Unity에서의 메모리 단편화 사례 구현 및 해결 결과 보고서

C277030 임경화

1. 메모리 단편화(Memory Fragmentation)란

메모리 단편화란, RAM에서 메모리의 공간이 작은 조각으로 나뉘어져, 사용가능한 메모리가 충분히 존재하지만 할당이 불가능한 상태를 의미한다.

메모리 단편화는 내부 단편화와 외부 단편화로 구분된다.

- **내부 단편화(Internal Fragmentation)**는 메모리 할당 시, 프로세스가 필요한 양보다 더 큰 메모리를 할당하여 메모리 공간이 낭비되는 상황을 의미한다.
- **외부 단편화(External Fragmentation)**는 사용가능한 메모리의 총량은 충분하지만, 필요한 만큼의 연속된 공간이 존재하지 않아 실제로 할당이 불가능한 상황을 의미한다.

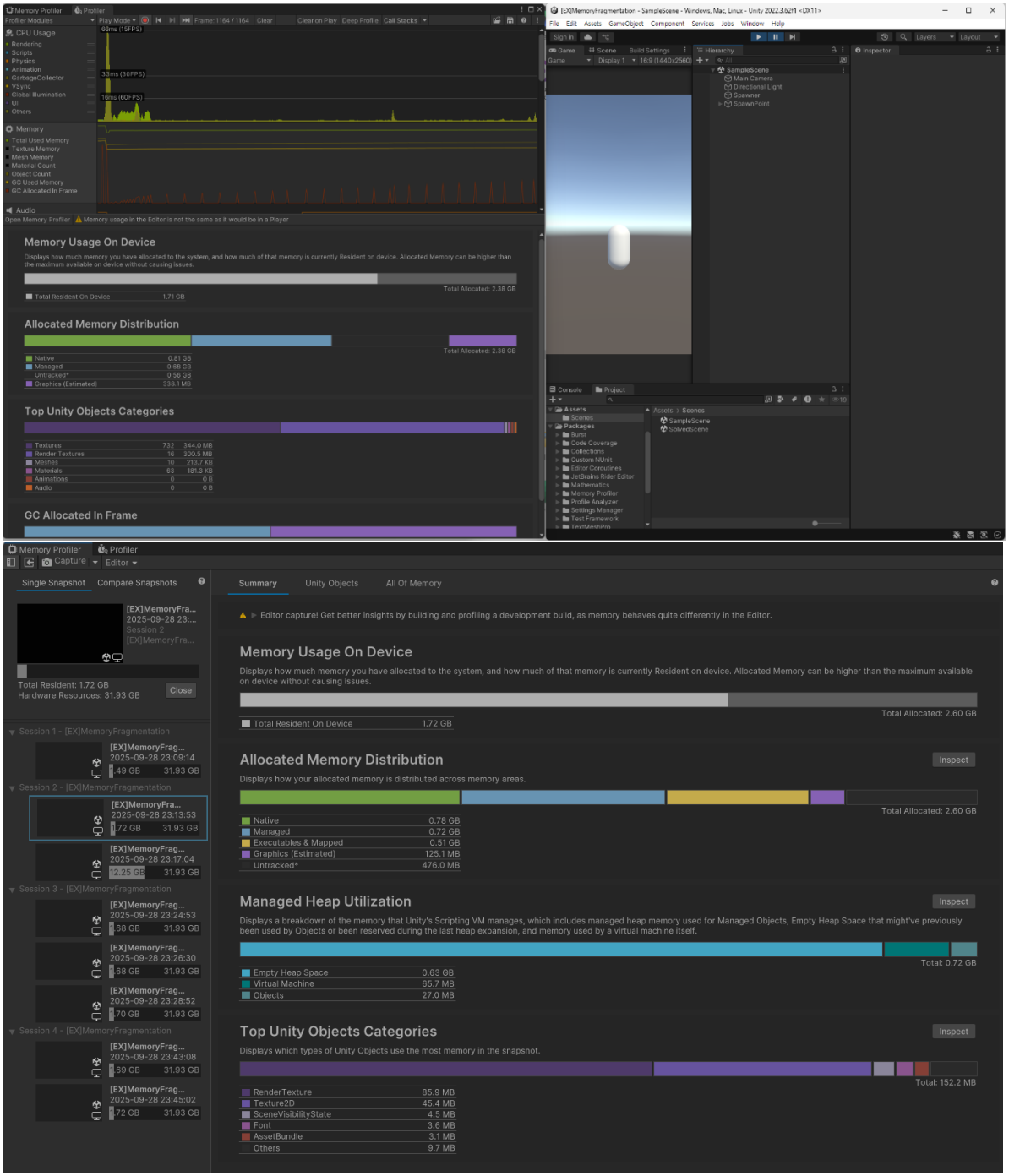
메모리 단편화를 해결하기 위해 다음과 같은 방법을 사용할 수 있다.

