中南大学

《动画与游戏程序设计》

课程项目报告

项目名称 基于 Unity 的坦克大战游戏开发

学生姓名 _____

学　　号 _____

专业班级 _____

小组成员 _____

指导教师 _____

学　　院 计算机学院

目　录

一、项目概述

1.1 项目名称

基于 Unity 的坦克大战游戏开发。

1.2 项目目的

通过该实践环节的训练，使学生能够更加全面和系统的掌握游戏设计的体系结构。通过对所做游戏的故事梗概、游戏类型以及设计制作过程中所涉及的相关技术的学习和掌握，提高学生的实践能力以及团队协作能力，为在计算机游戏设计这一领域进行深入研究做准备。

1.3 项目要求

利用提供的游戏资源构建坦克大战游戏项目(或自选项目)，该项目涉及游戏开发的多个方面，特别适合初学者提升自己的游戏设计水平。

清楚了解项目的意义和目的；项目功能阐述清晰，设计过程正确合理；项目结果演示正确顺利；鼓励添加项目新功能和创意的加分项；实现细节问题回答。

1.4 开发步骤指导

开发游戏的步骤主要包括：

1. 创建游戏 3D 场景。
2. 给场景添加坦克，控制坦克的前后移动。
3. 坦克的旋转和烟雾效果；
4. 控制坦克的子弹发射；
5. 控制炸弹的飞行和爆炸；
6. 控制炸弹对坦克的伤害；
7. 给坦克添加音效；
8. 给坦克添加生命值控制和游戏结束的播放。

加分项：

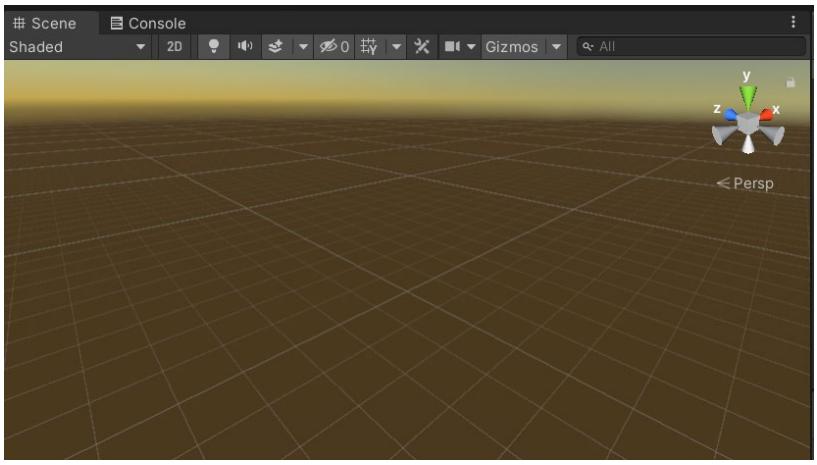
增加游戏的新效果和功能，例如英文介绍视频中的功能。

二、项目开发步骤

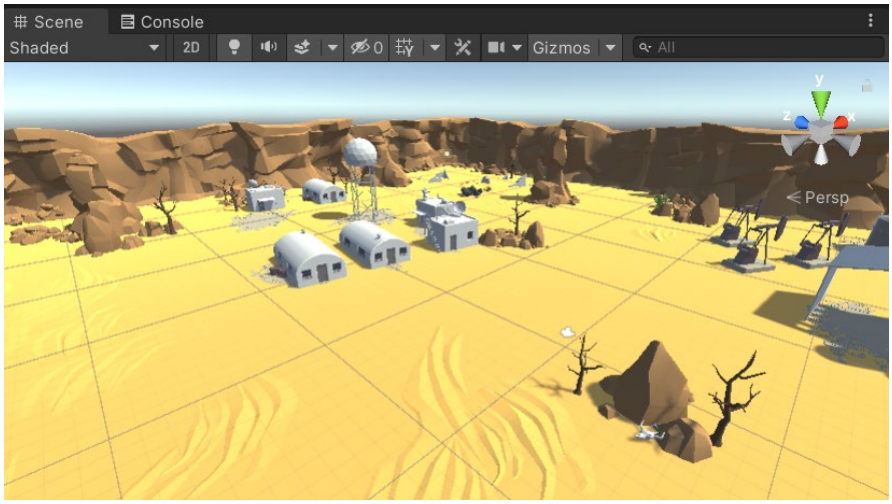
2.1 项目设置

1、创建项目并导入 Tanks Tutorial 资源。

2、创建一个空场景并命名为 Main，删除场景中的 Directional Light，因为在后面添加的游戏对象中会包含光源。

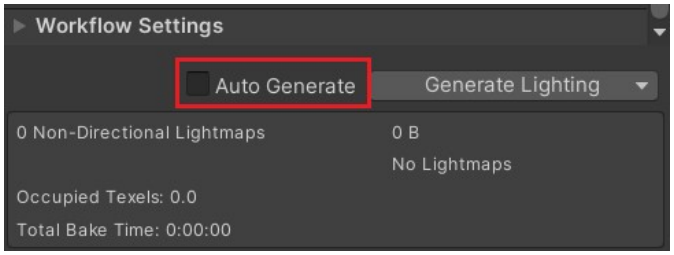


3、从 Prefabs 文件夹中找到环境模型 LevelArt，将其放入 Main 场景中，Position 设置为(0, 0, 0)。



4、在 Lighting 面板中设置光源属性：

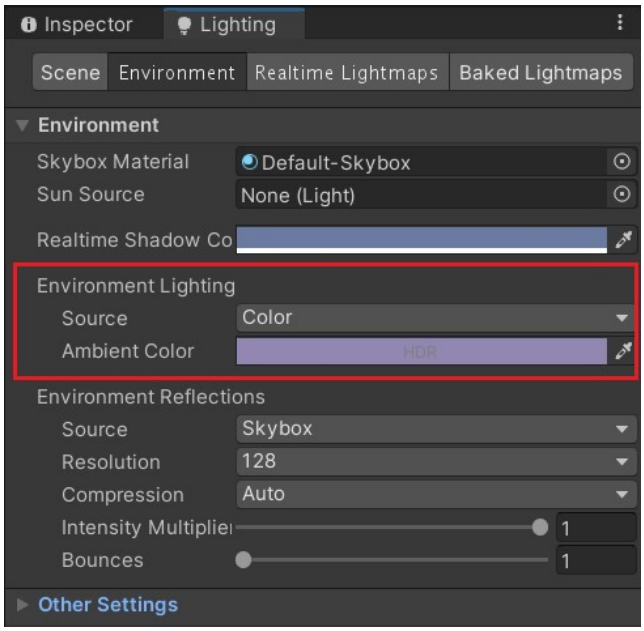
①在底部取消勾选 Auto Generate 选项；



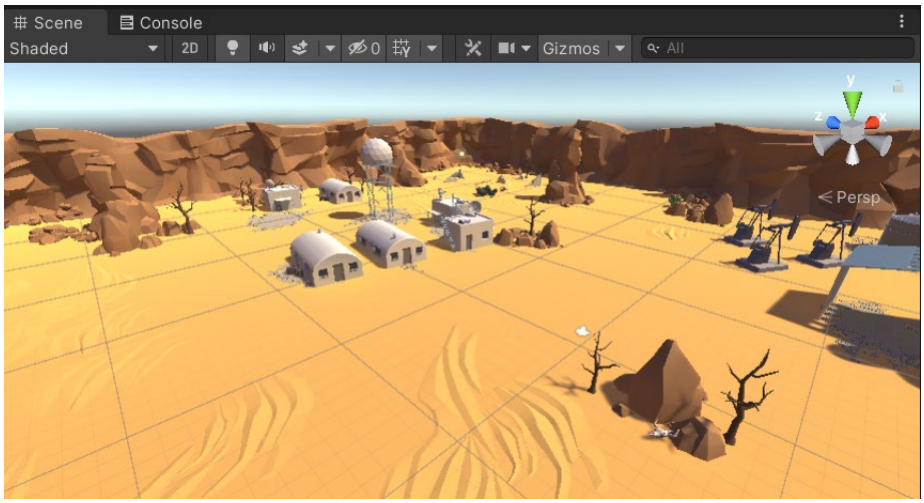
②打开 Mixed Lighting 组件，取消勾选 Backed Global Illumination；打开 Realtime Lighting 组件，勾选 Realtime Global Innumination。

③将 Indirect Resolution 的值设置为 0.5。

④在 Environment 栏中，将 Environment Lighting 的 Source 设置为 Color，Ambient Color 设置为(72, 62, 113)。



⑤设置完成后，点击下方的 Generate Lighting，完成光源设置。



5、Main Camera 设置

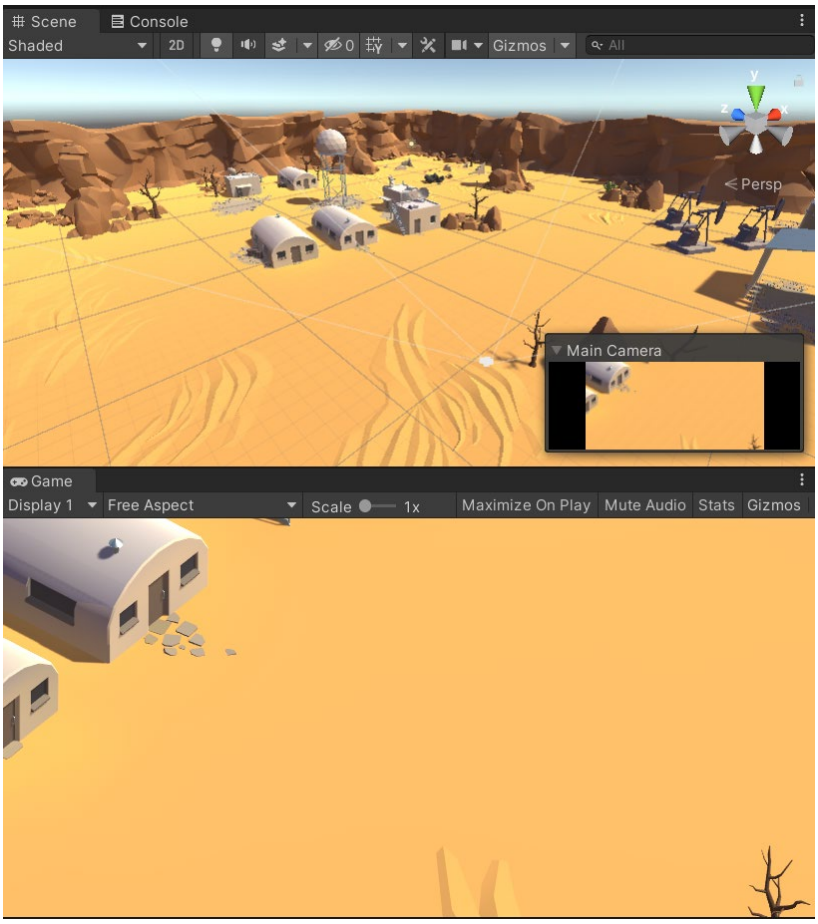
（1）设置 Transform 组件的 **Position** 为(-43, 42, -25)；

（2）设置 Transform 组件的 **Rotation** 为(40, 60, 0)；

（3）设置 Camera 组件的 **Projection** 为 Orthographic；

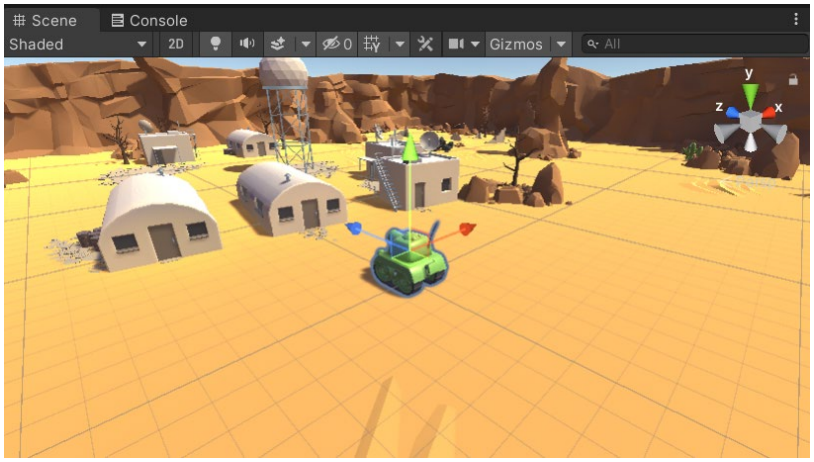
（4）设置 Camera 组件的 **Clear Flags** 为 Solid Color；

（5）设置 Camera 组件的 **Background** 为(80, 60, 50)；



2.2 坦克的创建

1、在 Models 文件夹中将 Tank 模型添加到场景，初始位置设置为(0, 0, 0)。



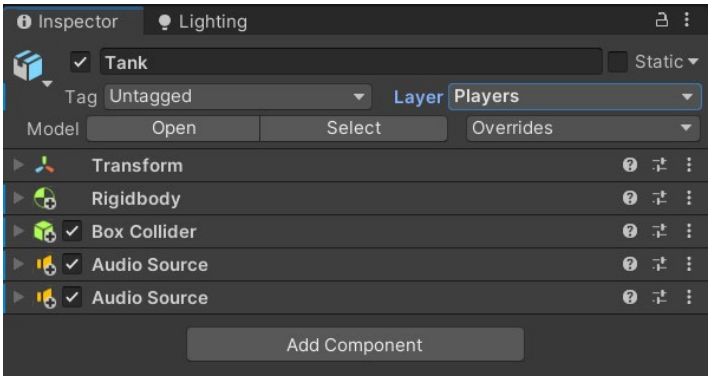
2、将 Tank 对象的 Layer 属性设置为 Players（在弹出的对话框中选择“No, this object only”），设置 Layer 的目的主要是便于在制作爆炸效果时，只需要销毁坦克，而不需要销毁其他游戏对象。

