**2023.2 Object Oriented Programming, Project #4**

**Snake Like Game Implementation**

**Class 02 [presentation on Dec. 11 (11am)], Team #8**

|  |  |  |
| --- | --- | --- |
| 20184256 | 박성민 | team leader |
| 20226041 | 김규리 |  |
| 20170530 | 장기봉 |  |
| 20220127 | 유선기 | presentation speaker |

**Table of Contents**

1. Briefing Project
2. How to Compile and Execution
3. Description on Functionality
4. Important Implementation Issues
5. UML Diagram (SW System Design)
6. Execution Results
7. OOP Concept Description
8. Conclusion

**Briefing Project**

In this project, we aim to develop a project reflecting object-oriented concepts, exploring and learning concepts such as abstraction, encapsulation, and polymorphism. Our team chose to develop a game similar to the 'snake game' using the Unity framework. Through this, we sought to gain experience with the features of the object-oriented language C# and explore its application in utilizing object-oriented principles.

**How to Compile and Execution**

[ 작성 필요 ]

**Description on Functionality**

게임 로직

1. 시작

게임을 시작하면 GameScene으로 넘어가면서 오브젝트에 붙어있는 Script들이 init 된다. MonoBehaviour 클래스를 상속하여 작성된 클래스들이 Unity Gameobject 들에 추가되어 있으면, overriding된 onEnable()와 Start() 함수들을 호출해 실행해준다.

OnEnable과 Start 함수를 실행하면,   
1. HeadMove, TailMove 에서 초기 변수 값들을 세팅해준다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

2. ItemManager에서 아이템을 필드에 생성한다.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

3. EventManager, ScoreManager와 UIManager의 데이터를 초기화하여 점수를 초기화한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

2. 움직임

- Update 함수 호출되어 이동

- 스페이스바 누르면 이동 방향 전환

게임 캐릭터의 HeadMove 스크립트와 Tail의 TailMove 스크립트의 Update 함수를 호출해 실행하면서 게임 오브젝트들이 움직이게 된다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

HeadMove 클래스를 가지고 있는 게임오브젝트는 Move를 통해 원을 그리며 이동하게 된다. 스페이스바를 입력으로 받으면 진행 방향을 반대로 바꾸며 새로운 이동 방향을 계산해 이동한다.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

TailMove 함수에서 캐릭터 뒤에 있는 Head가 이동하는 위치에 따라 Tail 에 있는 오브젝트들의 위치를 지정해준다.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

TailMove 클래스를 상속받은 꼬리 오브젝트들은 매 프레임 실행되는 Update 함수에서 Move 함수를 호출해, 다음 위치로 이동하게 된다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

3. 충돌 시

캐릭터 플레이어가 다른 오브젝트와 충돌하면 다음의 로직이 실행된다.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

만약 충돌된 오브젝트 대상이 Tail이나 Wall인 경우에는 EventManager의 게임 종료 함수를 호출한다.

만약 Item 게임오브젝트와 충돌한 경우에는 충돌한 Item에 해당하는 오브젝트를 꼬리에 추가하고 충돌한 아이템 오브젝트를 제거하며, 새로운 아이템을 다시 필드에 생성한다.

4. 종료

게임이 종료되면 EventManager를 통해 게임 종료 로직이 호출되어, 게임 종료와 관련된 UI 변경이 호출되며 사용자에게 점수와 최고 점수를 보여주게 된다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

**Important Implementation Issues**

**1. PointInfo**

**텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명**

PointInfo 클래스의 constructor를 overloading 하여, 2가지 시그니처를 만들어주었다. 인자로 전달받는 데이터가 없는 경우에는 값을 default로 초기화해주고, 데이터를 전달한 경우에는 해당 값으로 프로퍼티를 초기화해 PointInfo의 instance를 생성한다.

**2. HeadMove.cs**

****

HeadMove 클래스는 UnityEngine의 기본 클래스로, 유니티 엔진에서 Script로 사용하기 위해 상속받아야하는 클래스이다. MonoBehaviour 내부에는 유니티에서 지원하는 각종 메서드와 프로퍼티가 포함되어있다.

컴퓨터 환경과 별개로 동일한 오브젝트의 이동속도를 적용하기 위해 Update 함수 내에서 deltaTime을 통해 매 프레임에 걸리는 시간을 적용해 이동 거리를 계산해주었음. 이 함수는 MonoBehaviour가 가진 Update를 overriding한 함수로, 매 프레임마다 유니티 엔진이 호출하여 오브젝트의 이동을 처리할 수 있다.

스크린샷, 텍스트, 폰트이(가) 표시된 사진

자동 생성된 설명

매 프레임 Update 함수에 의해 Move가 호출되어 플레이어 캐릭터의 위치를 이동시킨다. 매 프레임마다 이동 방향이 변경되며, 키보드 입력을 통해 이동 중심과 회전 방향을 전환할 수 있다.

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

TailMove 함수를 호출하여, 플레이어 캐릭터의 이동 방향과 방향 전환에 대해 뒤따라오는 Tail 게임 오브젝트들에게 이를 전달해 알린다. Tail 오브젝트들은 이에 따라 위치 정보를 저장하는 PointInfo type을 수정해 다음 이동 위치를 결정할 수 있다.

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

꼬리 뒤에 Tail 게임오브젝트를 추가하기 위해 AddTail 함수를 사용한다. 꼬리가 원하는 위치에 생성되어 기존의 오브젝트들을 따라오도록 만들었다.



**3. TailMove.cs**



TailMove 클래스는 위에서 소개한 HeadMove 클래스를 상속하여 구현하였다. 이를 통해 HeadMove가 가지고 있는 이동과 관련된 여러 프로퍼티들을 다시 작성할 필요 없이 재사용하여 적용할 수 있다. Head와 다르게 동작해야 하는 Tail은 HeadMove의 이동 관련 메서드를 override 하여 동작을 수행하도록 설계하였다.

