中南大学

《动画与游戏程序设计》

课程实验报告

实验名称 基于 Unity 的 3D 人形动画游戏项目实验

学生姓名 _____

学　　号 _____

专业班级 _____

指导教师 _____

学　　院 _____

目　录

一、实验概述

1.1 实验名称

基于 Unity 的 3D 人形动画游戏项目实验。

1.2 实验目的

本次实验的总的目的是通过本次实验建立一个 3D 人形动画游戏项目。理解 unity 中 3D 人形动画的设计过程。

利用 unity-chan 动画资源创建一个包含女孩空闲，走路，跑步，跳跃等动作的动画项目，学习 Unity 中 3D 人形动画的主要组成部分 3D 模型和动画的导入，匹配骨骼，创建 avatar，创建 animator controller，创建动作有限状态机，创建脚本控制角色动作的方法。

1.3 实验要求

动画中包含女孩空闲，走路，跑步，跳跃等动作。

1.4 开发步骤指导

1. 导入资源包 Unity-chan Model.unitypackage。
2. 创建地面 plane，选择 Assets 下面 UnityChan-Stage-Matieral 中选择一种材质拖到地面上。
3. 选择 Assets 下面 Models-untychan 模型（如果没有创建替身和骨骼匹配需要进行设置），将其拖入 Hierarchy 视图创建游戏角色对象，放到（0,0,0）。
4. 将 unitychan 游戏对象上的组件 Animator 和脚本（Script）都删除掉。可以参考视频里内容来做。（一定要注意，否则我们的动画会和原有的动画相互干扰！）
5. Hierarchy 视图中选中 Main Camera，位置（0,1,2），旋转（0,180,0），Inspector 中 Camera-Solid Color，背景选黑色。
6. Project 视图中新建 chanAnimation 文件夹，在其中创建 Animation Controller（player），将其添加到角色上。
7. 在 Animator 里，将动作拖进来，查看动作播放时，如果将角色拖入到播放器上，就会使角色动作播放，否则是一个光头细长人，或者部分肢体活动信息。

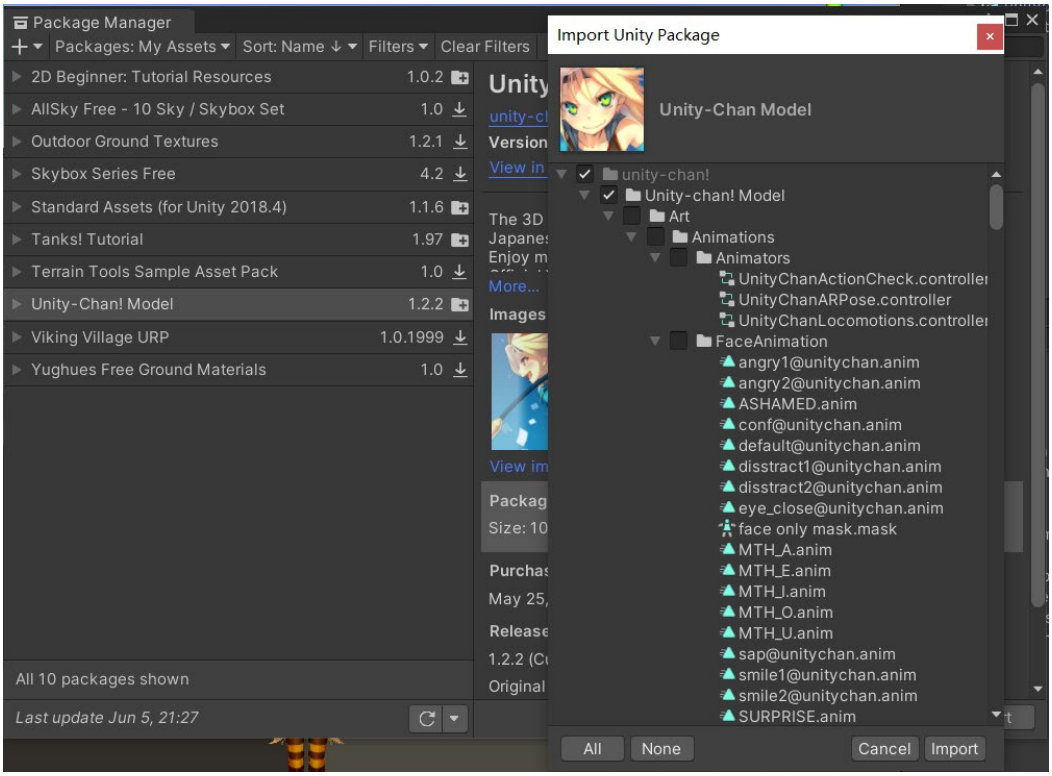
右键一个动作点 set as layer default state，就可以进入动画就进行播放。选择 Wait 动作演示。

8. 添加其他与环境相匹配的游戏对象，场景等，丰富游戏功能。

二、实验步骤

1、导入实验资源

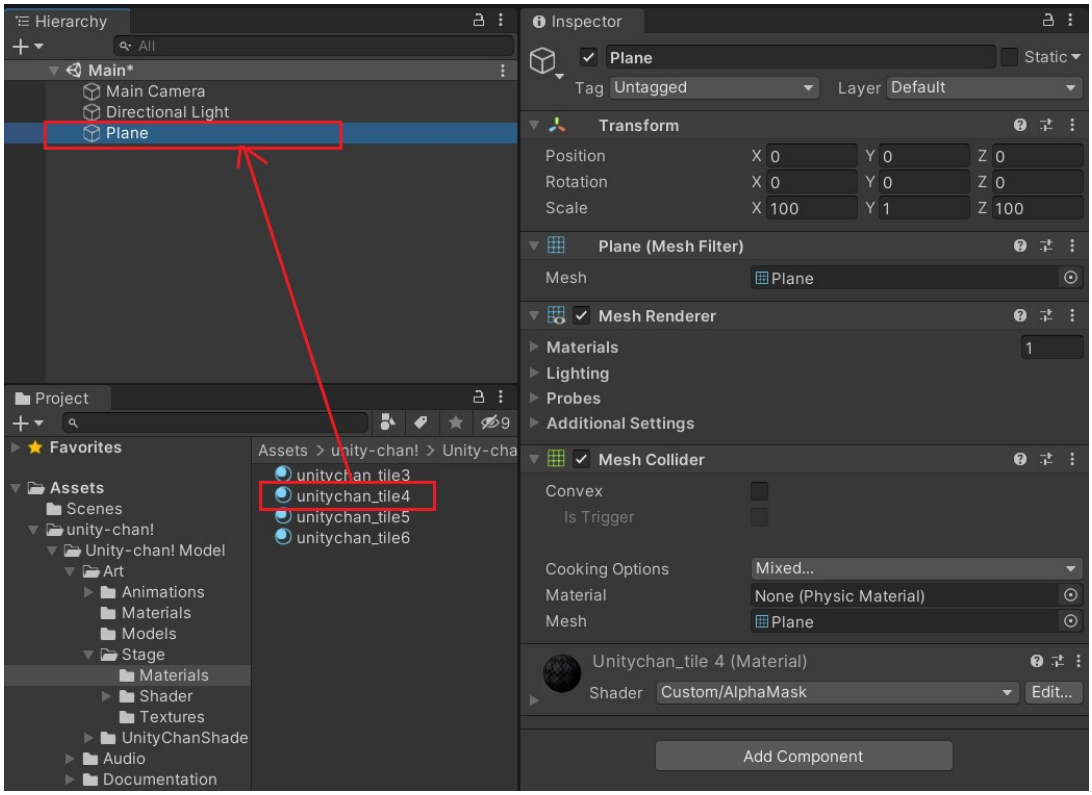
首先在 Assets Store 将 unity-chan 资源添加到我的资源，然后在 Unity 中通过 Package Manager 将资源包中所有内容导入。



2、创建地面

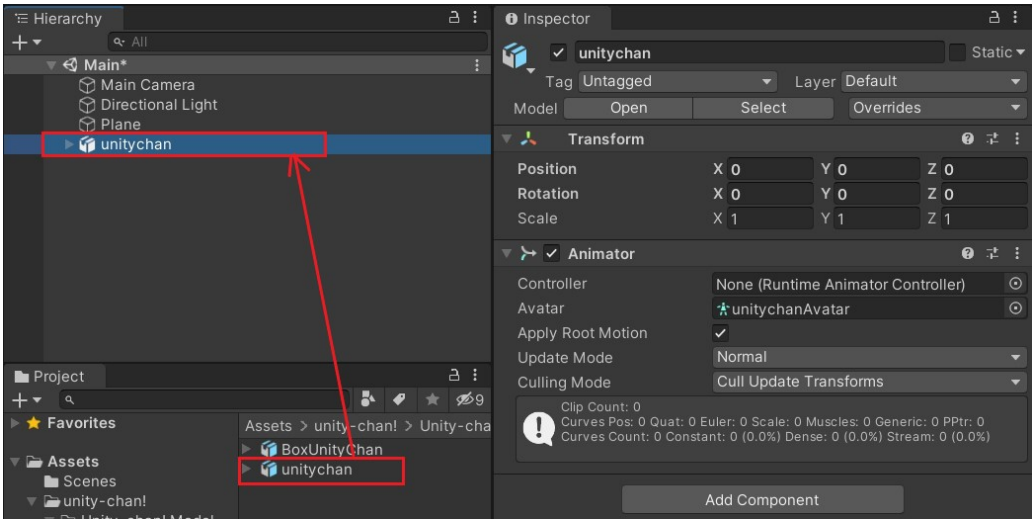
右键创建 Plane 对象，Position 设置为(0, 0, 0)，Rotation 设置为(0, 0, 0)，Scale 设置为(100, 1, 100)。

在 Project 面板的 unity-chan!/unity-chan!Model/Art/Stage/Materials 文件夹中选取一种材质设置为 Plane 的材质，我采用的是 unitychan_tile4。



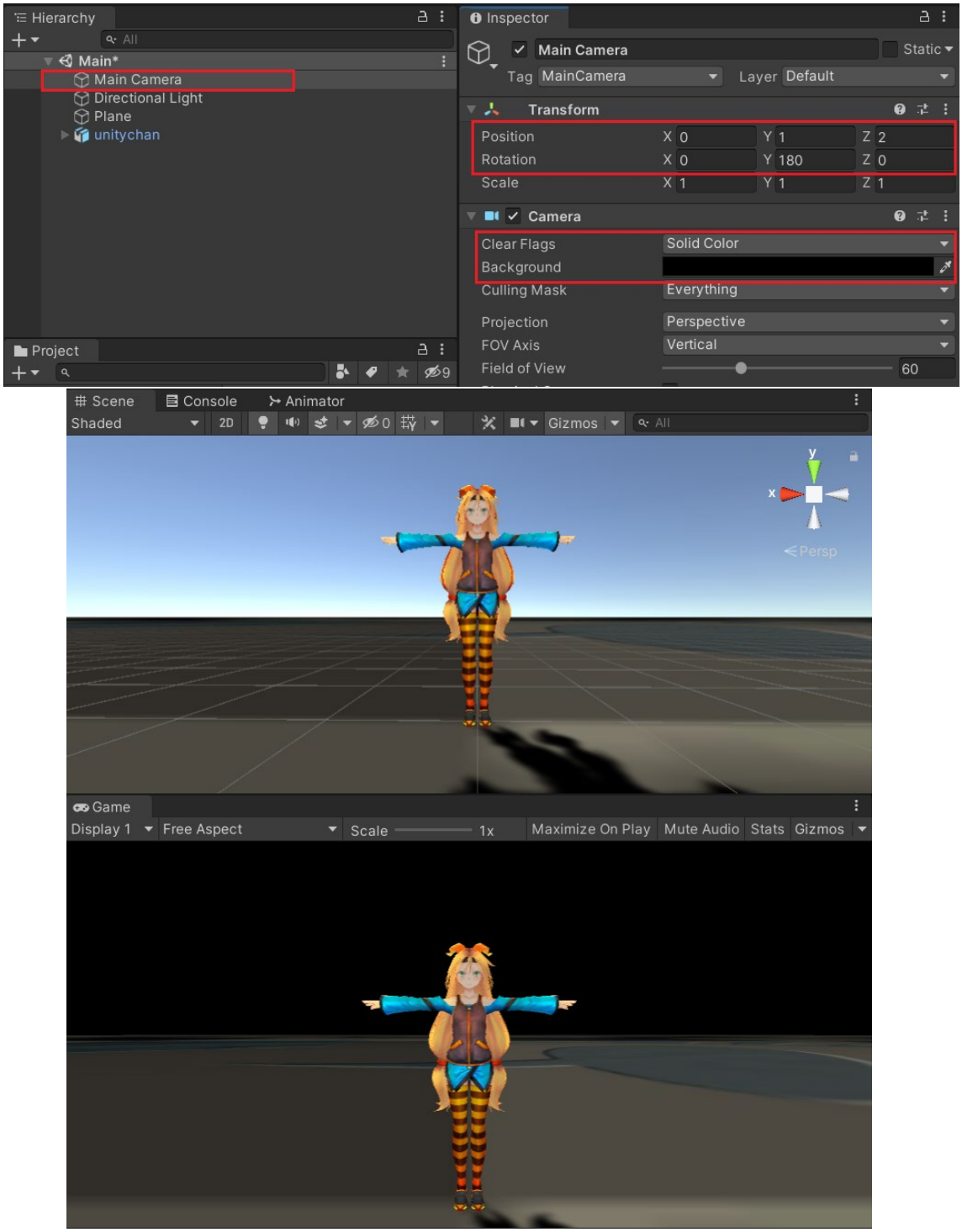
3、创建 Chan

在 Project 面板中，从 unity-chan!/unity-chan!Model/Art/Models 文件夹中将 unitychan 拖拽至 Hierarchy 面板创建 unitychan 对象，Position 设置为(0, 0, 0)。



