JustCut程序实现文档

目录

[一、 JustCut游戏介绍 3](#_Toc169206390)

[二、 游戏实现思路 4](#_Toc169206391)

[三、 主逻辑系统具体实现思路： 6](#_Toc169206392)

[（一） PlayerController： 6](#_Toc169206393)

[（1） Info 7](#_Toc169206394)

[（2） Model 7](#_Toc169206395)

[（3） Trigger 7](#_Toc169206396)

[（4） InputBuffer 8](#_Toc169206397)

[（5） InteractC 8](#_Toc169206398)

[（6） StateMachine 8](#_Toc169206399)

[（7） MotionController 10](#_Toc169206400)

[（8） FrameContoller 12](#_Toc169206401)

[（二） EffectObject 12](#_Toc169206402)

[（1） ObjectPool 13](#_Toc169206403)

[（2） AttackInfo生成AttackData 13](#_Toc169206404)

[（3） 异步事件 14](#_Toc169206405)

[（三） BOSSAI 14](#_Toc169206406)

[（1） StateMachine 14](#_Toc169206407)

[四、 周边系统实现思路 15](#_Toc169206408)

[（一） UI 15](#_Toc169206409)

[（二） CameraShake，AudioPlayer，Animator 16](#_Toc169206410)

[（三） World 16](#_Toc169206411)

[（四） SceneManager 16](#_Toc169206412)

[（五） GameRound 16](#_Toc169206413)

[（六） Debuger 16](#_Toc169206414)

[（1） PhysicsDebug 16](#_Toc169206415)

[（2） DebugLog 17](#_Toc169206416)

# JustCut游戏介绍

JustCut是一款2D平台动作游戏。以拼刀（与敌人同时攻击时发生的武器对抗）作为核心玩法。游戏特色在于击中敌人时的游戏反馈，和自由搭配时序技能链表以对齐敌人的攻击频率。

日程表

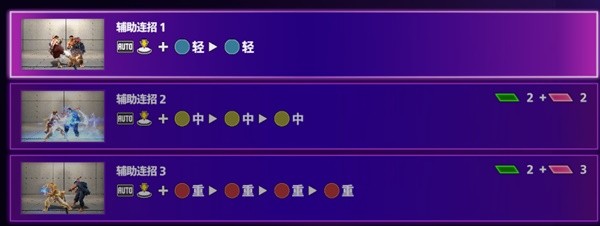
描述已自动生成

图表 1 武士零，一款2D平台动作游戏。

图片包含 猫, 游戏机, 火, 山

描述已自动生成

图表 2 怪物猎人 瞬时打击反馈。



图表 3 街霸6 现代模式辅助连招链表。

图片包含 建筑, 室内, 男人, 体育

描述已自动生成

图表 4 街霸6 高级4 连招链表。（见左侧）

# 游戏实现思路

游戏实现的核心难点是：记录和异步执行多种反馈的事件中心。精确到时间帧的多种状态变化。多种技能链表的数据读入和逻辑执行。BOSSAI的实现。数据与显示的分离。

游戏主要逻辑的实现框架分为三部分：玩家控制器，打击碰撞框，BOSSAI。

玩家控制器在读取玩家输入后执行移动、攻击等逻辑，产生打击碰撞框。

打击碰撞框对敌人受击框或者敌人的打击碰撞框进行碰撞检测。

敌人产生的打击碰撞框也会和玩家受击框和玩家打击框进行碰撞检测。

玩家控制器和BOSSAI通过打击碰撞框完成信息交换。

图示

描述已自动生成

图表 5 主逻辑的整体框架

除开玩家控制器，打击碰撞框，BOSSAI以外，还需要结合Unity游戏引擎实现更多系统。

表现系统：

UI：血条UI，架势UI，帧数表等

动画：Animator组件。

特效：特效播放器。

反馈系统（注册全局事件）：

音乐音效：音频管理器。

摄像机：屏幕震动，屏幕缩放。

世界：帧冻结。

流程系统：

世界：World，时间管理系统。

关卡：场景管理器。

辅助系统：

对象池：实现打击碰撞物体的实时创建与销毁。

# 主逻辑系统具体实现思路：

## PlayerController：

图示

描述已自动生成

可以把玩家控制器分为Info、Model、Trigger、StateMachine、InputBuffer、InteractC、MotionController、FrameController等模块。