- **페이징(Paging)**은 보조 기억장치의 가상메모리를 고정된 크기의 블록(페이지)로 나누어, 프로세스에 연속적이지 않은 메모리를 할당하더라도 연속된 논리 주소 공간처럼 보이도록 하는 방식이다. 외부 단편화를 해결할 수 있지만 페이지가 프로세스에 필요한 크기보다 클 경우, 내부 단편화가 발생할 수 있다.
- **세그멘테이션(Segmentation)**은 프로그램을 논리 단위의 세그먼트로 나누어 메모리를 할당하는 기법으로, 각 세그먼트는 크기가 다르며 필요한 만큼만 할당된다. 따라서 가변 크기의 세그먼트로 인한 외부 단편화가 발생할 수 있다.
- **메모리풀(Memory Pool)**은 미리 큰 메모리 블록을 확보하여 필요한 객체나 데이터를 블록 내에서 재사용하는 기법이다.
- **압축(Compaction)**과 **통합(Coalescing)**은 분산된 메모리 공간을 하나로 합쳐 큰 메모리 공간을 확보하는 기법이다. 압축은 모든 분산된 메모리 공간이 합쳐지도록 재배치되지만, 통합은 인접한 공간끼리만 통합한다는 점에서 차이가 있다. 두 방식은 다른 기법에 비해 비용이 많이 든다는 단점이 있다.

