1.玩家输入相关 键盘输入转为信号控制

使用Mathf.SmoothDamp 方法信号变化速率

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class PlayerInput : MonoBehaviour {

public bool inputEnabled = true;

public string KeyUp;

public string KeyDown;

public string KeyLeft;

public string KeyRight;

public float Dup;

public float Dright;

private float targetDup;

private float targetDright;

private float velocityDup;

private float velocityDright;

// Use this for initialization

void Start () {

}

// Update is called once per frame

void Update () {

//key转为signal

//signal -1-1之间

targetDup = Input.GetKey(KeyUp)?1.0f:0 - (Input.GetKey(KeyDown)? 1.0f : 0);

targetDright = Input.GetKey(KeyRight) ? 1.0f : 0 - (Input.GetKey(KeyLeft) ? 1.0f : 0);

if(inputEnabled==false)

{

targetDright = 0.0f;

targetDup = 0.0f;

}

Dup = Mathf.SmoothDamp(Dup, targetDup, ref velocityDup, 0.1f);

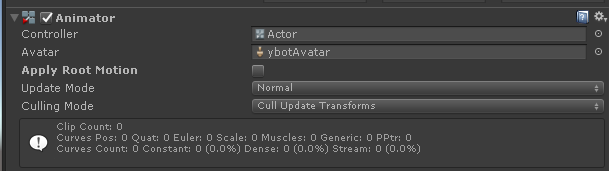
Dright = Mathf.SmoothDamp(Dright, targetDright, ref velocityDright, 0.1f);

}

}

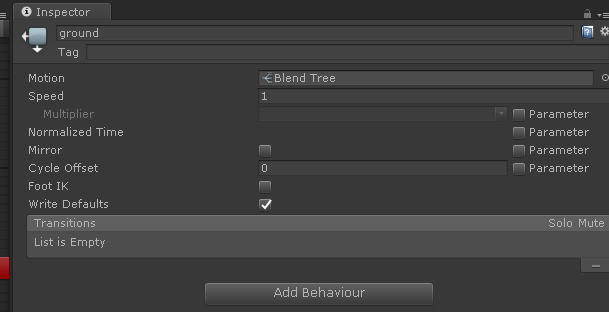
1. 导入模型 动画控制器

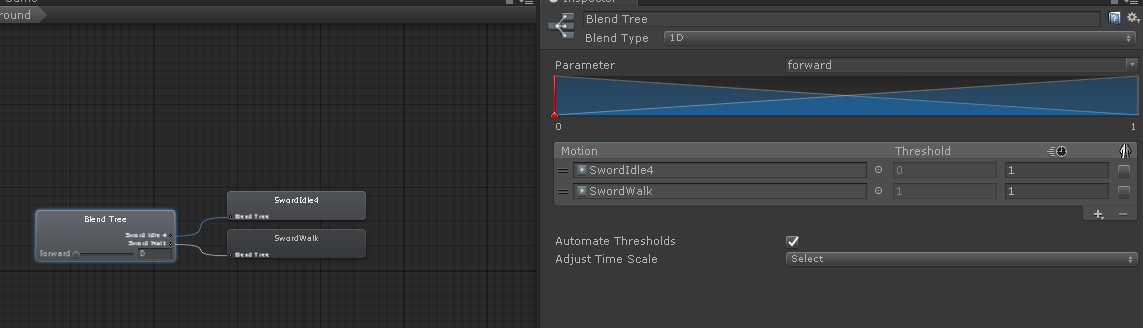
使用模型的idle 以及walk动画



Apply root motion 关闭防止模型播放动画一直远离根节点

防止动画控制器中动画过多 使用blend tree





Blend type 1D 只收一个参数改变

2D 两个参数

角色绑定 角色控制器相关脚本 跟按键绑定

public class ActorController : MonoBehaviour {

public GameObject model;

private PlayerInput pi;

[SerializeField]

private Animator anim;

// Use this for initialization

void Start () {

anim = model.GetComponent<Animator>();

pi = GetComponent<PlayerInput>();

}

// Update is called once per frame

void Update () {

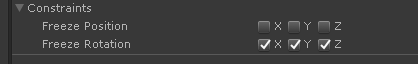
anim.SetFloat("forward", pi.Dup);

