UNITY
-CAHPTER4-2-

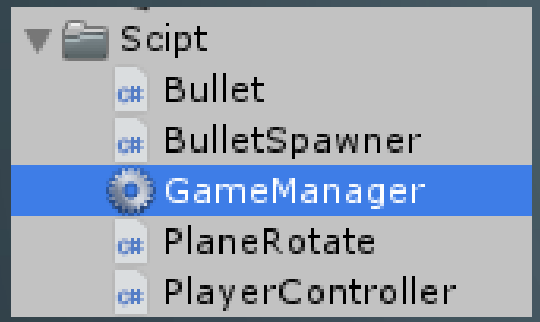
SOUL SEEK

목차

GAMEMANAGER로 컨트롤하기

1. GAMEMANAGER로 컨트롤하기

GameManager 생성



GameManager Script를 생성하고 생성한 **Component**를 추가할 **GameObject**를 준비한 뒤 **Component**를 생성한 **GameObject**에 추가한다.

GameManager가 해야 할 일.

1. **Timer**를 체크해서 생존시간을 기록 및 갱신한다.
2. 게임 종료 상황에서 나타나야 할 **UI**를 컨트롤 한다.
3. 게임을 재시작하고 **Game** 진행에 맞는 **Scene**을 생성한다.

GameManager Script 작성

먼저, 해당 작업을 하기 이전에 **System**에서 해당하는 **Compoent**들을 사용하기 위한 **namespace**의 사용을 추가 한다.

```
using UnityEngine.UI; // UI관련 라이브러리
using UnityEngine.SceneManagement; // Scene관련 라이브러리
```

1. GAMEMANAGER로 컨트롤하기

Game 관리와 진행에 필요한 **Type**들의 **Member** 변수를 선언한다.

```
public GameObject gameoverText; // 게임오버 시 활성화할 텍스트 GameObject
public Text timeText; // 생존 시간을 표시할 텍스트 Component
public Text recordText; // 최고 기록을 표시할 텍스트 Component

private float surviveTime; // 생존 시간
private bool isGameover; // 게임오버 상태
```

- **gameoverText** : 게임오버 시 활성화할 **Text GameObject**인 **Gameover Text**를 할당한다,
- **timeText** : 생존 시간을 표시할 **Text Component**인 **Time Text GameObject**의 **Text Component**를 할당한다.
- **recordText** : 최고 기록을 표시할 **TextComponent, Record Text GameObject**의 **Text Component**를 할당한다.
- **surviveTime** : 게임 시작 이후 현재까지 플레이어가 살아남은 시간
- **isGameover** : 게임오버 상태를 표현

Game이 시작하면 생존시간을 초기화 해야 하는 부분이 발생하고 **Play**하는 **Scene**을 계속 다시 **Load**하게 되면 비활성화와 활성화가 반복되면서 **Start() Method**가 항상 실행되기 때문에 초기화 부분을 작성할 수 있다.

```
void Start()
{
    //생존 시간과 게임오버 상태 초기화
    surviveTime = 0;
    isGameover = false;
}
```

1. GAMEMANAGER로 컨트롤하기

게임 종료부분의 **Method**를 준비한다.

```
//현재 게임을 게임오버 상태로 변경하는 Method
public void EndGame()
{

}
```

Update()에서 **Time.deltaTime**을 이용해서 생존시간을 지속적으로 체크할 것이다.

```
void Update()
{
    //게임오버가 아닌 동안
    if(!isGameover)
    {
        //생존 시간 갱신
        surviveTime += Time.deltaTime;
        //갱신한 생존 시간을 timeText의 Text Component를 이용해 표시
        timeText.text = "Time: " + (int)surviveTime;
    }
}
```

Game 진행 상태를 **bool** 형을 이용해서 체크 하고 있다. 만약 여러 상태 혹은 전체 **Scene**의 변화가 많을 을 때는 **bool**을 사용하는 것보다 **Enum**을 사용하는 것이 좋다.