4、设置相机

在 Hierarchy 视图中选中 Main Camera。在 Transform 组件中将其 Position 设置为(0, 1, 2)，Rotation 设置为(0, 180, 0)；在 Camera 组件中，将 Clear Flags 设置为 Solid Color，Background 设置为(0, 0, 0)。

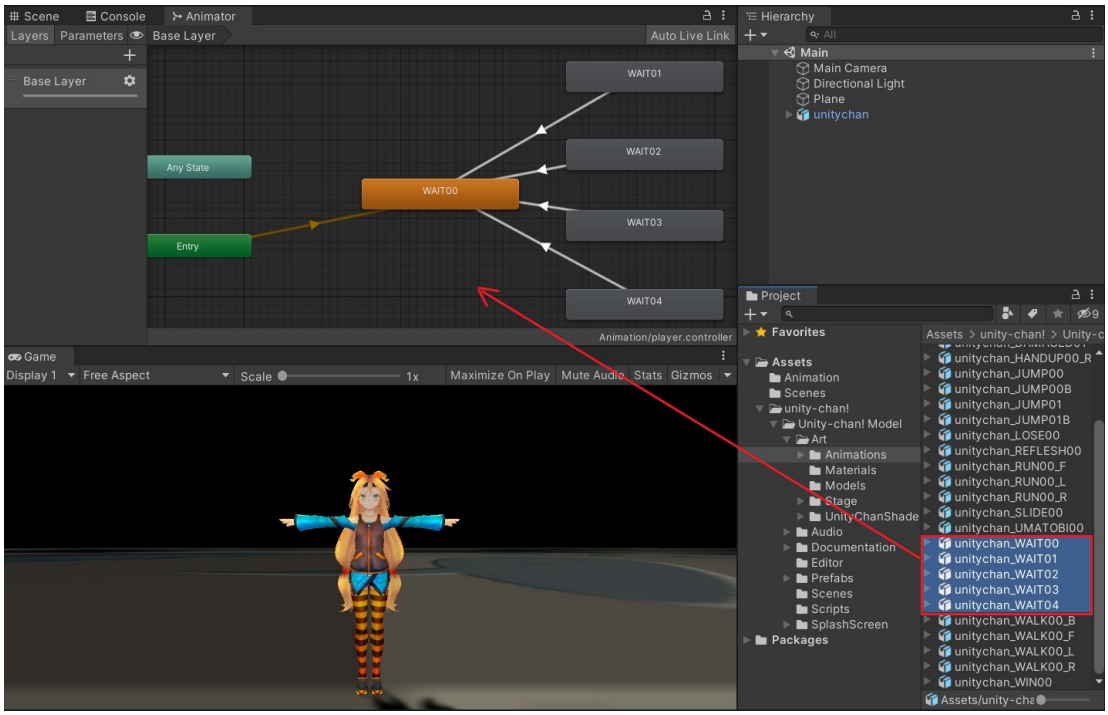


5、为 unitychan 设置动画

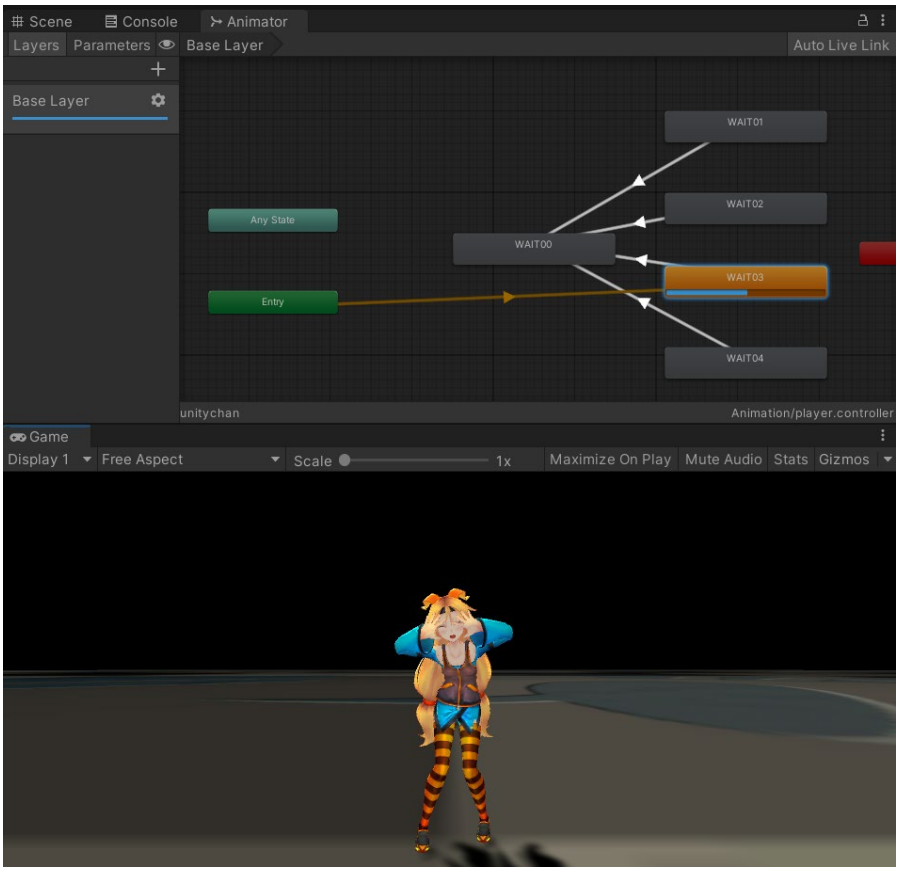
在 Project 面板中的 Assets 文件夹下新建 Animation 文件夹，在其中创建名为 player 的 Animation Controller，将其添加到 unitychan 对象的 Animator 组件的 Controller 属性上。并取消勾选 Animator 组件的 Apply Root Motion 选项。

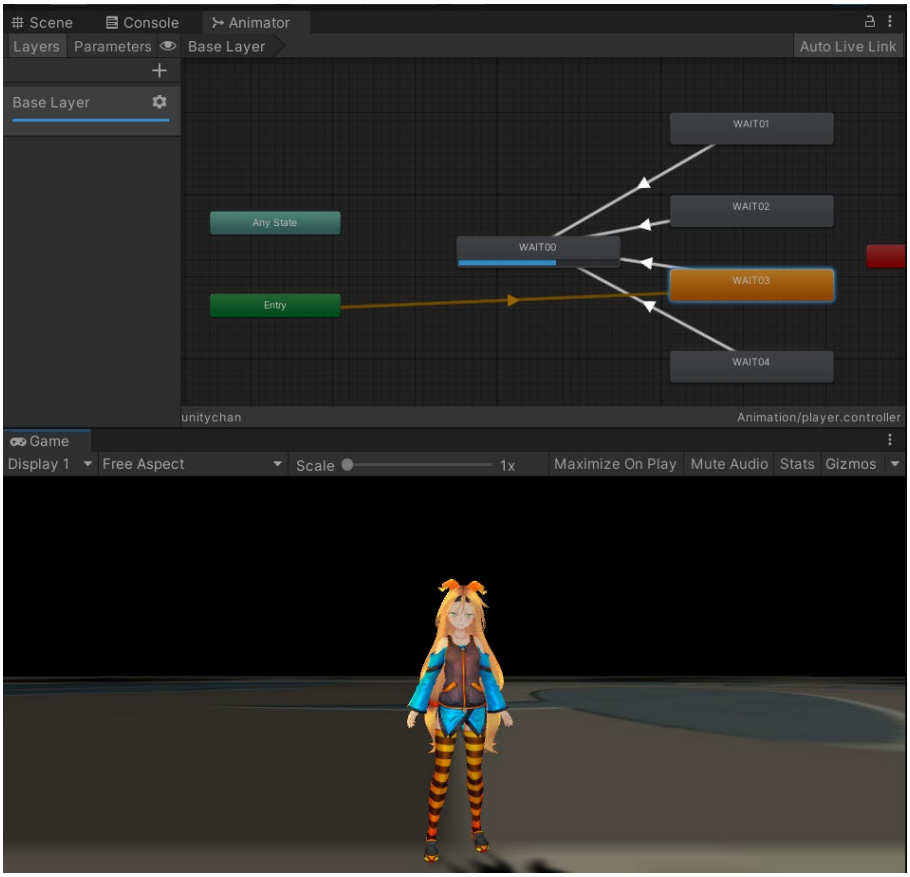
（1）添加等待动画

在 Project 面板中的 unity-chan!/unity-chan!Model/Art/Animations 将下图 5 个动画添加到控制器。并将 WAIT00 作为 Default State，其他 4 个都向 WAIT00 过渡。



然后分别将其他 4 个 WAIT01~WAIT04 设置为 Default State，运行后观察动画执行情况，发现初始会播放 Default State，然后按照 Transition 向指定 State 过渡。例如，将 WAIT03 设置为 Default State，运行游戏。如下，人物先执行 WAIT03 动画，执行完后，沿着过渡继续执行 WAIT00 动画，并最终循环播放 WAIT00，因为 WAIT00 未做任何过渡。





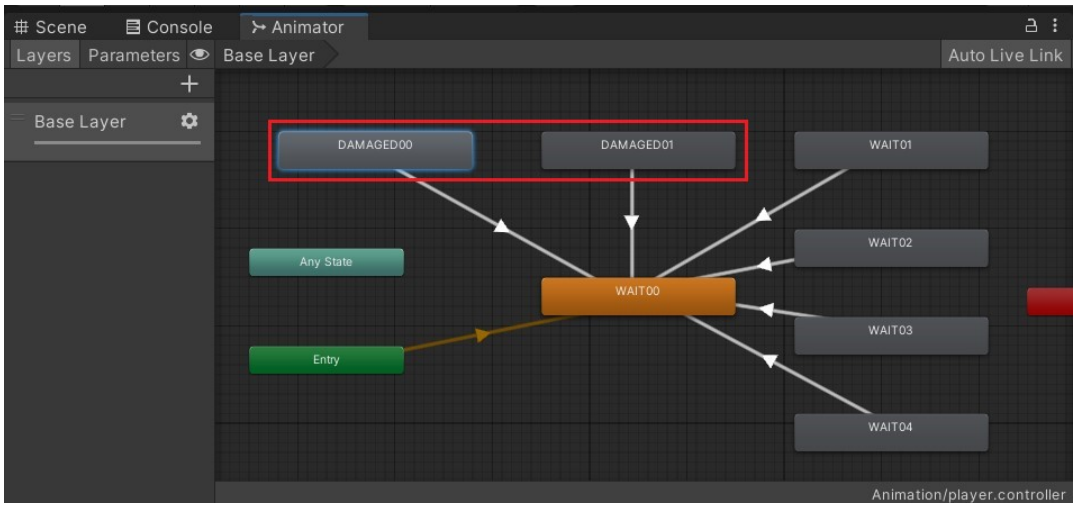
创建脚本 PlayerAnimationController.cs，并将其添加到 unitychan 对象上，用

以控制动画的转换。默认为 WAIT00，当按下“1”时，播放 WAIT01，按下“2”时，播放 WAIT02，依此类推到 WAIT04。脚本内容如下：

```
1. using System.Collections;
2. using System.Collections.Generic;
3. using UnityEngine;
4.
5. public class PlayerAnimationController : MonoBehaviour
6. {
7.     public Animator m_Animator;
8.
9.     // Start is called before the first frame update
10.    void Start()
11.    {
12.        // 获取动画控制器
13.        m_Animator = GetComponent<Animator>();
14.    }
15.
16.    // Update is called once per frame
17.    void Update()
18.    {
19.        if (Input.GetKeyDown("1"))
20.        {
21.            m_Animator.Play("WAIT01", -1, 0f);
22.        }
23.        else if (Input.GetKeyDown("2"))
24.        {
25.            m_Animator.Play("WAIT02", -1, 0f);
26.        }
27.        else if (Input.GetKeyDown("3"))
28.        {
29.            m_Animator.Play("WAIT03", -1, 0f);
30.        }
31.        else if (Input.GetKeyDown("4"))
32.        {
33.            m_Animator.Play("WAIT04", -1, 0f);
34.        }
35.    }
36.}
```

（2）添加受到伤害的动画

与上面相同，在同样的文件夹下找到动画 DAMAGED00 和 DAMAGED01，将其拖拽到 Animator 面板中，并使它们都向 WAIT00 过渡。

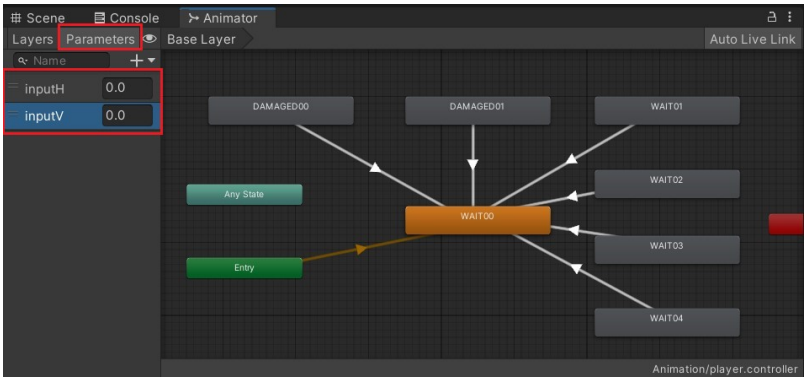


修改 PlayerAnimationController 脚本，添加受伤动画的播放控制，使用鼠标左键点击 unitychan 时，unitychan 随机播放两种受伤动画中的一种。这里只需要更改 Update()函数：

