Unity 游戏框架搭建
2017（四）简易有限状态机

为什么用有限状态机?

之前做过一款跑酷游戏,跑酷角色有很多状态:跑、跳、二段跳、死亡等等。一开始是使用if/switch来切换状态,但是每次角色添加一个状态（提前没规划好),所有状态处理相关的代码就会指数级增长,那样就会嗅出代码的坏味道了。在这种处理状态并且状态数量不是特别多的情况下,自然就想到了引入状态机。

优点:

1. 使代码整洁,状态容易扩展和管理。
2. 可复用。
3. 还没想到.....

缺点:

1. 也没想到.....

什么是有限状态机?

解释不清楚,看了下百度百科。反正是一种数据结构,一个解决问题的工具。

从百度百科可以看到,有限状态机最最最基础的概念有两个:状态和转移。

从刚才跑酷的例子来讲,跑、跳、二段跳等这些就是角色的状态。

如图所示:

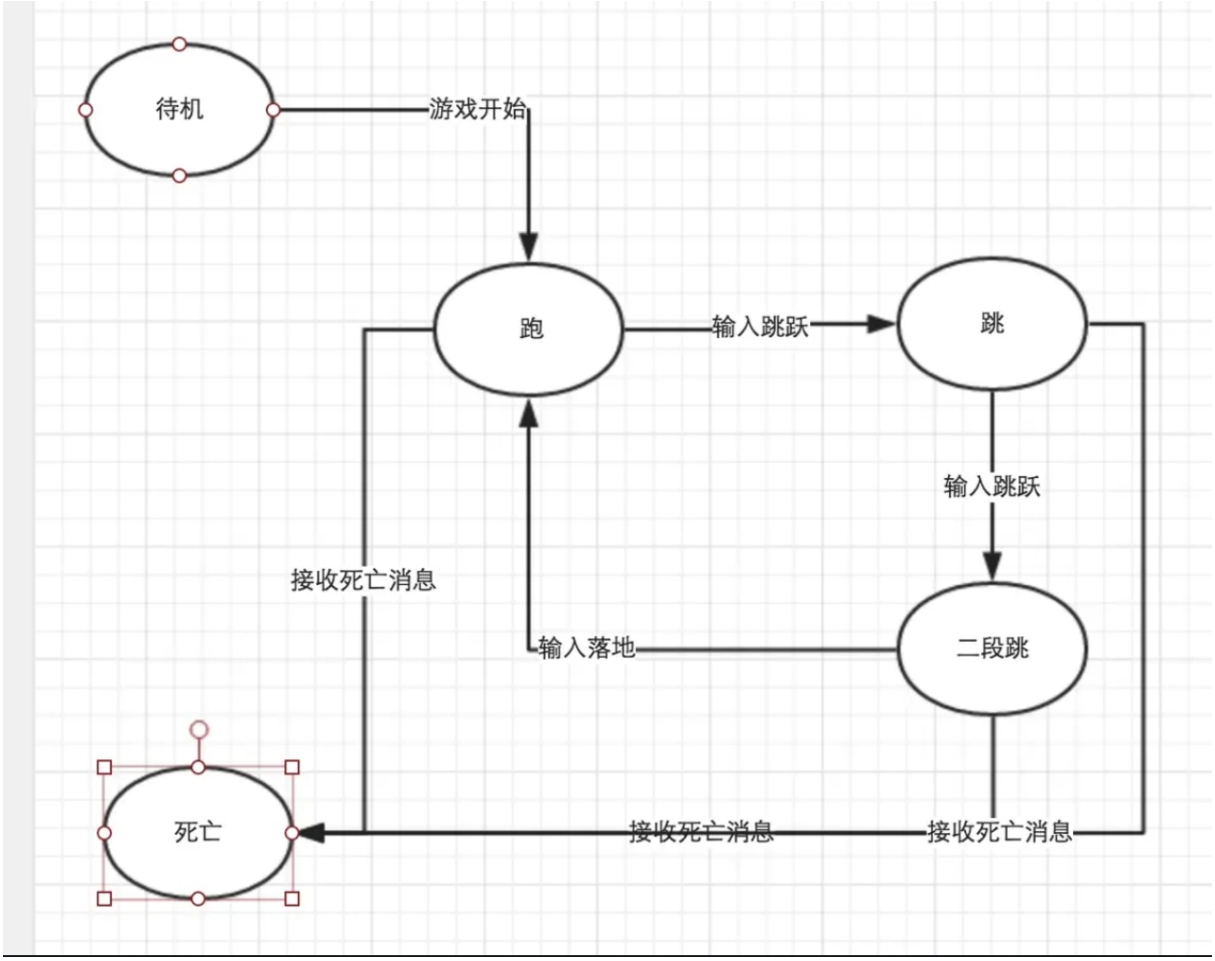

image.png

主角从跑状态切换到跳状态,从跳状态切换到二段跳状态,这里的切换就是指状态的转移。状态的转移是有条件的,比如主角从跑状态不可以直接切换到二段跳状态。但是可以从二段跳状态切换到跑状态。

另外,一个基本的状态有:进入状态、退出状态、接收输入、转移状态等动作。但是仅仅作为跑酷的角色的状态管理来说,只需要转移状态就足够了。有兴趣的同学可以自行扩展。

如何实现?

