MonsterPooling.cs

1. 오브젝트 풀링: 오브젝트 풀링은 다음과 같은 부분에서 이점이 있다.
   1. 힙 메모리 효율성: Instantiate는 호출과정에서 힙에 메모리를 할당하는데, 이때 오브젝트 풀링을 통해 필요한 메모리를 미리 할당함으로서 낭비되는 메모리 공간을 줄일 수 있다.
   2. Garbage Collection 대상을 줄임: Garbage Collection은 더이상 사용하지 않는 객체를 자동으로 수거하여 메모리 공간에서 삭제하는 시스템이다. 이때 삭제의 기준은 대부분의 객체가 생성 후 사용되지 않고 거의 바로 제거될 가능성이 높다는 약한 세대 가설을 기반으로 하여, 객체를 세대별로 나누고 각 세대에 할당된 메모리가 꽉 차게 되면 해당 메모리 내에서 사용되지 않는 객체는 삭제하고 사용 중인 객체를 다음 세대의 메모리로 승격된다. 이때, 삭제는 동기식으로 작동되어 객체를 삭제하는 동안 게임은 멈추게 됨으로 가비지 컬렉터의 실행을 줄이는 것은 최적화에 있어 매우 중요하다. 따라서 오브젝트 풀링을 통해 오브젝트들을 미리 선언하게 되면, 해당 오브젝트들은 플레이어 눈에는 보이지 않더라도 게임의 보이지 않는 곳에서 존재하게 됨으로 0세대 메모리가 꽉차게 되더라도 삭제되지 않고 다음 세대로 승격되게 된다. 즉, 가비지 컬렉션이 풀링된 오브젝트들을 관리할 때는 승격할 때만 동작하게 됨으로 런타임 중지를 방지할 수 있다.
   3. 깊은 복사 방지: Instantiate는 단순 복사가 아니라 게임 오브젝트의 Transform, SpriteRenderer등의 모든 컴포넌트를 복사하여 완전히 새로운 객체를 만들게 된다. 이러한 깊은 복사 작업은 고비용 작업으로 많은 리소스를 필요로 함으로 오브젝트 풀링을 통해서 이러한 작업을 필요한 만큼 미리해두어 게임 실행 중에 드는 비용을 줄일 수 있다.
2. 사용기술: 유니티 ObjectPool
3. 문제상황 및 해결과정: GC(Garbage Collector)의 간헐적 개입, 동적으로 생성된 몬스터들의 복잡한 초기화 로직 반복으로 인한 렉🡪UnityEngine.Pool.ObjectPool<T> API를 학습 후 도입
4. 배운점
   1. ObjectPool<T>을 단순한 재사용 도구가 아닌, GC 발생 제어 등 최적화 수단의 필요성을 이해하고 이를 구현함
   2. 깊은 복사, 비동기식 처리 모델 등에 대한 이해를 확장하여 최적화의 필요성을 학습함
5. 성장한 부분: C# Garbage Collection에 대한 이해, C#생성자에 대한 이해, 깊은 복사에 대한 이해
6. 향후 보완할 점
   1. 현재 코드에서는 단순히 오브젝트를 활성화/비활성화만 하고 있다, 추후에는 이를 수정하여 활성화 시 함께 실행할 코드를 GetMonster()에 추가하여 코드를 더 효율적으로 만들 계획이다.