3、给 Tank 对象添加 Rigidbody 组件，勾选 Freeze Position 的 Y 轴，因为坦克不需要上下移动；勾选 Freeze Rotation 的 X 轴和 Z 轴，因为坦克只需要在竖直轴上旋转即可。

4、给 Tank 对象添加 Box Collider 组件，将 Box Collider 的 Center 设置为(0, 0.85, 0)，Size 设置为(1.5, 1.7, 1.6)。

5、给 Tank 对象添加 Audio Source 组件，将 AudioClip 设置为 EngineIdle，并勾选 Loop 和 Play On Awake，使其从一开始便循环播放坦克引擎的声音。

6、再添加一个 Audio Source 组件，用于播放其他音源，例如子弹射击、蓄力等，这些音频将会在脚本中动态添加。取消勾选 Play On Awake 和 Loop。



7、设置完毕后，将 Tank 对象添加到 Prefabs 文件夹，作为预制体，便于后续创建多辆坦克对象，以及在坦克销毁后再次生成。保存场景。

2.3 坦克的移动

1、在 Prefabs 文件夹中找到“DustTrial”预制体，将其作为坦克行走时扬起的灰尘特效。将其拖放至 Tank 对象上，作为一个 Tank 的子对象。

2、再在 Hierarchy 面板中复制 DustTrail 对象，作为坦克两边轮子的扬尘，将两者分别重命名为“LeftDustTrail”和“RightDustTrail”。

3、LeftDustTrail 的 Position 设置为(-0.5, 0, -0.75)，RightDustTrail 的 Position 设置为(0.5, 0, -0.75)。

4、在 Scripts/Tank 文件夹中找到 TankMovement 脚本，将其添加到 Tank 对象上。编辑脚本用于控制坦克的移动，以及动态改变移动及停止时的音效，以及扬尘的

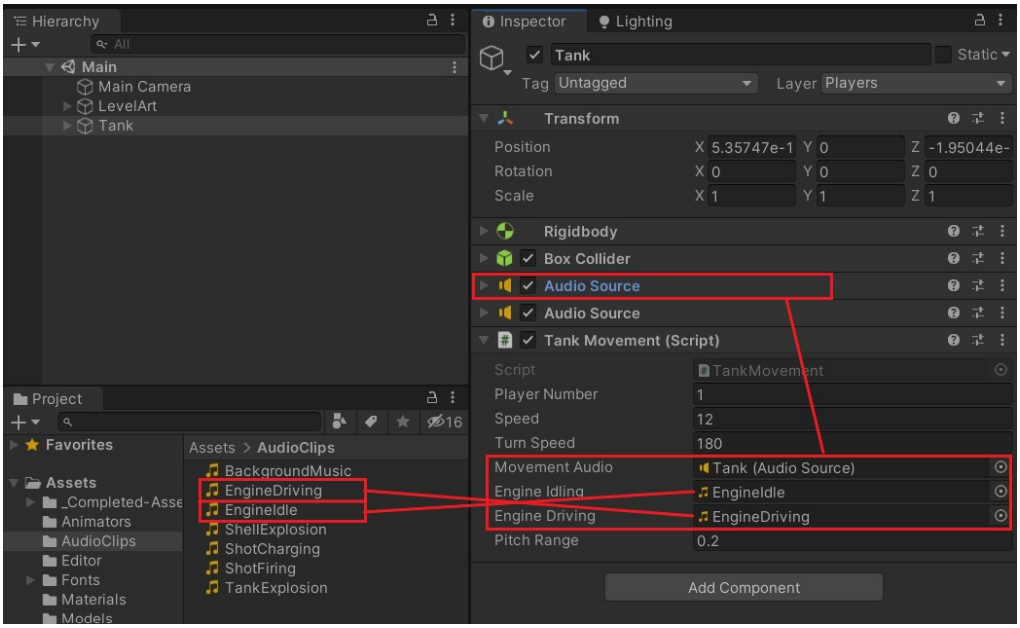
显示。脚本及注释内容如下：

```
1. using UnityEngine;
2.
3. public class TankMovement : MonoBehaviour
4. {
5.     public int m_PlayerNumber = 1;         // 玩家编号（玩家 1、玩家
   2）
6.     public float m_Speed = 12f;            // 坦克行驶的速度
7.     public float m_TurnSpeed = 180f;       // 坦克每次转过的角度
8.     public AudioSource m_MovementAudio;    // 坦克行驶时的音频组件，在
   Unity 中将 Tank 对象的第一个 Audio Source 赋给它
9.     public AudioClip m_EngineIdling;       // 音源 1，在 Unity 编辑器中赋
   值
10.    public AudioClip m_EngineDriving;      // 音源 2，在 Unity 编辑器中赋
   值
11.    public float m_PitchRange = 0.2f;
12.
13.
14.    private string m_MovementAxisName;     // 行驶方向轴的名称
   （Vertical1、Vertical2）
15.    private string m_TurnAxisName;         // 转弯方向轴的名称
   （Horizontal1、Horizontal2）
16.    private Rigidbody m_Rigidbody;         // 坦克对象的引用
17.    private float m_MovementInputValue;    // 行驶轴（Vertical）的输入
   大小
18.    private float m_TurnInputValue;        // 转弯轴（Horizontal）的输
   入大小
19.    private float m_OriginalPitch;
20.
21.    private void Awake()
22.    {
23.        // 获取到坦克对象的引用
24.        m_Rigidbody = GetComponent<Rigidbody>();
25.    }
26.
27.    private void OnEnable ()
28.    {
29.        // isKinematic 是使其不受任何力，即初始化坦克的受力为 0
30.        // 即不受物理引擎的影响，接受物理作用，但没有物理效果
31.        m_Rigidbody.isKinematic = false;
32.        // 输入重置为 0
33.        m_MovementInputValue = 0f;
34.        m_TurnInputValue = 0f;
35.    }
```

```
36.
37.     private void OnDisable ()
38.     {
39.         m_Rigidbody.isKinematic = true;
40.     }
41.
42.     private void Start()
43.     {
44.         // 根据玩家编号当前初始化输入轴的名称，便于后续处理输入
45.         m_MovementAxisName = "Vertical" + m_PlayerNumber;
46.         m_TurnAxisName = "Horizontal" + m_PlayerNumber;
47.
48.         // Tank 第一个 Audio Source 组件的 pitch
49.         m_OriginalPitch = m_MovementAudio.pitch;
50.     }
51.
52.     // 存储玩家的输入，并且确保引擎的声音被播放
53.     private void Update()
54.     {
55.         // 随后会自动调用 FixedUpdate()函数根据用户输入实现坦克的移动
56.         m_MovementInputValue = Input.GetAxis(m_MovementAxisName);
57.         m_TurnInputValue = Input.GetAxis(m_TurnAxisName);
58.         // 播放音频
59.         EngineAudio();
60.     }
61.
62.     // 基于坦克是否正在移动，以及坦克当前播放的音源播放音频
63.     private void EngineAudio()
64.     {
65.         // 如果坦克没有移动，播放 EngineIdling 音源
66.         if (Mathf.Abs(m_MovementInputValue) < 0.1f && Mathf.Abs(m_Tu
    rnInputValue) < 0.1f)
67.         {
68.             // 如果当前播放的就是 EngineDriving，改成 EngineIdling
69.             if (m_MovementAudio.clip == m_EngineDriving)
70.             {
71.                 m_MovementAudio.clip = m_EngineIdling;
72.                 m_MovementAudio.pitch = Random.Range(m_OriginalPitch
     - m_PitchRange, m_OriginalPitch + m_PitchRange);
73.
74.                 // 更改音源后应该重新播放
75.                 m_MovementAudio.Play();
76.             }
77.         }
```

```
78.         // 如果坦克正在移动，播放 EngineDriving 音源
79.         else
80.         {
81.             if (m_MovementAudio.clip == m_EngineIdling)
82.             {
83.                 m_MovementAudio.clip = m_EngineDriving;
84.                 m_MovementAudio.pitch = Random.Range(m_OriginalPitch
    - m_PitchRange, m_OriginalPitch + m_PitchRange);
85.
86.                 m_MovementAudio.Play();
87.             }
88.         }
89.     }
90.
91.     private void FixedUpdate()
92.     {
93.         // 坦克的移动
94.         Move();
95.         Turn();
96.     }
97.
98.     // 基于用户的输入，计算 Tank 该前进多远，调整 Tank 的 Position
99.     private void Move()
100.        {
101.            Vector3 movement = transform.forward * m_MovementInputVa
   lue * m_Speed * Time.deltaTime;
102.            m_Rigidbody.MovePosition(m_Rigidbody.position + movement
   );
103.        }
104.
105.        // 基于用户的输入，调整 Tank 的 Rotation
106.        private void Turn()
107.        {
108.            // 计算角度 Quaternion（transform 组件的 Rotation 属性）
109.            float turn = m_TurnInputValue * m_TurnSpeed * Time.delta
   Time;
110.            // 将 Vector3 转换为 Quaternion 角度，沿 Y 轴旋转
111.            Quaternion turnRotation = Quaternion.Euler(0f, turn, 0f)
   ;
112.            m_Rigidbody.MoveRotation(m_Rigidbody.rotation * turnRota
   tion);
113.        }
114.    }
```

在 Unity 编辑器中设置 TankMovement 脚本的 public 变量，如下所示：



2.4 相机设置

1、在 Hierarchy 面板创建一个空游戏物体并重命名为 CameraRig。Position 设置为(0, 0, 0)，Rotation 设置为(40, 60, 0)。

2、将 Main Camera 拖放至 CameraRig 对象上，作为子对象。将 Main Camera 的 Position 设置为(0, 0, -65)，Rotation 设置为(0, 0, 0)。

3、在 Project 面板中，从 Scripts/Camera 文件夹下将 CameraControl 脚本添加到 CameraRig 游戏物体上。

4、双击打开脚本，利用脚本控制相机跟随坦克移动，并且随着坦克的运动而放大或缩小。脚本内容如下：

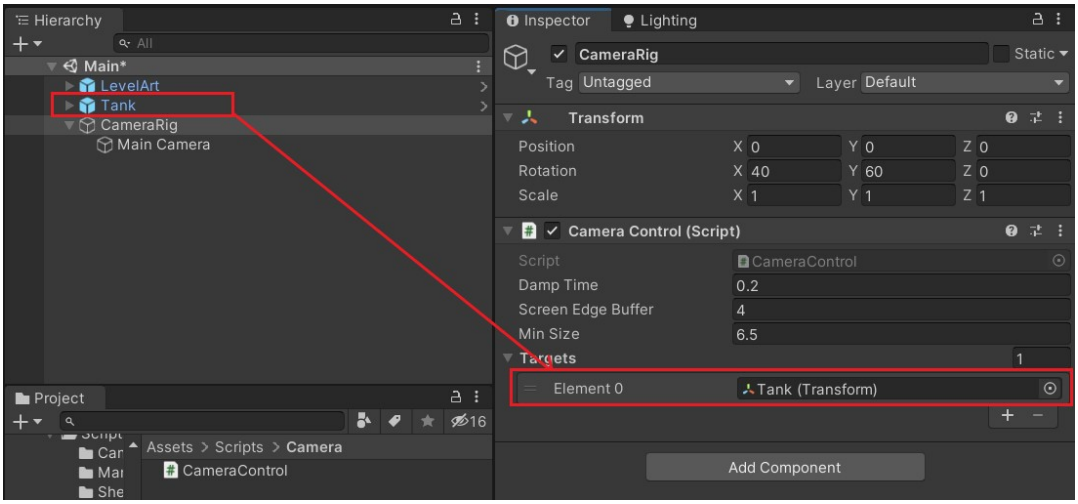
```
1. using UnityEngine;
2.
3. public class CameraControl : MonoBehaviour
4. {
5.     // 镜头移动的时间间隔
6.     public float m_DampTime = 0.2f;
7.     public float m_ScreenEdgeBuffer = 4f;
8.     // 最小的镜头大小，避免一直放大
9.     public float m_MinSize = 6.5f;
10.    // HideInInspector 表示不在 Inspector 面板中显示，即不可在 Unity 编辑器中指定
11.    /*[HideInInspector]*/ public Transform[] m_Targets;    // 场景中所有坦克的 transform 组件数组
12.
13.
```

```
14.     private Camera m_Camera;                // 镜头对象的引用
15.     private float m_ZoomSpeed;
16.     private Vector3 m_MoveVelocity;
17.     private Vector3 m_DesiredPosition;
18.
19.
20.     private void Awake()
21.     {
22.         // GetComponentInChildren 获取到的是满足条件的第一个子对象
23.         m_Camera = GetComponentInChildren<Camera>();
24.     }
25.
26.     // 因为 Tank 的移动是在 FixedUpdate()函数中实现的，而镜头需要跟随移动，
    所以镜头也在该函数中移动
27.     private void FixedUpdate()
28.     {
29.         Move();
30.         Zoom();
31.     }
32.
33.
34.     private void Move()
35.     {
36.         // 1.找到场景中存活坦克间的平均距离
37.         FindAveragePosition();
38.         // 平滑移动到指定位置
39.         transform.position = Vector3.SmoothDamp(transform.position,
    m_DesiredPosition, ref m_MoveVelocity, m_DampTime);
40.     }
41.
42.     // 找到所有被激活坦克中间的位置，为 m_DesiredPosition 赋值
43.     private void FindAveragePosition()
44.     {
45.         Vector3 averagePos = new Vector3();
46.         int numTargets = 0;
47.
48.         for (int i = 0; i < m_Targets.Length; i++)
49.         {
50.             // 如果坦克没有被激活，就不计入计算，跳过
51.             if (!m_Targets[i].gameObject.activeSelf)
52.                 continue;
53.
54.             // 将所有被激活 Tank 的 position 求和
55.             averagePos += m_Targets[i].position;
```

```
56.             // 被激活坦克的数量加 1
57.             numTargets++;
58.         }
59.
60.         // 如果被激活坦克的数量大于 0，求 position 的平均值
61.         if (numTargets > 0)
62.             averagePos /= numTargets;
63.
64.         // 确保镜头不会上下偏移，因为 Tank 的 y 轴 Position 是固定的，因此镜头的 y 轴也不需要改变，每次保持和前面一致就好了
65.         averagePos.y = transform.position.y;
66.
67.         m_DesiredPosition = averagePos;
68.     }
69.
70.
71.     private void Zoom()
72.     {
73.         // 计算需要的 orthographic 镜头的大小
74.         float requiredSize = FindRequiredSize();
75.         // 设置 orthographic 镜头大小为计算得到的大小
76.         m_Camera.orthographicSize = Mathf.SmoothDamp(m_Camera.orthographicSize, requiredSize, ref m_ZoomSpeed, m_DampTime);
77.     }
78.
79.
80.     private float FindRequiredSize()
81.     {
82.         // 根据镜头的相对位置计算 CameraRig 的 Global 坐标
83.         Vector3 desiredLocalPos = transform.InverseTransformPoint(m_DesiredPosition);
84.
85.         float size = 0f;
86.
87.         // 找到所有坦克距离中心的距离，取最大值
88.         for (int i = 0; i < m_Targets.Length; i++)
89.         {
90.             if (!m_Targets[i].gameObject.activeSelf)
91.                 continue;
92.
93.             Vector3 targetLocalPos = transform.InverseTransformPoint(m_Targets[i].position);
94.
```

```
95.             Vector3 desiredPosToTarget = targetLocalPos - desiredLoc
   alPos;
96.
97.             size = Mathf.Max (size, Mathf.Abs (desiredPosToTarget.y)
   );
98.
99.             size = Mathf.Max (size, Mathf.Abs (desiredPosToTarget.x)
    / m_Camera.aspect);
100.            }
101.
102.            size += m_ScreenEdgeBuffer;
103.
104.            size = Mathf.Max(size, m_MinSize);
105.            return size;
106.        }
107.
108.
109.        public void SetStartPositionAndSize()
110.        {
111.            FindAveragePosition();
112.
113.            transform.position = m_DesiredPosition;
114.
115.            m_Camera.orthographicSize = FindRequiredSize();
116.        }
117.    }
```