恰好之前看到过一个还算简易的实现(简易就是指我能看得懂- -,希望大家也是),原版是用lua实现的,我的跑酷游戏是用C#实现的,所以直接贴出C#代码。

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
public class FSM {
    // 定义函数指针类型
    public delegate void FSMTranslationCallfunc();    ///
<summary>
    /// 状态类
    /// </summary>
    public class FSMState
    {
        public string name;

        public FSMState(string name)
        {
            this.name = name;
        }
        /// <summary>
        /// 存储事件对应的条转
        /// </summary>
        public Dictionary <string,FSMTranslation> TranslationDict
= new Dictionary<string,FSMTranslation>();
    }
    /// <summary>
    /// 跳转类
    /// </summary>
    public class FSMTranslation
    {
        public FSMState fromState;
        public string name;
        public FSMState toState;
        public FSMTranslationCallfunc callfunc; // 回调函数

        public FSMTranslation(FSMState fromState,string name,
FSMState toState,FSMTranslationCallfunc callfunc)
        {
            this.fromState = fromState;
            this.toState   = toState;
            this.name = name;
            this.callfunc = callfunc;
        }
    }
```

```
    // 当前状态
    private FSMState mCurState;

    Dictionary <string,FSMState> StateDict = new
Dictionary<string,FSMState>();
    /// <summary>
    /// 添加状态
    /// </summary>
    /// <param name="state">State.</param>
    public void AddState(FSMState state)
    {
        StateDict [state.name] = state;
    }
    /// <summary>
    /// 添加条转
    /// </summary>
    /// <param name="translation">Translation.</param>
    public void AddTranslation(FSMTranslation translation)
    {
        StateDict [translation.fromState.name].TranslationDict
[translation.name] = translation;
    }
    /// <summary>
    /// 启动状态机
    /// </summary>
    /// <param name="state">State.</param>
    public void Start(FSMState state)
    {
        mCurState = state;
    }
    /// <summary>
    /// 处理事件
    /// </summary>
    /// <param name="name">Name.</param>
    public void HandleEvent(string name)
    {
        if (mCurState != null &&
mCurState.TranslationDict.ContainsKey(name)) {
            Debug.LogWarning ("fromState:" + mCurState.name);

            mCurState.TranslationDict [name].callfunc ();
            mCurState = mCurState.TranslationDict [name].toState;
```

```
            Debug.LogWarning ("toState:" + mCurState.name);
        }
    }
}
```

测试代码(需自行修改):

```
//        Idle,                 闲置
//        Run,                  跑
//        Jump,                 一段跳
//        DoubleJump,           二段跳
//        Die,                  挂彩

        // 创建状态
        FSM.FSMState idleState = new FSM.FSMState("idle");
        FSM.FSMState runState  = new FSM.FSMState("run");
        FSM.FSMState jumpState = new FSM.FSMState("jump");
        FSM.FSMState doubleJumpState = new
FSM.FSMState("double_jump");
        FSM.FSMState dieState  = new FSM.FSMState("die");
        // 创建跳转
        FSM.FSMTranslation touchTranslation1 = new
FSM.FSMTranslation(runState,"touch_down",jumpState,Jump);
        FSM.FSMTranslation touchTranslation2 = new
FSM.FSMTranslation(jumpState,"touch_down",doubleJumpState,DoubleJ
ump);

        FSM.FSMTranslation landTranslation1 = new
FSM.FSMTranslation(jumpState,"land",runState,Run);
        FSM.FSMTranslation landTranslation2 = new
FSM.FSMTranslation(doubleJumpState,"land",runState,Run);

        // 添加状态
        PlayerModel.Instance ().fsm.AddState (idleState);
        PlayerModel.Instance ().fsm.AddState (runState);
        PlayerModel.Instance ().fsm.AddState (jumpState);
        PlayerModel.Instance ().fsm.AddState (doubleJumpState);
        PlayerModel.Instance ().fsm.AddState (dieState);

        // 添加跳转
```

```
        PlayerModel.Instance ().fsm.AddTranslation
(touchTranslation1);
        PlayerModel.Instance ().fsm.AddTranslation
(touchTranslation2);
        PlayerModel.Instance ().fsm.AddTranslation
(landTranslation1);
        PlayerModel.Instance ().fsm.AddTranslation
(landTranslation2);

        PlayerModel.Instance ().fsm.Start (runState);
```

就这些,想要进一步扩展的话,可以给 FSMState 类添加 EnterCallback 和 ExitCallback 等委托,然后在 FSM 的 HandleEvent 方法中进行调用。当时对跑酷的项目来说够用了,接没继续扩展了,我好懒- -,懒的借口是:没有最好的设计,只有最适合的设计,233333。

此篇的内容就这些。

转载请注明地址：凉鞋的笔记：liangxiegame.com

更多内容

- QFramework 地址：https://github.com/liangxiegame/QFramework
- QQ 交流群：623597263
- **Unity 进阶小班**：
  - 主要训练内容：
    - 框架搭建训练（第一年）
    - 跟着案例学 Shader（第一年）
    - 副业的孵化（第二年、第三年）
  - 权益、授课形式等具体详情请查看《小班产品手册》：https://liangxiegame.com/master/intro
- 关注公众号：liangxiegame 获取第一时间更新通知及更多的免费内容。