}

}

角色移动 使用刚体

添加刚体 设置如下 游戏运行过程中不断转动 偶尔会出现rotation很小 由于重力导致物体倒下去



刚体的操作不要放在update放在FixedUpdate 两个方法的时间不同

// Update is called once per frame

void Update () {

anim.SetFloat("forward", pi.Dmag);

//角色转向

if(pi.Dmag>0.1f)

{

model.transform.forward = pi.Dright \* transform.right + pi.Dup \* transform.forward;

}

movingVec = pi.Dmag \* model.transform.forward;

}

private void FixedUpdate()

{

//刚体相关在这里设置

rigid.position += movingVec \* Time.fixedDeltaTime;

}

加入walkSpeed控制玩家行走速度

movingVec = pi.Dmag \* model.transform.forward\*walkSpeed;

Rigid的两种移动方式

rigid.position += movingVec \* Time.fixedDeltaTime;

//rigid.velocity = new Vector3(movingVec.x, rigid.velocity.y, movingVec.z);

角色奔跑

增加奔跑按键以及对应的信号

public string KeyA;

public string KeyB;

public string KeyC;

public string KeyD;

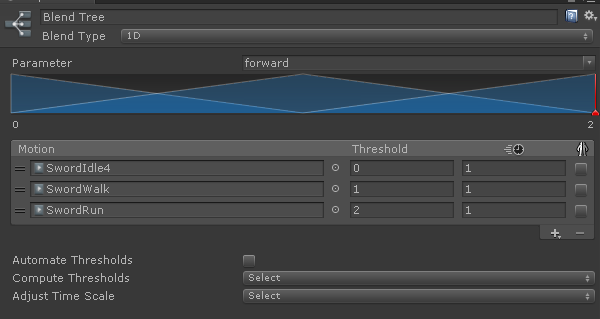
public bool run;

run = Input.GetKey(KeyA);



增加blend tree中的动画 增加一个run动画

设置时间



角色控制器

// Update is called once per frame

void Update () {

anim.SetFloat("forward", pi.Dmag\* ((pi.run) ? 2.0f : 1.0f));

//角色转向

if(pi.Dmag>0.1f)

{

model.transform.forward = pi.Dright \* transform.right + pi.Dup \* transform.forward;

}

movingVec = pi.Dmag \* model.transform.forward\*walkSpeed\*((pi.run)?2.0f:1.0f);

}

private void FixedUpdate()

{

//刚体相关在这里设置

rigid.position += movingVec \* Time.fixedDeltaTime;

//rigid.velocity = new Vector3(movingVec.x, rigid.velocity.y, movingVec.z);

}

使用球星插值优化 角色转向

if(pi.Dmag>0.1f)

{

model.transform.forward = Vector3.Slerp(model.transform.forward, pi.Dright \* transform.right + pi.Dup \* transform.forward, 0.2f);

}

线性插值优化 动画过渡

anim.SetFloat("forward", pi.Dmag\* Mathf.Lerp(anim.GetFloat("forward"),targetRunMuti,0.5f));

斜角行走速度过快问题解决

使用方法 修改 xy

private Vector2 SquareToCircle(Vector2 input)

{

Vector2 v = Vector2.zero;

v.x = input.x \* Mathf.Sqrt(1 - (input.y \* input.y) / 2.0f);

v.y = input.y \* Mathf.Sqrt(1 - (input.x \* input.x) / 2.0f);

return v;

}

跳跃实现

增加jump新号 根据输入信号控制变量

bool tempJump = Input.GetKey(KeyB);

if(tempJump!=lastJump && tempJump==true)

{

jump = true;

}

else

{

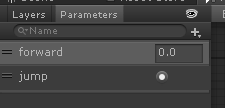
jump = false;

}

lastJump = tempJump;