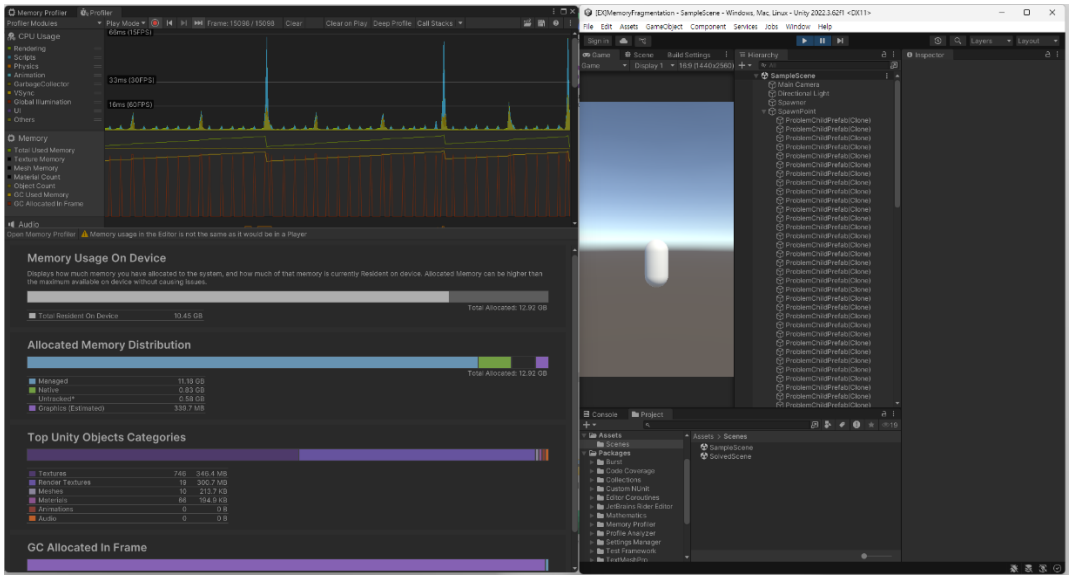
2. Unity에서의 메모리 단편화 사례 구현

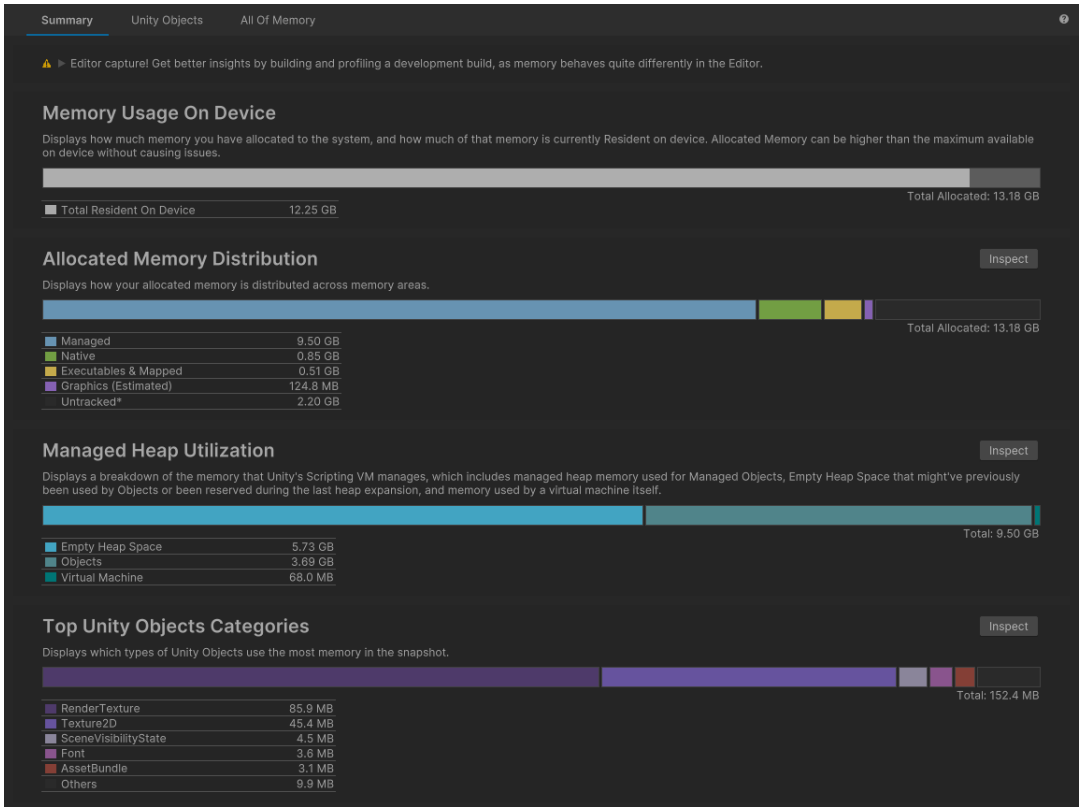
오브젝트 스포너는 0.01초마다 새로운 오브젝트를 생성한다. 생성한 오브젝트는 char 배열을 멤버변수로 가지는데, 오브젝트가 생성될 때마다 생성된 오브젝트의 배열의 크기는 1000씩 증가한다.

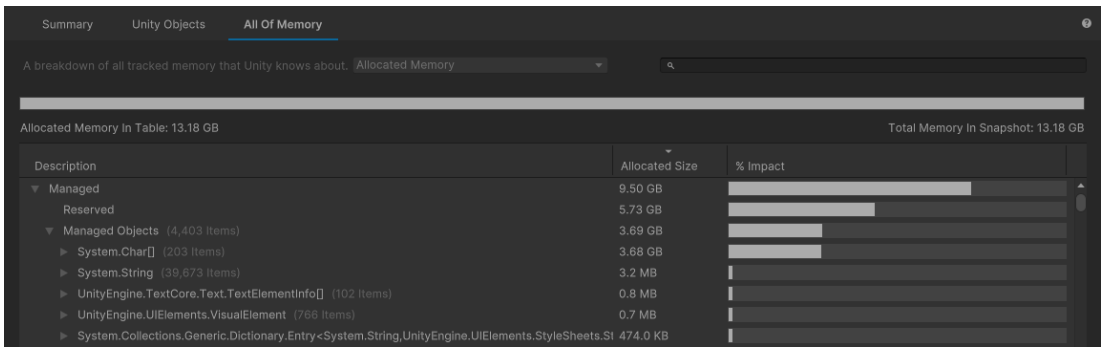
Github: https://github.com/lkh7764/-2025-2-PracticalProgrammingPractices



