 경우엔 cost가 낮은 함수를 이용하도록 하여 성능을 향상했다. 또한 충돌 감지 주기를 인위적으로 조절하여 성능 향상에 조금 더 힘을 보탰다. / **유효** (12.09 ~ 12.10) |  |  | | --- | | **동작 원리** | | 게임 내 오브젝트는 3가지 클래스를 이용해 정의할 수 있다. 오직 위치, 스케일, 회전 정보만을 가지며 충둘 감지를 하지 않는 Transform 객체, 지정된 Collider를 가지며 충돌 감지와 속도 변화를 모두 담은 CollisionDetecion 객체, 변의 크기를 통해 자동적으로 Collider를 생성하는 BoxCollider 객체이다. Transform으로 생성된 객체는 지속적인 움직임, 스케일 변화, 회전 변화를 통해 화면에 본인의 Transform 정보를 지속적으로 나타낸다. Collision이 포함된 클래스로 생성된 객체는 본인의 Collider 정보를 바탕으로 지속적으로 충돌 여부를 검사한다. 이때, 최적화를 위해 일정 범위 내에 있는지를 검사하는 overlapCircleArea 함수를 통해 본인 근처에 객체가 있는지 먼저 확인한 뒤 충돌 확률이 1에 가까워 즉시 충돌할 여지가 있는 대상에 대해 Raycast를 이용한 충돌 감지를 수행한다. 만약 충돌이 일어났다면 정해진 공식에 의해 각 충돌한 두 객체의 속도가 변한다. 만약 충돌 대상이 벽인 경우에는 다른 공식을 적용하여 충돌을 수행한다. 추가적으로 CollisionDetection과 그를 상속하는 객체는 부채꼴 형태의 범위를 지정하여 해당 범위 내에 객체가 있는지를 확인할 수 있는데, 현재 테스트 코드에는 플레이어 객체가 이를 수행하고 있다.  테스트 코드 실행 시 6개의 적 객체와 4개의 벽, 1개의 플레이어 객체가 존재하며 방향키 입력 시 플레이어가 움직이면서 속도 벡터 기준 양 옆 30도씩, 즉 전방 60도 범위 내에 객체가 존재하는지 지속적으로 확인하며, 화면에 존재하는 모든 객체들은 충돌을 지속 감지하면서 충돌 발생 시 속도에 변화를 주게되어 시간이 지날수록 화면은 혼돈에 빠지는 상태가 된다. |  |  | | --- | | **자체 평가** | | 사소한 버그는 당연히 존재했지만, 강의노트를 바탕으로 충돌 감지와 impulsive Force를 구현하는 과정에서 너무나도 많은 버그와 논리적 오류, 성능 저하를 겪을 수 있었고 이를 해결하기 위해 수많은 고민과 구글링을 겪으면서 새로운 관점을 획득할 수 있었다. 평소 유니티에서 가벼운 마음으로 충돌을 다루어 왔는데, 이번 기회를 통해 유니티가 얼마나 충돌에 대해 섬세하고 정확하게 개발을 했는지 어느정도 이해할 수 있었고, 이번 강의를 통해 획득한 지식과 프로그래밍적 사고를 이용해 완벽하지는 않으나 비슷한 기능을 직접 개발할 수 있어서 게임 개발자로서 한걸음 나아간 기분이 들었다. 최종적으로 구현하고자 했던 부분들이 정상적으로 동작하는 모습을 보며 뿌듯함과 성취감을 느꼈고 이 시간 이후로 조금 더 개발에 대한 자신감을 가지며 개인의 기술을 향상시키고 더 많은 지식을 얻고 싶다는 목적 의식을 가지게 되었다. | |