


2018-2학기 세종창의학기제 주간학습보고서

이름	오민규	학과(전공)	컴퓨터공학과		
학번	128206	학년	4		
연락처	010-8843-4757	e-Mail	hotalsrb12@naver.com		
과목명	자기주도 창의전공 I, II, III, IV	신청학점	12	분반	3
학습기간	2018. 09. 03 ~ 2018. 09. 09	학습주차	1	학습시간	60
창의과제	네트워크를 통해 상대방과의 실감나는 대전형 슈팅 3D 모바일게임 개발				
금주 학습목표	<ul style="list-style-type: none">- 대전형 슈팅 3D 모바일게임 코어 설계를 위한 학습- Actor 클래스 구현- 싱글톤 / 모노 싱글톤 클래스 구현				
학습내용	<p>1. 대전형 슈팅 3D 모바일게임 코어 설계를 위한 학습</p> <p>게임에서 어느 부분(UI, 캐릭터, 이펙트, 맵, 이벤트 등)이든 공통적으로 사용하는 클래스를 설계해야한다. 따라서 공통으로 사용하는 클래스를 설계해야하고 이 클래스를 더 효율적으로 설계를 위한 디자인 패턴을 선택해야한다. 패턴의 경우 많은 지식이 없기 때문에 학습이 필요하였다. 기존에 구매했던 게임 프로그래밍 패턴 문헌을 가지고 학습을 하고 개발하는 게임에 적합한 패턴을 선정하여 구현을 해야 했다. 코어를 설계하기 위하여 전체적인 게임의 구조를 그리는 과정에서 시간을 많이 사용하였다. 실제 서비스를 위한 전체 구조를 설계하는 것은 무리가 있었기 때문에 Prototype 개발을 목적으로 한 구조를 바탕으로 설계를 시작하였다. 앞으로 사용해야 할 데이터를 static으로 설계하여 게임이 구동될 때 계속 사용하기 위한 싱글톤 패턴을 사용하기로 채택하였다. 그리고 내 캐릭터와 AI 캐릭터가 같은 기능을 처리하기 위해 명령 패턴이 적용된 Actor클래스를 구현하고 이 클래스를 상속받음으로써 캐릭터가 행동을 하도록 해야 한다. 디자인 패턴 문헌을 학습을 진행하면서 어려운 점이 많이 있었다. 단순히 코딩 공부하는 것이 아닌 소프트웨어를 설계하는 방법이 제안되어있기 때문에 책을 읽으며 이해가 잘 되지 않은 부분이 많이 있어서 학습하는데 시간이 많이 걸리게 되었다. 이러한 이유로 이 문헌을 삼일 안에 책 전체를 학습 할 수 없다고 생각되어 필요한 부분만 찾아서 학습을 진행하였다.</p> <div></div>				



2. Actor 클래스 구현

한 개의 클래스를 사용하여 내 캐릭터와 적 캐릭터의 행동을 할 수 있는 클래스를 구현하여 재활용을 높이고 코드의 가독성과 수정을 용이하기 위해 명령패턴을 구축하여 적용하였다. 책에서는 명령패턴에 대해서 C++ 또는 sudo코드를 사용하여 설명을 하였다. 하지만 게임제작에 사용하는 언어는 C#이며 같은 OOP 개념을 사용한 언어라도 사용 방식에서 차이가 많이 있기 때문에 실제 구현하면서 메모리 관련하여 고려해야할 부분이 너무 많았다. 명령 패턴에서는 사용자가 특정 행동을 캐릭터에게 시키기 위해 버튼을 누르게 된다. 그에 따른 처리를 위하여 Command 클래스를 생성하여 Excute()함수를 가상함수로 둔 껍데기를 클래스이자 최상위 클래스를 생성하였다. 그 후 실제 명령에 사용될 클래스인 CommandThrow, CommandJump, CommandDash, CommandMove 클래스를 생성하여 Command를 상속하여 한 개의 Command 객체를 이용하여 게임이 실행될 때 생성되는 각 Command행동 클래스들의 객체를 교체하며 사용할 수 있도록 구현하였다. 여기까지 구현한 부분은 내 캐릭터에서 밖에 사용하지 못한다. 즉 주체가 고정되어 있는 상태이다. 따라서 이 문제를 해결하기 위해 Actor 클래스를 생성하여 실제 행동하는 함수들을 작성하였다. 기능은 추후에 구현하기 때문에 일단 로그를 생성하여 테스트하는 문장만 생성하였다. Command 클래스의 Excute함수에 Actor를 매개변수로 넘겨주어 동적으로 모든 캐릭터가 사용 할 수 있도록 하였으며 각 명령행동 클래스에 맞는 Actor의 함수를 실행시키도록 구현하였다.