1000 프레임에서 메모리 프로파일링 결과







15000 프레임에서의 메모리 프로파일링 결과

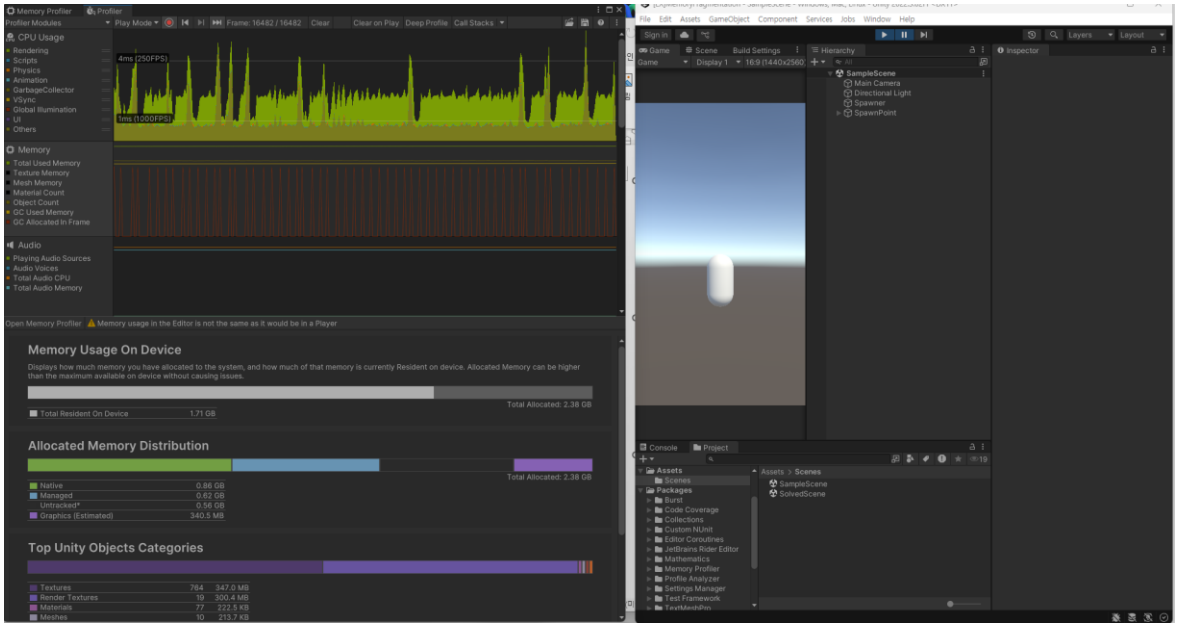
char 배열의 크기 증가로 메모리 단편화가 발생하였다.

