1주차 프로토타입

제목 프로메테우스 프로젝트
분류 시뮬레이션
작성자 김호곤

가. 진행 상황

1. 지구 기준 물리량에서 평형을 유지한다.

   물리량 계산은 PlayManager 클래스의 Update 함수에서 한다. 물리 공식은 계획서와 거의 유사하고, 수정된 공식은 아래 변경 사항에 있다. 지구 기준의 물리량을 입력하면 평형을 유지하나 의도했던 수치와는 약간의 오차가 있다.

2. Language 클래스에서 언어를 관리한다.

   여러 언어를 쓸 것을 생각해서 Language는 Resources 폴더의 json 파일로부터 읽어온 언어를 제공한다. 또한 언어에 따라 폰트가 달라져야 하므로 Resources 폴더에 폰트 파일도 들어있어서 언어 변경 시 폰트도 같이 제공한다. Language 클래스에 대리자도 생성되어 있어서 언어 변경 시 동작할 것은 모두 대리자에 등록되어있다. 대리자 덕분에 같은 타이밍에 모든 동작을 할 수 있다.

3. json 파일 생성은 JsonCreator 클래스가 한다.

   Language 클래스는 MonoBehaviour를 상속받지 않기 때문에 json 파일을 생성하는 함수를 누군가가 호출해줘야 한다. json 파일 생성은 기본적으로 에디터에서만 수행할 것이기 때문에, JsonCreator 컴포넌트가 부착된 prefab을 만들어 필요할때마다 계층 구조 창으로 끄집어낸다. JsonCreator 컴포넌트는 Editor 스크립트와 연결되어있어 버튼으로 간단하게 json 파일을 생성할 수 있다. 또한 Editor 스크립트에 번역본이 제대로 적용이 되는지 확인하는 기능을 추가했다.

4. AutoTranslate 컴포넌트를 만들어 UI 텍스트에 자동으로 번역을 적용한다.

   모든 텍스트를 찾아 번역을 시도하면, 단순히 숫자만 표시하는 텍스트처럼 번역이 불필요한 텍스트까지 번역을 적용하려 할 것이다. 따라서 번역이 필요한 텍스트에만 AutoTranslate 컴포넌트를 붙여서 AutoTranslate가 언어 변경 시 동작을 대리자에 등록한다. 번역이 불필요한 텍스트의 경우, 폰트를 통일 시키기 위해서, 폰트만 변경할 것인지 bool 체크를 한다. bool 체크가 되어 있으면 번역 없이 폰트만 변경한다.

5. UIString 클래스는 공용으로 사용할 문자열을 관리한다.

   문자열 생성을 최소한으로 하기 위해 UIString 클래스를 만들었다. UIString은 누군가가 문자열을 참조하고 싶을 때 문자열을 생성하고, 문자열 생성 시의 변수값을 기억하고 있다가 다음 누군가가 참조하고 싶을 때 변수값이 변경되지 않았으면 문자열을 새로 생성하지 않고 원래의 문자열을 반환한다.

6. Language, UIString, Playmanager 클래스는 인덱서를 만들었다.

위 세 클래스를 참조할 때 가장 많이 하는 것은 어떤 배열의 인덱스 접근이다. 따라서 위 세 클래스에 인덱서라는 것을 만들어, 마치 인스턴스에서 직접 인덱스에 접근하는 것처럼 만들었다. Language는 해쉬테이블처럼 한국어 문자열을 인덱스처럼 입력하면 번역된 값이 반환된다. Playmanager는 반환할 값의 타입에 따라 인덱서를 오버로드했고, 원하는 인덱스는 열거형으로 입력한다. UIString도 Playmanager와 동일한 값을 입력해서 원하는 값의 문자열을 반환한다.

7. 메뉴 화면에서 상태 패턴 적용을 시도했다.

부모 클래스는 IState 인터페이스를 상속받고 IState에는 Execute, ChangeState 함수가 있다. GeneralMenuButtons 클래스에서는 부모 형식의 클래스를 참조하고 Execute, ChangeState만 호출한다. 그러나 화면 이동 조건은 결국 GeneralMenuButtons가 제공하는 데다가 참조 변수의 개수를 줄이기 위해 자식 객체의 참조도 같은 클래스에 몰아넣은 바람에 상태 패턴의 의미가 사라졌다. 게다가 Execute, ChangeState 함수도 모든 객체가 같은 동작을 하게 돼서, 부모 클래스에 정의하면 자식 클래스는 오버라이드할 필요가 없다. 현재 메뉴 화면의 상태 패턴은 형식적으로만 존재한다.

8. AnimationManager 클래스에서 모든 애니메이션을 관리한다.

이 게임의 애니메이션은 게임 플레이에 영향을 주지 않는다. 그러므로 모든 애니메이션을 한 클래스에 넣고 [SerializeField]를 통해 Unity 인스펙터 창에서 관련 값을 조절할 수 있게 하는 것이 관리가 훨씬 더 편하다.