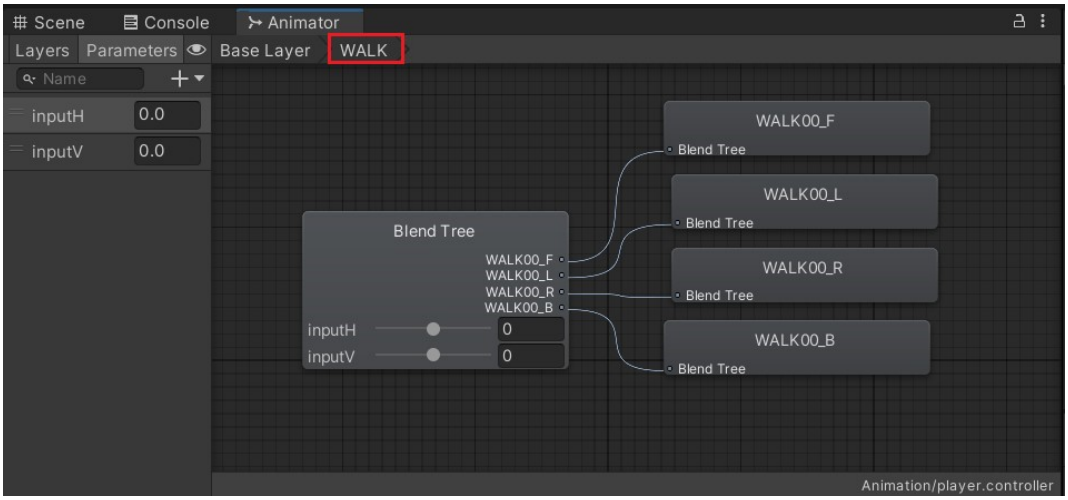
```
1. public class PlayerAnimationController : MonoBehaviour
2. {
3.     ……
4.     // Update is called once per frame
5.     void Update()
6.     {
7.         ……
8.         if (Input.GetMouseButtonDown(0))
9.         {
10.            int type = Random.Range(0, 2);
11.            m_Animator.Play("DAMAGED0" + type, -1, 0f);
12.        }
13.    }
14.}
```

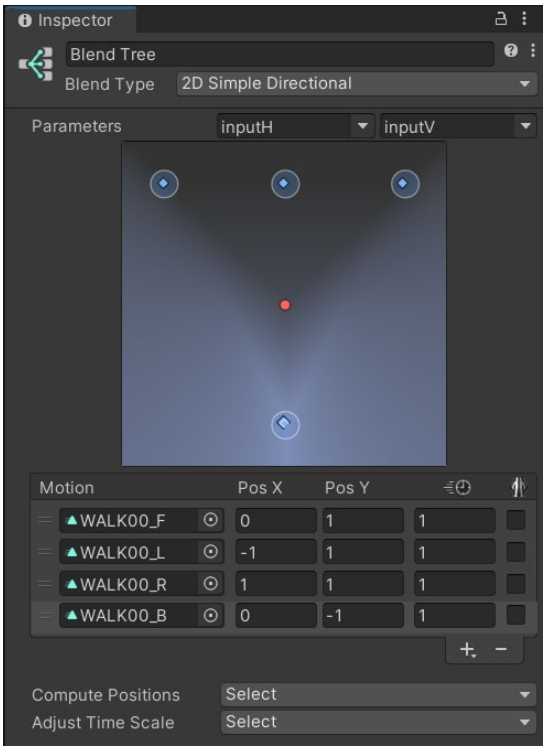
（3）添加移动的动画

首先为动画控制器添加两个参数 inputH、inputV，分别用于接收前后左右的输入。

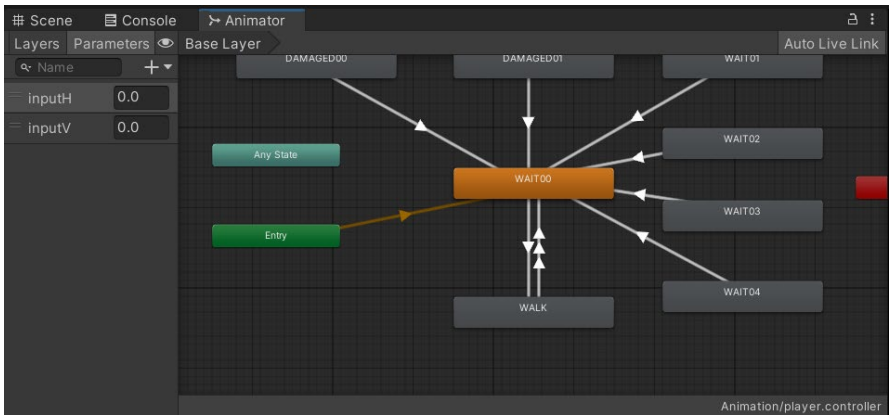


在 Animator 面板创建一个 Blend Tree 重命名为 WALK，用于控制角色的移动动画。为 WALK 添加 4 个动画，分别为前后左右移动的动画，参数设置如下：

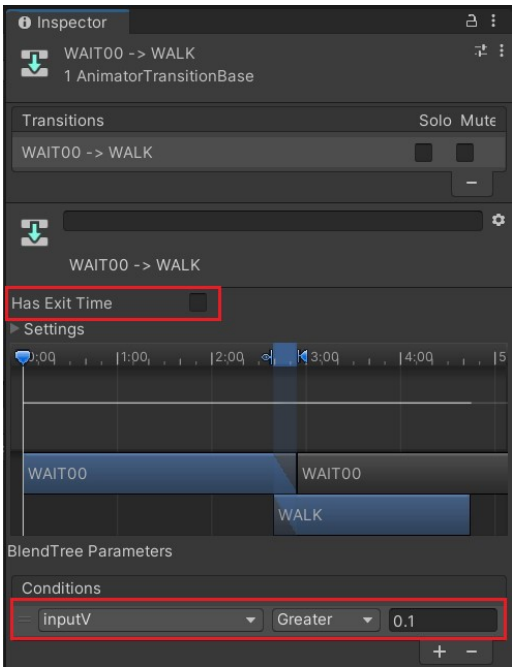




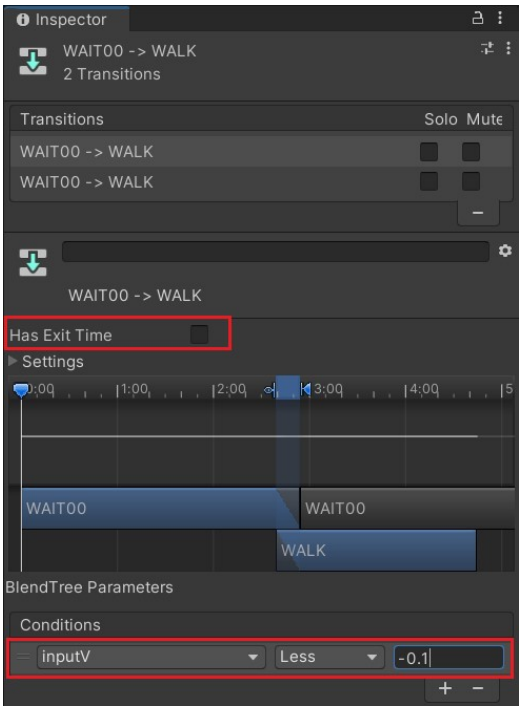
将 WALK 过渡到 WAIT00，添加 3 个两者间的过渡：



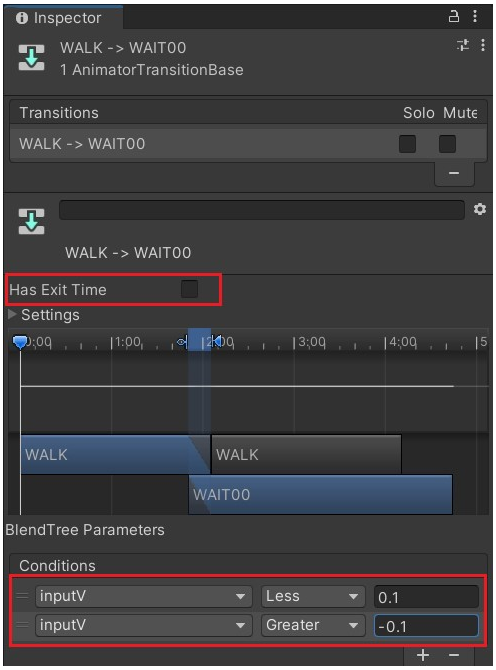
① WALK 到 WAIT00：向前或者左右走。设置如下：



② WALK 到 WAIT00：向后走。设置如下：

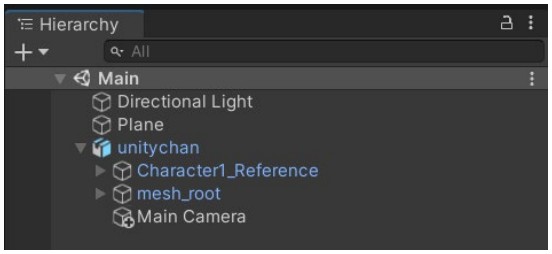


③ WAIT00 到 WALK：从行走到停下。设置如下：

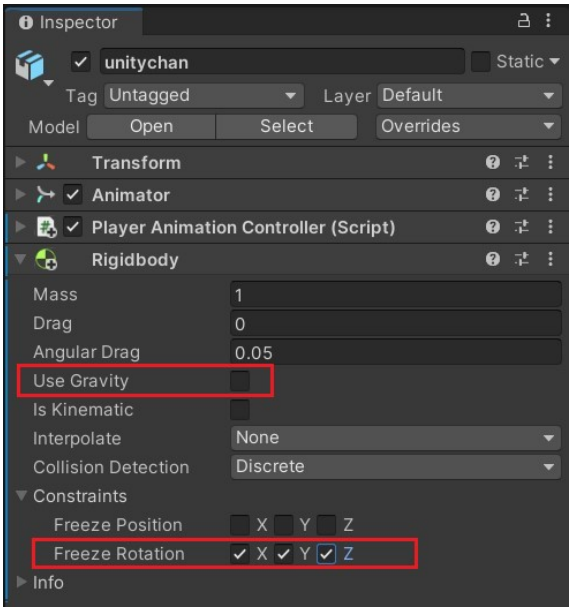


（4）实现角色移动

首先将Main Camera设置为unitychan对象的子对象，使得unitychan移动时，镜头也会随之移动。



第二，给 unitychan 添加 Rigidbody 组件，以便控制移动。取消勾选该组件的 Use Gravity，勾选 Freeze Rotation 的 X、Y、Z 选项，这样角色就不会转身。



打开 PlayerAnimationController 脚本，更改脚本内容，实现角色移动。做出如下修改：

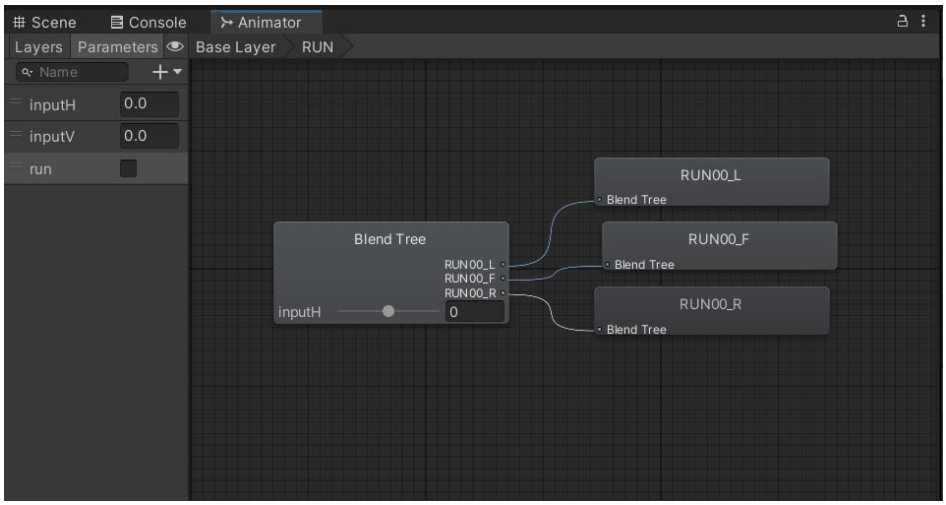
```
1. public class PlayerAnimationController : MonoBehaviour
2. {
3.     public Animator m_Animator;
4.     public Rigidbody m_ChanRigidbody;
5.
6.     private float inputH;
7.     private float inputV;
8.
9.     // Start is called before the first frame update
10.    void Start()
11.    {
12.        // 获取动画控制器
13.        m_Animator = GetComponent<Animator>();
14.        m_ChanRigidbody = GetComponent<Rigidbody>();
15.    }
16.
17.    // Update is called once per frame
18.    void Update()
19.    {
20.        ……
21.
22.        // 接收前后左右移动的输入
23.        inputH = Input.GetAxis("Horizontal");
24.        inputV = Input.GetAxis("Vertical");
25.
26.        // 根据输入控制动画
27.        m_Animator.SetFloat("inputH", inputH);
28.        m_Animator.SetFloat("inputV", inputV);
29.
30.        // 控制角色移动
31.        float moveX = inputH * 20f * Time.deltaTime;
32.        float moveZ = inputV * 50f * Time.deltaTime;
33.        if (moveZ <= 0)
34.        {
35.            // 如果没有向前走，那么不能左右移动
36.            moveX = 0;
37.        }
38.        m_ChanRigidbody.velocity = new Vector3(moveX, 0f, moveZ);
39.    }
40.}
```

