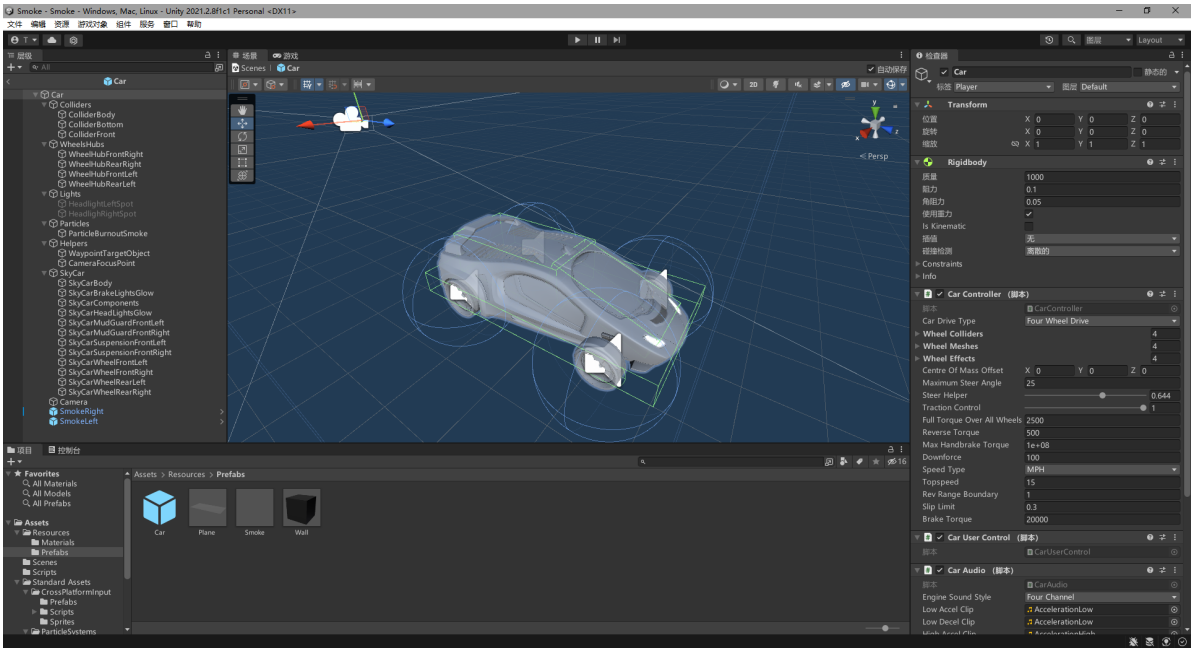
完善 “汽车尾气”模拟

✓ 使用官方资源资源 Vehicle 的 car， 使用 Smoke 粒子系统模拟启动发动、运行、故障等场景效果

粒子系统是模拟一些不确定、流动现象的技术，它采用许多形状简单且赋予生命的微小粒子作为基本元素来表示物体(一般由点或很小的多边形通过纹理贴图表示)，表达物体的总体形态和特征的动态变化。人们经常使用粒子系统模拟的现象有火、爆炸、烟、水流、火花、落叶、云、雾、雪、尘、流星尾迹或者象发光轨迹这样的抽象视觉效果等等。