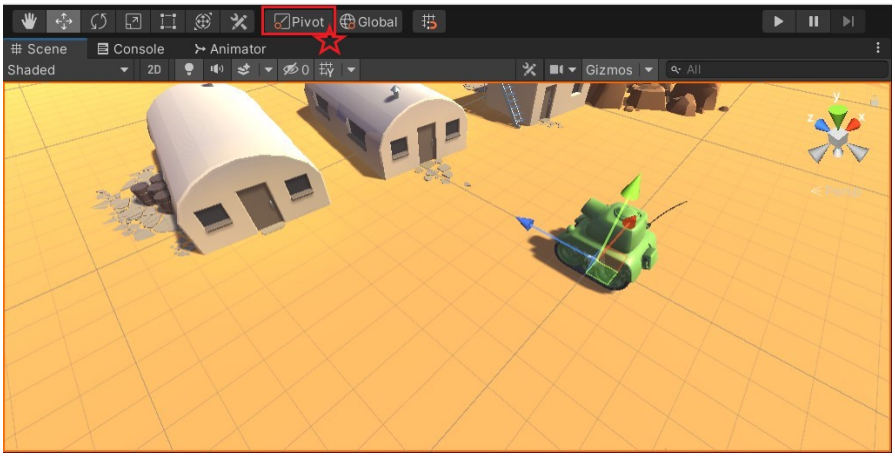
在 Unity 编辑器中，将 Tank 拖放至 Targets 变量中，添加游戏对象，如下图所示。



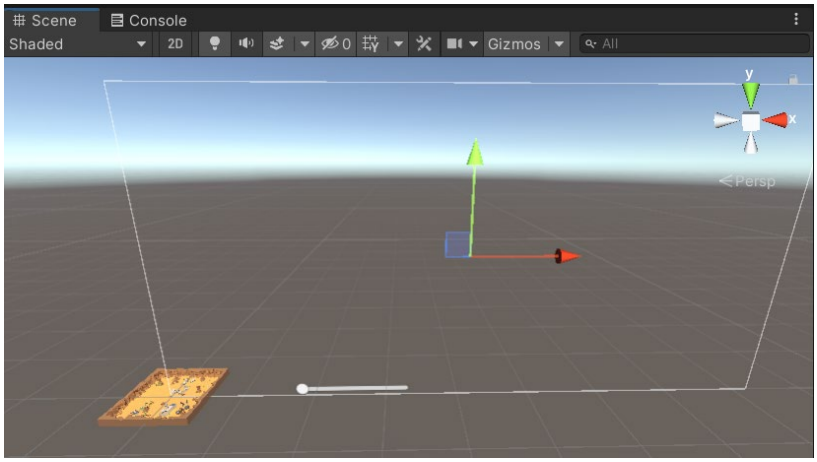
运行游戏，镜头会随着坦克的移动而跟随移动，自动放大缩小功能因为需要多辆坦克，这里暂时先不演示。

2.5 Tank 生命值显示

1、将 Scene 面板上方的 Transform Toggle 设置为 Pivot。



2、创建游戏物体 Slider：右键/【GameObject】->【UI】->【Slider】。

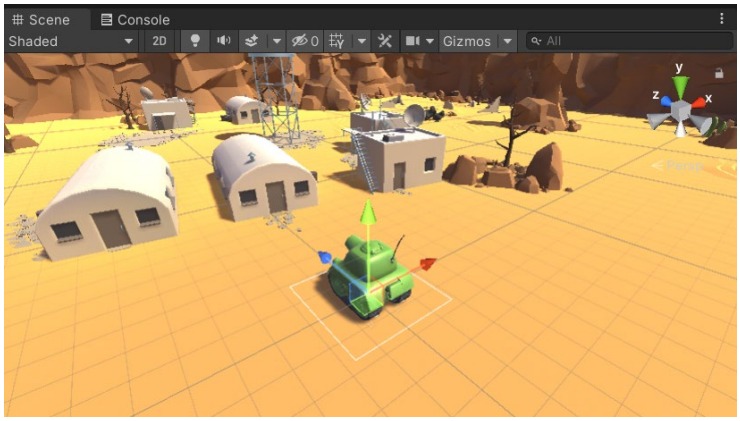


3、设置 EventSystem，在其 Standalone Input Module 组件中将 Horizontal Axis 设置为 HorizontalUI；将 Vertical Axis 设置为 VerticalUI。

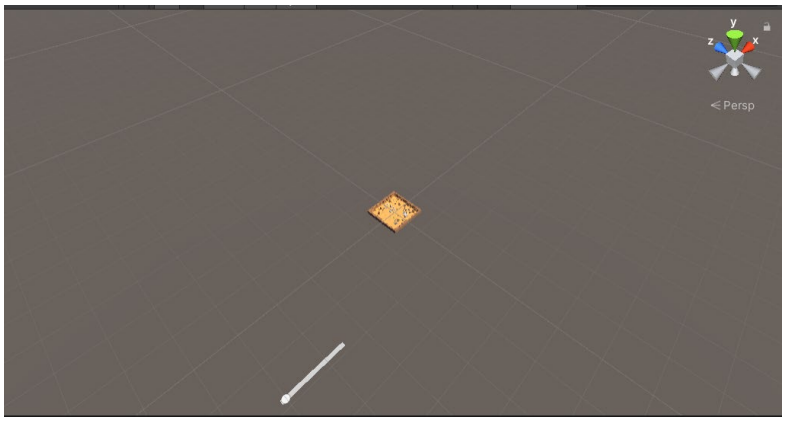
4、选中 Canvas 对象，在其 Canvas Scaler 组件中，将 Reference Pixels per Unit 设置为 1；在其 Canvas 组件中，将 Render Mode 设置为 World Space，将其固定在 World 中。

5、在 Hierarchy 面板中，将 Canvas 对象拖拽到 Tank 对象上，使之成为 Tank 的子对象；

6、选中 Canvas，将其 Rect Transform 组件的 PosX、PosY、PosZ 属性设置为(0, 0.1, 0)；将 Width 和 Height 设置为 3.5；将 Rotation 设置为(90, 0, 0)。UI 到了如下位置：

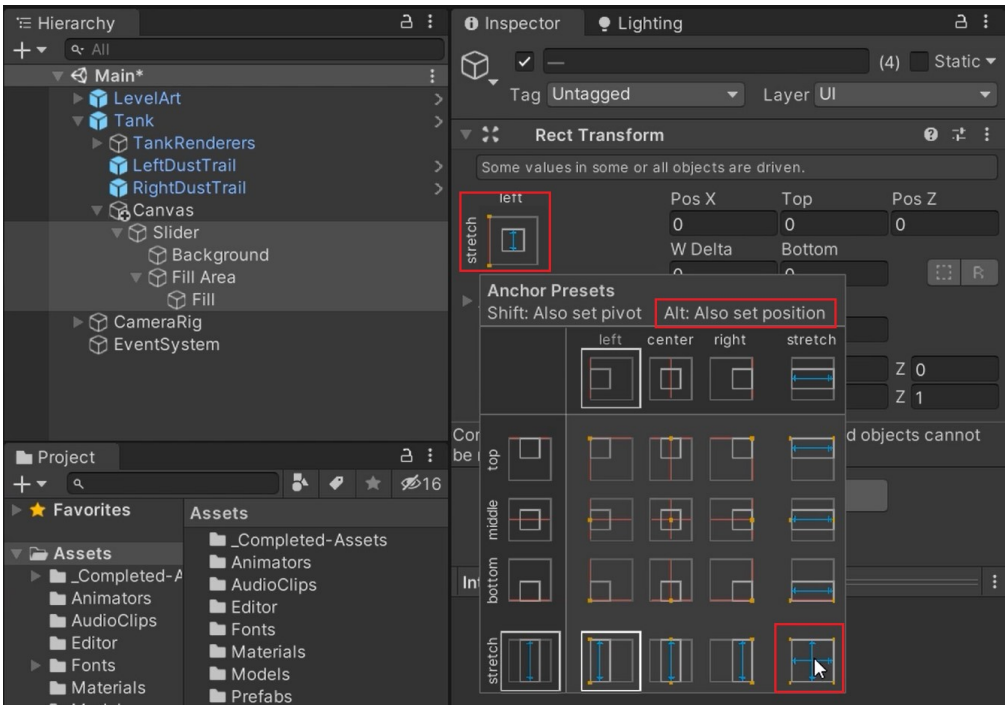


但此时，表示血量的滑条还是在外面：

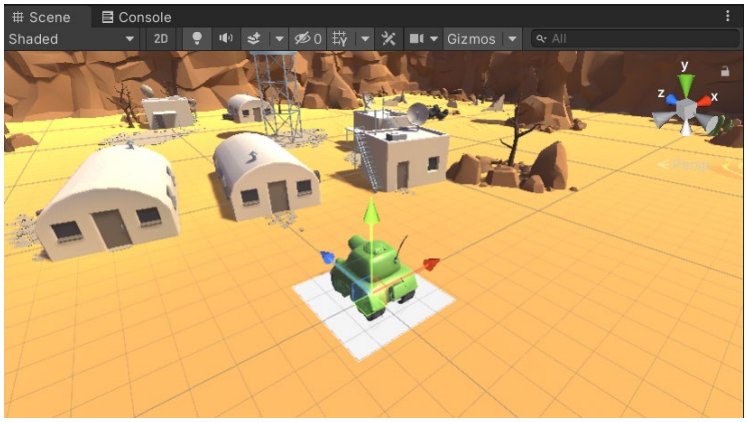


7、删除 Slider 对象的 HandleSlideArea 子对象。

8、同时选中 Slider，Background，Fill Area 和 Fill 对象，在右侧的 Inspector 面板中点击设置 stretch 属性。在弹出的 Anchor Presets 窗口，按住 Alt，点击最右下角的图形。



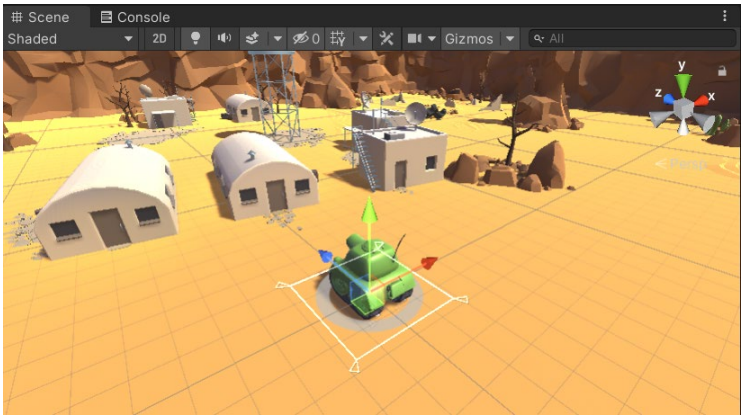
现在，Slider 就在 Tank 对象下面了。



9、选中 Slider 对象，将其重命名为 HealthSlider 在其 Slider 组件中，取消勾选 Interactable；将 Transition 设置为 None；将 Max Value 和 Value 设置为 100。