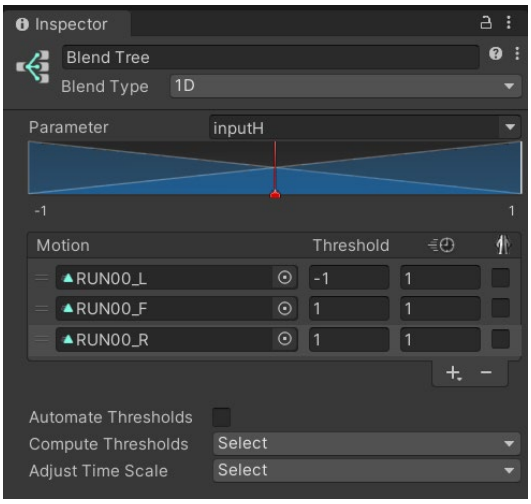
运行游戏，unitychan 便能在场景中移动了，并且镜头会跟随其移动。

（5）添加奔跑动画

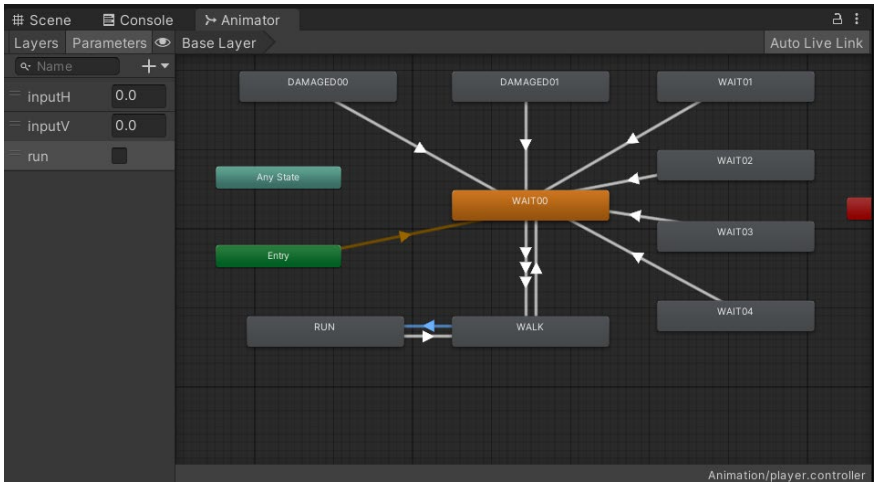
首先为动画控制器添加一个 bool 参数 run，用于控制奔跑动画。

在 Animator 面板创建一个 Blend Tree 重命名为 RUN，用于控制角色的奔跑动画。为 RUN 添加 3 个动画，分别为向前、向左、向右跑的动画，参数设置如下：

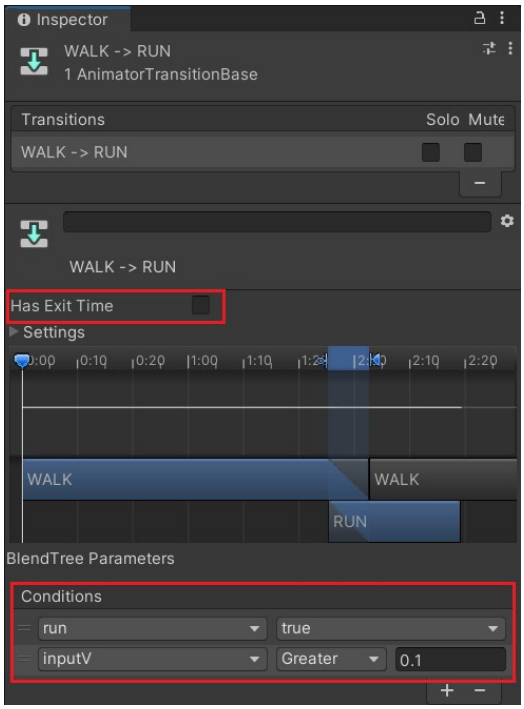




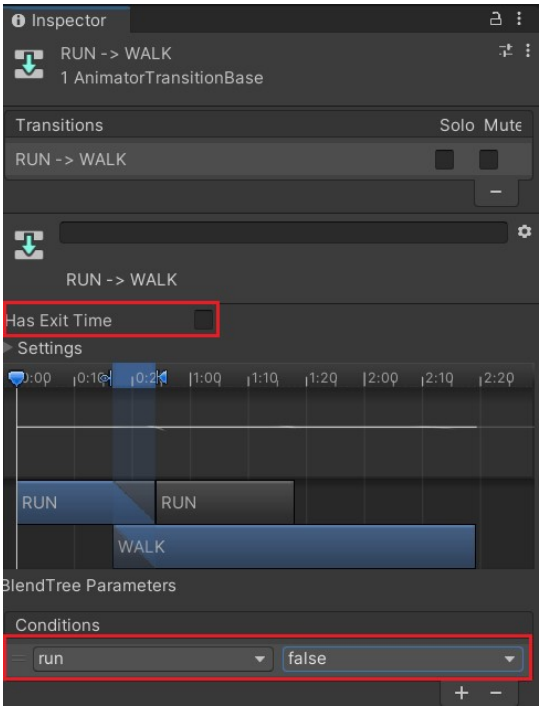
在 RUN 和 WALK 之间，添加 2 个两者间的过渡：



① WALK 过渡到 RUN，参数如下：



② RUN 过渡到 WALK，参数如下：



（6）控制由行走到奔跑

打开 PlayerAnimationController 脚本，更改脚本内容，实现角色奔跑。当按下键盘左侧的 Shift 键时即可奔跑。做出如下修改：

```
1. using System.Collections;
2. using System.Collections.Generic;
```

```
3. using UnityEngine;
4.
5. public class PlayerAnimationController : MonoBehaviour
6. {
7.     public Animator m_Animator;
8.     public Rigidbody m_Chan;
9.
10.    private float inputH;
11.    private float inputV;
12.    private bool run;
13.
14.    // Start is called before the first frame update
15.    void Start()
16.    {
17.        // 获取动画控制器
18.        m_Animator = GetComponent<Animator>();
19.        m_Chan = GetComponent<Rigidbody>();
20.        run = false;
21.    }
22.
23.    // Update is called once per frame
24.    void Update()
25.    {
26.        ……
27.        if (Input.GetKey(KeyCode.LeftShift))
28.        {
29.            run = true;
30.        }
31.        else
32.        {
33.            run = false;
34.        }
35.
36.        // 接收前后左右移动的输入
37.        inputH = Input.GetAxis("Horizontal");
38.        inputV = Input.GetAxis("Vertical");
39.
40.        // 根据输入控制动画
41.        m_Animator.SetFloat("inputH", inputH);
42.        m_Animator.SetFloat("inputV", inputV);
43.        m_Animator.SetBool("run", run);
44.
45.        // 控制角色移动
46.        float moveX = inputH * 20f * Time.deltaTime;
```

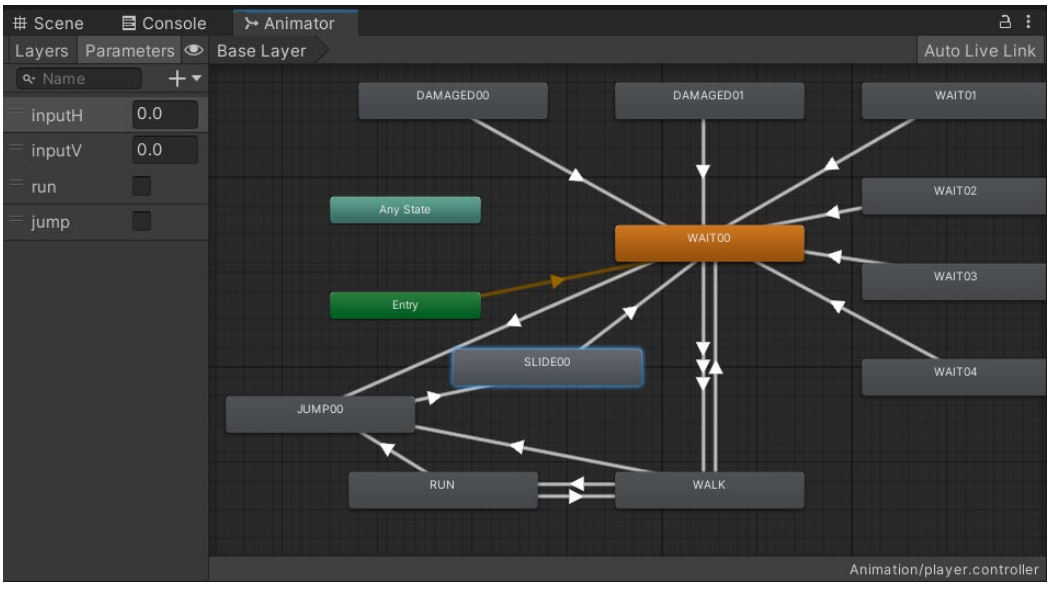
```
        float moveZ = inputV * 50f * Time.deltaTime;
        if (moveZ <= 0)
        {
            // 如果没有向前走，那么不能左右移动
            moveX = 0;
        }
        else
        {
            if (run)
            {
                moveX *= 3f;
                moveZ *= 3f;
            }
        }
        m_Chan.velocity = new Vector3(moveX, 0f, moveZ);
    }
}
```

