2주차 정리

개요
C# 프로그래밍 기초 학습
자료형
변수
제어문
스크립트 작성과 컴포넌트 연결 방법 학습
스크립트 작성
컴포넌트 연결
이벤트 처리와 변수 활용 실습

개요

▸C# 프로그래밍 기초 학습

▸스크립트 작성과 컴포넌트 연결 방법 학습

▸이벤트 처리와 변수 활용 실습

C# 프로그래밍 기초 학습

▼ 자료형

1. **기본 자료형 (값 형식): 기본 자료형은 스택 메모리에 데이터를 저장합니다.**

정수형:

- `int` **: 32비트 정수. 범위는 -2,147,483,648 ~ 2,147,483,647.**
- `long` **: 64비트 정수. 범위는 -9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807.**
- `short` **: 16비트 정수. 범위는 -32,768 ~ 32,767.**
- `byte` **: 8비트 정수. 범위는 0 ~ 255.**
- `sbyte` **: 8비트 정수. 범위는 -128 ~ 127 (부호가 있는 자료형).부동 소수점형:**
- `float` **: 32비트 실수. 일반적으로 7자리 정밀도를 가짐. 접미사 'f'가 필요함.**
- `double` **: 64비트 실수. 일반적으로 15~17자리 정밀도를 가짐.문자형:**
- `char` **: 16비트 유니코드 문자.논리형:**
- `bool` **: 불리언 값. true 또는 false.**

2. **참조 형식: 참조 형식은 힙 메모리에 데이터를 저장합니다.**

문자열형:

- `string` **: 문자의 연속이 저장되는 참조형 자료형인 문자열(immutable).객체 형식:**

- `object`: 모든 C# 클래스의 기본 클래스. 모든 참조형 자료형을 object 형식에 저장할 수 있습니다.

3. 사용자 정의 자료형 (null 형식):

사용자 정의 형식:

- `class`: 사용자가 원하는 데이터 및 함수를 포함하는 클래스를 정의할 수 있습니다.
- `struct`: 클래스와 유사하나 값 형식으로 동작하는 사용자 정의 형식입니다.
- `enum`: 여러 개의 명명된 상수들의 집합을 정의하는 타입입니다.
- `delegate`: 사용자가 정의하는 함수 포인터 형식입니다.

▼ 변수

변수는 프로그램에서 사용하는 데이터를 저장하는 공간입니다. C#에서 변수를 사용할 때, 다음 요소를 고려해야 합니다.

데이터의 자료형(Type): 변수에 저장되는 데이터의 종류를 결정합니다.
예를 들어, `int`, `float`, `bool`, `string` 등이 있습니다.변수의 이름(Identifier): 변수를 식별할 수 있는 고유한 이름을 지정합니다.
예를 들어, score, playerName, isGameOver과 같은 이름을 사용할 수 있습니다. 명명 규칙을 따르는 것이 좋습니다.값(Value): 변수가 저장하는 데이터입니다.
예를 들어, 변수에 정수 100을 저장하거나 문자열 "Hello, World!"를 저장할 수 있습니다.

```
int number;           // 정수형 변수 선언
float score;          // 부동 소수점 변수 선언
bool isGameOver;      // 불리언 변수 선언
string playerName;    // 문자열 변수 선언
char grade;           // 문자 변수 선언

number = 100;          // 변수에 값을 할당
score = 95.5f;
isGameOver = false;
playerName = "John";
grade = 'A';
```

▼ 제어문

제어문은 프로그램의 실행 흐름을 제어하는데 사용되는 문장들입니다. 주요 제어문은 다음과 같습니다.