10、选中 HealthSlider 的子对象 Background，将其 Image 组件的 Source Image 改为 HealthWheel；Color 的透明度为 80。

11、选中 HealthSlider 的子对象 Fill，将其 Image 组件的 Source Image 属性设置为 Health Wheel；将其 Color 属性的不透明度设置为 150；将其 Image Type 设置为 Filled；将其 Fill Origin 属性设置为 Left；取消勾选 Clockwise。



12、在 Scripts/UI 文件夹下，找到 UIDirectionControl 脚本，并将该脚本添加到 HealthSlider 对象上。然后双击打开，编辑脚本，脚本内容如下：

```
1. using UnityEngine;
2.
3. public class UIDirectionControl : MonoBehaviour
4. {
5.     public bool m_UseRelativeRotation = true;
6.
7.     private Quaternion m_RelativeRotation;
8.
9.     private void Start()
10.    {
11.        m_RelativeRotation = transform.parent.localRotation;
12.    }
13.
14.    private void Update()
15.    {
16.        if (m_UseRelativeRotation)
17.            transform.rotation = m_RelativeRotation;
18.    }
19.}
```

13、将 Tank 的更改同步到 Prefabs 中的 Tank 预制体。

2.6 制作 Tank 爆炸特效

1、在 Prefabs 文件夹中，找到 TankExplosion 预制体，将其拖入 Hierarchy 面板。

2、给 TankExplosion 对象添加 Audio Source 组件。将 Audio Source 的 AudioClip 属性为 TankExplosion，并且取消勾选 Play On Awake。

3、选中 TankExplosion 对象，将更改同步到 Prefab。

4、同步更改后，删除 Hierarchy 面板中的 TankExplosion 对象。

2.7 Tank 血量控制

1、在 Scripts/Tank 文件夹中找到 TankHealth 脚本，将其添加到 Tank 对象上。

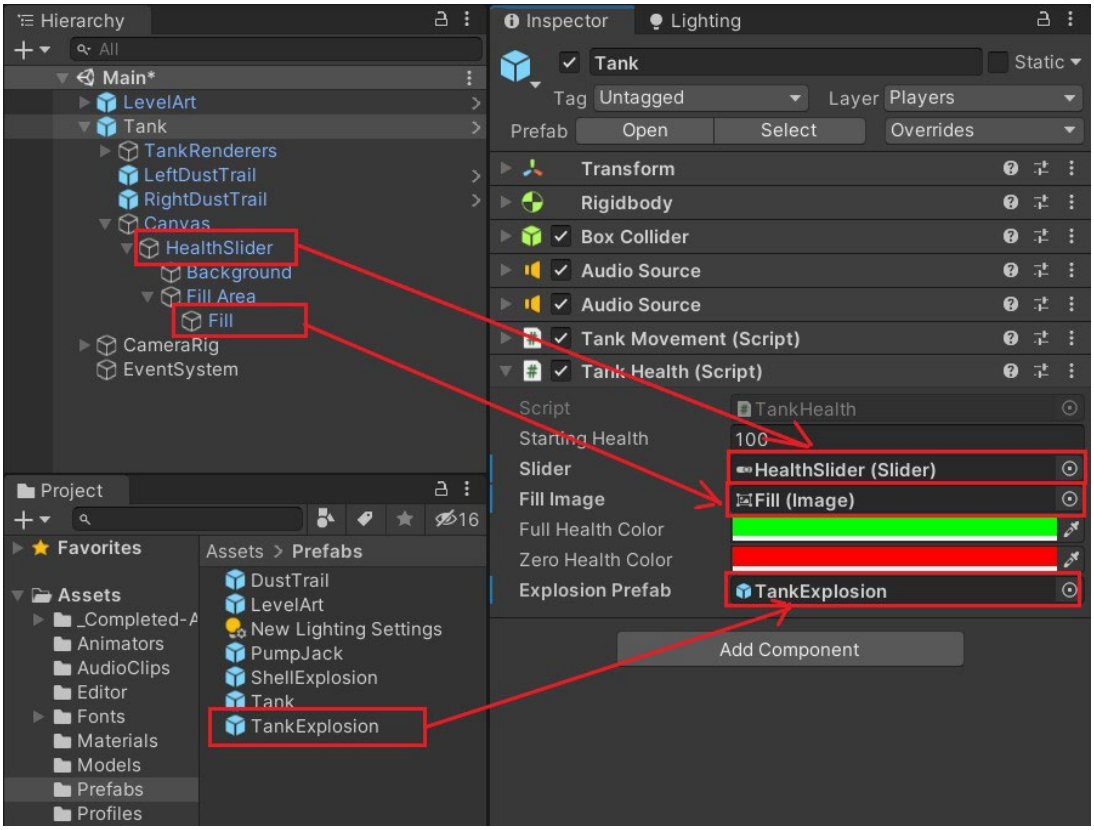
2、双击打开并编辑脚本，该脚本控制 Tank 受到伤害时的血量变化，以及血量为 0 时播放 Tank 的爆炸特效。脚本内容如下：

```
1. using UnityEngine;
2. using UnityEngine.UI;
3.
4. public class TankHealth : MonoBehaviour
5. {
6.     public float m_StartingHealth = 100f;           // 初始时的血量
7.     public Slider m_Slider;                         // 血条
   HealthSlider 对象的引用，在 Unity 中指定
8.     public Image m_FillImage;                       // 填充图 Fill 对象
   的引用，在 Unity 中指定
9.     public Color m_FullHealthColor = Color.green;  // 满血时的颜色
10.    public Color m_ZeroHealthColor = Color.red;    // 没血时的颜色
11.    public GameObject m_ExplosionPrefab;           // 坦克爆炸的特效
   TankExplosion 对象
12.
13.
14.    private AudioSource m_ExplosionAudio;          // 坦克爆炸音效的引
   用 AudioSource 即可
15.    private ParticleSystem m_ExplosionParticles;   // 坦克爆炸特效
16.    private float m_CurrentHealth;                 // 当前血量
17.    private bool m_Dead;                           // 坦克是否已死亡
   （血量耗尽以否）
18.
19.
20.    private void Awake()
21.    {
22.        // 创建 TankExplosion 对象，并获取该对象的粒子系统
23.        m_ExplosionParticles = Instantiate(m_ExplosionPrefab).GetCom
   ponent<ParticleSystem>();
24.        m_ExplosionAudio = m_ExplosionParticles.GetComponent<AudioSo
   urce>();
25.
26.        // 创建后先设置为未激活状态，后续需要使用时只需要激活即可
27.        m_ExplosionParticles.gameObject.SetActive(false);
28.    }
29.
30.
31.    private void OnEnable()
```

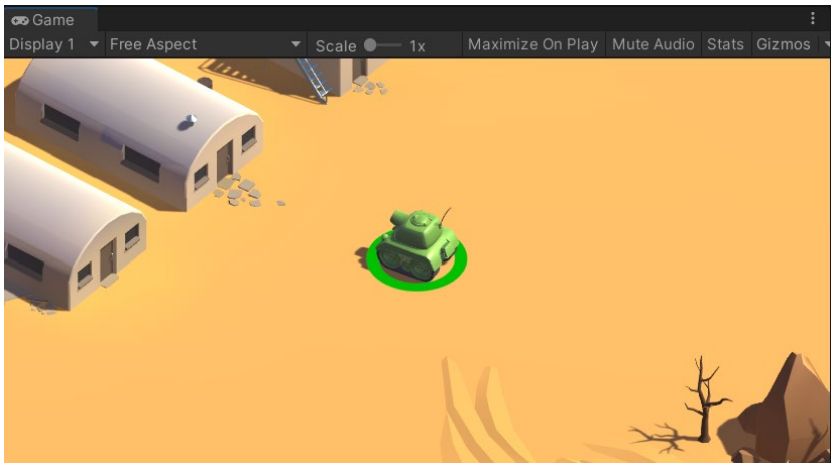
```
32.     {
33.         // 当 Tank 被激活时，初始化血量为满血，死亡状态为“未死”
34.         m_CurrentHealth = m_StartingHealth;
35.         m_Dead = false;
36.
37.         // 更新血条显示
38.         SetHealthUI();
39.     }
40.
41.     // Tank 承受伤害的函数
42.     // 参数 amount 即为子弹造成的伤害
43.     // 调整 Tank 的当前血量，基于当前血量更新血条的 UI 显示，并判断 Tank 是否死亡
44.     public void TakeDamage(float amount)
45.     {
46.         // 更新当前血量
47.         m_CurrentHealth -= amount;
48.         // 基于当前血量，更新血条显示
49.         SetHealthUI();
50.         // 判断 Tank 血量是否需要死亡，如果小于 0，则调用死亡函数
51.         if (m_CurrentHealth <= 0 && !m_Dead)
52.         {
53.             OnDeath();
54.         }
55.     }
56.
57.     // 根据当前血量设置血条的大小和颜色
58.     private void SetHealthUI()
59.     {
60.         m_Slider.value = m_CurrentHealth;
61.         // 根据满血和没血确定残血的颜色
62.         m_FillImage.color = Color.Lerp(m_ZeroHealthColor, m_FullHealthColor, m_CurrentHealth / m_StartingHealth);
63.     }
64.
65.     // 播放 Tank 的爆炸特效，并且取消激活 Tank 对象
66.     private void OnDeath()
67.     {
68.         m_Dead = true;
69.         // 播放特效
70.         m_ExplosionParticles.transform.position = transform.position;
71.         m_ExplosionParticles.gameObject.SetActive(true);
72.
```

```
73.         m_ExplosionParticles.Play();
74.         m_ExplosionAudio.Play();
75.
76.         gameObject.SetActive(false);
77.     }
78. }
```

在 Unity 中指定该脚本的 public 变量，如下图所示：



运行游戏，Tank 初始血条为绿色。



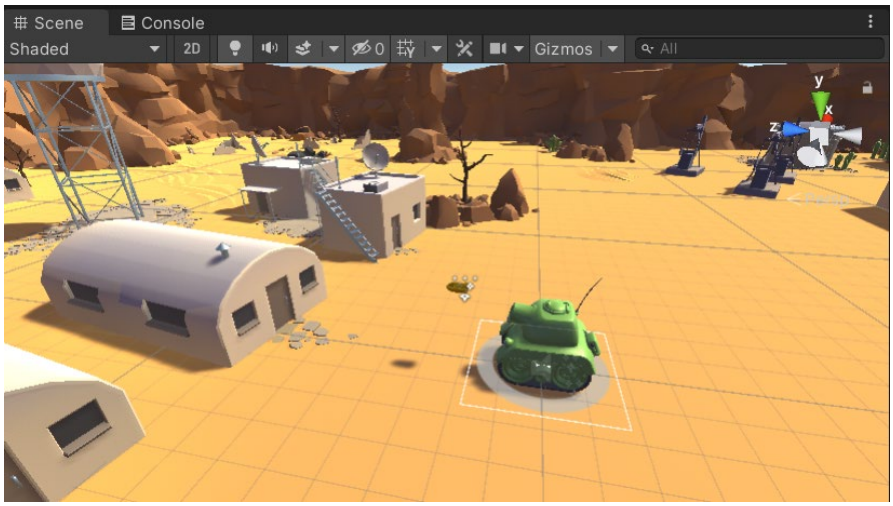
3、更新 Tank 对象的更改到对应的 Prefab，保存场景。

2.8 Shell 预制体的制作

1、在 Models 文件夹中找到名为 Shell 的模型，将其添加到场景中。

2、给 Shell 对象添加 Capsule Collider 组件，勾选 Is Trigger；将 Direction 属性设置为 Z-Axis；将 Center 设置为(0, 0, 0.2)；将 Radius 设置为 0.15；将 Height 设置为 0.55。

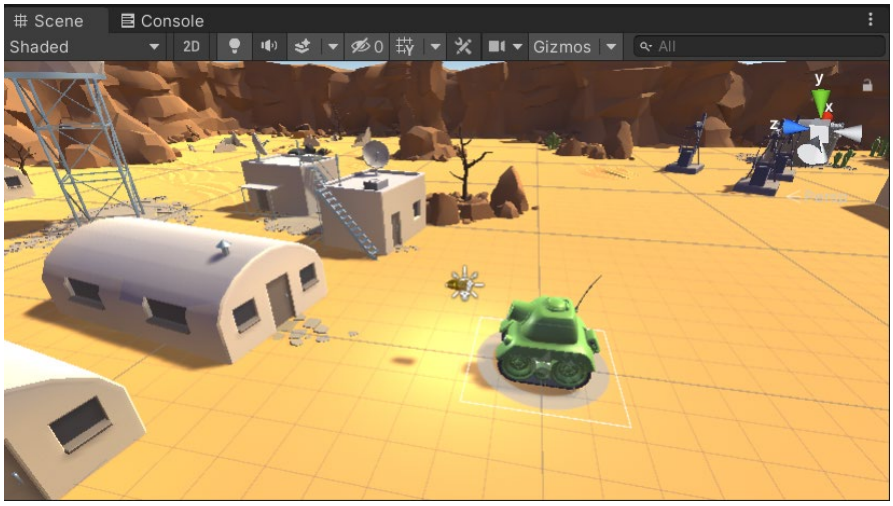
3、为 Shell 对象添加 Rigidbody 组件。



4、在 Prefabs 文件夹下找到 ShellExplosion 预制体，将其拖拽到 Shell 对象上，作为 Shell 的子对象。

5、为 ShellExplosion 对象添加 Audio Source 组件，设置 AudioClip 组件为 ShellExplosion；取消勾选 Play On Awake 选项。