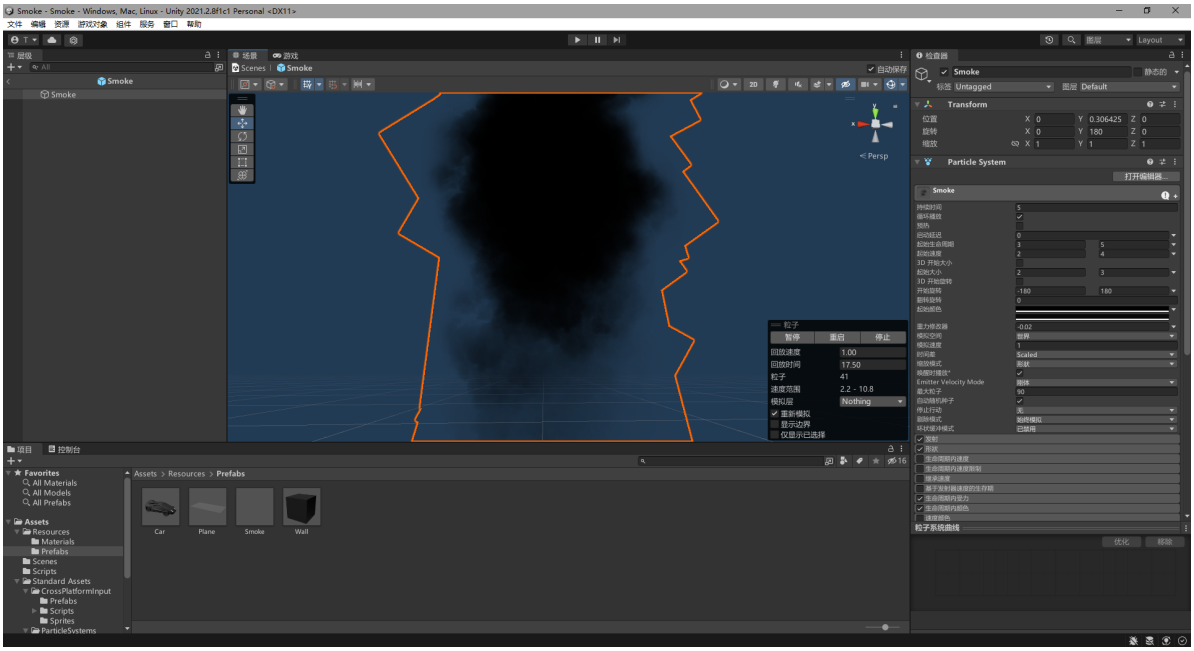
作为粒子系统，每个粒子运动一般具有简单的数学模型和它们之间具有自形似的运动过程。通过引入特定的随机分布作用于粒子，使得系统整体呈现复杂的现象，这是粒子系统的本质。