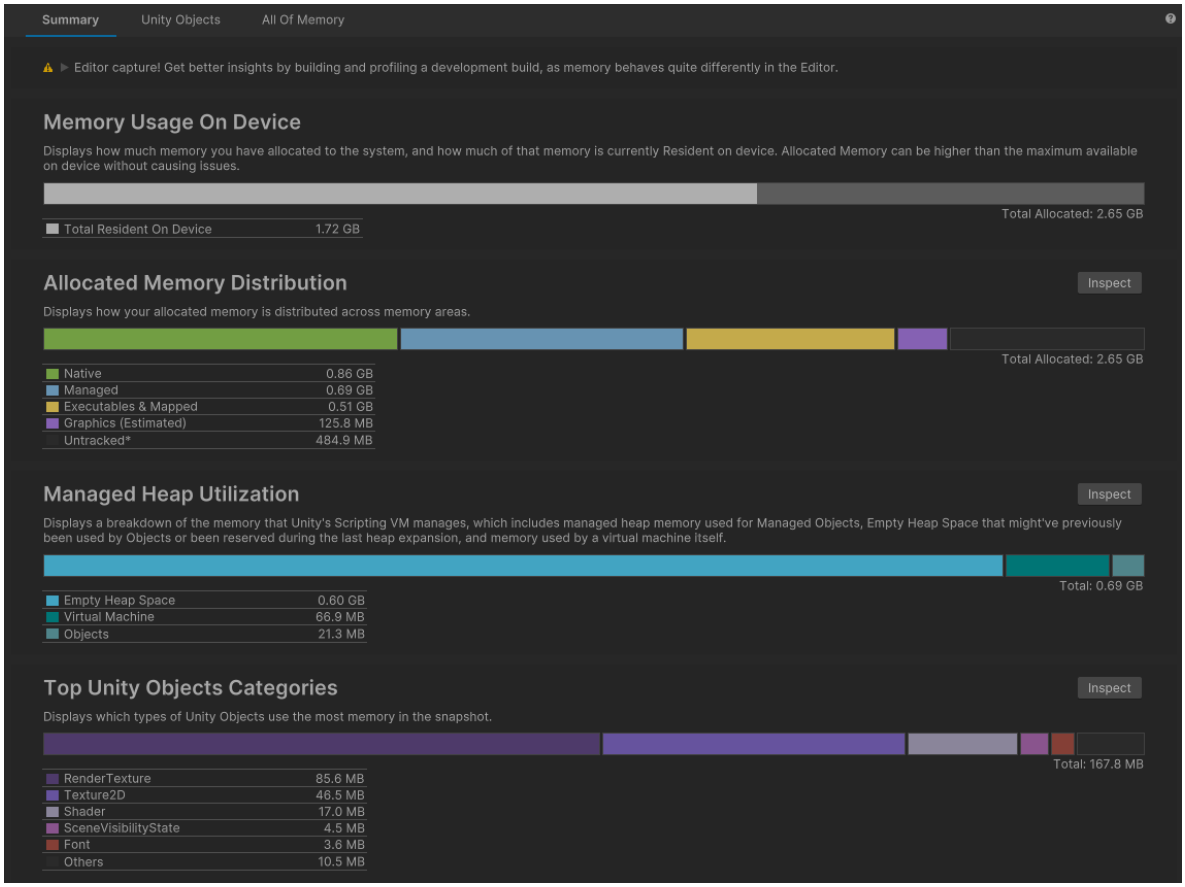
3. Unity에서의 메모리 단편화 해결

Unity는 OS/런타임 수준의 메모리 관리에 직접 접근할 수 없기 때문에 페이징, 세그멘테이션, 압축, 통합과 같은 방법으로 문제를 해결하기는 어렵다. 대신 Unity 환경에서는 오브젝트 풀링과 같은 메모리풀 기반 기법을 활용하여 최소한의 메모리만 할당하고, 이를 재사용하는 방식으로 메모리 사용을 최적화할 수 있었다.

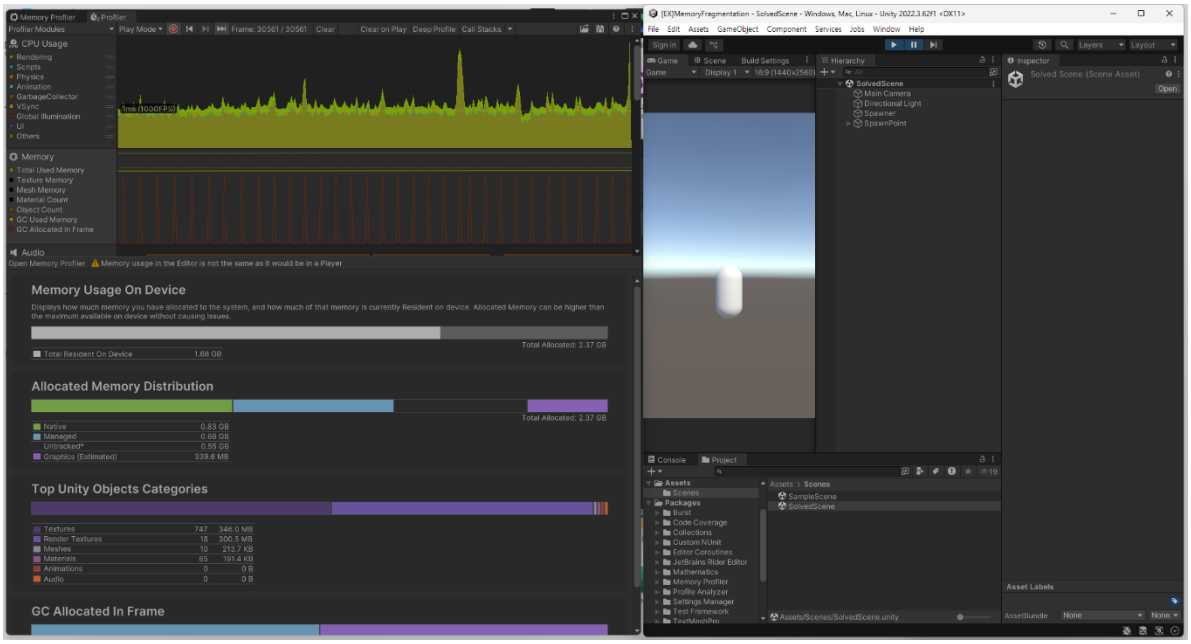
또한, 배열을 리스트로 변경하여 적용해본 결과, Unity 내부에서 자동으로 capacity 관리 및 확장이 이루어지며 GC Alloc 발생 빈도가 줄어들고, 메모리 파편화로 인한 불필요한 증가가 크게 감소하는 효과를 확인할 수 있었다.