- **Time.deltaTime**을 누적하여 **surviveTime**을 갱신
- **timeText**가 표시중인 **UI Text** 내용을 갱신된 생존 시간으로 변경

1. GAMEMANAGER로 컨트롤하기

게임 종료부분의 **Method**를 준비한다.

```
//현재 게임을 게임오버 상태로 변경하는 Method
public void EndGame()
{

}
```

Update()에서 **Time.deltaTime**을 이용해서 생존시간을 지속적으로 체크할 것이다.

```
void Update()
{
    //게임오버가 아닌 동안
    if(!isGameover)
    {
        //생존 시간 갱신
        surviveTime += Time.deltaTime;
        //갱신한 생존 시간을 timeText의 Text Component를 이용해 표시
        timeText.text = "Time: " + (int)surviveTime;
    }
}
```

Game 진행 상태를 **bool** 형을 이용해서 체크 하고 있다. 만약 여러 상태 혹은 전체 **Scene**의 변화가 많을 을 때는 **bool**을 사용하는 것보다 **Enum**을 사용하는 것이 좋다.

- **Time.deltaTime**을 누적하여 **surviveTime**을 갱신
- **timeText**가 표시중인 **UI Text** 내용을 갱신된 생존 시간으로 변경

1. GAMEMANAGER로 컨트롤하기

Text Component를 **Inspector**창에서 변경을 할 수 있다는 것은 **public** 형태의 프로퍼티이다.
그렇기 때문에 **GameManager**에서도 직업 해당 프로퍼티들의 값을 대입해서 변경할 수 있다.

Ex)

```
timeText.fontSize = 33;
timeText.color = Color.red;
timeText.color = new Color(255, 0, 0);
```

GameObject 상황에서 특정 **KeyCode**를 입력해서 재 시작을 진행하여 순환구조를 만들자.

```
else
{
    // 게임오버 상태에서 R키를 누른 경우
    if(Input.GetKeyDown(KeyCode.R))
    {
        // GameScene Scene을 로드
        SceneManager.LoadScene("GameScene");
    }
}
```

```
using UnityEngine.SceneManagement; // Scene관련 라이브러리
```

해당 **namespace**를 사용한다고 지정해야 **SceneManager**를 이용할 수 있다. **Scene**이 여러 개 일 경우 해당 현재 **Scene**을 **UnLoad**하고 원하는 **Scene**을 **Load**할 경우 **LoadScene**을 사용한다.

1. GAMEMANAGER로 컨트롤하기

EndGame() Method를 구현한다.

- 게임오버 상태 **isGameover**를 **true**로 변경

```
//현재 게임을 게임오버 상태로 변경하는 Method
public void EndGame()
{
    //현재 상태를 게임오버 상태로 전환
    isGameover = true;
    //게임오버 텍스트 게임 오브젝트를 활성화
    gameoverText.SetActive(true);
}
```

- 현재 생존 시간 기록과 최고 생존시간 기록을 비교
- 게임오버 **UI**를 활성화하고 최고 기록을 비교

```
//BestTime 키로 저장된 이전까지의 최고 기록 가져오기
float bestTime = PlayerPrefs.GetFloat("BestTime");

//이전까지의 최고 기록보다 현재 생존 시간이 더 크다면
if(surviveTime > bestTime)
{
    //최고 기록 값을 현재 생존 시간 값으로 변경
    bestTime = surviveTime;
    //변경된 최고 기록을 BestTime 키로 저장
    PlayerPrefs.SetFloat("BestTime", bestTime);
}

//최고 기록을 recordText Text Component를 이용해 표시
recordText.text = "BestTime: " + (int)bestTime;
```

1. GAMEMANAGER로 컨트롤하기

파일 입출력을 통해서 기록을 할 수도 있다, 하지만 간단한 **Player**에 대한 정보 기록을 위해 **Unity**에서 **1MB** 정도의 용량으로 제공해주는 **PlayerPrefs**라는 것을 이용할 수 있다.