1. 조건문 (Conditional Statements): 조건문은 주어진 조건에 따라 실행 흐름을 분기합니다.

if 문: 조건이 참인 경우에만 해당 블록의 코드를 실행합니다.

```
if (condition)
{
    // 조건이 참일 때 실행되는 코드
}
```

if-else 문: 조건이 참인 경우와 거짓인 경우에 각각 다른 코드를 실행합니다.

```
if (condition)
{
    // 조건이 참일 때 실행되는 코드
}
else
{
    // 조건이 거짓일 때 실행되는 코드
}
```

switch 문: 변수의 값에 따라 여러 블록 중 하나를 실행합니다.

```
switch (variable)
{
    case value1:
        // 변수 값이 value1일 때 실행되는 코드
        break;
    case value2:
        // 변수 값이 value2일 때 실행되는 코드
        break;
    default:
        // 변수 값이 일치하는 case가 없을 때 실행되는 코드
        break;
}
```

2. **반복문 (Loop Statements): 반복문은 특정 조건이 만족되는 동안 코드 블록을 여러 번 실행합니다.**

for 문: 초기값, 조건식, 증감식을 사용하여 반복을 제어합니다.

```
for (int i = 0; i < 10; i++)
{
    // 반복 실행되는 코드
}
```

while 문: 주어진 조건이 참인 동안 계속 반복합니다.

```
while (condition)
{
    // 반복 실행되는 코드
}
```

do-while 문: 코드 블록을 먼저 실행한 뒤, 조건이 참인 동안 반복을 계속합니다.

```
do
{
    // 반복 실행되는 코드
} while (condition);
```

3. **제어 구문 (Control Flow Statements):**

break 문: switch 문이나 반복문의 실행을 즉시 종료합니다.

```
csharp
for (int i = 0; i < 10; i++)
{
    if (i == 5)
    {
        break;
    }
// 반복 실행되는 코드
}
```

continue 문: 반복문의 나머지 부분을 건너뛰고 다음 반복으로 진행합니다.

```
csharp
for (int i = 0; i < 10; i++)
{
    if (i % 2 == 0)
    {
        continue;
    }
// 반복 실행되는 코드
}
```

이러한 제어문을 활용하여 프로그램의 실행 흐름을 조작하며 다양한 로직을 구현할 수 있습니다.

스크립트 작성과 컴포넌트 연결 방법 학습

▼ 스크립트 작성

1. **스크립트 작성: 유니티에서 먼저 C# 스크립트를 생성해야 합니다. Project 창에서 마우스 오른쪽 버튼을 클릭하고 Create > C# Script 를 선택합니다. 스크립트의 이름을 "ObjectMover"로 변경합니다.**
2. **ObjectMover 스크립트를 더블 클릭하여 코드 편집기를 열고 다음 코드를 작성합니다.**

```
using UnityEngine;

public class ObjectMover : MonoBehaviour
{
```

```
    public float speed = 1.0f;

    void Update()
    {
        float moveHorizontal = Input.GetAxis("Horizontal");
        float moveVertical = Input.GetAxis("Vertical");

        Vector3 movement = new Vector3(moveHorizontal, 0.0f, moveVertical);
        transform.position += movement * speed * Time.deltaTime;
    }
}
```