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

// 명령 패턴에 사용될 최상위 추상 클래스
public abstract class Command
{
    public virtual void Excute(Actor actor) { }
}

// Idle명령 클래스
public class CommandAlert : Command
{
    public override void Excute(Actor actor)
    {
        actor.Alert();
    }
}

// Dash명령 클래스
public class CommandDash : Command
{
    public override void Excute(Actor actor)
    {
        actor.Dash();
    }
}

// Attack명령 클래스
public class CommandAttack : Command
{
    public override void Excute(Actor actor)
    {
        actor.Attack();
    }
}

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class InputHandler
{
    // 명령 변수
    private Command buttonNone; // 유희 명령
    private Command buttonF; // 대쉬 명령
    private Command buttonMouseX; // 공격 명령

    // 명령 초기화 함수
    public void SetHandler()
    {
        buttonNone = new CommandAlert();
        buttonF = new CommandDash();
        buttonMouseX = new CommandAttack();
    }

    // 눌러지는 키에 따라서 해당하는 명령을 반환하는 함수
    public Command IsPressedKey()
    {
        if (Input.GetKeyDown(KeyCode.F))
        {
            return buttonF;
        }
        else if (Input.GetMouseButtonDown(0))
        {
            return buttonMouseX;
        }

        return buttonNone;
    }
}
```

<Command Class>

<InputHandler Class>

3. 싱글톤 / 모노 싱글톤 구현

싱글톤을 사용하여 게임 시작 시 한 개의 공통으로 사용하는 객체를 생성하고 이 객체를 참조하여 공용 데이터를 사용할 수 있고 객체를 계속 생성하고 삭제하는 작업을 하지 않으므로

가비지 컬렉터의 역할이 줄어들기 때문에 게임 성능이 더 좋아지게 된다. 추상 클래스와 템플릿을 사용하여 싱글톤 클래스를 제작하였고 평소에 사용하지 않았던 C#의 기능 중 프로퍼티를 사용하여 get을 통해 생성하거나 반환한다. 그리고 유니티에서 제공하는 기능들을 사용하기 위해서는 MonoBehaviour를 상속받아 각종 컴포넌트들을 사용해야한다. 기본 싱글톤은 제네릭 형태의 클래스를 사용할 때 사용되므로 유니티 기능이 포함된 클래스는 사용하지 못한다. 따라서 MonoBehaviour를 상속받은 싱글톤을 구현해야한다. 내용은 기본 싱글톤과 다른 점이 없어서 게임 오브젝트를 사용하여 생성하고 이름을 붙이는 작업과 생성 시 파괴되지 않는 함수인 DontDestroyOnLoad를 적용하여 구현하였다.