PlayerPrefs는 게임이 삭제되면 함께 사라지기 때문에 게임에 대한 옵션 같은 비교적 복잡하지 않고 간단한 **Data**를 저장하기 위해 주로 사용된다.
Key, Value 형태로 저장되며 각 **Type**형태에 따라 사용하는 **Method**는 달라진다.

```
public static void SetFloat(string key, float value);
public static float GetFloat(string key);
```

- **SetXXX()** 형태의 **Method**를 이용해 저장할 **key**를 문자열형태로 **value**를 저장한다.\
  →**key**가 없으면 **key**를 만들 것이고 이미 해당 **key**가 존재하고 **value**의 **type**이 같다면 해당 **key**의 **value**를 갱신하게 된다, **type**이 다르면 덮어 씌운다.

- **GetXXX()** 형태의 **Method**를 이용해 **key**에 저장된 값을 가져온다.
  →해당하는 키가 존재 하지 않으면 기본값인 **0**으로 반환하며 **Value Type String**일 경우 **“”**이 반환한다.

- **HasKey(“KeyName”)**으로 해당한 **Key**로 **Value**를 저장한 것이 존재하는지를 알 수 있다.
  → **SetXXX, GetXXX**를 하기 이전에 **Key**의 존재 여부를 미리 체크하는 것이 좋다.

1. GAMEMANAGER로 컨트롤하기

EndGame() Method가 실행될 지점을 찾아야 한다.
→게임오버가 되는 경우는 **Player**가 **Bullet**이랑 충돌했을 경우이다.
→**PlayerController**에서 **Die() Method**에서 **Bullet**과의 충돌여부를 판별하여 **EndGame()**을 호출한다.

```
public void Die()
{
    //자신의 GameObject를 비활성화
    gameObject.SetActive(false);

    //Scene에 존재하는 GameManager 타입의 오브젝트를 찾아서 가져오기
    GameManager gameManager = FindObjectOfType<GameManager>();

    //가져온 GameManager 오브젝트의 EndGame() Method 실행
    gameManager.EndGame();
}
```

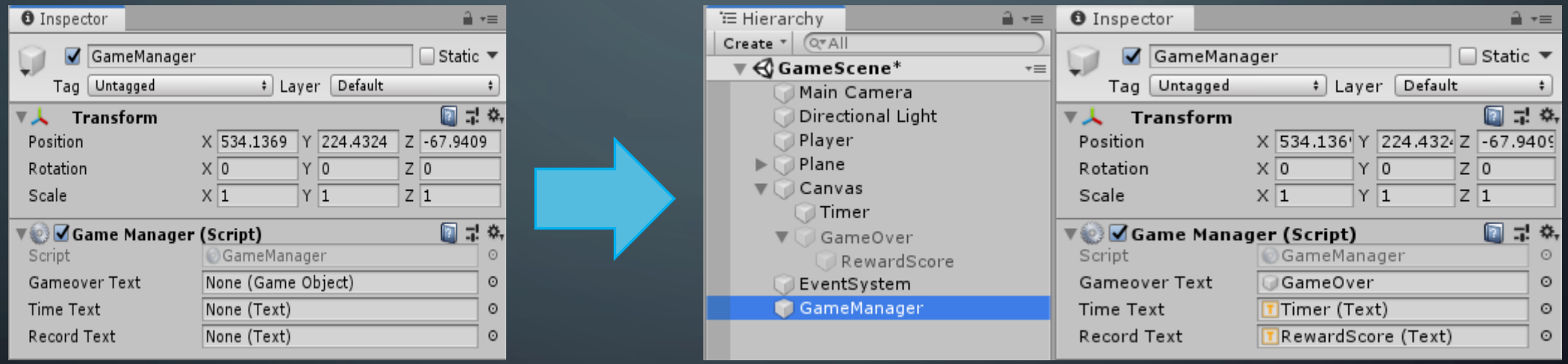
GameManager Object를 찾아서 **Component**를 가져온다.

만약, **Player**도 **Prefab**이라면 **GameManager**에서 생성해서 **Player**를 설정하면서 **GameManager**의 **Component**를 넘겨주는 방법을 사용할 수 있다.

모든 **Script**를 작성하였으면 **GameManager**에서 **Add**되어야할 다른 **Object**의 **Component**를 연결하자.

1. GAMEMANAGER로 컨트롤하기

다른 **Object**들의 **Component** 연결



1. GAMEMANAGER로 컨트롤하기