### Info

图示, 示意图

描述已自动生成

记录游戏流程中动态变化的玩家信息，比如HP、Posture等信息，在信息数据变化时，会发送信息变化事件，来提醒显示系统更新信息（观察者模式）

### Model

Model记录游戏流程中不会改变的数据。继承ScriptObject存储在游戏文件中。

数据比如：每段攻击的帧数，每段攻击的位移，基础移动速度，移动加速度，跳跃速度曲线，最大下落速度，基础伤害，攻击强度。玩家初始血量，最大架势值等数据都记录与此。

### Trigger

图示

描述已自动生成

Trigger中发射射线检测玩家的位置关系。是否接触地面，是否卡在角落。检测到之后给StateMachine发送消息提醒状态变换。

### InputBuffer

图示

描述已自动生成

存储和处理玩家输入信号。帮助StateMachine转换状态。

### InteractC

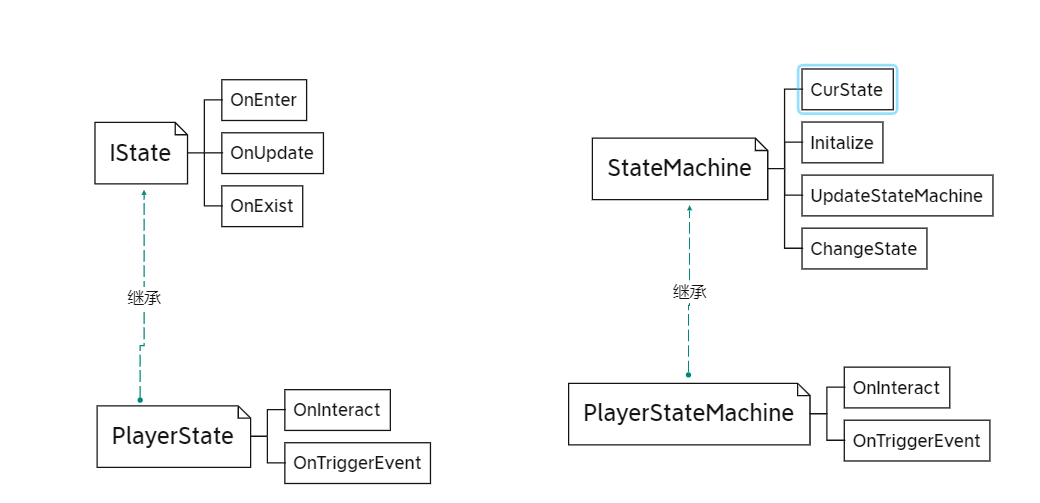
图示

描述已自动生成

与外界交互的组件，接收来自敌人碰撞框的打击信息。交给StateMachine处理。

### StateMachine

运用状态模式的玩家状态机模块。模块分为状态机类和状态类



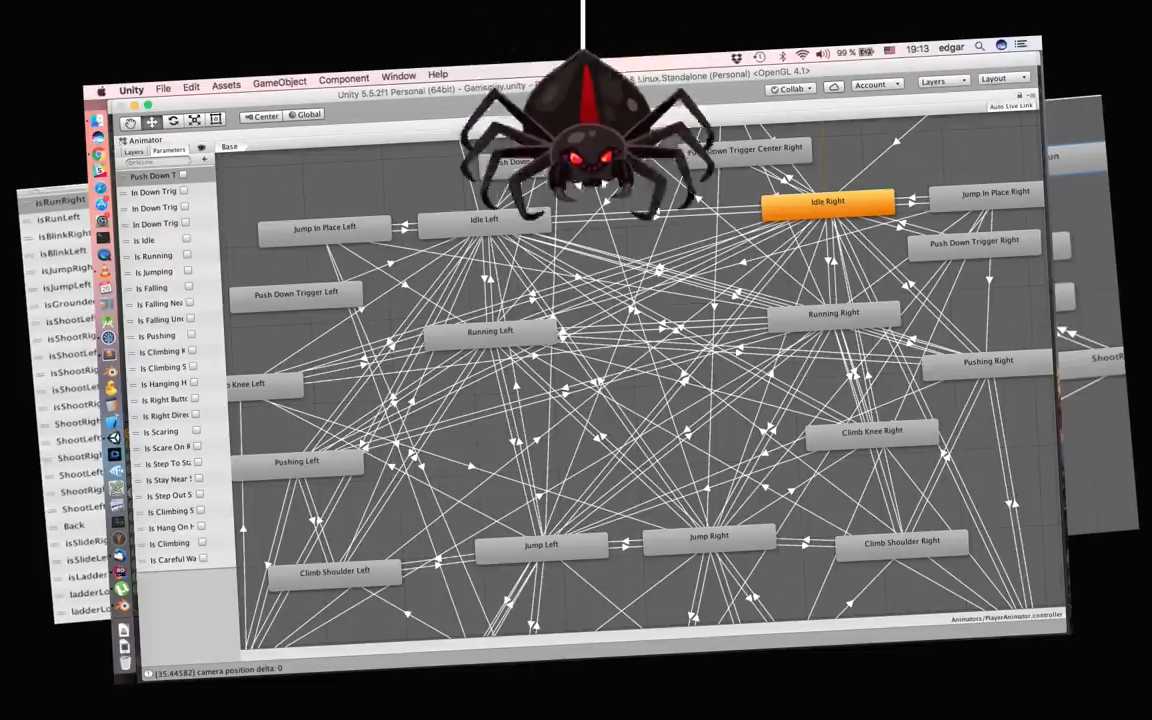
状态机类负责确保只有一个状态被驱动。

状态类则负责具体情况下的逻辑实现。

图示

描述已自动生成

Player拥有的状态数量（图中的状态转换没有连接完全）（状态转换情况复杂，全部连接会变成蜘蛛网）。



#### HeavyAttackState -> RollState -> RollHeavyAttackState

游戏中为了保证玩家在躲避时不会掉连段，HeavyAttackState转到RollState时需要保存HeavyAttack状态信息。在翻滚时能继续连段。

可以用白板模式在状态机内部创建一个白板来存储所有状态的共享信息。

也可以在状态机类中使用状态栈，保存上一个状态的信息。

#### AttackState:LightAttack1State,HeavyAttack1State………

非常多的状态转换具有相同的转换条件。所有的攻击状态中，都可以接收翻滚、移动、跳跃指令。因此采用分层状态机的方式，所有攻击状态继承一个父类攻击状态。相同的转换条件只需要写一次。

### MotionController

负责物理位移计算

图示

描述已自动生成

内部含有玩家的物理坐标向量。

通过状态机传入参数，驱动位移计算。

#### Move(Vector3 dir,AnimationCurve Speed)

传入一个贝赛尔曲线，根据时间每帧读取曲线中速度值，加上运动位移，计算出物理位置。

#### Run

OnStart，标志着玩家输入移动命令的开始。根据加速度，当前速度，最大速度计算位移值。

OnEnd，标志着玩家输入移动命令的终止。根据减速度，当前速度，最大速度计算位移。

#### Roll

OnStart（AnimationCurve），根据运动贝塞尔曲线，计算速度与位移。

#### Jump

OnStart，OnEnd标志玩家输入的开始与停止。

根据跳跃速度，跳跃最大有效时间，正常重力规模，跳跃顶端时重力规模，跳跃最大下落速度，来计算当前速度与位移。

### FrameContoller

图示

描述已自动生成