播放跳跃动画

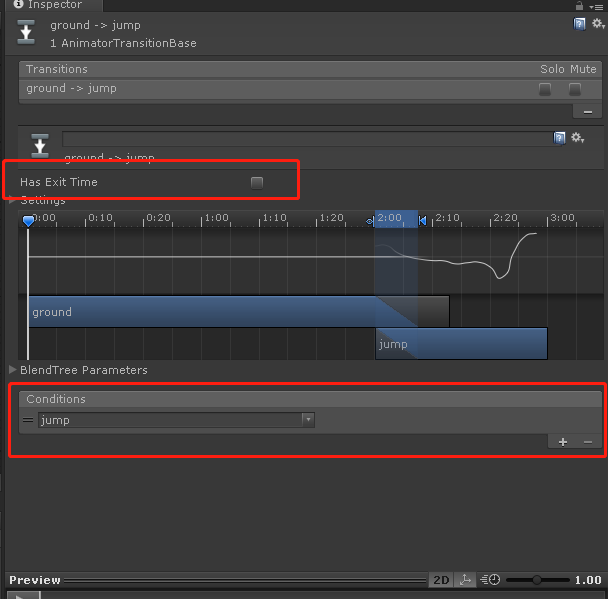
动画控制器增加一个 jump状态 动画控制器增加一个参数 trigger



Jump 与 ground 之间状态连接 起来

设置ground转换到jump的一些属性

去掉勾选 Has Exit Time 增加条件



玩家控制代码 满足条件播放jump动画

if(pi.jump)

{

anim.SetTrigger("jump");

}

修正连跳问题 动画trigger清空

ground 动画状态挂载一个 FSMClearSignal 脚本 改脚本继承StateMachineBehaviour

定义一个需要清空的数组

public string[] clearAtEnter;

public string[] clearAtExit;

// OnStateEnter is called when a transition starts and the state machine starts to evaluate this state

override public void OnStateEnter(Animator animator, AnimatorStateInfo stateInfo, int layerIndex) {

foreach (var signal in clearAtEnter)

{

animator.ResetTrigger(signal);

}

}

避免进入了 ground动画中 之前的trigger还在生效 enter的时候清空之前的signal

人物在跳跃时候 输入控制不能进行其他操作 比如移动等 这时候需要在 jump动画状态增加脚本

FSMOnEnter

public string[] onEnterMsg;

// OnStateEnter is called when a transition starts and the state machine starts to evaluate this state

override public void OnStateEnter(Animator animator, AnimatorStateInfo stateInfo, int layerIndex) {

foreach (var msg in onEnterMsg)

{

animator.gameObject.SendMessage(msg);

}

}

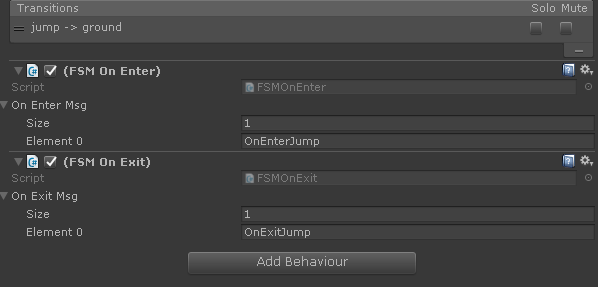
使用SendMessage需要在对应的animator上挂载脚本 包含对应msg的方法 这样显得比较麻烦

所以使用 SendMessageUpwards方法

animator.gameObject.SendMessage(msg);

这样可以直接在 动画控制脚本中 增加对应的回调方法

同样jump执行完成 也可以增加回调 通知输入控制器



public void OnEnterJump()

{

print("OnEnterJump");

}

public void OnExitJump()

{

print("OnExitJump");

}

当然 SendMessage不是最好的方法 可以使用动画事件

跳起时候 增加变量 控制角色继续水平方向位移

private bool lockPlanar; //控制水平方向是否位移

if(lockPlanar==false)

{

planarVec = pi.Dmag \* model.transform.forward \* walkSpeed \* ((pi.run) ? 2.0f : 1.0f);

}

跳起角色向上移动的处理

增加一个向上的向量

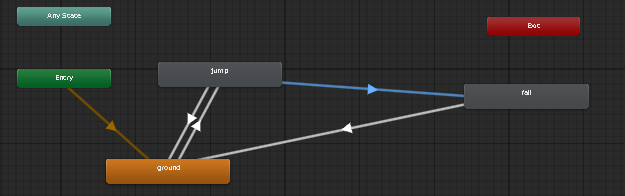
在fixedupdate中更新角色的坐标

rigid.position += planarVec \* Time.fixedDeltaTime;

rigid.position += jumpThrust;

jumpThrust = Vector3.zero;