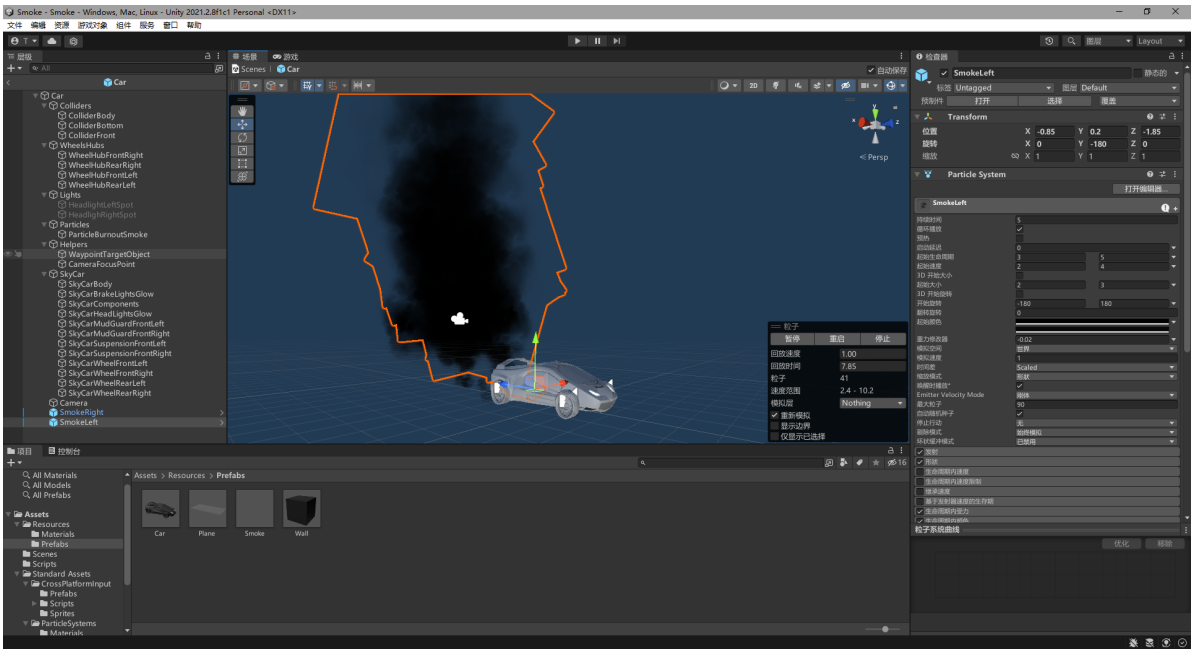
导入 Standard Assets 素材包，实例化其中的 Car 预制：



我们复制其自带的 `ParticleBurnoutSmoke` 为 `SmokeLeft` 和 `SmokeRight` ，作为左右排气管的尾气粒子系统。在右侧的 Inspector 菜单，我们设置 `Force over Lifetime` 和 `Size over Lifetime` 属性，分别设置粒子的运动方向和粒子群的大小变化情况，具体设置如下图所示。

粒子效果和汽车释放尾气的粒子效果：





官方素材提供的 Car 的运动是需要较大的运动空间，并且我们需要模拟车辆与障碍物的碰撞，在这里，我们采用了代码生成地图场景的方案，具体实现位于 `MainController` 类。

在该类的 `Awake` 方法中，我们调用了 `LoadResources` 方法，用于生成地图场景和车辆，其具体代码如下：

```
using UnityEngine;

namespace Smoke
{
    public class MainController : MonoBehaviour
    {
        private GameObject car;

        private void Awake()
        {
            // 装载资源。
            LoadResources();
        }
```

```
14
15          // 装载地图资源。
16          private void LoadResources()
17          {
18              // 加载地图平面。
19              var plane = Instantiate(Resources.Load<GameObject>
("Prefabs/Plane"));
20              plane.name = "Plane";
21
22              // 加载车辆资源。
23              car = Instantiate(Resources.Load<GameObject>("Prefabs/Car"));
24              car.name = "Car";
25
26              // 加载边界预制。
27              var wallPrefab = Resources.Load<GameObject>("Prefabs/Wall");
28              // 绘制地图边界。
29              {
30                  GameObject wall = Instantiate(wallPrefab);
31                  wall.transform.position = new Vector3(-25, 2, 0);
32                  wall.transform.localScale = new Vector3(1, 4, 100);
33              }
34              {
35                  GameObject wall = Instantiate(wallPrefab);
36                  wall.transform.position = new Vector3(25, 2, 0);
37                  wall.transform.localScale = new Vector3(1, 4, 100);
38              }
39              {
40                  GameObject wall = Instantiate(wallPrefab);
41                  wall.transform.position = new Vector3(0, 2, 50);
42                  wall.transform.localScale = new Vector3(50, 4, 1);
43              }
44              {
45                  GameObject wall = Instantiate(wallPrefab);
46                  wall.transform.position = new Vector3(0, 2, -50);
47                  wall.transform.localScale = new Vector3(50, 4, 1);
48              }
49          }
50      }
51  }
```

实现这个一个需求：当引擎转速越高，尾气粒子的运动速度也越快。首先，我们如何获取车辆的运行速度？游戏对象 Car 绑定了 `CarController` 脚本，其含有 `Revs` 值，即引擎转速值。根据此信息，我们实现了 `SetEmissionRate` 函数，用于根据引擎转速设置粒子速度。

```
1   using UnityEngine;
2   using UnityStandardAssets.Vehicles.Car;
3
4   namespace Smoke
5   {
6       public class SmokeParticle : MonoBehaviour
7       {
8           // 车辆句柄。
9           private GameObject car;
10          private CarController carController;
11          // 粒子系统。
12          private ParticleSystem exhaust;
13
```