6、为 Shell 对象添加 Light 组件。



7、在 Project 面板中的 Scripts/Shell 文件夹下找到 ShellExplosion 脚本，将其添加到 Shell 对象中，并双击打开。该脚本主要控制子弹的爆炸特效播放。脚本内容如下：

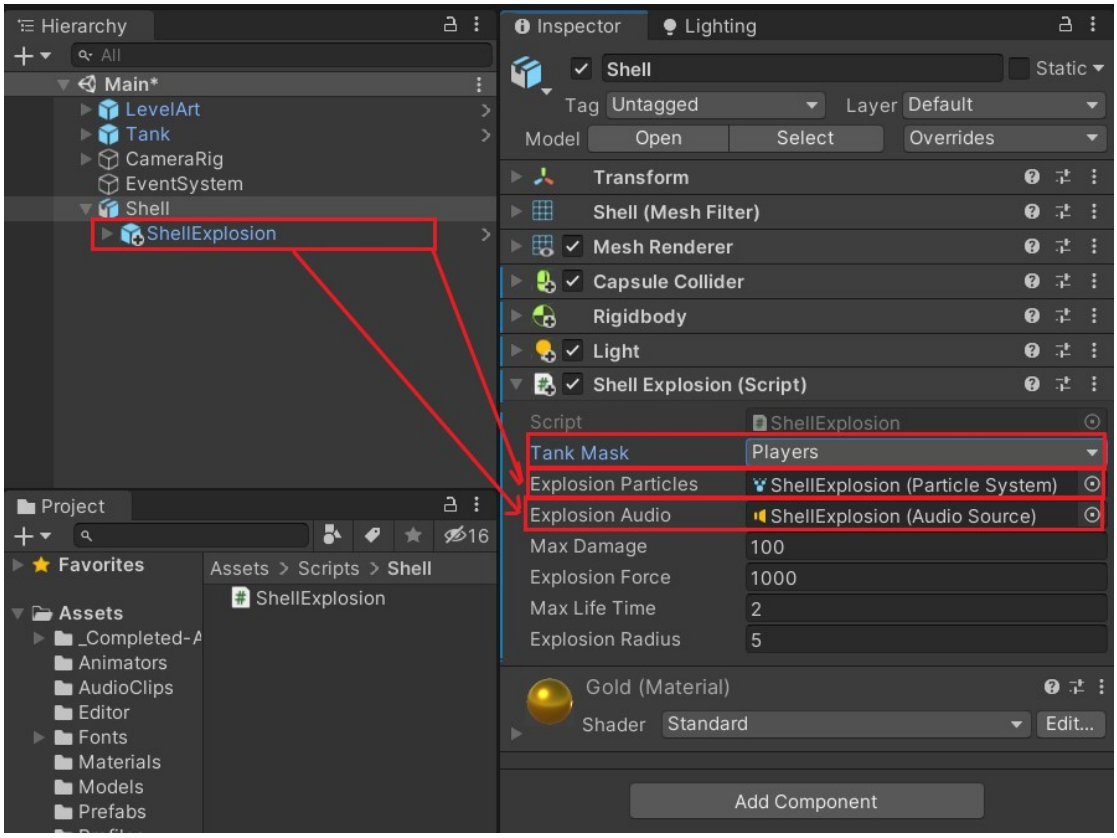
```
1. using UnityEngine;
```

```
2.
3. public class ShellExplosion : MonoBehaviour
4. {
5.     public LayerMask m_TankMask;
6.     public ParticleSystem m_ExplosionParticles;    // 子弹爆炸的特
   效
7.     public AudioSource m_ExplosionAudio;           // 子弹爆炸的音
   效
8.     public float m_MaxDamage = 100f;               // 造成的最大伤
   害
9.     public float m_ExplosionForce = 1000f;         // 最大的爆炸
   力
10.    public float m_MaxLifeTime = 2f;               // 子弹最长的生命周
   期
11.    public float m_ExplosionRadius = 5f;           // 爆炸半径
12.
13.
14.    private void Start()
15.    {
16.        // 在 m_MaxLifeTime 时间后销毁 Shell 对象
17.        Destroy(gameObject, m_MaxLifeTime);
18.    }
19.
20.
21.    private void OnTriggerEnter(Collider other)
22.    {
23.        // 找到所有在 shell 对象 Trigger 范围内的 Tank，对它们造成伤害
24.        // Trigger 范围的圆心为 transform.position，半径为
   m_ExplosionRadius，并且只捕获 Players 层的对象，即 Tank 对象
25.        Collider[] colliders = Physics.OverlapSphere(transform.posit
   ion, m_ExplosionRadius, m_TankMask);
26.
27.        // 遍历该层的所有对象
28.        for (int i = 0; i < colliders.Length; ++i)
29.        {
30.            // 查看碰撞器中是否找得到 Rigidbody
31.            Rigidbody targetRigidbody = colliders[i].GetComponent<Ri
   gidbody>();
32.            // 如果找不到，说明触发到的不是 Tank，继续下次循环
33.            if (!targetRigidbody)
34.            {
35.                continue;
36.            }
```

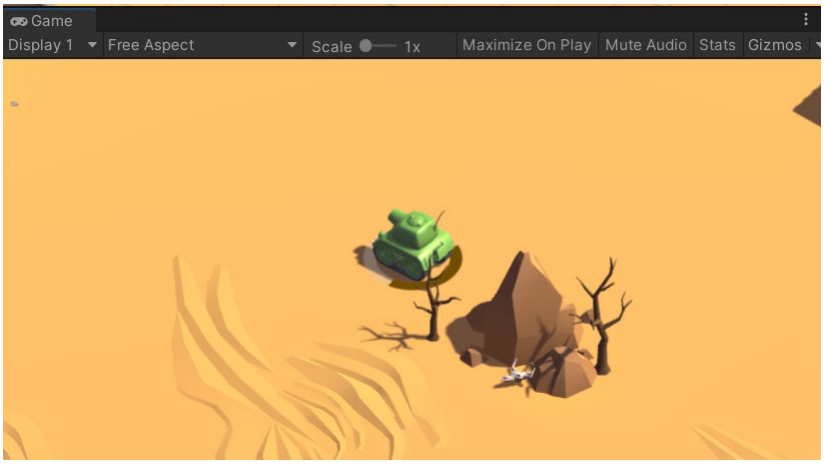
```
37.             // 如果是 Rigidbody，说明 targetRigidbody 是 Tank 对象，为 Tank 对象添加爆炸力
38.             targetRigidbody.AddExplosionForce(m_ExplosionForce, transform.position, m_ExplosionRadius);
39.             TankHealth targetTankHealth = targetRigidbody.GetComponent<TankHealth>();
40. 
41.             if (!targetTankHealth)
42.             {
43.                 continue;
44.             }
45. 
46.             float damage = CalculateDamage(targetRigidbody.position);
47.             targetTankHealth.TakeDamage(damage);
48.         }
49. 
50.         m_ExplosionParticles.transform.parent = null;
51.         // 播放特效和音效
52.         m_ExplosionParticles.Play();
53.         m_ExplosionAudio.Play();
54.         // 在子弹的生命周期后销毁粒子对象
55.         ParticleSystem.MainModule mainModule = m_ExplosionParticles.main;
56.         Destroy(m_ExplosionParticles.gameObject, mainModule.duration);
57. 
58.         Destroy(gameObject);
59.     }
60. 
61.     // 基于 Tank 的位置计算应该受到的伤害
62.     private float CalculateDamage(Vector3 targetPosition)
63.     {
64.         // 计算坦克和子弹之间的矢量距离
65.         Vector3 explosionToTarget = targetPosition - transform.position;
66. 
67.         float explosionDistance = explosionToTarget.magnitude;
68.         float relativeDistance = (m_ExplosionRadius - explosionDistance) / m_ExplosionRadius;
69.         // 根据距离比例计算伤害值
70.         float damage = relativeDistance * m_MaxDamage;
71.         damage = Mathf.Max(0f, damage);
72. 
```

```
        return damage;
    }
}
```

在 Unity 编辑器中指定 ShellExplosion 的 public 变量，如下所示：



运行游戏，当子弹落下后，坦克受到伤害，血条减少，颜色改变，被击退。

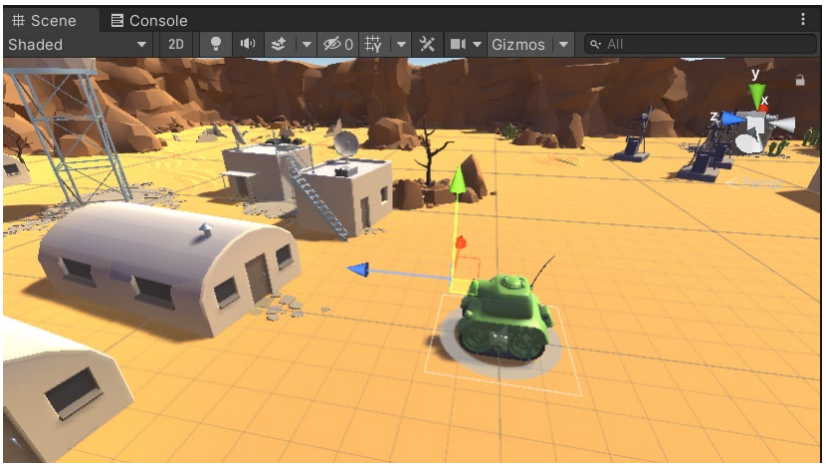


8、将 Shell 作为 Prefab 存入 Prefabs 文件夹，保存后删除 Shell 对象，因为子弹是动态生成的，不需要一开始就存在于场景中。删除后，保存场景。

2.9 Tank 发射效果的制作

1、选中 Tank 对象，在 Tank 对象中创建空的子对象 Empty Object，并重命名为

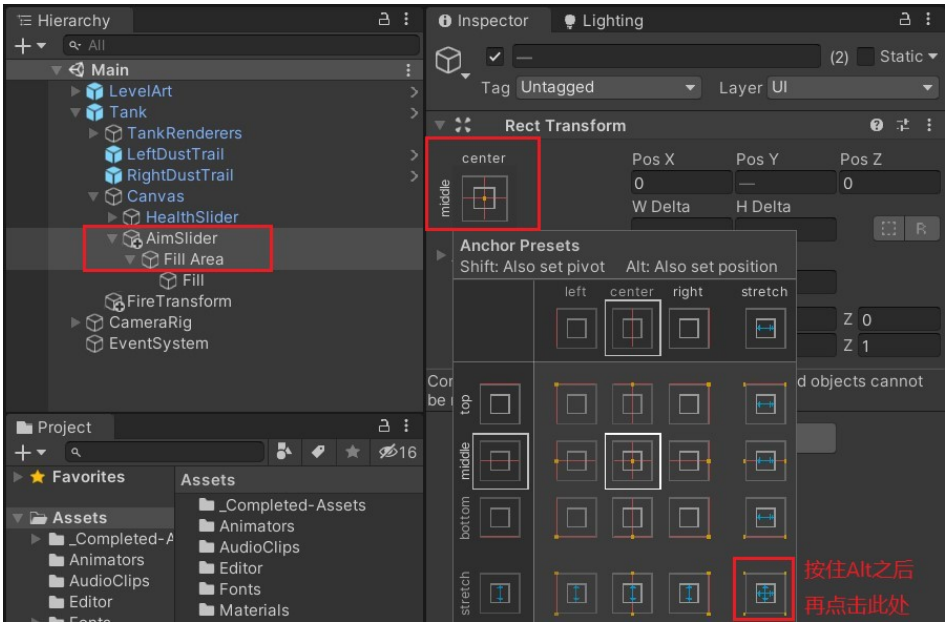
FireTransform。将 FireTransform 的 Position 设置为(0, 1.7, 1.35)，Rotation 设置为(350, 0, 0)。



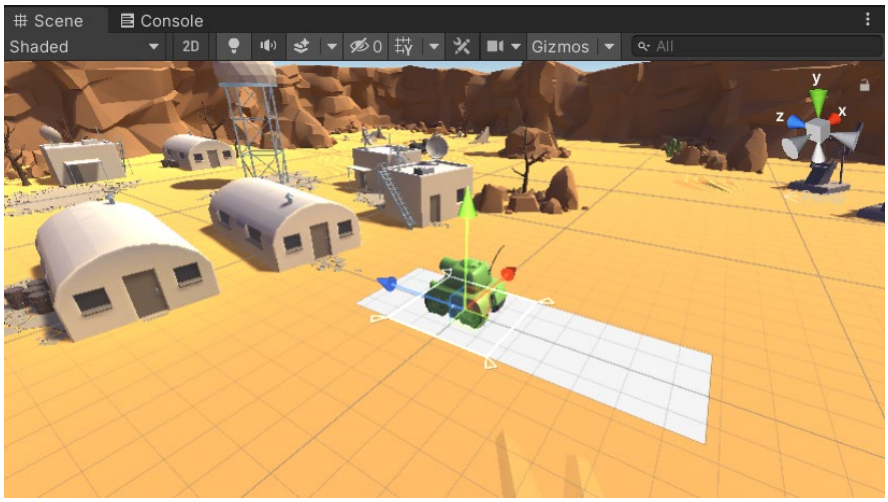
2、在 Tank/Canvas 对象下再创建一个 Slider，命名为 AimSlider，用于显示武器的蓄力效果。删除 AimSlider 的 Background 和 Handle Slide Area 子对象。

3、选中 AimSlider 对象，在其 Slider 组件中，取消勾选 Interactable；设置 Transition 为 None；设置 Direction 为 Bottom To Top；设置 Min Value 为 15；设置 Max Value 为 30。