角色跳起后下落动画 下落动画拖进角色控制器



修改jump的状态转移顺序



角色可以播放下落动画

但是存在问题 下落动画一结束 就会播放走路动画 加入角色还未落地就会很奇怪

需要判断角色是否落地

使用一个虚拟的碰撞胶囊体来检测是否跟地面碰撞

Unity提供了对应的接口 Physics.OverlapCapsule(point1, point2, radius,LayerMask.GetMask("Ground"));

为了只检测地板 需要增加layer层

public class OnGroundSenser : MonoBehaviour {

public CapsuleCollider capCol;

private Vector3 point1;

private Vector3 point2;

private float radius;

private void Awake()

{

radius = capCol.radius;

}

// Update is called once per frame

void FixedUpdate () {

point1 = transform.position + transform.up\*radius;

point2 = transform.position + transform.up \* capCol.height - transform.up \* radius;

Collider[] outputCols = Physics.OverlapCapsule(point1, point2, radius,LayerMask.GetMask("Ground"));

if (outputCols.Length!=0)

{

print("collision");

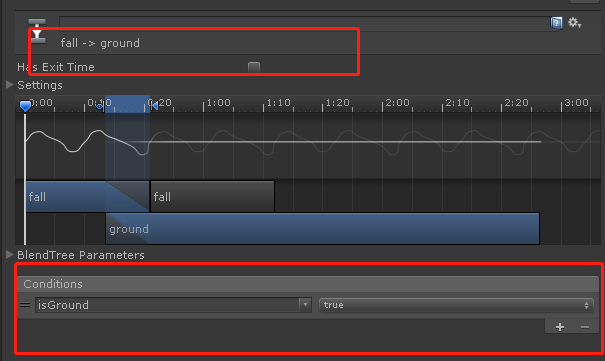
}

}

}

连接动画控制器中 fall到ground之间的状态

增加bool 参数 为ture的时候 fall转换为ground



这样在碰撞检测代码处增加处理

Collider[] outputCols = Physics.OverlapCapsule(point1, point2, radius,LayerMask.GetMask("Ground"));

if (outputCols.Length!=0)

{

SendMessageUpwards("IsGround");

}

else

{

SendMessageUpwards("IsNotGround");

}

动画控制代码增加对应的方法监听

public void IsGround()

{

print("IsGround");

anim.SetBool("isGround", true);

}

public void IsNotGround()

{

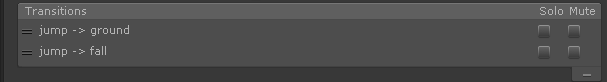
print("IsNotGround");

anim.SetBool("isGround", false);

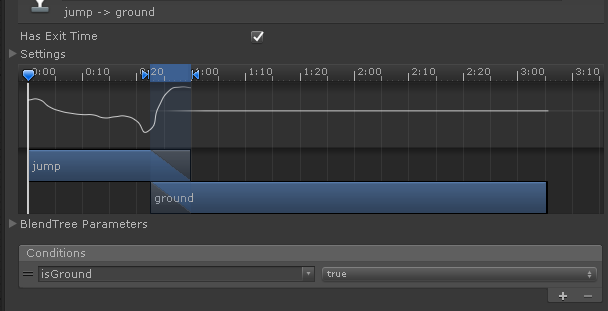
}

Jump切换到 fall 跟ground两个状态 优先判断是否着地 再判断fall

修改之前的判断顺序



Jump到ground之间 增加bool的控制



下落失去速度问题

角色失去了 水平方向的速度 原因在于 之前的跳起动画结束之后 进行的设置

现在跳起之后还有 ground和fall状态 可以直接在ground进入状态重新设置

删去之前的 OnExitJump

增加 OnEnterGround

public void OnEnterGround()

{

pi.inputEnabled = true;

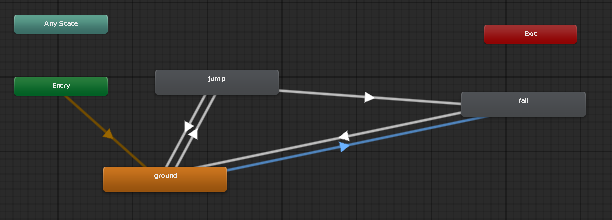
lockPlanar = false;