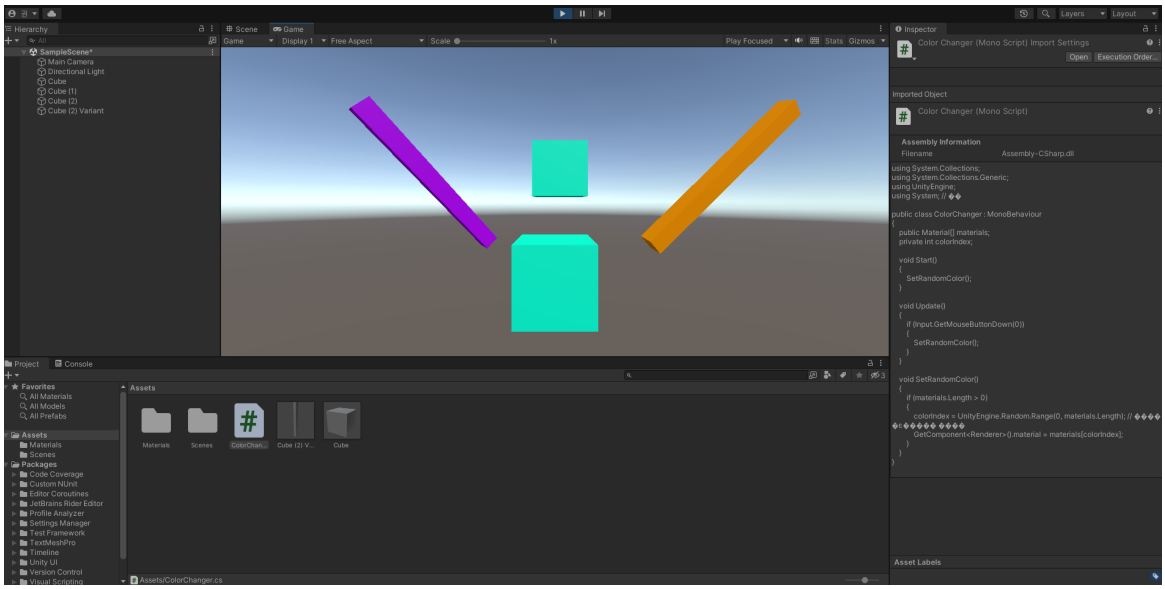
이 스크립트는 입력받은 방향키에 따라 오브젝트를 움직이는 로직을 구현한 것입니다.

▼ 컴포넌트 연결

컴포넌트 연결: 이제 작성된 스크립트를 원하는 게임 오브젝트에 적용해야 합니다. 이를 위해 Hierarchy 창에서 움직이게 하려는 게임 오브젝트를 선택합니다. Inspector 창에서 Add Component 버튼을 클릭하고 Scripts 카테고리에서 "ObjectMover" 스크립트를 선택합니다. 또는 작성된 스크립트를 직접 게임 오브젝트로 드래그 앤 드롭으로 추가할 수도 있습니다.

위의 과정을 완료하면 "ObjectMover" 스크립트가 선택된 게임 오브젝트의 컴포넌트로 추가됩니다. 이제 게임을 실행하면 방향키를 누를 때 오브젝트가 이동하는 것을 확인할 수 있습니다.
스크립트 작성과 컴포넌트 연결 방법인 이를 익히고 나면 다양한 게임 로직을 작성하여 게임 오브젝트에 적용하고 조작할 수 있게 됩니다.

▼ 이벤트 처리와 변수 활용 실습



```
using System.Collections;
using System.Collections.Generic;
```

```
using UnityEngine;
using System; // 추가

public class ColorChanger : MonoBehaviour
{
    public Material[] materials;
    private int colorIndex;

    void Start()
    {
        SetRandomColor();
    }

    void Update()
    {
        if (Input.GetMouseButtonDown(0))
        {
            SetRandomColor();
        }
    }

    void SetRandomColor()
    {
        if (materials.Length > 0)
        {
            colorIndex = UnityEngine.Random.Range(0, materials.Length); // 랜덤 인덱스로 변경
            GetComponent<Renderer>().material = materials[colorIndex];
        }
    }
}
```

위 `ColorChanger` 스크립트는 게임 오브젝트의 색상을 마우스 왼쪽 버튼 클릭 시 랜덤으로 변경하는 기능을 수행합니다. `Update()` 메서드에서 클릭 이벤트를 감지하고, `SetRandomColor()` 메서드를 호출하여 배열에 저장된 색상 중 무작위로 하나의 색상을 선택해 오브젝트의 렌더러의 Material을 변경하도록 구현되어 있습니다.

참고 자료

[Unity] 유니티 기초 탈출기 - Part.03 - 스크립트 편 (사용자 입력 이벤트)