분석 문제: 분석한 내용을 직접 작성하고, 강의의 코드를 다시 한번 작성하며 복습해봅시다.

1. 별도의 UI 스크립트를 만드는 이유에 대해 객체지향적 관점에서 생각해보세요.
   * UI스크립트는 ‘보이는’ 부분을 담당하는 특성 상, 변화나 제거가 자주 발생합니다. 그렇기 때문에 UI 스크립트에서 실 처리를 담당하게 된다면 추후 간단한 UI 수정이나 기능 수정에도 대처해야 하는 코드의 양이 많아질 가능성이 생깁니다.  
     특히 해당 UI는 제거하지만 그 UI에서 담당하던 기능은 필요한 경우, 잘못하여 해당 UI를 지워버린 경우에는 해당 기능을 다시 작성해야 합니다.  
     때문에, UI는 기능 영역에 혼선이 발생하지 않는 범위 안에서 가능한 한 많은 종류로 나눠서 작성하는 것이 좋습니다.
2. 인터페이스의 특징에 대해 정리해보고 구현된 로직을 분석해보세요.
   * 인터페이스는 ‘공통된 특징’을 명시하기 위해서 사용하는 요소입니다.  
     동시에 같은 인터페이스를 상속받은 클래스라면 인터페이스에서 명시된 함수에 한하여 같은 코드로 호출할 수 있게 됩니다.  
     이러한 방식을 통해서 코드의 유연성을 크게 증진시킬 수 있습니다.
   * 본 프로그램에 작성된 interface는 IDamageable과 IInteractable의 2종류가 있습니다.
     + IDamageable: ‘피격 시 판정’을 담당하는 인터페이스입니다.
       - 피격 시 처리에 대한 함수인 TakePhysicalDamage(int) 함수 하나 만 가지고 있습니다.
     + IInteratable: 상호작용 가능한 물체에 추가되는 인터페이스입니다.
       - GetInteractPrompt 함수는 해당 객체의 내부에 존재하는 이름 및 설명을 외부 클래스에서 받아올 수 있도록 만들어주는 함수입니다.
       - OnInteract 함수는 상호작용 시 발생하는 효과에 대한 함수입니다.
3. 핵심 로직을 분석해보세요. (UI 스크립트 구조, Campfire, DamageIndicator)
   * UI스크립트 구조
     + UI를 관리하는 스크립트가 해당 기능의 스크립트에 포함되어 있거나, 해당 기능의 UI 스크립트에 기능 구현이 포함되어 있는 경향이 있습니다.
     + 대표적으로 인벤토리는 UIInventory 스크립트에서 아이템 습득 및 사용, 버리기 등의 기능이 구현되어 있고, ItemSlot 스크립트에서 해당 슬롯의 아이템 정보를 지니고 있으면서도 아이템 슬롯에 따른 UI 변동 함수를 지니고 있는 구성입니다.
     + 반면에, 각 기능 별로 확실히 나뉘어져 있어, 기능 상의 의존관계 이외에는 별다른 의존관계가 발생하지는 않고 있습니다.
   * Campfire
     + Collider의 trigger 범위 안에 들어온 객체를 리스트에 저장하고 있다가 일정 주기로 저장된 객체들에게 피해를 주는 방식입니다.
     + Trigger 범위에서 나갈 경우 해당 정보를 리스트에서 빼서 더 이상 campfire에 의한 피해를 입지 않도록 조치합니다.
     + 데미지를 주는 방식은 InvokeRepeating을 이용한 반복 유발으로 구성되어 있습니다.
   * DamageIndicator
     + 플레이어 캐릭터가 공격을 받으면 화면이 빨개졌다가 되돌아오는 구조입니다.
     + Coroutine을 이용해서 시간 경과에 따른 변화를 구현하였습니다.
     + Condition의 onTakeDamage 이벤트를 통해서 피격 관련 기능들을 한 번에 불러오는 구조입니다.