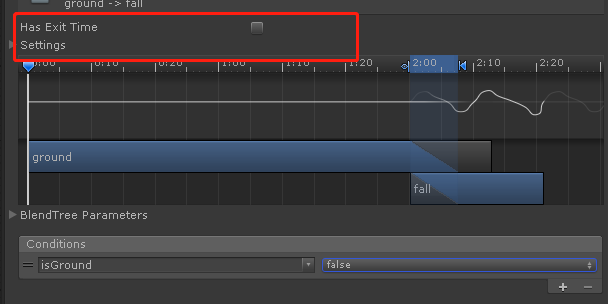
print("OnEnterGround");

}

角色直接走到高处跳下 不播放fall动画

需要增加 ground到fall之间的转换 增加转换条件 isGround为false



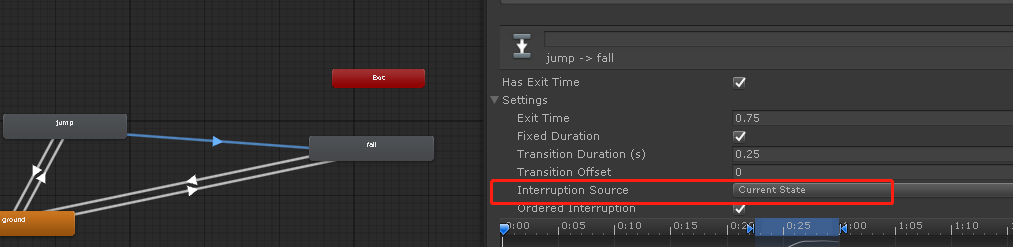


设置之后 角色走到高处掉落 也会丢失水平方向的速度

在fall动画增加fsmonenter控制

进入fall状态 锁死玩家控制

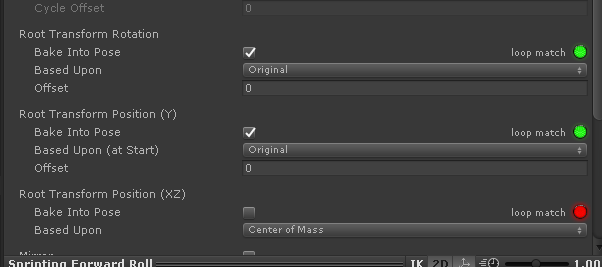
角色跳起 高度过高时候 已经落地还在播放 fall的问题



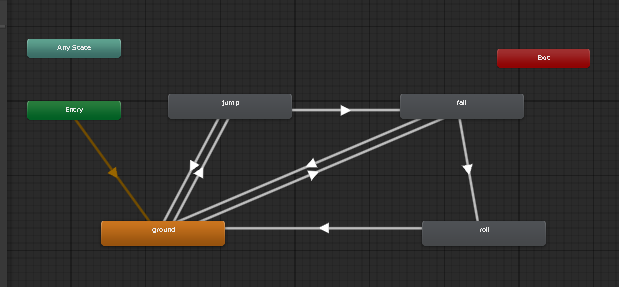
当已经着地 还在播放fall时候 需要中断回到jump状态到ground

前滚翻

增加一个前翻滚动画 动画的中心远离模型中心 可以修改模型相关参数



动画拖入状态控制器中

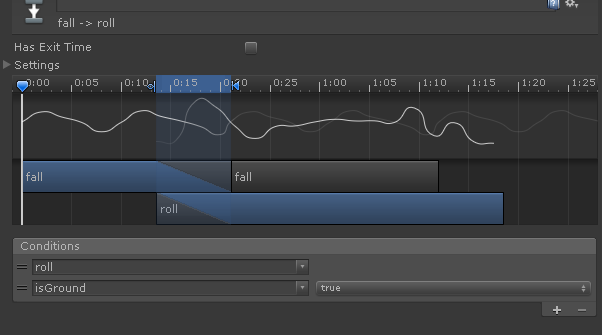


Fall 可以切换到 roll状态 roll也可以切换到ground

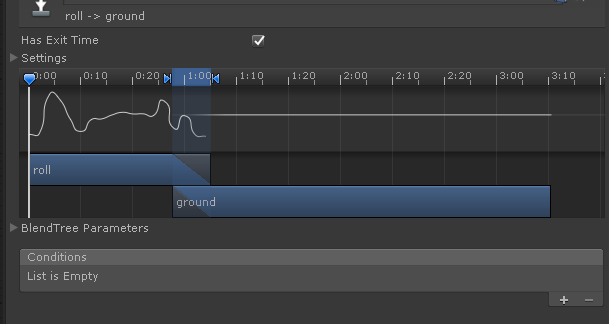
增加一个trigger参数 roll

Fall到roll ground 两个状态 设置优先顺序 先判断roll

转换状态条件 一个是roll 一个是 isground



Roll切换到ground



增加测试roll

if(rigid.velocity.magnitude>5)