游戏中每个攻击动作视觉上都有前摇期，有效期，后摇期，逻辑上有蓄力变招期，GP（拼刀时间点），取消期。每个关键帧阶段都需要通过回调来提醒状态机的状态转换。

## EffectObject

BOSSAI和玩家进行的信息交换通过EffectObject来完成。

图示

描述已自动生成

其中运用到对象池技术，根据AttackInfo生成并交换AttackData，通过事件异步调用音效，粒子特效，帧冻结，屏幕抖动等效果。

### ObjectPool

对象池。

ObjectPool.Create(Type type);检查池中有无对应Type的对象，如果没有，通过反射创建出对应Type的对象，调用OnCreate();，并返回对象。

ObjectPool.Dispose(object obj);调用obj.Dispose();进行析构。析构后的对象放入池中。如果需要再次创建则直接激活并且返回。

对象池可以方便的对需要多次创建并销毁的实例进行管理。属于原型模式的一种。

### AttackInfo生成AttackData

通过对象池中返回的实例，进行传入攻击参数（AttackInfo）的初始化。

EffectObject通过Unity的碰撞回调函数判断是否击中敌人，是否击中敌人攻击框。如果击中，数据整合进AttackData中传递给敌人。

### 异步事件

异步效果可以用Unity封装好的Coroutine进行。

事件可以让效果实现类注册全局静态事件。

## BOSSAI

图示

描述已自动生成

BOSSAI 的实现方式与PlayerController基本相同。

其中取消了InputBuffer作为信息的接入。

增加了更为细致的Trigger判断。判断玩家是否在某项攻击的范围内。

其中重写的部分有：StateMachine，Info，Model，Trigger。

可复用的部分有：MotionController，FrameController，InteractC，Animator。

### StateMachine

BOSSAI逻辑分散在StateMachine中。StateMachine根据具体的Trigger状态以及Info来执行攻击和移动命令。

图示, 文本, 信件

描述已自动生成

BOSSAI分散在各个状态之中。状态读取Trigger和Info中的信息。

比如HP小于50%时，下一次进入AttackState会变成ShowTimeState。BOSS会释放一系列攻击对玩家进行考试。

Posture小于0时，会进入BreakDownState，让玩家有机会进行情绪释放。

# 周边系统实现思路

## UI

当UI对应的具体数值发生变化时，调用UI。

UI方面可以注册全局事件，Info只需要在数据变化时发送指令就行。

或者使用事件中心作为转接口（支持合作编程）。

图示

描述已自动生成

## CameraShake，AudioPlayer，Animator

这些组件Unity提供封装。可以直接获取并调用。

## World

单例模式。

自定义Timescale，自定义脚本生命周期。

注册顿帧事件。

## SceneManager

静态变量，提供场景转换工具方法。

## GameRound

支持游戏流程转换。

加载关卡信息。

处理胜利条件（转换场景等）。

## Debuger

### PhysicsDebug

静态类。

画出射线标识。碰撞框等。

### DebugLog

静态类。

输出日志文件，方便调试。