运行游戏，当角色在向前或左右运动时，按住 Shift 即可实现奔跑。

（7）添加跳跃动画

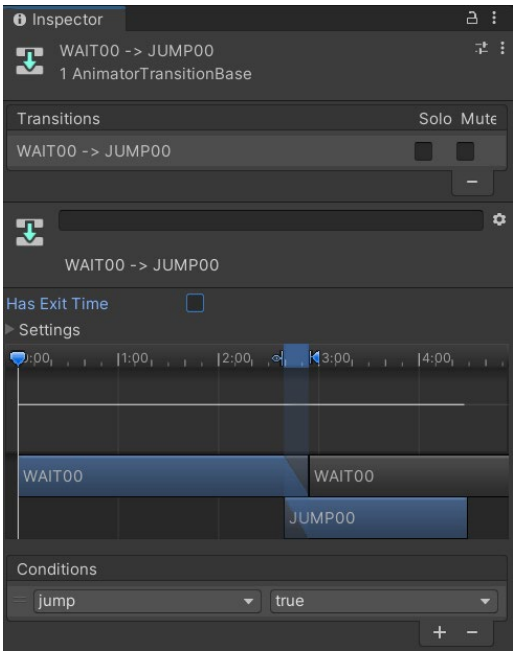
首先为动画控制器添加一个 bool 参数 jump，用于控制跳跃动画。

与第(1)、(2)步相同，在同样的文件夹下找到动画 JUMP00 和 SLIDE00，将它们拖拽到 Animator 面板中。并添加过渡：



过渡的设置如下：

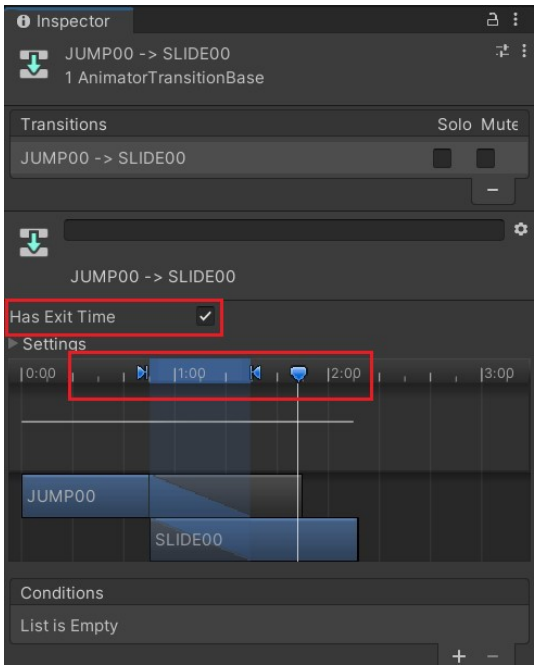
① WAIT00 过渡到 JUMP00，设置如下：



② WALK 过渡到 JUMP00，设置和①中一样；

③ RUN 过渡到 JUMP00，设置和①中一样；

④ JUMP00 到 SLIDE00，设置如下：



⑤ SLIDE00 过渡到 WAIT00，使用默认设置。

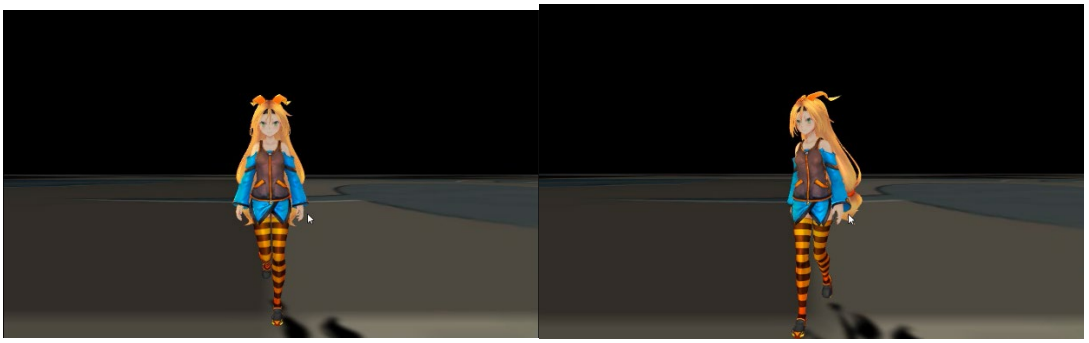
（8）控制跳跃

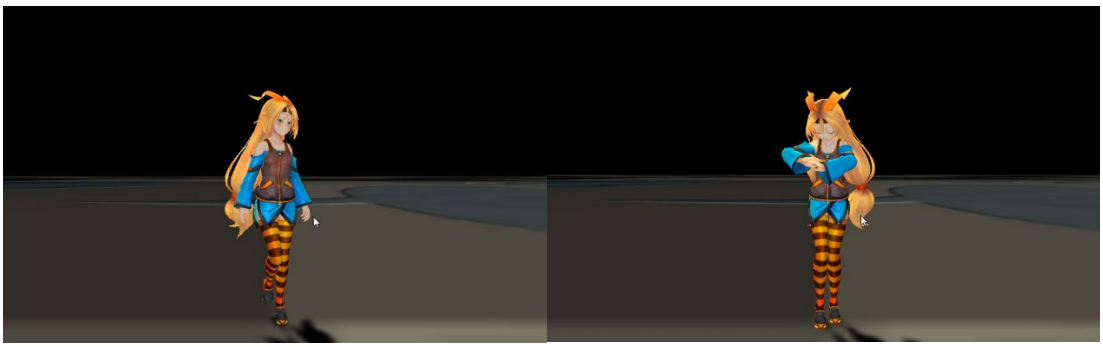
打开 PlayerAnimationController 脚本，更改脚本内容，实现角色跳跃。当按下空格键时即可跳跃。做出如下修改：

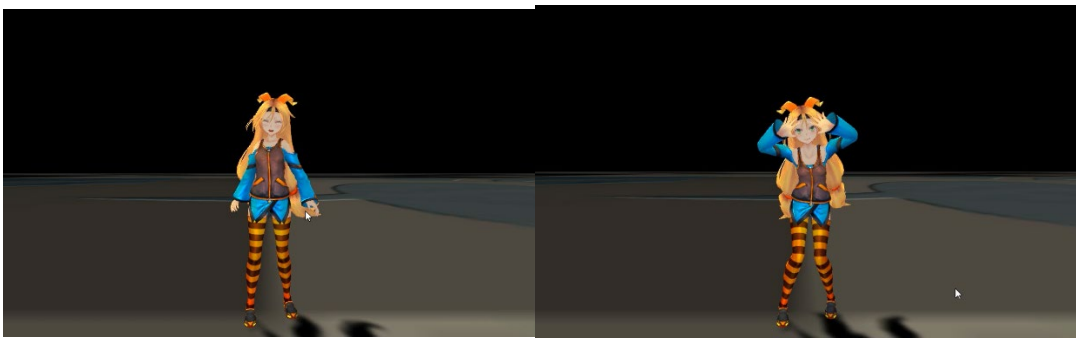
```
1. public class PlayerAnimationController : MonoBehaviour
```

```
{
    ……
    // Update is called once per frame
    void Update()
    {
        ……
        if (Input.GetKey(KeyCode.Space))
        {
            m_Animator.SetBool("jump", true);
        }
        else
        {
            m_Animator.SetBool("jump", false);
        }
        ……
    }
}
```

运行游戏，游戏运行截图如下：

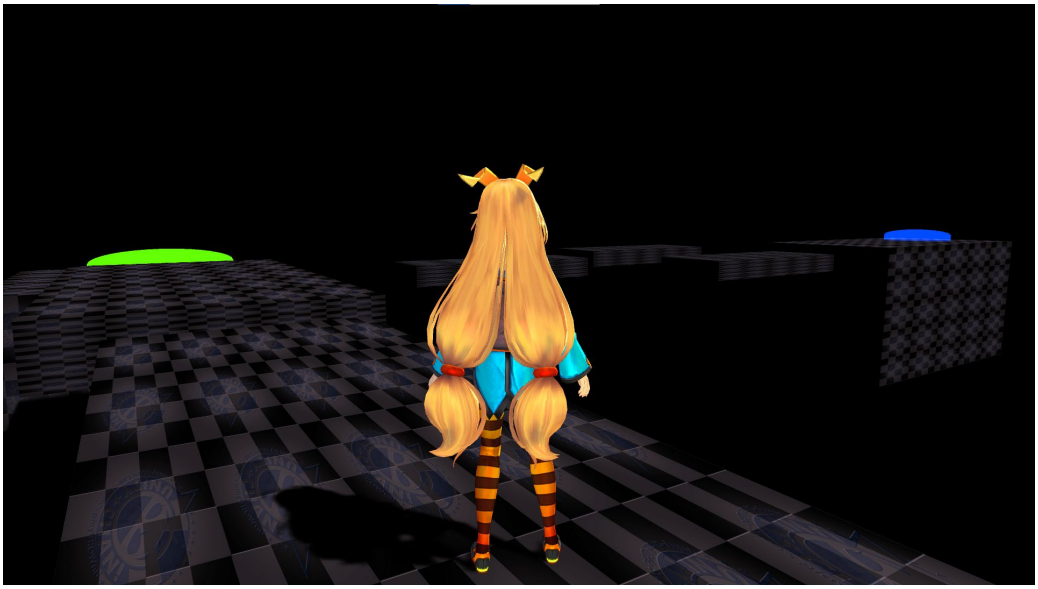






6、为 unity-chan 添加游戏（加分项）

前面利用 unity-chan 开发包内的动画资源实现了 unity-chan 内基本的动画控制效果，做完之后，我们觉得这样的游戏未免有些枯燥，只能简单的行走跳跃以及播放一些空闲的动画。因此，我们打算添加一些游戏场景，使得 unity-chan 成为一个闯关游戏，如下图所示。



在该游戏中，存在若干个长短宽窄不一的悬浮平台(Cube)，某些平台上有不同颜色的圆形的高亮的区域，绿色的称为复活点，蓝色的称为终点。unity-chan 可以在这些悬浮平台间跳跃，当 unity-chan 从平台上掉下时，如果已经到过复活点，那么在复活点复活；如果没到过复活点，那么在关卡起始点复活。

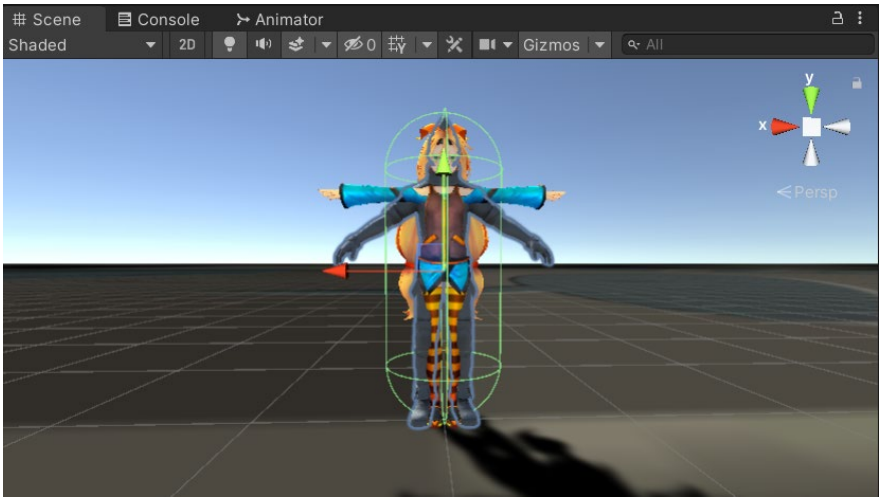
（1）添加 Standard Assets 资源包

在 Package Manager 中导入 Standard Assets 资源包，并删除 SampleScenes 文件夹。

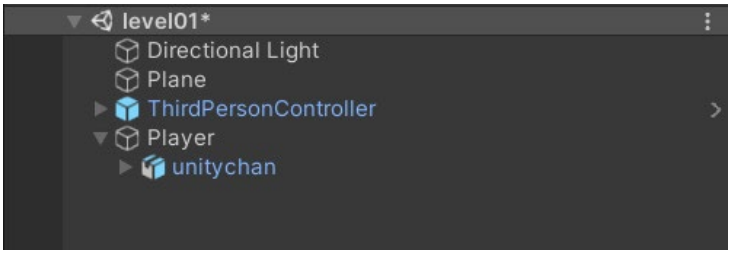
（2）为 unity-chan 添加第三人称角色效果

① 删除在前面一节为 unity-chan 设置的除 Transform 以外的所有组件；

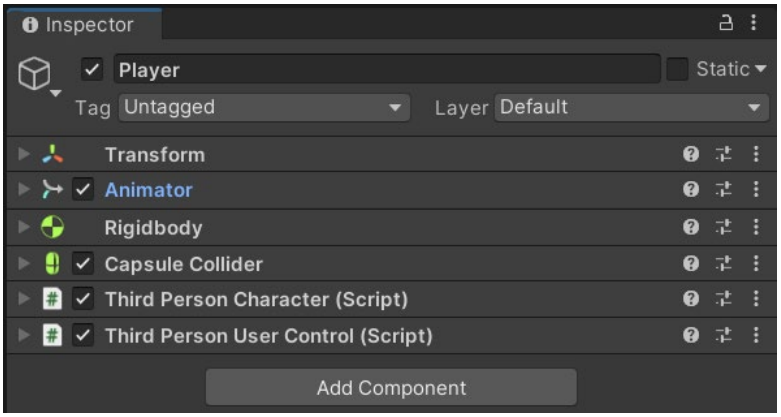
② 在文件夹 Assets/Standard Assets/Characters/ThirdPersonCharacter/Prefabs 中找到 ThirdPersonController，将其拖拽至 Hierarchy 面板。



③ 在 Hierarchy 面板创建一个空游戏对象，重命名为 Player，Position 设置为(0, 0, 0)，并将 unity-chan 拖拽到 Player 对象下，成为 Player 的子对象；



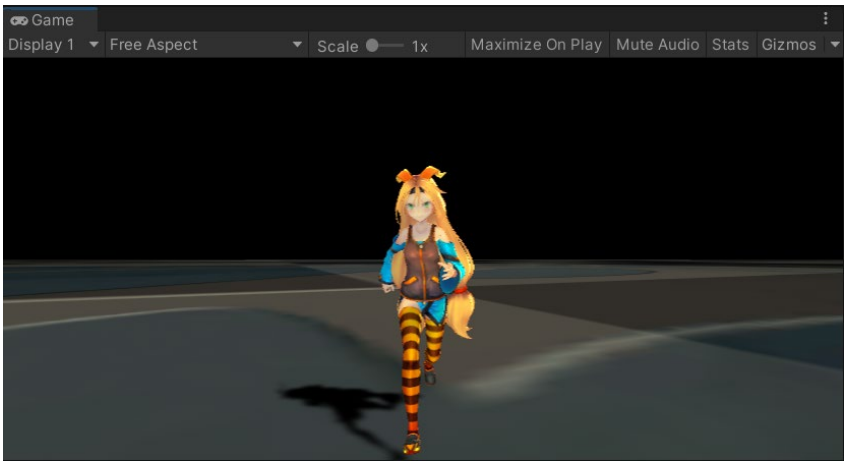
④ 将 ThirdPersonController 对象的所有组件复制给 Player 对象。



⑤ 修改 Player 的 Animator 组件的 Avatar 为“unitychanAvatar”；

⑥ 删除 ThirdPersonController 对象。

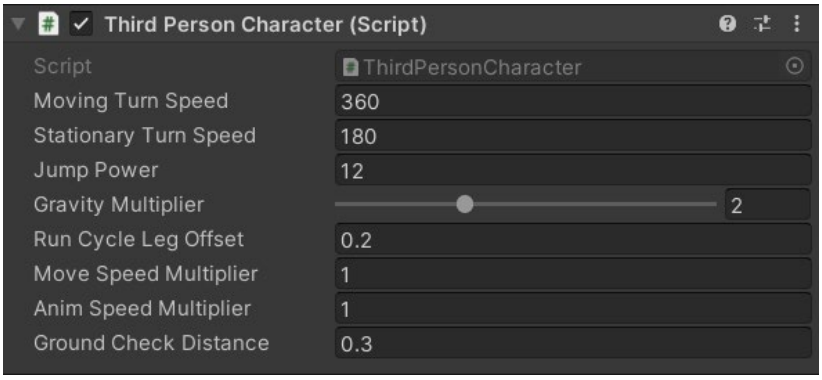
运行游戏，unity-chan 便可以像 Ethan 一样运动了，如下图，但是这样的话 unity-chan 使用的还是 Ethan 的动画，因此需要将他们替换为 unity-chan 的动画。