{

anim.SetTrigger("roll");

}

需求是类似黑魂的控制 慢走时候 按J键是前滚翻 快走时候是跳跃落下前滚翻

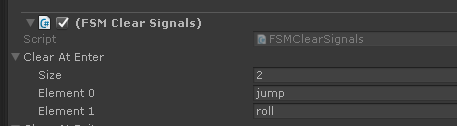
翻滚状态相关切换

慢走前滚翻

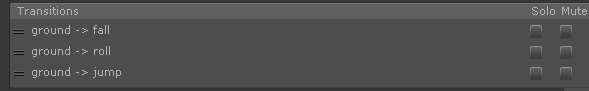
Onground 转到 roll 切换条件是 jump新号触发

Forward速度小于1.1

Ground进入的时候 也要清空roll信号



Ground状态转换检测优先级调整



修改roll触发条件

if(rigid.velocity.magnitude>1.0f)

{

anim.SetTrigger("roll");

}

触发roll 前翻滚时候 也要锁死角色控制以及位移

所以 enterroll 增加触发事件

同时给前滚翻增加一个 冲量

public void OnEnterRoll()

{

pi.inputEnabled = false;

lockPlanar = true;

jumpThrust = new Vector3(0, rollVelocity, 0);

print("OnEnterRoll");

}

这里把刚体位移相关方法修改

private void FixedUpdate()

{

//刚体相关在这里设置

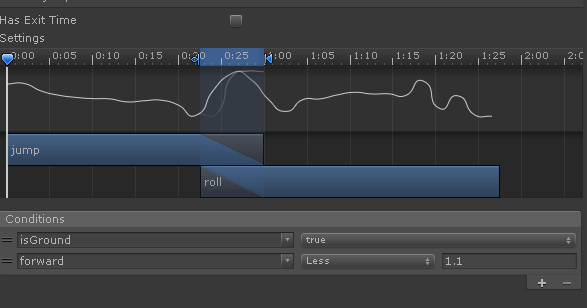
rigid.velocity = new Vector3(planarVec.x, rigid.velocity.y, planarVec.z)+jumpThrust;

jumpThrust = Vector3.zero;

}

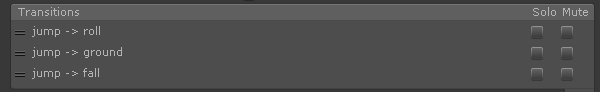
跳起后进行翻滚

增加jump到roll的状态转换



条件转换控制

Jump状态转换优先级



增加后跳动画

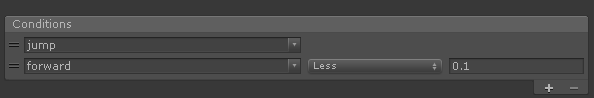
动画增加到动画控制器 jab

ground到 jab jab也可以转到ground

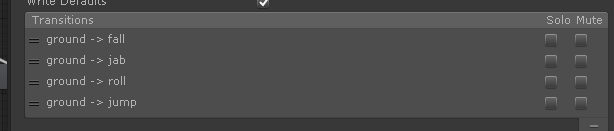
当角色静止时候 按j 执行后跳操作

此时forward为0

设置ground到jab的转换条件



此时ground有四个状态转换 调整优先级



同样在 jab动画状态 绑定脚本 设置进入状态时候 锁死角色移动 增加向后冲量

public void OnEnterJab()

{

pi.inputEnabled = false;

lockPlanar = true;

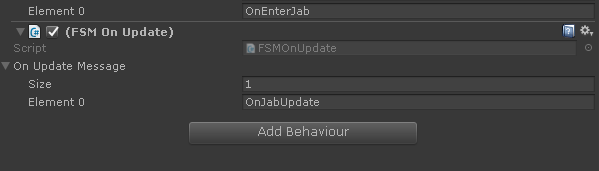
jumpThrust =model.transform.forward\*(-jabVeclocity);

print("OnEnterJab");