4、同时选中 AimSlider 和 Fill Area 对象，在右侧的 Inspector 面板中点击设置 stretch 属性。在弹出的 Anchor Presets 窗口，按住 Alt，点击最右下角的图形。

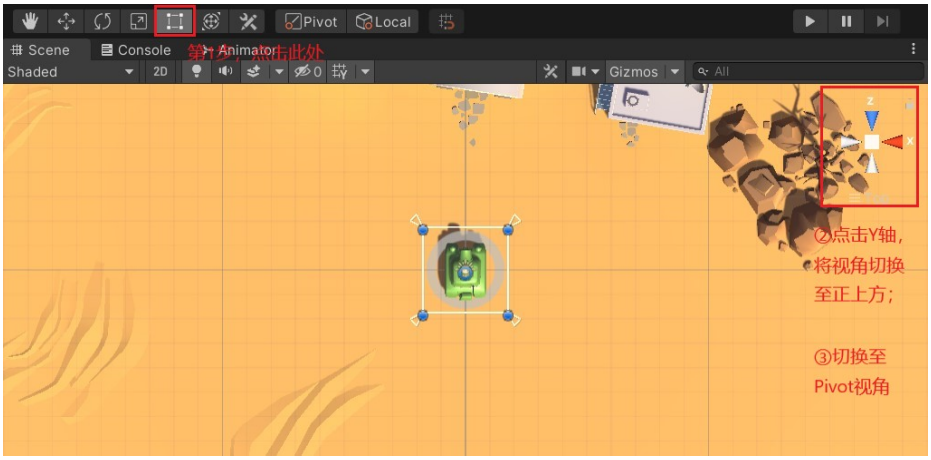


场景中显示如下：

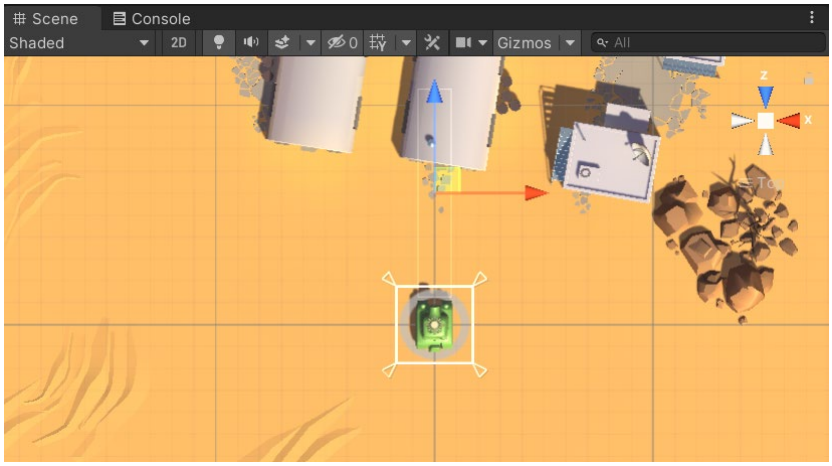


5、选中 Fill 对象，设置其 Rect Transform 组件的 Height 属性为 0，以删除偏移；设置其 Image 组件的 Source Image 为 Aim Arrow。

6、选中 AimSlider 对象，切换到 Rect Tool 视图，将视角调整至坦克正上方（点击右上角坐标的 Y 轴即可），切换至 Pivot 视角。



7、将 AimSlider 的左右两边贴紧坦克的两边，将前面的边拉长，然后将其后面的边拖拽到坦克炮筒前方的位置。也可通过直接设置 Rect Transform 的属性来实现，其 Rect Transform 的值大概是(1, -9, -1, 1, 3)。



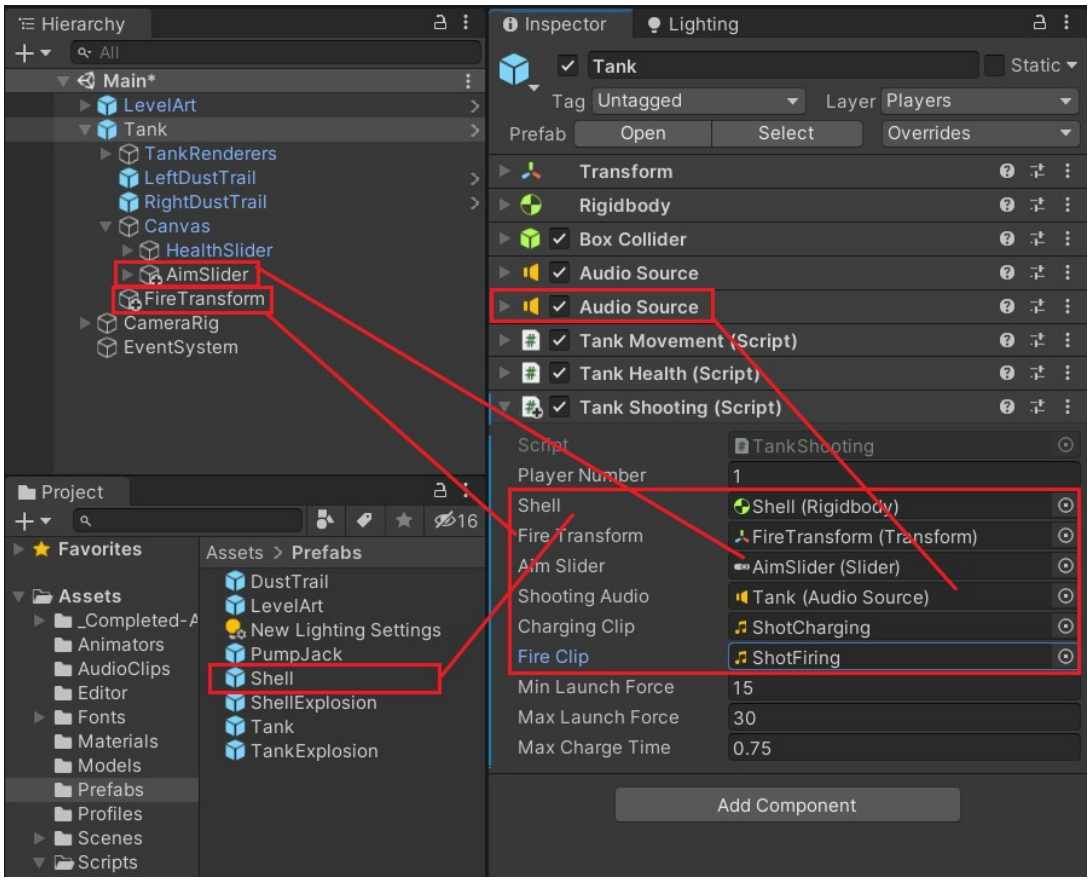
8、在 Scripts/Tank 文件夹中找到 TankShooting 脚本，将其添加到 Tank 对象上。双击打开脚本，使该脚本控制子弹的发射，以及武器蓄力。脚本内容如下：

```
using UnityEngine;
using UnityEngine.UI;

public class TankShooting : MonoBehaviour
{
    public int m_PlayerNumber = 1;          // 玩家编号
    public Rigidbody m_Shell;               // 子弹对象的引用
    public Transform m_FireTransform;       // FireTransform 对象的引用
    public Slider m_AimSlider;              // 瞄准滑条的引用
    public AudioSource m_ShootingAudio;     // Tnak 对象的第 2 个 Audio Source 组件，在 Unity 中指定
    public AudioClip m_ChargingClip;        // 蓄力的音频
    public AudioClip m_FireClip;            // 发射子弹的音频
    public float m_MinLaunchForce = 15f;    // 与 AimSlider 的最值相同
    public float m_MaxLaunchForce = 30f;
    public float m_MaxChargeTime = 0.75f;   // 蓄能时间


    private string m_FireButton;            // "Fire" + m_PlayerNumber
    private float m_CurrentLaunchForce;
    private float m_ChargeSpeed;
    private bool m_Fired;


    private void OnEnable()
    {
        m_CurrentLaunchForce = m_MinLaunchForce;
        m_AimSlider.value = m_MinLaunchForce;
    }


    private void Start()
    {
        m_FireButton = "Fire" + m_PlayerNumber;
        // 速度 = （最大值 - 最小值）/ 蓄能时间
        m_ChargeSpeed = (m_MaxLaunchForce - m_MinLaunchForce) / m_MaxChargeTime;
    }

```

```
38.     // 追踪当前玩家开火键的状态（是否按下、长按），并根据当前的发射力做出决
    定
39.     private void Update()
40.     {
41.         m_AimSlider.value = m_MinLaunchForce;
42.
43.         // 如果蓄能到最大值，并且没有发射
44.         if (m_CurrentLaunchForce >= m_MaxLaunchForce && !m_Fired)
45.         {
46.             m_CurrentLaunchForce = m_MaxLaunchForce;
47.             Fire();
48.         }
49.         // 是不是第一次按下开火键？
50.         else if (Input.GetButtonDown(m_FireButton))
51.         {
52.             m_Fired = false;
53.             m_CurrentLaunchForce = m_MinLaunchForce;
54.
55.             // 播放蓄能音频
56.             m_ShootingAudio.clip = m_ChargingClip;
57.             m_ShootingAudio.Play();
58.         }
59.         // 按住了开火键，但是还没有发射
60.         else if (Input.GetButton(m_FireButton) && !m_Fired)
61.         {
62.             m_CurrentLaunchForce += m_ChargeSpeed * Time.deltaTime;


63.
64.             m_AimSlider.value = m_CurrentLaunchForce;
65.         }
66.         // 松开了开火键，但是还没有发射
67.         else if (Input.GetButtonUp(m_FireButton) && !m_Fired)
68.         {
69.             Fire();
70.         }
71.     }
72.
73.     // Instantiate and launch the shell.
74.     private void Fire()
75.     {
76.         m_Fired = true;
77.
78.         Rigidbody shellInstance = Instantiate(m_Shell, m_FireTransfo
    rm.position, m_FireTransform.rotation) as Rigidbody;
```

```
79.         shellInstance.velocity = m_CurrentLaunchForce * m_FireTransf
    orm.forward;
80.
81.         m_ShootingAudio.clip = m_FireClip;
82.         m_ShootingAudio.Play();
83.
84.         m_CurrentLaunchForce = m_MinLaunchForce;
85.     }
86. }
```

在 Unity 编辑器中指定 TankShooting 脚本的 public 变量，如下所示：



9、更新 Tank 对象的更改到对应的 Prefab，保存场景。

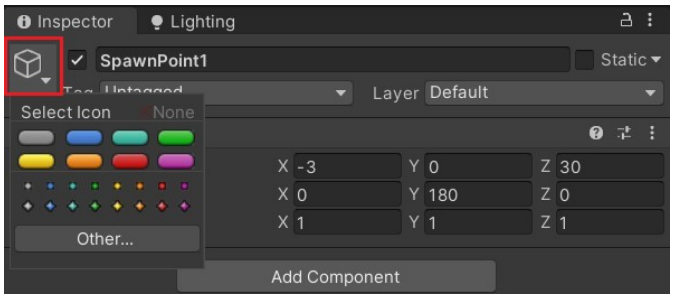
10、到此，Tank 对象已经制作完成，将其从场景中删除，后续由 Game Manager 创建，保存场景。

2.10 游戏控制器 Game Managers

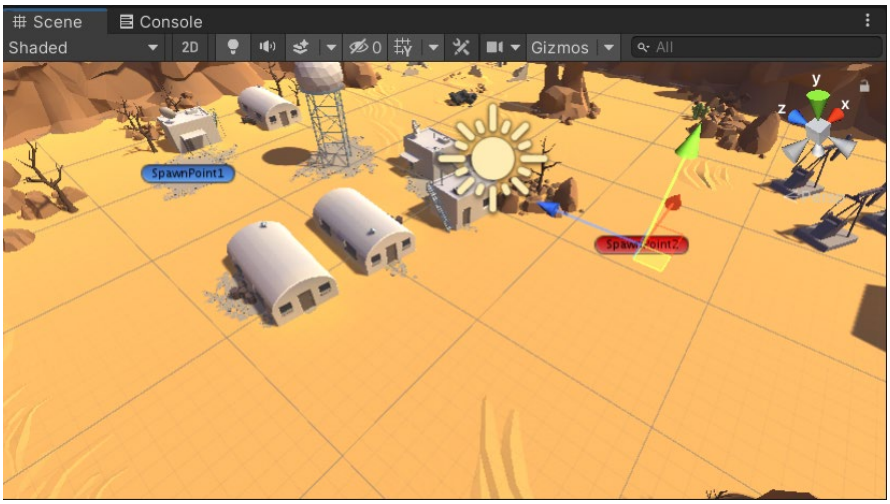
1、创建两个空游戏对象，分别命名为 SpawnPoint1 和 SpawnPoint2。SpawnPoint1 的 Position 设置为(-3, 0, 30)，Rotation 设置为(0, 180, 0)；SpawnPoint2 的 Position 设置为(13, 0, -5)，Rotation 设置为(0, 0, 0)