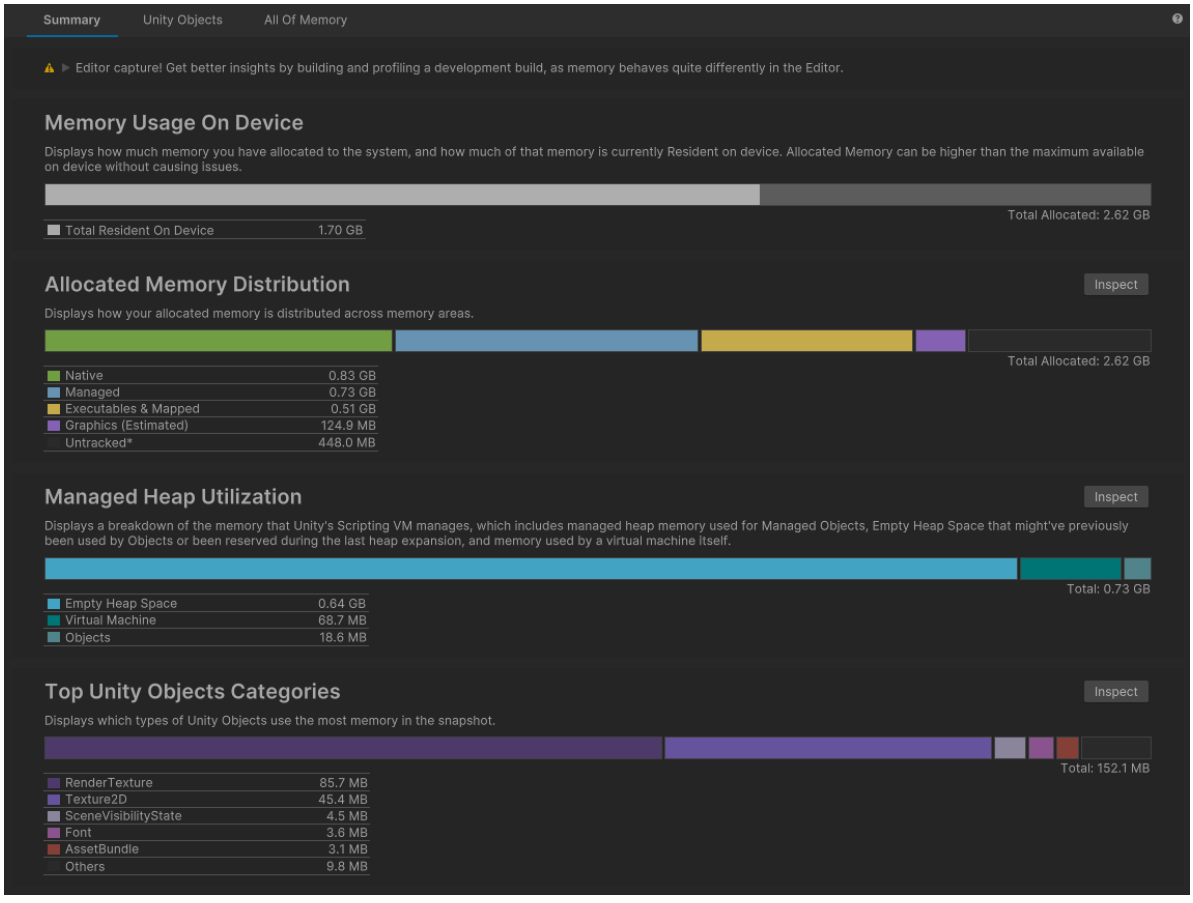
따라서 C# 및 Unity가 제공하는 자료형과 컨테이너를 적극 활용하고, 상황에 맞는 오브젝트 풀링과 같은 메모리 관리 기법을 적용하는 것이 효율적이다. 이러한 접근은 특히 대규모 프로젝트 및 메모리의 할당과 해제가 빈번한 프로젝트에서는 반드시 필요한 과정일 것이다.





배열을 리스트로 변경 후, 15000 프레임에서의 프로파일링 결과. 변경 전에 비해 최종 사용 메모리의 크기가 대폭 줄었다.





배열을 리스트로 변경하고 오브젝트 풀링을 적용한 후, 30000 프레임에서의 메모리 프로파일링 결과. 2.62GB로 가장 적은 최종 사용 메모리를 확인할 수 있다.