}

直接在enter里面设置有点僵硬 可以使用动画fsmupdate方法

在jab动画挂载脚本 FSMOnUpdate



增加对应的方法

public void OnJabUpdate()

{

jumpThrust = model.transform.forward \* anim.GetFloat("jabVelocity");

}

这里使用了动画的 curve功能

在动画控制器增加参数 jabVeclocity

在后跳动画源文件 curve增加一个key

设置value为-3.0 设置曲线

代码中获取anim.GetFloat("jabVelocity") 修改角色后跳的位置改变

相机运动

增加四个按键两个信号 跟最开始的wasd一样

相机控制使用 箭头方向键 左右控制水平方向 上下控制垂直方向

相机信号不做smooth处理

Jup = Input.GetKey(JKeyUp) ? 1.0f : 0 - (Input.GetKey(JKeyDown) ? 1.0f : 0);

Jright = Input.GetKey(JKeyRight) ? 1.0f : 0 - (Input.GetKey(JKeyLeft) ? 1.0f : 0);

相机运动 一般需要把相机放在角色当做子物体

所以 实现物体层级 创建一个空物体camera handle 放在角色脖子处 main camera当做他的子物体

坐标reset

这样移动playerhandle 相机水平方向运动

使用 camera handle 实现垂直方向运动

先做水平方向

增加脚本 CameraCotroller

public class CameraController : MonoBehaviour {

private GameObject playerHandle;

private GameObject cameraHandle;

public float horizontalSpeed = 100;

public PlayerInput pi;

// Use this for initialization

void Awake () {

cameraHandle = transform.parent.gameObject;

playerHandle = cameraHandle.transform.parent.gameObject;

}

// Update is called once per frame

void Update () {

playerHandle.transform.Rotate(Vector3.up, pi.Jright \* 10.0f \* Time.deltaTime\* horizontalSpeed);

}

}

垂直方向 注意角度控制 还有同位角问题

tempEulerX -= pi.Jup \* Time.fixedDeltaTime \* verticalSpeed;

tempEulerX = Mathf.Clamp(tempEulerX, -40, 30);

cameraHandle.transform.localEulerAngles = new Vector3(tempEulerX, 0, 0);

model.transform.eulerAngles = tempModelEuler;

镜头跟随 修改场景中物体与camera之间的层级关系 camera增加smoothdamp

private GameObject camera;

private Vector3 cameraDampVelocity;

//camera.transform.position = Vector3.Lerp(camera.transform.position, transform.position, 0.2f);

camera.transform.eulerAngles = transform.eulerAngles;

camera.transform.position = Vector3.SmoothDamp(camera.transform.position, transform.position, ref cameraDampVelocity, cameraDampValue);

