**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**

1. Game Start

When the game starts and transitions to the GameScene, the scripts attached to the objects are initialized. Classes that inherit from the MonoBehaviour class and are added to Unity GameObjects have their overridden OnEnable() and Start() functions called, triggering the following:  
1. In the HeadMove and TailMove scripts, initialize the variable values.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

2. Create Items on the field in ItemManager Script

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

3. Initialize Score by initializing EventManager, ScoreManager and UIManager.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

2. Player Movement

Calling the Update function of the HeadMove script for the game character's head and the TailMove script for the tail initiates the movement of the game objects.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

The game object with the HeadMove class moves in a circular pattern through the Move function. Upon receiving input from the spacebar, it changes the direction of movement to the opposite direction and calculates the new movement direction for continued motion.텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

In the TailMove function, the positions of objects in the Tails are determined based on the movement of the Head located forward them.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

The tail objects with the TailMove class invoke the Move function in the Update function, which runs every frame, to move to the next position.텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

3. On Collision

When the player character collides with another object, the following logic is executed.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

If the collided object is either Tail or Wall, the GameOver function of the EventManager is called.

If the collision is with an Item object, the corresponding object associated with the collided Item is added to the tail, the collided Item object is removed, and a new item is respawned on the field.

4. GameOver

When the game is over, the GameOver logic is invoked through the EventManager, triggering UI changes related to game termination. It displays the user's score and the highest score achieved.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

**Important Implementation Issues**

**1. PointInfo**

**텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명**

We created two different signatures for the constructor of the PointInfo class by overloading it. In one signature, when no data is passed as an argument, the values are initialized with defaults. In the other signature, when data is provided, the properties are initialized with those values to create an instance of PointInfo.

**2. HeadMove.cs**

****

The HeadMove class is a fundamental class in UnityEngine that needs to be inherited to be used as a script in the Unity engine. MonoBehaviour contains various methods and properties supported by Unity.

To apply the same movement speed for an object regardless of the computer environment, deltaTime was utilized within the Update function to calculate the movement distance based on the time taken per frame. This function is an override of the Update function in MonoBehaviour, and Unity engine calls it every frame to handle the object's movement.

스크린샷, 텍스트, 폰트이(가) 표시된 사진

자동 생성된 설명

Every frame, the Move function is called by the Update function, causing the player character's position to change. The movement direction changes every frame, and it's possible to switch between the movement center and rotation direction through keyboard input.

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

The TailMove function is called to notify the trailing Tail game objects about the player character's movement direction and direction changes. The Tail objects, based on this information, modify the PointInfo type that stores positional data, enabling them to determine the next movement position.

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

자동 생성된 설명

To add Tail game objects behind the tail, the AddTail function is used. The Tail is created at the desired location, allowing it to follow the existing objects.



**3. TailMove.cs**



The TailMove class inherits from the previously introduced HeadMove class. This allows for the reuse of various properties related to movement that HeadMove possesses, without the need to rewrite them. Unlike the Head, Tail is designed to perform its own actions by overriding the movement-related methods of HeadMove.

Like the Head object, the Tail game object also uses deltaTime to ensure it moves the same distance regardless of CPU performance.

스크린샷, 블랙, 텍스트이(가) 표시된 사진

자동 생성된 설명

The SetInfo function employs Method Overloading, providing two different signatures. This design allows for the flexibility of passing different arguments to handle the necessary logic as needed.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

**4. CollisionController.cs**

Inheriting from the MonoBehaviour class in UnityEngine, and overloading the OnTriggerEnter2D method, I call the logic to be executed upon collision with other objects. In this function, different logic is performed based on the tag value of the colliding game object.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

**4. ItemInfoHolder & ItemManager**

In the ItemInfoHolder class, I implemented getter and setter methods. The getTail function is designed to return the desired GameObject when accessing the variable. The internal property, tails, is declared as an array of GameObject type, which is the ancestor class of all Unity objects. It holds and returns objects that inherit from GameObject and can be used as tails.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

In the ItemManager, there is a logic for randomly generating items at random positions on the game board. The logic is designed to adjust the position of the generated item based on the current distance from the player.

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** | The script classes utilize public access modifiers for elements that need to be accessed externally, private for elements that should only be accessed internally, and protected for elements that require access within the inheritance hierarchy. In C#, the default access modifier is private, so variables and functions without explicit modifiers are automatically set to private. |
| **Inheritance** | All classes related to objects used in the game inherit from the basic Unity engine object class, GameObject. Script classes that can be added to GameObjects through a composition relationship inherit from the MonoBehaviour class and are implemented using overriding in Unity's framework. The framework calls various methods such as OnStart and Update, implemented in these script classes, to execute the game. |
| **Polymorphism** | In this project, Overloading and Overriding were extensively utilized. All classes related to scripts inherit from the MonoBehaviour class by default. The Unity framework then calls methods defined in MonoBehaviour, such as OnStart and Update. Users can override these methods and provide their own implementations. For instance, TailMove class inherits from HeadMove, enabling the direct use of properties from HeadMove. By overriding methods like Update and Move in TailMove, developers can modify and utilize them to suit the specific functionality of TailMove.  The TailMove class also implements the SetInfo function with two different method signatures using overloading. This allows for different logic execution based on the provided arguments. |
| **Composition** | Most of Unity's game object features, such as script classes and colliders, can be added to objects through composition. |

**Conclusion**

Through game development, I gained significant enjoyment in handling graphic elements, creating visually appealing game screens. This experience emphasized the importance of the synergy between design and development in enhancing the quality of games.

In Unity game development, I actively applied core concepts of object-oriented programming. Utilizing Composition for flexible object composition and implementing Polymorphism enhanced code maintainability and strengthened project scalability through modular design.

The team project provided a valuable opportunity to apply object-oriented programming concepts effectively. Writing modular code using classes and objects facilitated smooth collaboration among team members. Applying inheritance and polymorphism allowed us to independently develop our assigned components while maintaining overall system consistency, improving project efficiency, and quality.