2、在 Inspector 面板中，在 GameObject 名字的旁边有一个立方体，表示 GameObject

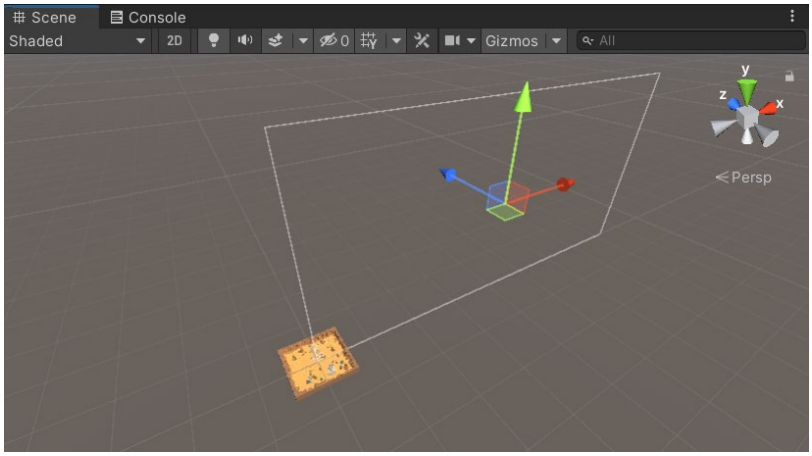
的 gizmo。如下图所示：



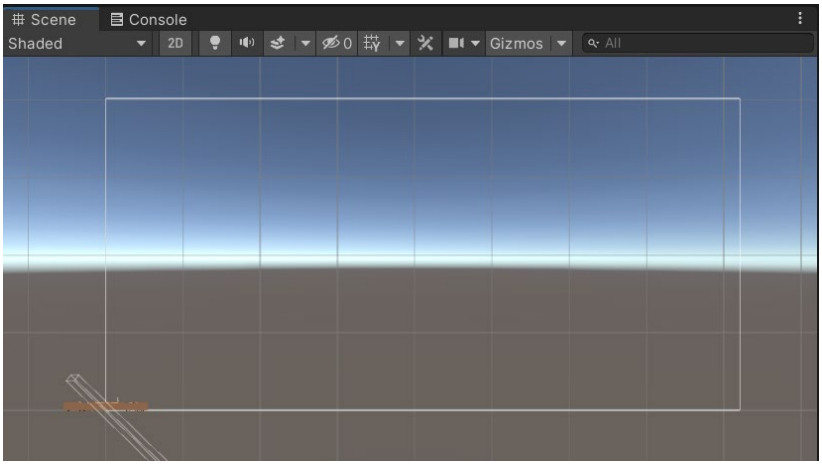
将 SpawnPoint1 的 gizmo 设置为蓝色，SpawnPoint 的 gizmo 设置为红色，这样，在场景中就能看到它们的位置。如下图：



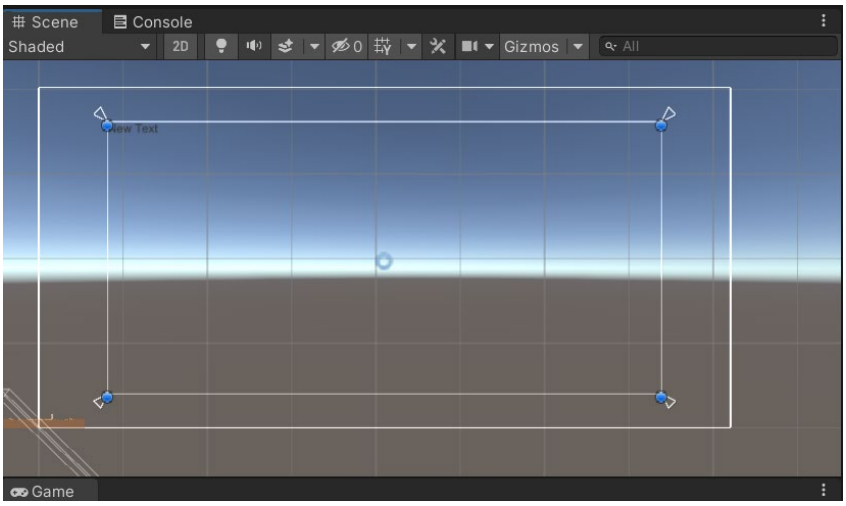
3、新建一个 Canvas 用于显示游戏提示信息，例如“Round 1”等等，将其命名为 MessageCanvas。



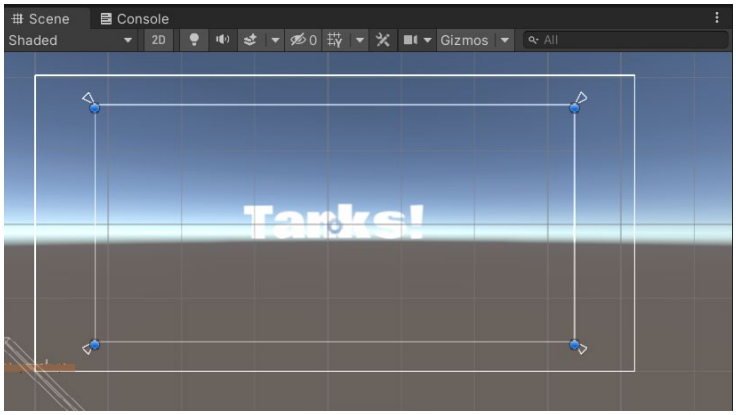
4、进入 2D 视图，对 MessageCanvas 进行操作。



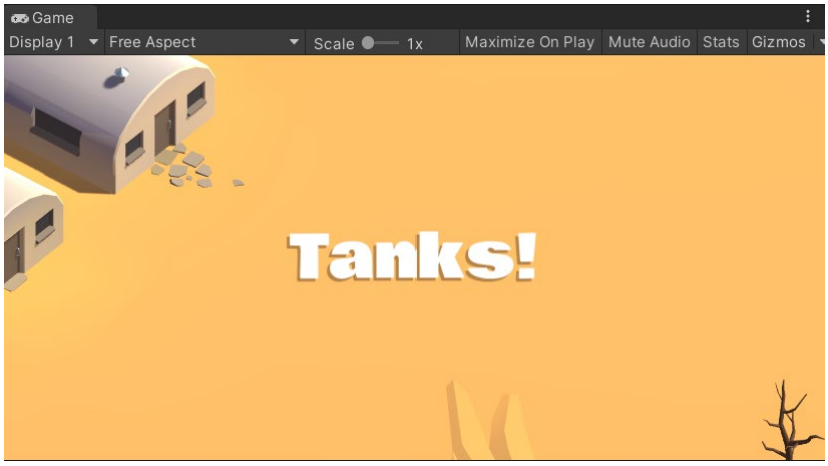
5、右键 MessageCanvas 创建一个 Text 子对象。在其 Rect Transform 组件中，将其 Anchors 属性的 Min 设置为(0.1, 0.1)，Max 设置为(0.9, 0.9)。将 Left, Right, Top, Bottom 和 Pos Z 都设置为 0。



6、文本设置为“Tanks!”，字体为 BowlbyOne-Regular，字体对齐方式为上下居中、左右居中。勾选 Best Fit；Max Size 设置为 60，Color 设置为(255, 255, 255, 255)。



7、为 Text 对象添加 Shadow 组件，Effect Color 设置为(114, 71, 40, 128)，Effect Distance 设置为(-3, -3)。

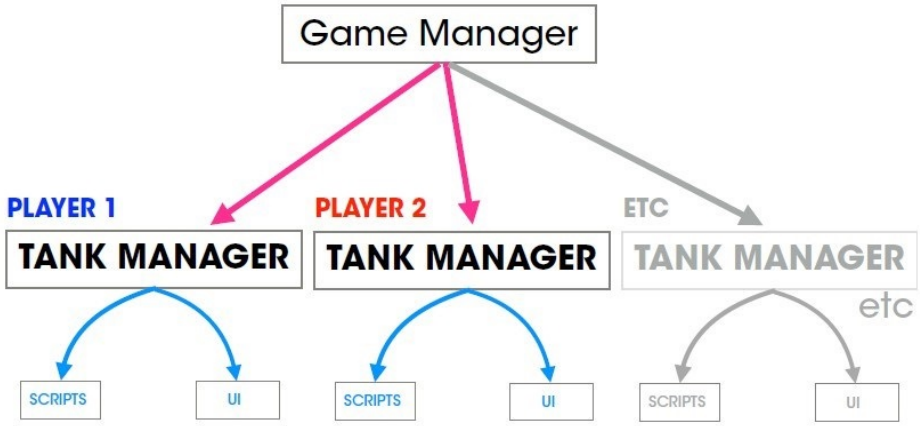


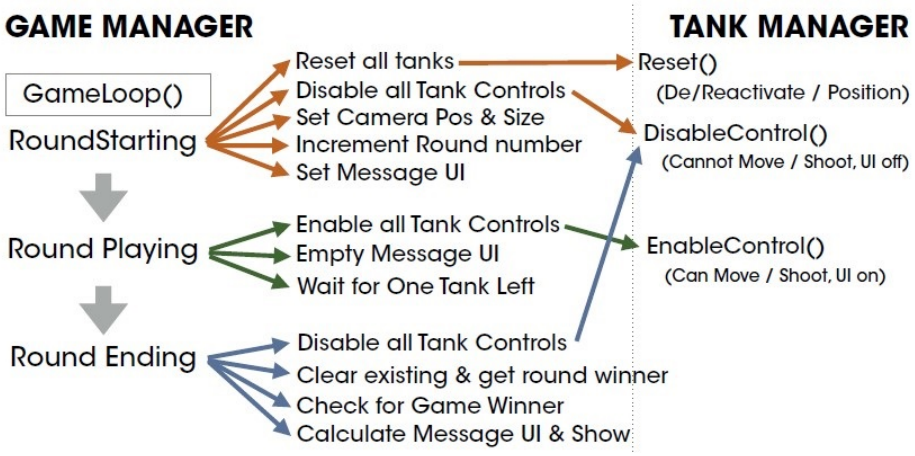
8、选择 CameraRig，在其 CameraControl 脚本组件中，设置 Targets 值为 0，并将 CameraControl 脚本中的 m_Targets 变量前的修饰去注释。

```
[HideInInspector] public Transform[] m_Targets;
```

9、在 Hierarchy 面板创建一个空对象并命名为 GameManager，在 Scripts/Managers 文件夹中找到 GameManager 脚本，将其添加至 GameManager 对象。双击打开并编辑脚本，该脚本控制坦克的创建以及游戏的轮数等等。

其中，存在一个自定义类 TankManager，每一个玩家通过 TankManager 控制 Tank，GameManager 控制 TankManager。两者之间的关系如下：





（1）TankManager 脚本内容如下：

```
1. using System;
2. using UnityEngine;
3.
4. [Serializable]   // 该修饰符的作用是使得其在被实例化时可以在 Inspector 面板中显示
5. public class TankManager
6. {
7.     public Color m_PlayerColor;
8.     public Transform m_SpawnPoint;
9.     [HideInInspector] public int m_PlayerNumber;
10.    [HideInInspector] public string m_ColoredPlayerText;  // 玩家的对应字体的颜色
11.    [HideInInspector] public GameObject m_Instance;       // 存储实例化的 Tank 对象
12.    [HideInInspector] public int m_Wins;
13.
14.
15.    private TankMovement m_Movement;        // TankMovement 脚本引用
16.    private TankShooting m_Shooting;        // TankShooting 脚本引用
17.    private GameObject m_CanvasGameObject;          // 用于显示 UI 界面，例如“ROUND1”……
18.
19.
20.    public void Setup()
21.    {
22.        m_Movement = m_Instance.GetComponent<TankMovement>();
23.        m_Shooting = m_Instance.GetComponent<TankShooting>();
24.        // 从 Canvas 子对象中获取 Text 对象
25.        m_CanvasGameObject = m_Instance.GetComponentInChildren<Canvas>().gameObject;
26.
27.        m_Movement.m_PlayerNumber = m_PlayerNumber;
```

```
28.         m_Shooting.m_PlayerNumber = m_PlayerNumber;
29.
30.         m_ColoredPlayerText = "<color=#" + ColorUtility.ToHtmlString
    RGB(m_PlayerColor) + ">PLAYER " + m_PlayerNumber + "</color>";
31.
32.         // 将 Tank 所有颜色都换为指定的颜色
33.         MeshRenderer[] renderers = m_Instance.GetComponentsInChildre
    n<MeshRenderer>();
34.         for (int i = 0; i < renderers.Length; i++)
35.         {
36.             renderers[i].material.color = m_PlayerColor;
37.         }
38.     }
39.
40.
41.     public void DisableControl()
42.     {
43.         // 使 Tank 对象上的 TankMovement、TankShooting 脚本失效
44.         m_Movement.enabled = false;
45.         m_Shooting.enabled = false;
46.
47.         m_CanvasGameObject.SetActive(false);
48.     }
49.
50.
51.     public void EnableControl()
52.     {
53.         // 激活 Tank 对象上的 TankMovement、TankShooting 脚本
54.         m_Movement.enabled = true;
55.         m_Shooting.enabled = true;
56.
57.         m_CanvasGameObject.SetActive(true);
58.     }
59.
60.
61.     public void Reset()
62.     {
63.         m_Instance.transform.position = m_SpawnPoint.position;
64.         m_Instance.transform.rotation = m_SpawnPoint.rotation;
65.
66.         m_Instance.SetActive(false);
67.         m_Instance.SetActive(true);
68.     }
69. }
```