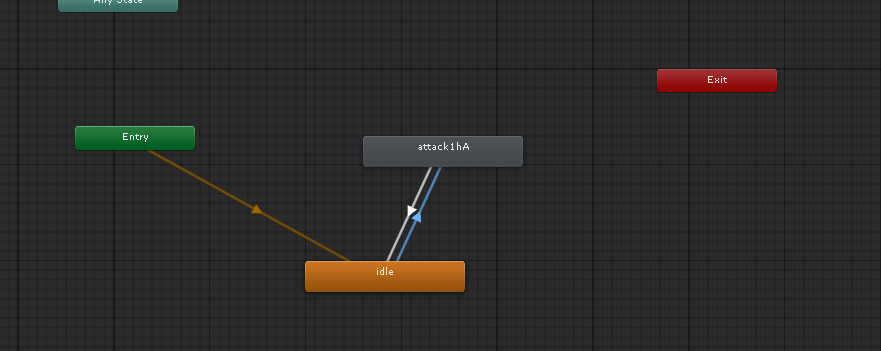
增加攻击动画

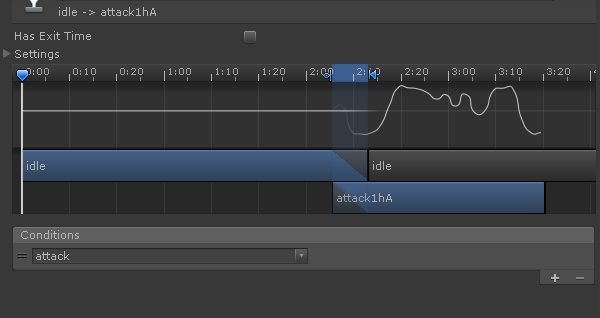
新建一个动画层级attack

攻击动画拖进状态机中

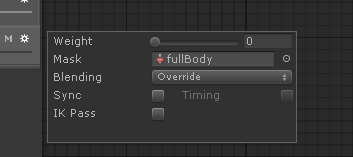
为了实现状态转换 增加触发器trigger参数 attack

闲置状态转为攻击状态 需要满足attack为true



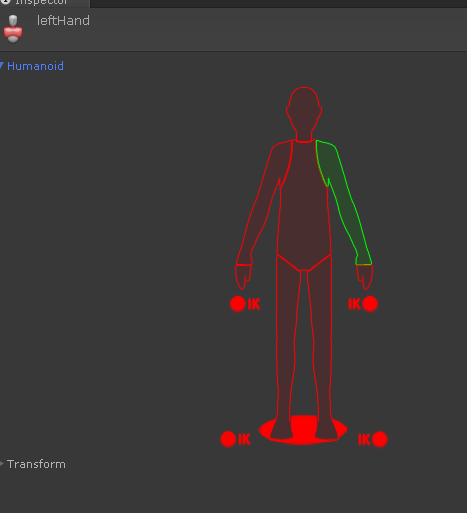


动画图层属性



Weight控制是否执行动画层

创建avatarMask



代码实现动画层级调整

进入attack attack的动画层级改为1 进入idle改为0

同时进入attack 锁死控制 进入idle放开并清掉attack新号

实现attack新号 以及上面

public bool attack;

private bool lastAttack;

bool tempAttack = Input.GetKey(KeyC);

if (tempAttack != lastAttack && tempAttack == true)

{

attack = true;

}

else

{

attack = false;

}

lastAttack = tempAttack;

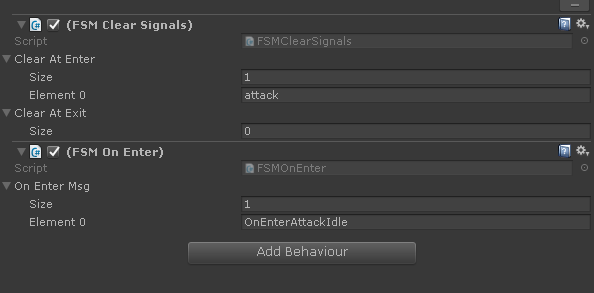
print(attack);

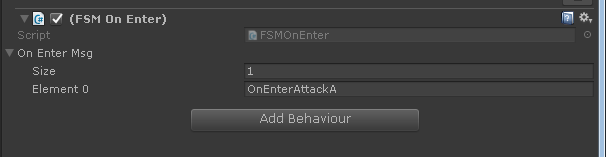
if(pi.attack)

{

anim.SetTrigger("attack");

}





public void OnEnterAttackA()

{

pi.inputEnabled = false;

lockPlanar = true;

anim.SetLayerWeight(anim.GetLayerIndex("attack"), 1.0f);

}

public void OnEnterAttackIdle()

{

pi.inputEnabled = true;

lockPlanar = false;

anim.SetLayerWeight(anim.GetLayerIndex("attack"), 0.0f);

}

增加攻击的冲量 以及移动时候不能攻击

跟后跳方法一样 使用动画曲线

public void OnUpdateAttackA()

{

jumpThrust = model.transform.forward \* anim.GetFloat("attack1hAVelocity");

}

到这里 角色在空中还是能攻击 增加控制条件 使得角色只有在地面才可以攻击

1. 判断当前是否是ground的动画状态 增加方法

private bool CheckState(string stateName,string layerName="Base Layer")

{

return anim.GetCurrentAnimatorStateInfo(anim.GetLayerIndex(layerName)).IsName(stateName);

}

然后判断trigger是否是地面 同时增加变量 控制 是否能攻击

canAttack 接受跳跃信号为false

进入地面为true

//限制攻击条件 动画状态ground

if (pi.attack && CheckState("ground") &&anim.GetBool("isGround") && canAttack)

{

anim.SetTrigger("attack");

}

最后遇到材质问题 当角色撞到墙壁不掉落

因此动态修改角色胶囊体的物理材质

进入地面 使用具有摩擦力的 离开地面使用没有摩擦力的

增加攻击状态B

导入动画资源 更新动画控制器