```

4 // Generic 싱글톤
5
6 public abstract class Singleton<T> where T : class {
7     protected static T Instance = null;
8     public static T Instance
9     {
10         get
11         {
12             if (Instance == null)
13             {
14                 Instance = System.Activator.CreateInstance(typeof(T)) as T;
15             }
16             if (Instance == null)
17             {
18                 Debug.LogError(typeof(T) + " is not found");
19             }
20             return Instance;
21         }
22     }
23 }
24
25
4 // Mono 싱글톤
5
6 public class MonoSingleton<T> : MonoBehaviour where T : MonoBehaviour {
7     protected static T Instance = null;
8     public static T Instance
9     {
10         get
11         {
12             Instance = FindObjectOfType(typeof(T)) as T;
13             if (Instance == null)
14             {
15                 Instance = new GameObject("8*" + typeof(T).ToString(), typeof(T)).GetComponent<T>();
16                 DontDestroyOnLoad(Instance);
17             }
18             if (Instance == null)
19             {
20                 Debug.LogError(typeof(T) + " is not found");
21             }
22             return Instance;
23         }
24     }
25 }
26
27
28
29

```

4. 느낀점

이번 프로젝트를 진행하면서 명령패턴을 처음 사용해보았다. 학교에서 과제나 프로젝트를 진행하면 이런 패턴을 적용해 볼 수 있는 기회가 많지가 않았다. 그저 과제를 해내기 위하여 구현을 하고 결과를 나타내는데 급급하여 소프트웨어의 설계가 엉망이거나 단순 절차적인 로직으로만 돌아가도록 제작하였지만 큰 프로젝트를 진행을 위하여 전체적인 구조를 설계하고 패턴들을 사용함으로써 기술적으로 도움이 될 스킬과 패턴들을 사용하여 프로그래밍 실력에 큰 도움이 될 것이라고 생각되며 나중에 취업을 하게 된다면 실무에서 큰 도움이 될 것이라고 생각된다. Actor 클래스를 설계하고 적용하는 과정에서는 시간이 많이 들었지만 싱글톤을 구현하는 과정은 생각보다 빨리 작업이 마무리되어 문제없이 첫 주를 보낼 수 있었다. 하지만 추후 네트워크가 적용된다면 이 명령패턴이 효율적으로 사용될지 의문이다. 단순히 하나의 함수로 처리하는 방식의 행동을 구현하였을 때 호출하여 사용한다면 함수 하나를 처리하고 전송하면 클라이언트 처리도 빠르고 전송 또한 빠르게 될 것이지만 명령패턴을 사용하여 네트워크를 전송한다면 호출되는 함수의 수도 증가되므로 전송과 처리속도가 떨어질 것이다. 이 문제는 추후 교수님에게 조언을 얻거나 유니티에서 성능을 확인할 수 있는 프로파일러창을 통해 테스트를 해봐야겠다. 명령패턴을 사용하지 않더라도 Actor 함수의 기능을 다 제작하기 때문에 별도로 구현할 필요는 없다고 생각한다. 시작이 좋은 출발이듯 앞으로도 문제없이 프로젝트를 잘 진행되었으면 좋겠다.

참고자료 및 문헌

- 절대강작! 유니티 2018
- 게임 프로그래밍 패턴
- <https://docs.unity3d.com/kr/2018.1/Manual/UnityManual.html> (유니티 공식 매뉴얼)

학습방법	게임 프로그래밍 패턴 문헌을 통하여 게임 개발에 많이 사용되는 패턴들에 대해 학습하고 참고하여 구현을 한다. 유니티 툴을 제어하고 사용하기 위한 요소들을 절대강좌! 유니티 문헌과 유니티 가이드를 통해 학습하였다.
학습성과 및 목표달성도	<ul style="list-style-type: none"> - Actor 클래스 구현 완료 (100%) - 싱글톤 / 모노 싱글톤 클래스 구현 완료 (100%)
내주 계획	Actor 클래스를 구현하였으니 실제 캐릭터가 하는 행동을 구현해야한다. 기능을 코드와 에디터에서 오브젝트에 필요한 컴포넌트를 부착하고 캐릭터가 행동에 맞춰 애니메이션도 필요하기 때문에 애니메이션 매니저를 설계한다.

2018. 09. 10 .

지도교수

(인)