9. 탐사 기능을 추가했다.

탐사를 진행하면 일정 시간 후에 토지를 찾아낸다. 토지를 찾으면 토지 목록에 토지 슬롯이 추가되는데, 토지 슬롯은 prefab으로 만들어져있고 LandSlot이라는 컴포넌트가 붙어있다. 슬롯을 생성하는 쪽은 LandSlot에 접근해서 슬롯의 이름 등을 정할 수 있다.

10. 소리 재생은 AudioManager 클래스가 한다.

소리 재생 중에 씬을 이동하면 소리가 끊어진다. 따라서 소리를 재생할 오브젝트는 씬을 이동해도 파괴되지 않는 것이 좋다. AudioManager는 싱글턴패턴을 적용하고 DontDestroyOnLoad도 적용한다. 에디터에서 씬에 AudioManager를 미리 생성해놓으면 생성되어있는 씬으로 이동할 때마다 중복된 AudioManager를 파괴하게 되는데, 이를 방지하기 위해 AudioManager를 참조하는 정적변수가 null일 때만 AudioManager를 동적 생성하게 했다. 씬 이동 시 AudioManager 생성, 파괴를 반복하지 않아도 된다.

11. 소리 채널을 여러 개 생성했다.

여러 소리를 동시에 재생할 것을 대비해 소리 채널을 여러 개 만들고, 소리를 재생할 때마다 한 채널씩 소리를 할당해서 재생한다. 채널 개수는 인스펙터 창에서 수정 가능하다.

12. 상수는 Constants 클래스에 담았다.

공통으로 사용할 상수거나 중요한 상수는 Constants라는 정적 클래스에 담았다. 수정이 필요하면 Constants에서 찾아서 수정하면 되고, 정적 클래스기 때문에 인스턴스 생성이 필요 없다.

나. 변경 사항

1. 생물 안정도 공식을 수정했다.

계획서에있는 광합성, 호흡 생물 안정도 공식은 잘못됐다. 대기압, 온도 등에 따른 안정도를 0에서 1사이로 먼저 계산 후 액체 상태 물의 양에 따라 안정도 값을 조정하는 것으로 했다.

2. Language 클래스에서 생성한 해쉬테이블에 한국어로 된 키를 입력하면 값으로 배열의 인덱스가 반환된다.

json 파일에서 언어를 읽어오면 배열의 형태로 저장되는데 이를 다시 해쉬테이블에 저장하는 것은 별로 효율적으로 보이지 않는다. 따라서 언어는 배열 상태 그대로 두고, 해쉬테이블은 한국어 '키'와 배열의 인덱스 '값'을 담도록 한다. 그러면 한국어 '키'로 검색했을 때 인덱스 '값'이 반환되고 그 인덱스로 원하는 배열의 요소로 접근할 수 있다.

3. PlayManager는 JsonData의 배열을 프로퍼티로 일일이 구분해주지 않는다.

계획서에서는 JsonData의 배열로 존재하는 데이터를 PlayManager가 프로퍼티로 구분해주는 것으로 되어있다. 그러나 스크립트에서 프로퍼티를 추가, 삭제할 때마다 인덱스를 구분해야되고 프로퍼티의 개수만큼 JsonData의 배열 공간을 확장해야 된다는 불편함이 있다. 따라서 프로퍼티를 통한 인덱스 구분이 아니라 열거형을 통한 인덱스 구분을 선택했다. 원하는 열거형의 요소를 입력하면 거기에 해당하는 배열 인덱스로 접근할 수 있어서 접근을 위한 함수 하나면 배열 하나를 모두 접근할 수 있고, 필요한 변수를 추가, 삭제할 때로 열거형만 수정하면 된다. JsonData의 배열 공간은 열거형의 제일 마지막 End라는 이름의 요소를 가져와서 배열의 길이를 정할 수 있다.

4. 메뉴 화면은 MonoBehaviour를 상속받는다.

메뉴 화면은 UI의 동작을 수행하기 때문에 Unity UI와 연결하기 위해 Mono Behaviour를 상속받는다. 그로 인해 메뉴 화면의 상태 패턴에서 상태 변경은 해당 오브젝트의 활성화, 비활성화로 이루어진다.

5. 계획 상에 있던 IInitiateForSingleton 인터페이스는 제거했다.

계획서에서는 MonoBehaviour를 상속받는 클래스는 Awake 함수에서 초기화할 수 있다는 점 때문에 MonoBehaviour를 상속받지 않는 클래스는 초기화 함수를 따로 만들어 누군가가 호출해줘야 하는 것으로 생각했다. 그러나 MonoBehaviour를 상속받지 않는 클래스는 생성자를 정의할 수 있기 때문에 IInitiateForSingleton는 필요하지 않다.