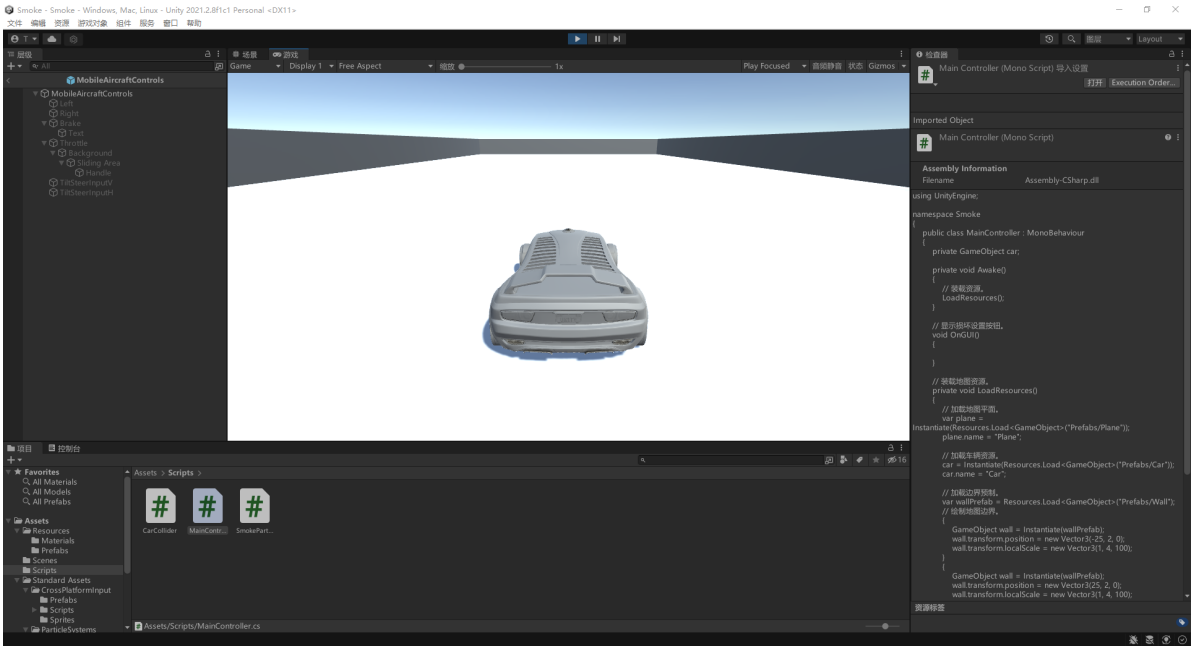
```
        void Start()
        {
            car = transform.parent.gameObject;
            carController = car.GetComponent<CarController>();
            exhaust = GetComponent<ParticleSystem>();
        }

        [System.Obsolete]
        void Update()
        {
            // 设置粒子释放速率。
            SetEmissionRate();
            // 设置粒子颜色。
            SetColor();
        }

        // 根据引擎转速设置粒子释放速率。
        private void SetEmissionRate()
        {
            // 比例系数。
            var K = 5000;
            // 注意：若使用 exhaust.emission.rateOverTime = K * carController.Revs; 会返回语法错误。
            var emission = exhaust.emission;
            emission.rateOverTime = K * carController.Revs;
        }

        // 根据车辆损坏程度设置粒子颜色。
        private void SetColor()
        {
            // 获取粒子颜色句柄。
            var color = exhaust.colorOverLifetime;
            // 获取车辆损坏情况。
            var damage = car.GetComponent<CarCollider>().GetDamage();
            // 根据损坏情况设置颜色深浅，损坏越严重，颜色越深。
            var gradient = new Gradient();
            var colorKeys = new GradientColorKey[] { new GradientColorKey(Color.white, 0.0f), new GradientColorKey(new Color(214, 189, 151), 0.079f), new GradientColorKey(Color.white, 1.0f) };
            var alphaKeys = new GradientAlphaKey[] { new GradientAlphaKey(0.0f, 0.0f), new GradientAlphaKey(damage / 255f + 10f / 255f, 0.061f), new GradientAlphaKey(0.0f, 1.0f) };
            gradient.SetKeys(colorKeys, alphaKeys);
            color.color = gradient;
        }
    }
}
```

最终呈现效果：



参考：https://github.com/Jiahonzheng/Unity-3D-Learning/tree/master/HW7