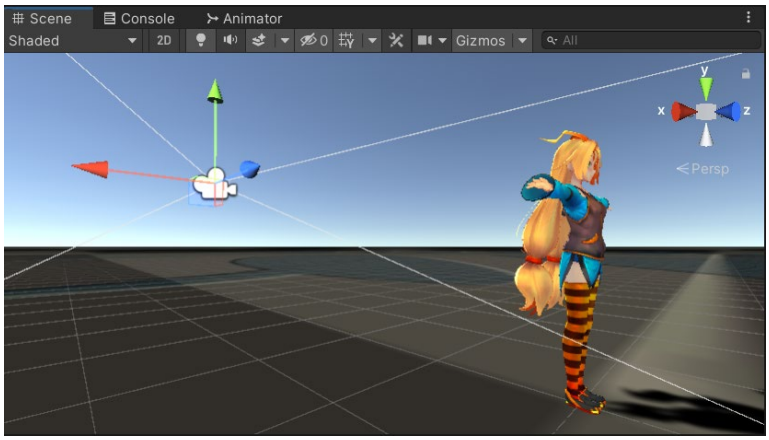
（3）设置 Camera

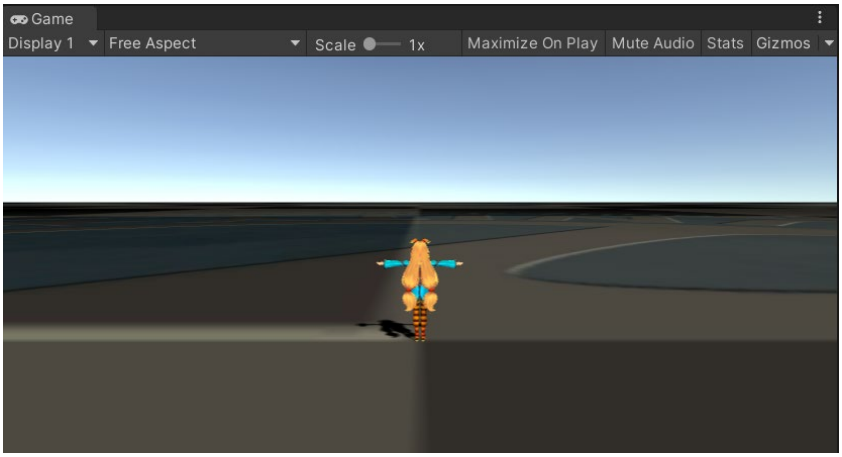
① 将 unity-chan 对象中的 Main Camera 拖放至最外层；

② 选中 Player 对象，Reset 脚本 ThridPersonCharacter 的参数，并重新设置如下：



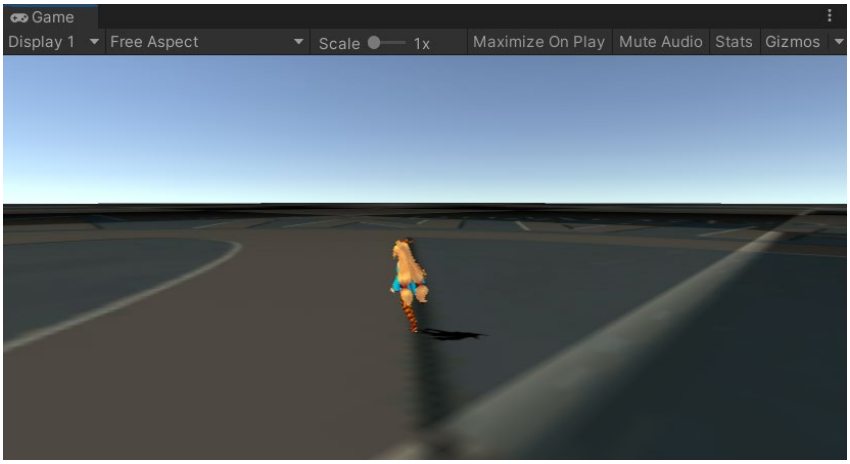
③ 删除 Main Camera 对象，在 Assets/Standard Assets/Cameras/Prefabs 文件夹中找到 MultipurposeCameraRig，将其添加到 Hierarchy 面板。



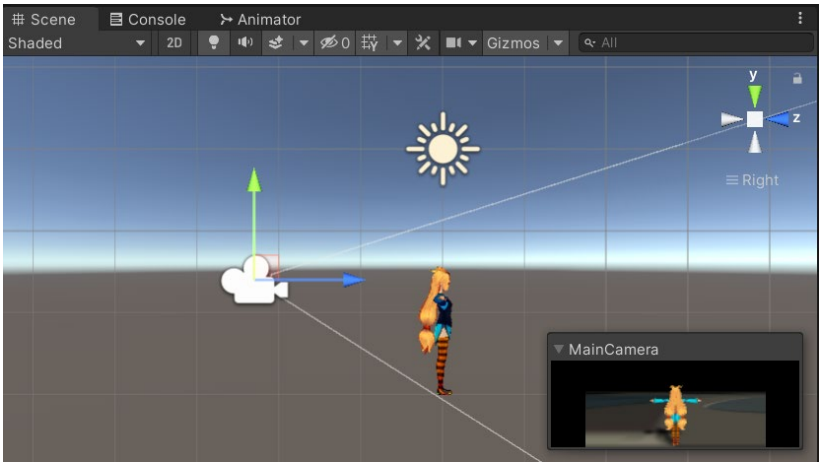


④ 在 MultipurposeCameraRig 的 Auto Cam 组件中，设置其 Target 参数为 Player，并勾选 Follow Velocity 选项；同时设置 Player 对象的标签为 Player。

如下图所示，现在镜头便可以跟随 unity-chan 移动了。

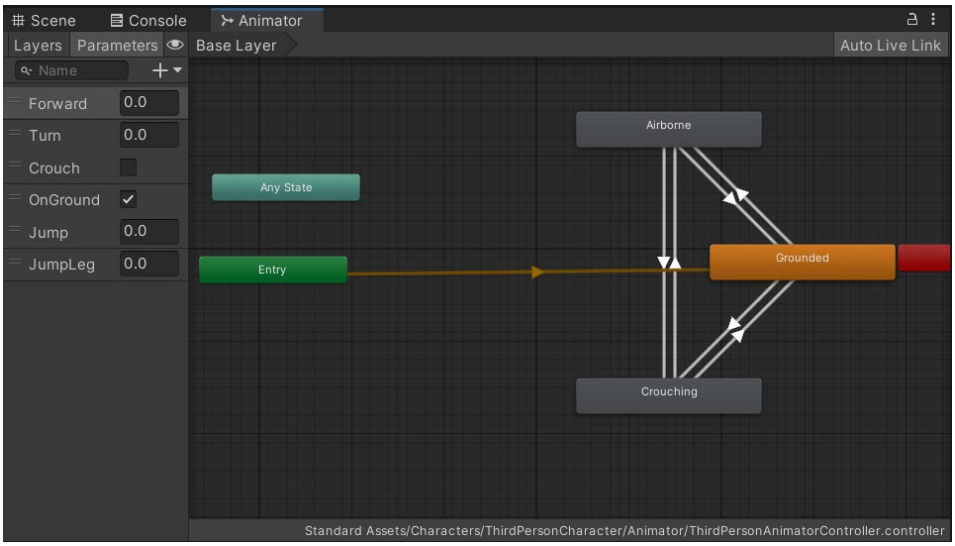


⑤ 选中 MultipurposeCameraRig 的 Pivot 子对象，设置 Position 属性为(0, 1.4, 0)；再选中 Pivot 对象的 MainCamera 子对象，设置 Position 属性为(0, 0, -2.3)，设置其 Camera 组件的 Clear Flags 为 Solider Color，Background 为(0, 0, 0)。



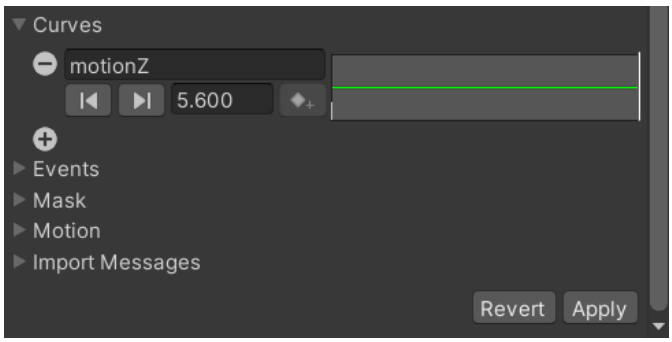
（4）修改 Player 的动画

① 打开 Player 的 Animator 组件中的 ThirdPersonAnimatorController 动画控制器



② 打开 Grounded 混合树，将其中的动画根据需求更改为对应 unity-chan 中的动画。

③ 在 unity-chan 的 Animations 文件夹中找到 RUN00_F，添加 Curves 如下，添加后，点击 Apply 应用。



然后在 Animator 面板添加一个 float 类型的 motionZ 参数。然后，打开 Player 的 ThirdPersonCharacter 脚本，在 OnAnimatorMove()函数中添加以下内容：

```
1. public void OnAnimatorMove()
2. {
3.     // we implement this function to override the default root motio
   n.
4.     // this allows us to modify the positional speed before it's app
   lied.
5.     if (m_IsGrounded && Time.deltaTime > 0)
6.     {
7.         // 使用 motionZ 向前移动
8.         Vector3 moveForward = transform.forward * m_Animator.GetFloa
   t("motionZ") * Time.deltaTime;
9.
```

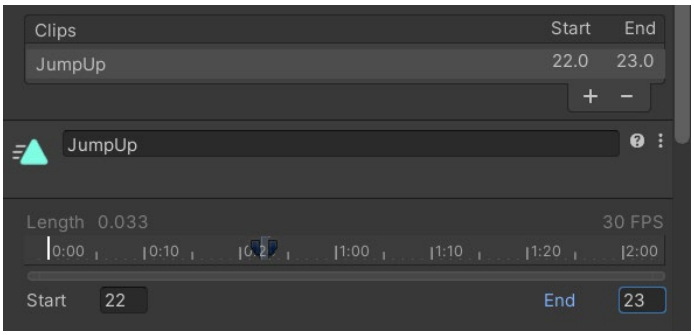
```
        Vector3 v = ((m_Animator.deltaPosition + moveForward) * m_MoveSpeedMultiplier) / Time.deltaTime;

        // we preserve the existing y part of the current velocity.
        v.y = m_Rigidbody.velocity.y;
        m_Rigidbody.velocity = v;
    }
}
```

这样，添加了 RUN00_F 动画的 unity-chan 在奔跑时便可以移动了。

④ 同理，为其他的 RUN 动画和 WALK 动画添加 Curves，RUN00_L 和 RUN00_R 添加 motionZ，大小为 3；WALK00_F 添加 motionZ，大小为 1.55；WALK00_L 和 WALK00_R 添加 motionZ，大小为 1。

⑤ 选中 unity-chan 的 JUMP01 动画，在 Inspector 面板修改其参数如下，然后点击 Apply 应用。



⑥ 同理，选中 unity-chan 的 JUMP00B 动画，在 Inspector 面板修改其 Clips 名为 MidAir，起止帧分别为(22, 23)，然后点击 Apply 应用；选中 unity-chan 的 JUMP00 动画，在 Inspector 面板修改其 Clips 名为 Fall，起止帧分别为(30, 31)，然后点击 Apply 应用。

⑦ 打开 Airborne 混合树，将其中的动画根据需求更改为对应 unity-chan 中的跳跃动画。

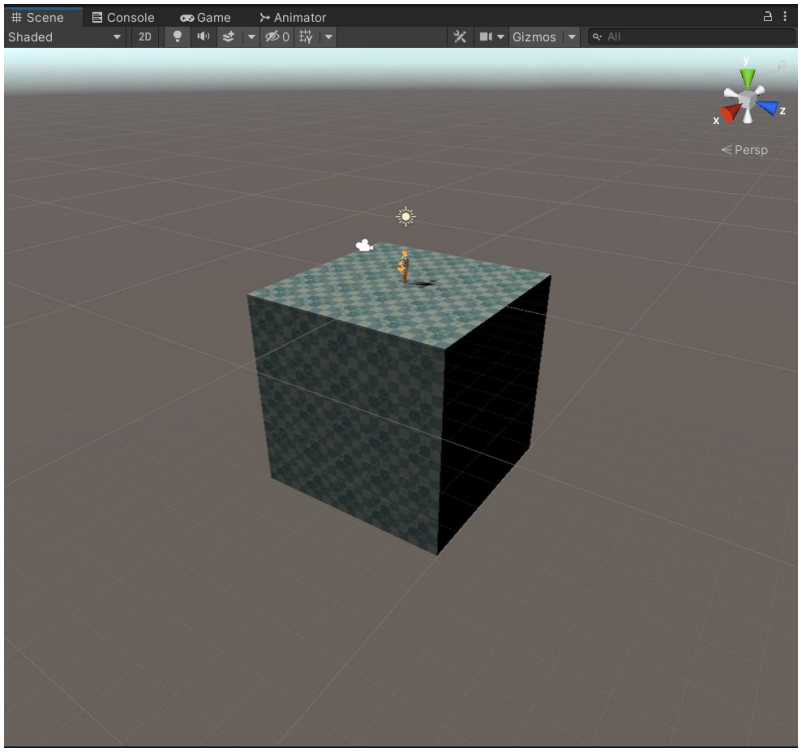
⑧ 修改 ThirdPersonCharacter 脚本，让 unity-chan 跳得更远。因此，修改脚本中的 HandleGroundedMovement()函数如下：

```
void HandleGroundedMovement(bool crouch, bool jump)
{
    // check whether conditions are right to allow a jump:
    if (jump && !crouch && m_Animator.GetCurrentAnimatorStateInfo(0).IsName("Grounded"))
    {
        // jump!
```