Tail 게임오브젝트도 Head 오브젝트와 동일하게 deltaTime을 사용해 CPU 성능에 상관없이 동일한 거리를 이동할 수 있도록 하였다.

스크린샷, 블랙, 텍스트이(가) 표시된 사진

자동 생성된 설명

SetInfo 함수는 Method Overloading을 통해 2가지의 시그니처를 가지고 있어, 필요시에 인자를 다르게 전달해 필요한 로직을 처리할 수 있게 설계하였다.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

**4. CollisionController.cs**

UnityEngine의 MonoBehaviour 클래스를 상속받고, 해당 클래스 내의 OnTriggerEnter2D 메서드를 Overload 하여 다른 오브젝트와 충돌 시 수행할 로직을 호출한다. 이 함수에서는 충돌한 게임 오브젝트가 가지고 있는 tag값에 따라 다른 로직을 수행하도록 구현하였다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

**4. ItemInfoHolder & ItemManager**

ItemInfoHolder 클래스에서는getter setter를 구현해, getTail 함수로 변수에 접근할 때 원하는 GameObject를 반환하도록 구현하였다. 내부 프로퍼티인 tails에는 Unity의 모든 객체의 조상 class인 GameObject 타입의 배열로 선언해, tail로 사용할 수 있는 GameObject를 상속하는 객체들을 담아두고 반환해주도록 설계하였다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

ItemManager 에서는 게임 보드 위 무작위 위치에 아이템을 생성하는 로직을 수행한다. 현재 플레이어와의 위치 거리를 기준으로 생성 시 위치를 조정하도록 로직을 설계하였다.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

**UML Diagram (SW System Design)**

**1. Class Diagram**

**도표, 평면도, 기술 도면, 개략도이(가) 표시된 사진

자동 생성된 설명**

**2. Usecase Diagram**

**도표, 그림, 원, 스케치이(가) 표시된 사진

자동 생성된 설명**

**3. Activity Diagram**

**도표, 라인, 그래프, 스크린샷이(가) 표시된 사진

자동 생성된 설명**

**Execution Results**

To ensure the correctness of operations between all signs, the program was tested to check all cases.

**OOP Concept Description**

| **OOP Concept** | **Explanation** |
| --- | --- |
| **Encapsulation** | 스크립트 클래스에서 외부에서 활용해야 하는 요소들은 Public으로, 내부에서만 접근 가능해야 하는 요소들은 Private, Protected로 지정하여 사용하였다. C#의 기본 접근자는 private으로 되어있어, 명시되어 있지 않은 변수와 함수는 private으로 적용되어있다. |
| **Inheritance** | 게임 내에서 사용되는 모든 오브젝트와 관련된 클래스는 유니티 엔진의 기본 오브젝트 클래스인 GameObject를 상속받아 구현된다.  게임오브젝트에 Composition 관계로 추가될 수 있는 스크립트 클래스들은 MonoBehaviour 클래스를 상속받아 구현되며, 유니티 프레임워크에서 스크립트 클래스에 overriding을 통해 구현된 OnStart, Update를 비롯한 다양한 메서드를 호출해 게임을 실행시킨다. |
| **Polymorphism** | 본 프로젝트에서는 Overloading과 Overriding이 다수 사용되었다.  기본적으로 모든 스크립트와 관련된 클래스가 MonoBehaviour 클래스를 상속하여 만들어지고, 유니티 프레임워크에서는 MonoBehaviour 에 작성되어있는 메서드들을 호출해준다. 여기에서 OnStart, Update 등에 필요한 내용들을 override하여 작성해 사용할 수 있다.  TailMove 클래스가 HeadMove 클래스를 상속받아 구현되면서, HeadMove의 프로퍼티를 그대로 사용할 수 있다. TailMove에서 overriding을 통해 Update, Move함수를 수정해 사용할 수 있다.  TailMove 클래스 내에서는 SetInfo 함수를 overloading을 통해 2가지의 메서드 시그니처로 작성해, 인자에 따라 다르게 함수 로직이 다르게 실행되도록 만들 수 있다. |
| **Composition** | Unity의 게임 오브젝트에 대부분의 스크립트 클래스와 Collider 등의 특성은 Composition을 통해 오브젝트에 추가해 사용할 수 있다. |

**Conclusion**

게임 개발을 통해 그래픽 요소를 다루는 경험을 쌓으면서, 화려하고 시각적으로 매력적인 게임 화면을 창조하는 과정에서 큰 즐거움을 느꼈습니다. 이를 통해 디자인과 개발의 조화로 게임의 퀄리티를 향상시키는 중요성을 몸소 체험했습니다.

유니티를 이용한 게임 개발에서 객체지향프로그래밍의 핵심 개념들을 적극적으로 활용했습니다. Composition을 통한 유연한 객체 구성과 Polymorphism을 활용한 다형성 구현은 코드의 유지보수성을 높이고, 모듈화된 디자인을 통해 프로젝트의 확장성을 강화하는 데에 큰 도움이 되었습니다.

팀 프로젝트에서 객체지향프로그래밍의 컨셉들을 적극적으로 활용한 경험이 매우 유익했습니다. 클래스와 객체를 효과적으로 활용하여 모듈화된 코드를 작성함으로써, 팀원들 간의 협업이 원활하게 진행되었습니다. 상속과 다형성을 적용하여 각자 맡은 부분을 독립적으로 개발하면서도 전반적인 시스템이 일관성 있게 유지되어 프로젝트의 효율성과 품질을 향상시킬 수 있었습니다.