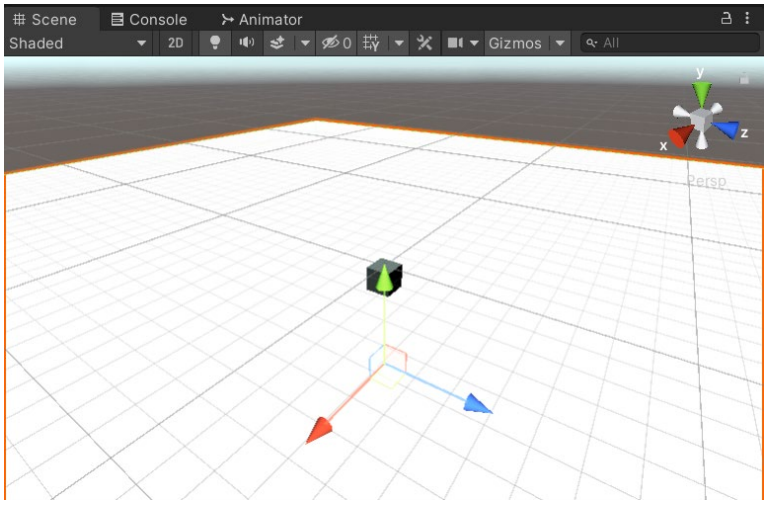
```
7.          m_Rigidbody.velocity = new Vector3(m_Rigidbody.velocity.x *
    2f, m_JumpPower, m_Rigidbody.velocity.z * 2f);
8.          m_IsGrounded = false;
9.          m_Animator.applyRootMotion = false;
10.         m_GroundCheckDistance = 0.1f;
11.     }
12.}
```

（5）创建游戏场景

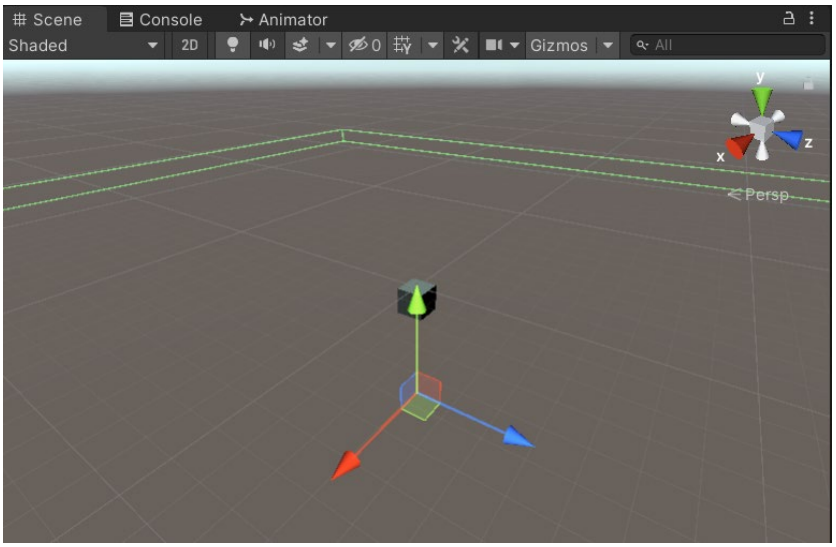
① 删除之前创建的 Plane，并创建一个 Cube 作为出生点，如下图：



② 再创建一个 Cube 名为 FallZone 作为落下后死亡的地区，并设置其 tag 标签为 death，如下图：



③ 删除 FallZone 的 Cube 和 Mesh Renderer 组件，设置 Box Collider 为 Trigger。

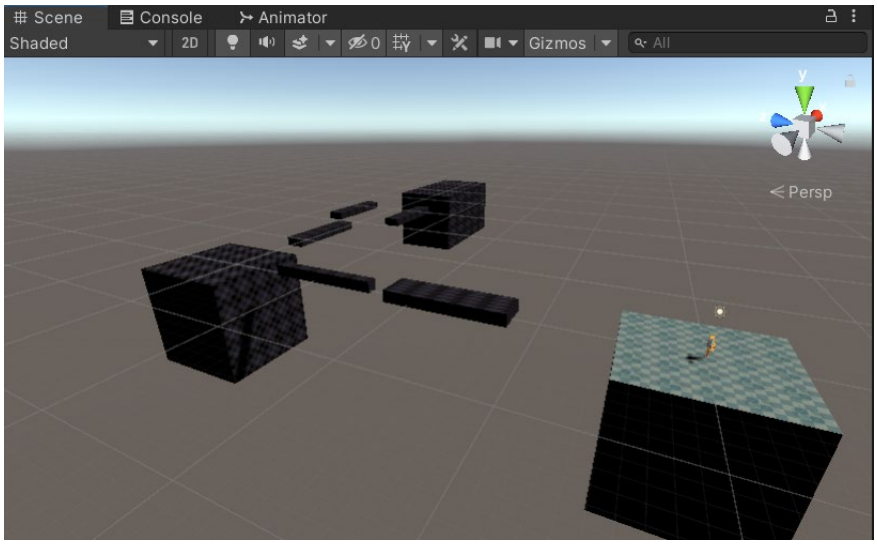


④ 创建脚本 Respawn，用于控制游戏对象的死亡与复活，并将其添加到 Player 对象，控制死亡的脚本内容如下：

```
1. public class Respawn : MonoBehaviour
2. {
3.     private Vector3 startPosition;
4.     private Quaternion startRotation;
5.
6.     // Start is called before the first frame update
7.     void Start()
8.     {
9.         startPosition = transform.position;
10.        startRotation = transform.rotation;
11.    }
12.
13.    // 触发检测
14.    private void OnTriggerEnter(Collider other)
15.    {
16.        if (other.tag == "death")
17.        {
18.            transform.position = startPosition;
19.            transform.rotation = startRotation;
20.        }
21.    }
22.}
```

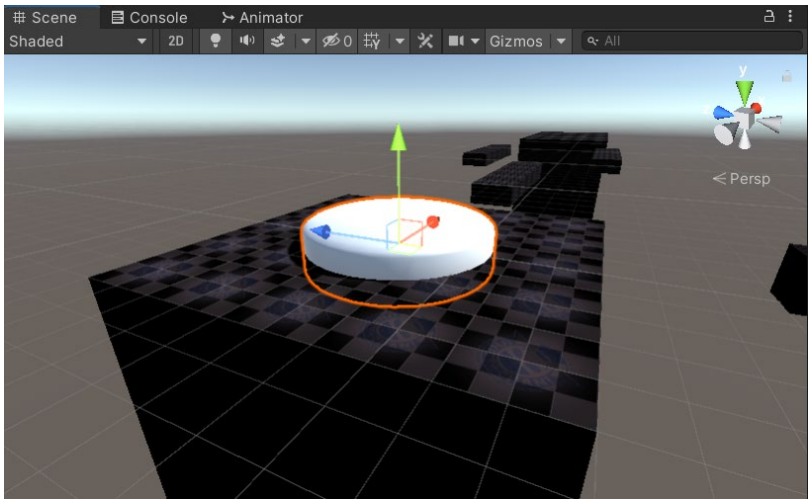
这样，当主角掉下悬浮的平台时便会在设置的复活点复活。

⑤ 在场景中再创建几个 Cube，作为悬浮的平台，创建完成后如下图所示：

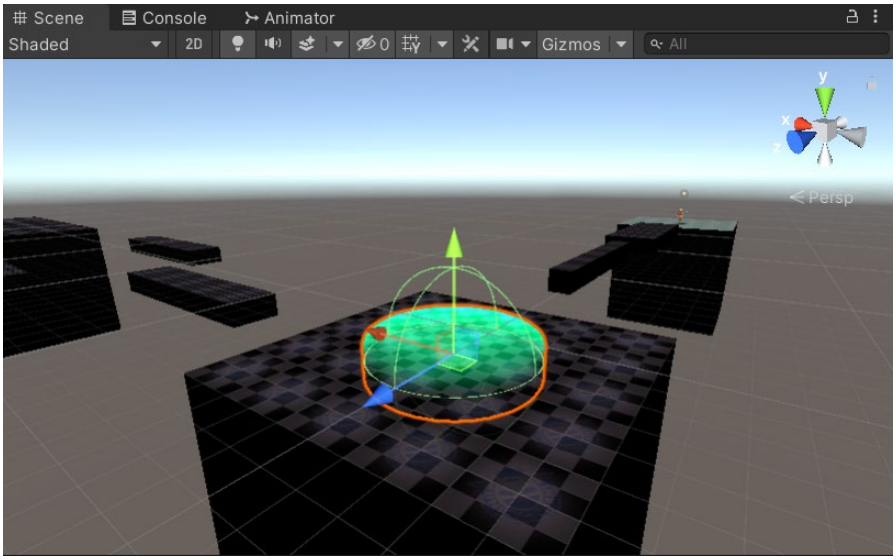


⑥ 创建复活点预制体。

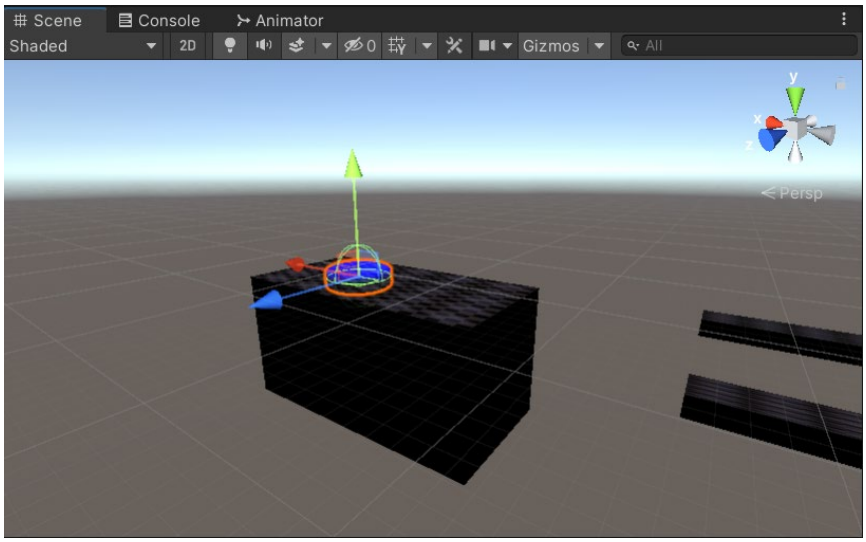
在 Hierarchy 面板右键创建 Capsule，命名为 CheckPoint，设置 tag 标签为 check，Capsule Collider 设置为 isTrigger。



为其添加 Material 后如下所示，将其保存至 Prefabs 文件夹。



⑦ 创建终点预制体。与复活点创建方法一样，只是颜色不同，tag 标签设置为 final。

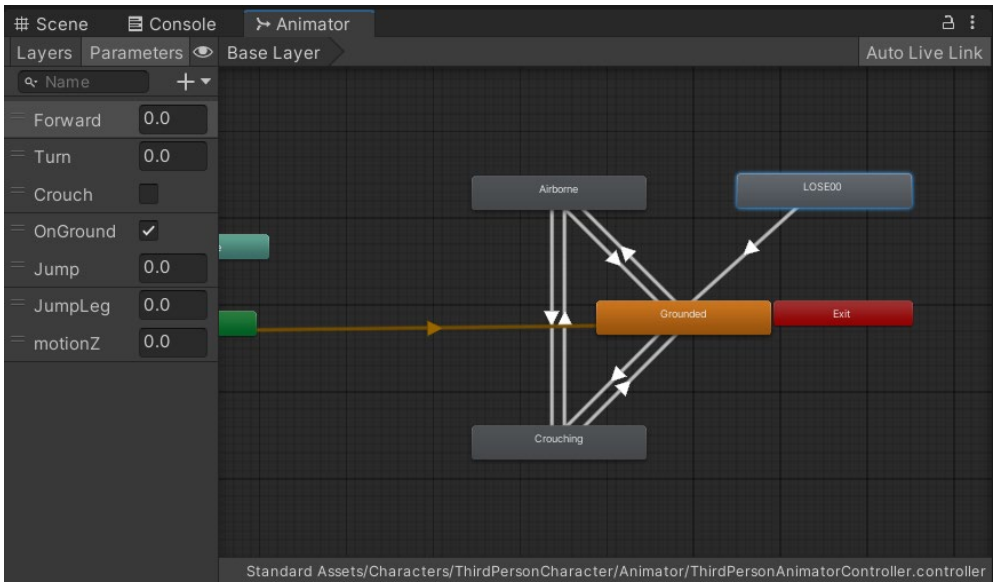


（6）添加主角死亡、复活与胜利的动画

① 新建空对象，命名为 UnityChanPrefab，Position 设置为(0, 0, 0)，将 Player 和 MultipurposeCameraRig 对象拖拽至 UnityChanPrefab 成为其子对象。然后将 UnityChanPrefab 保存至 Prefabs 文件夹。将 FallZone 对象也保存至 Prefabs 文件夹。

② 添加死亡后复活时的动画。

在 unity-chan 的 Animations 文件夹中找到 LOSE00 动画，将其添加到 Animator 面板，并且向 Grounded 混合树转换。

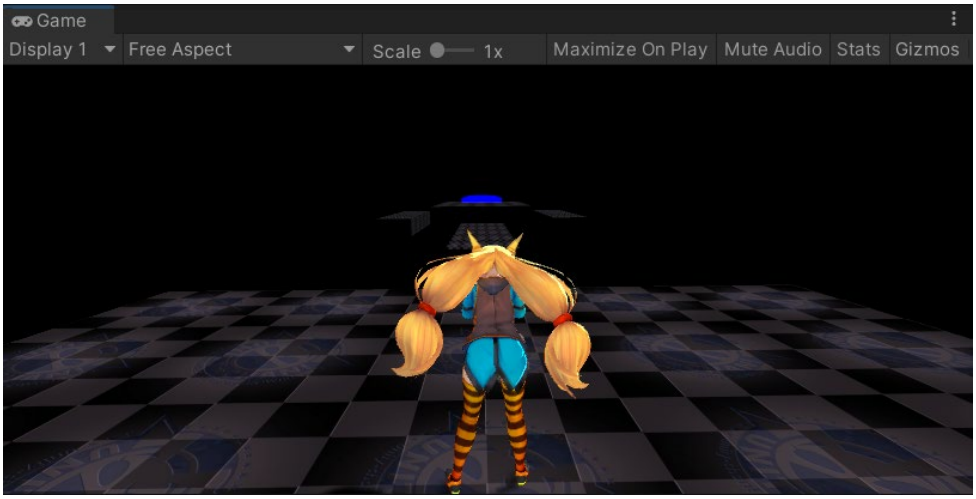


修改 Respawn 脚本如下，控制复活时动画的播放。

```
1. public class Respawn : MonoBehaviour
2. {
```

```
    // 触发检测
    private void OnTriggerEnter(Collider other)
    {
        if (other.tag == "death")
        {
            ……

            // 复活时播放失败的动画
            GetComponent<Animator>().Play("LOSE00", -1, 0f);
            // 复活时不可移动
            GetComponent<Rigidbody>().velocity = new Vector3(0, 0, 0
);
            GetComponent<Rigidbody>().angularVelocity = new Vector3(
0, 0, 0);
        }
    }
}
```



③ 添加当落到复活点时的脚本控制。

当落到复活点时需要将复活位置设置为新的复活点的位置，并且销毁该复活点对象。因此修改 Respawn 脚本的 OnTriggerEnter()函数，如下：

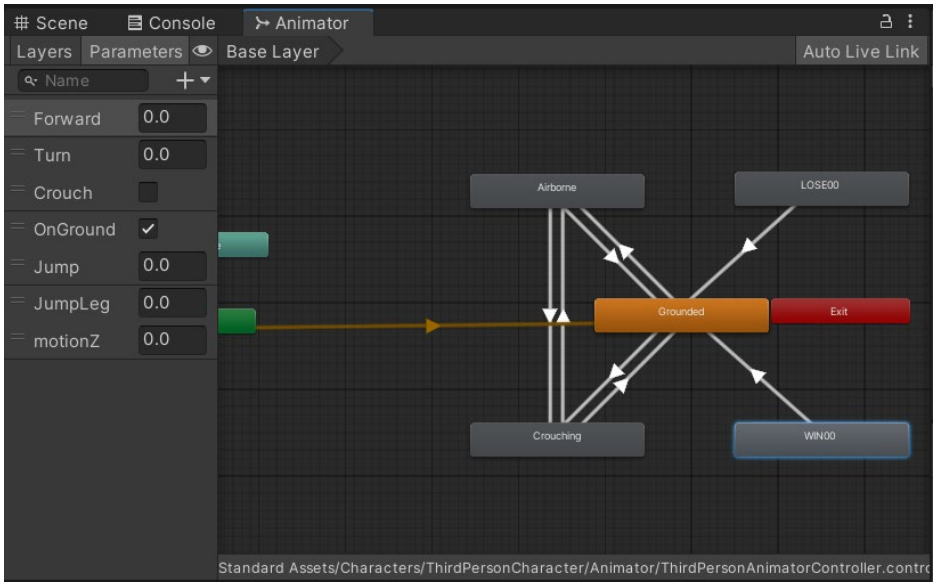
```
private void OnTriggerEnter(Collider other)
{
    // 如果是坠落到 FallZone
    if (other.tag == "death")
    {
        transform.position = startPosition;
        transform.rotation = startRotation;

        // 复活时播放失败的动画
        GetComponent<Animator>().Play("LOSE00", -1, 0f);
        // 复活时不可移动
```

```
12.         GetComponent<Rigidbody>().velocity = new Vector3(0, 0, 0);
13.         GetComponent<Rigidbody>().angularVelocity = new Vector3(0, 0
    , 0);
14.     }
15.     // 如果是复活点
16.     else if (other.tag == "check")
17.     {
18.         startPosition = other.transform.position;
19.         startRotation = other.transform.rotation;
20.         Destroy(other.gameObject);
21.     }
22. }
```

④ 添加当落到终点时的动画及脚本控制。

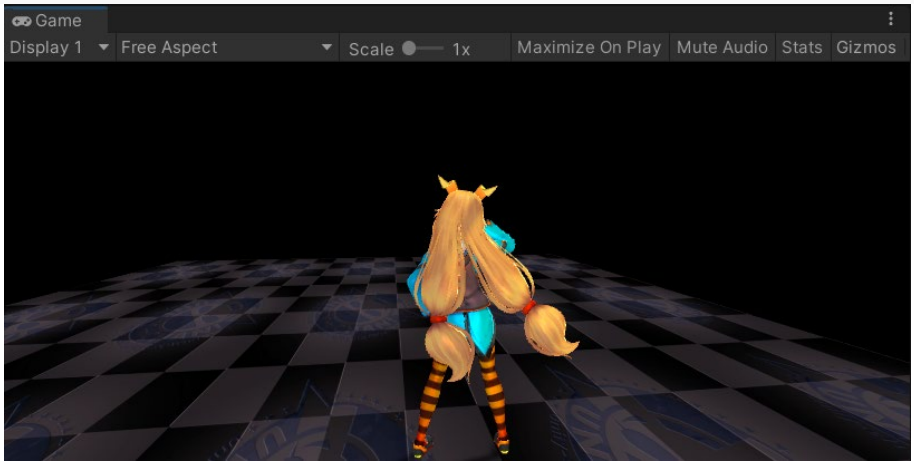
在 unity-chan 的 Animations 文件夹中找到 WIN00 动画，将其添加到 Animator 面板，并且向 Grounded 混合树转换。



修改 Respawn 脚本的 OnTriggerEnter()函数，如下，控制到达终点时动画的播放。

```
1. private void OnTriggerEnter(Collider other)
2. {
3.     // 如果是坠落到 FallZone
4.     if (other.tag == "death")
5.     {
6.         transform.position = startPosition;
7.         transform.rotation = startRotation;
8.
9.         // 复活时播放失败的动画
10.        GetComponent<Animator>().Play("LOSE00", -1, 0f);
11.        // 复活时不可移动
12.        GetComponent<Rigidbody>().velocity = new Vector3(0, 0, 0);
```

```
13.         GetComponent<Rigidbody>().angularVelocity = new Vector3(0, 0
    , 0);
14.     }
15.     // 如果是复活点
16.     else if (other.tag == "check")
17.     {
18.         startPosition = other.transform.position;
19.         startRotation = other.transform.rotation;
20.         Destroy(other.gameObject);
21.     }
22.     // 如果是终点
23.     else if (other.tag == "final")
24.     {
25.         Destroy(other.gameObject);
26.         GetComponent<Animator>().Play("WIN00", -1, 0f);
27.     }
28. }
```



三、最终游戏效果

在该游戏中，存在若干个长短宽窄不一的悬浮平台(Cube)，某些平台上有不同颜色的圆形的高亮的区域，绿色的称为复活点，蓝色的称为终点。unity-chan 可以在这些悬浮平台间跳跃，当 unity-chan 从平台上掉下时，如果已经到过复活点，那么在复活点复活；如果没到过复活点，那么在关卡起始点复活。复活时会播放哭的动画，表示掉下了平台。当 unity-chan 经过一次次的尝试最终到达终点时，会播放庆祝的动画，表示游戏胜利。并进入下一个关卡。

游戏演示视频见附件：Untiy Chan 演示视频.mp4。

四、游戏制作遇到的问题及解决方法

实验过程中遇到的第一个问题是左右播放的动画播放不正常。初始时，为WALK00_R 设置的 PosX、PosY 参数为(1, 0)，为 WALK00_L 设置的 PosX、PosY 为(-1, 0)，此时运行游戏发现，按左右按钮时动画播放不正常，虽然是在往左右走，但是播放的动画不是直接左右移动的。后面预览动画发现，该预制的动画不是单纯的左右行走，而是向左前和右前方行走。因此将其分别修改为(1, 1)、(-1, 1)，代表向左前和右前方行走。

第二个问题为，为 unity-chan 添加的移动脚本控制后，按移动键时，对象虽然会正常播放动画，但是其位置并不会移动。后面发现，是因为未给对象添加 Rigidbody 组件，添加该组件后，对象可以正常移动。

第三个是移动脚本正常添加后，即使 unity-chan 没有向前走，但是其位置也可以左右移动，只是不会播放动画而已。这与游戏效果不符，修改脚本，添加以下判断，当 unity-chan 没有向前走时，也不会左右移动。

```
1. if (moveZ <= 0)
2. {
3.     // 如果没有向前走，那么不能左右移动
4.     moveX = 0;
5. }
```

上面就是在做基本要求时遇到的问题，下面是做加分项遇到的问题。

第一个问题就是，单纯地将 Ethan 的动画替换为 untiy-chan 的动画后，unity-chan 不会移动了，只是会播放动画，位置并不会改变。后面发现两者的动画是不一致的，Ethan 的动画是带有速度的，即是会移动的，而 unity-chan 的动画是原地播放的，不会移动。因此需要修改 unity-chan 的动画的参数，为其添加 curves 组件，具体操作如上所示。

第二个问题是跳跃的动画制作问题，因为 unity-chan 的原始动画直接替换 Ethan 的动画的话会影响跳跃的流畅性，并且效果很生硬。后面发现 Ethan 的动画使用的是一个个的片段，因此，通过从网上阅读资料，发现可以根据自己的实际需要截取 unity-chan 的动画片段来作为跳跃的每一帧，这样将每一帧连起来就是一个完整的跳跃和下落过程。

第三个问题是，当 unity-chan 触碰到 CheckPoint 时，在落下的过程中会有一个卡顿的现象。仔细思考发现是当其落到 CheckPoint 上后会试图在 CheckPoint 上行走。解决方法为修改 ThridPersonCharacter 脚本的 CheckGroundStatus()函数，

使其落到 Trigger 上时不继续行走。该函数修改后如下所示：

```
1. void CheckGroundStatus()
2. {
3.     RaycastHit hitInfo;
4. #if UNITY_EDITOR
5.     // helper to visualise the ground check ray in the scene view
6.     Debug.DrawLine(transform.position + (Vector3.up * 0.1f), transfo
   rm.position + (Vector3.up * 0.1f) + (Vector3.down * m_GroundCheckDis
   tance));
7. #endif
8.     // 0.1f is a small offset to start the ray from inside the chara
   cter
9.     // it is also good to note that the transform position in the sa
   mple assets is at the base of the character
10.    if (Physics.Raycast(transform.position + (Vector3.up * 0.1f), Ve
   ctor3.down, out hitInfo, m_GroundCheckDistance))
11.    {
12.        // 如果落点是 Trigger，不继续行走
13.        if (!hitInfo.collider.isTrigger)
14.                {
15.            m_GroundNormal = hitInfo.normal;
16.            m_IsGrounded = true;
17.            m_Animator.applyRootMotion = true;
18.        }
19.    }
20.    else
21.    {
22.        m_IsGrounded = false;
23.        m_GroundNormal = Vector3.up;
24.        m_Animator.applyRootMotion = false;
25.    }
26.}
```

五、总结与体会

Mecanim 是 Unity 的一个丰富且精密的动画系统，它提供了为人形角色提供的简易的工作流和动画创建能力；Retargeting 运动重定向功能，即把动画从一个角色模型应用到另一个角色模型上。针对 Animation Clips(动画片段)的简易工作流，即针对动画片段以及它们之间的过渡和交互过程的预览能力。这样可以使动画师更加独立地进行工作，而不用过分地依赖于程序员，从而在编写游戏逻辑代码之前即可预览动画效果。并且提供了通过不同逻辑来控制不同身体部位运动的

能力。

本次实验首先通过完成实验的基本任务，学会了如何使用 Unity 编辑器进行动画的过渡预览，以及如何通过添加参数的方式来控制动画的播放，如果通过脚本获取到动画的参数以及通过键盘输入控制动画的过渡。并且通过创建混合树以及编辑混合树，学会了如何通过参数来控制混合树中动画的播放。

在实现扩展功能时，通过将 Ethan 的 Avatar 替换为 unity-chan 的 Avatar，了解了如何通过重定向来实现把动画从一个角色模型应用到另一个角色模型。以及如何调整动画的参数，使其播放更加流畅。并且，还学习到如何通过 Curves 来为原地播放的动画添加移动的效果，即不通过脚本而是通过动画来控制主角运动的速度，以及运动的方向。