（2）GameManager 脚本内容如下：

```
1. using UnityEngine;
2. using System.Collections;
3. using UnityEngine.SceneManagement;
4. using UnityEngine.UI;
5.
6. public class GameManager : MonoBehaviour
7. {
8.     public int m_NumRoundsToWin = 5;
9.     public float m_StartDelay = 3f;
10.    public float m_EndDelay = 3f;
11.    public CameraControl m_CameraControl;
12.    public Text m_MessageText;
13.    public GameObject m_TankPrefab;
14.    public TankManager[] m_Tanks;
15.
16.
17.    private int m_RoundNumber;
18.    private WaitForSeconds m_StartWait;
19.    private WaitForSeconds m_EndWait;
20.    private TankManager m_RoundWinner;
21.    private TankManager m_GameWinner;
22.
23.
24.    private void Start()
25.    {
26.        m_StartWait = new WaitForSeconds(m_StartDelay);
27.        m_EndWait = new WaitForSeconds(m_EndDelay);
28.
29.        SpawnAllTanks();
30.        SetCameraTargets();
31.
32.        StartCoroutine(GameLoop());
33.    }
34.
35.
36.    private void SpawnAllTanks()
37.    {
38.        for (int i = 0; i < m_Tanks.Length; i++)
39.        {
40.            m_Tanks[i].m_Instance =
41.                Instantiate(m_TankPrefab, m_Tanks[i].m_SpawnPoint.po
   sition, m_Tanks[i].m_SpawnPoint.rotation) as GameObject;
42.            m_Tanks[i].m_PlayerNumber = i + 1;
```

```
43.             m_Tanks[i].Setup();
44.         }
45.     }
46.
47.
48.     private void SetCameraTargets()
49.     {
50.         Transform[] targets = new Transform[m_Tanks.Length];
51.
52.         for (int i = 0; i < targets.Length; i++)
53.         {
54.             targets[i] = m_Tanks[i].m_Instance.transform;
55.         }
56.
57.         m_CameraControl.m_Targets = targets;
58.     }
59.
60.
61.     private IEnumerator GameLoop()
62.     {
63.         yield return StartCoroutine(RoundStarting());
64.         yield return StartCoroutine(RoundPlaying());
65.         yield return StartCoroutine(RoundEnding());
66.
67.         if (m_GameWinner != null)
68.         {
69.             SceneManager.LoadScene(0);
70.         }
71.         else
72.         {
73.             StartCoroutine(GameLoop());
74.         }
75.     }
76.
77.     // 每一轮开始时执行的代码
78.     private IEnumerator RoundStarting()
79.     {
80.         // 初始化所有坦克
81.         ResetAllTanks();
82.         // 在开始阶段不能移动坦克
83.         DisableTankControl();
84.         // 初始化镜头位置
85.         m_CameraControl.SetStartPositionAndSize();
86.         // 轮数加 1
```

```
87.         m_RoundNumber++;
88.         m_MessageText.text = "ROUND " + m_RoundNumber;
89.
90.         yield return m_StartWait;
91.     }
92.
93.
94.     private IEnumerator RoundPlaying()
95.     {
96.         // 激活坦克的控制脚本
97.         EnableTankControl();
98.         // UI 不再显示内容
99.         m_MessageText.text = string.Empty;
100.            // 循环执行，直到只有一个坦克留下来，即一轮游戏结束
101.            while (!OneTankLeft())
102.            {
103.                yield return null;
104.            }
105.        }
106.
107.
108.        private IEnumerator RoundEnding()
109.        {
110.            // 游戏结束时坦克不能移动
111.            DisableTankControl();
112.
113.            // 获取每一轮的赢家
114.            m_RoundWinner = null;
115.            m_RoundWinner = GetRoundWinner();
116.
117.            if (m_RoundWinner != null)
118.            {
119.                m_RoundWinner.m_Wins++;
120.            }
121.
122.            // 获取整场游戏的赢家
123.            m_GameWinner = GetGameWinner();
124.
125.            // 设置 UI 显示输赢情况
126.            string message = EndMessage();
127.            m_MessageText.text = message;
128.
129.            yield return m_EndWait;
130.        }
```

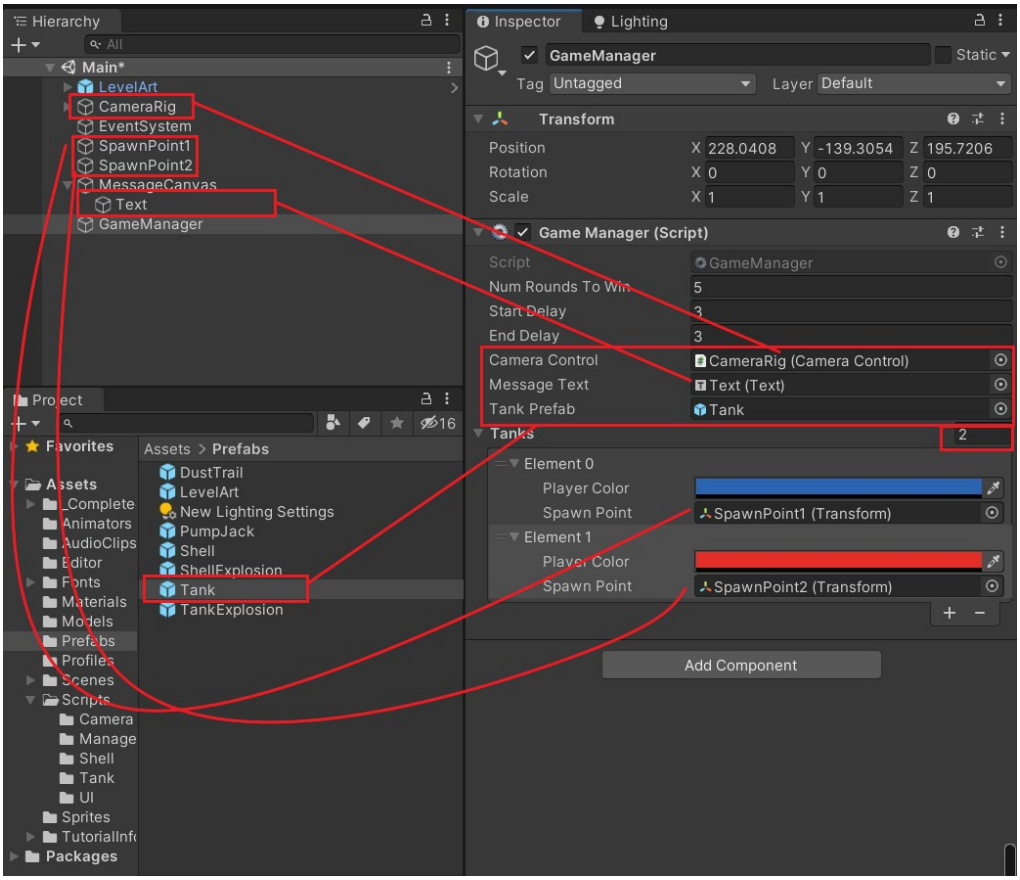
```
131.
132.
133.        private bool OneTankLeft()
134.        {
135.            int numTanksLeft = 0;
136.
137.            for (int i = 0; i < m_Tanks.Length; i++)
138.            {
139.                if (m_Tanks[i].m_Instance.activeSelf)
140.                    numTanksLeft++;
141.            }
142.
143.            return numTanksLeft <= 1;
144.        }
145.
146.        // 获取每轮游戏的赢家
147.        private TankManager GetRoundWinner()
148.        {
149.            for (int i = 0; i < m_Tanks.Length; i++)
150.            {
151.                if (m_Tanks[i].m_Instance.activeSelf)
152.                    return m_Tanks[i];
153.            }
154.
155.            return null;
156.        }
157.
158.        // 获取游戏的赢家
159.        private TankManager GetGameWinner()
160.        {
161.            for (int i = 0; i < m_Tanks.Length; i++)
162.            {
163.                if (m_Tanks[i].m_Wins == m_NumRoundsToWin)
164.                    return m_Tanks[i];
165.            }
166.
167.            return null;
168.        }
169.
170.
171.        private string EndMessage()
172.        {
173.            string message = "DRAW!";
174.
```

```
175.            if (m_RoundWinner != null)
176.                message = m_RoundWinner.m_ColoredPlayerText + " WINS THE ROUND!";
177.
178.            message += "\n\n\n\n";
179.
180.            // 输出所有 Tank 的分数
181.            for (int i = 0; i < m_Tanks.Length; i++)
182.            {
183.                message += m_Tanks[i].m_ColoredPlayerText + ": " + m_Tanks[i].m_Wins + " WINS\n";
184.            }
185.
186.            if (m_GameWinner != null)
187.                message = m_GameWinner.m_ColoredPlayerText + " WINS THE GAME!";
188.
189.            return message;
190.        }
191.
192.
193.        private void ResetAllTanks()
194.        {
195.            for (int i = 0; i < m_Tanks.Length; i++)
196.            {
197.                m_Tanks[i].Reset();
198.            }
199.        }
200.
201.
202.        private void EnableTankControl()
203.        {
204.            for (int i = 0; i < m_Tanks.Length; i++)
205.            {
206.                m_Tanks[i].EnableControl();
207.            }
208.        }
209.
210.
211.        private void DisableTankControl()
212.        {
213.            for (int i = 0; i < m_Tanks.Length; i++)
214.            {
215.                m_Tanks[i].DisableControl();
```

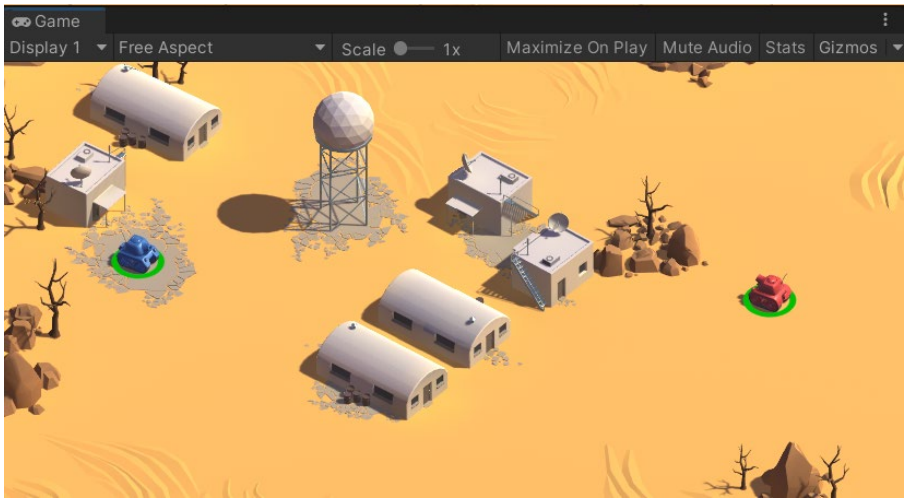
```
216.             }
217.         }
218.     }
```

在 Unity 中设置 GameManager 脚本的 public 变量，如下所示：

其中 Tanks 变量的 Element0 的 Color 属性设置为(42, 100, 178)，Element1 的 Color 属性设置为(229, 46, 40)。



运行游戏，截图如下：



2.11 添加游戏音乐

1、给 GameManager 对象添加 Audio Source 组件，设置 Audio Source 组件的 Audio Clip 为 BackgroundMusic，并勾选 Loop 选项。

2、在 Project 面板创建空文件夹 AudioMixers，右键创建 Audio Mixer 对象，命名为 MainMix。

3、打开 Audio Mixer 面板，在 Groups 栏，在 Master 下面添加 3 子 Group，分别命名为 Music、SFX 和 Driving。

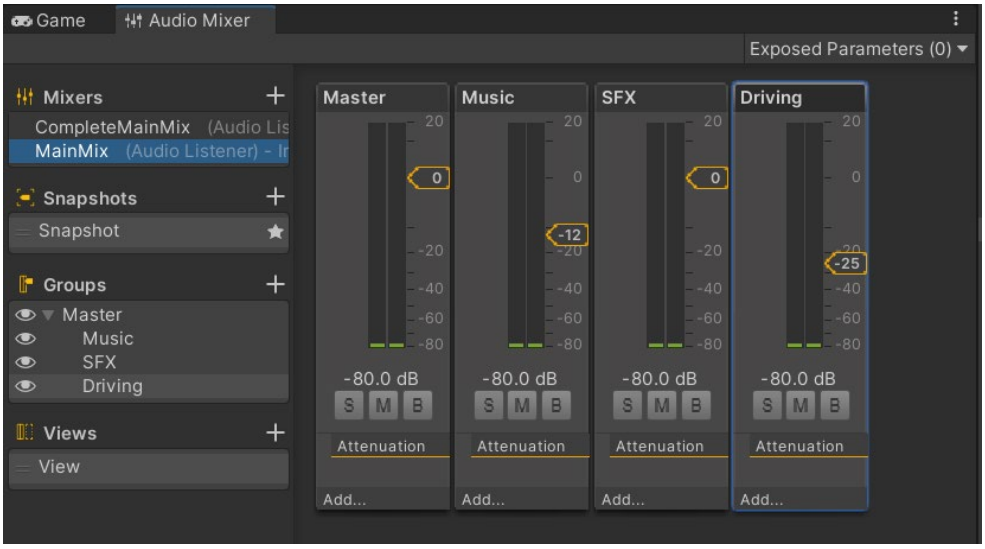
4、在 Prefabs 文件夹打开 Tank 预制体，在第一个 Audio Source 组件中，Output 设置为 MainMix 的 Driving；第二个 Audio Source 组件中，Output 设置为 MainMix 的 SFX。

5、在 Prefabs 文件夹中打开 Shell 预制体，选择子对象 ShellExplosion，将 Audio Source 组件的 Output 设置为 MainMix 的 SFX。

6、在 Hierarchy 面板选中 GameManager，将 Audio Source 组件的 Output 设置为 MainMix 的 Music。

7、在 Prefabs 文件夹中打开 TankExplosion 预制体，将 Audio Source 组件的 Ouput 设置为 MainMix 的 SFX。

8、在 Audio Mixer 面板，选中 MainMix，将 Music 的 Attenuation 设置为-12，Driving 的 Attenuation 设置为-25。



9、选中 Music，在 Inspector 面板中点击“Add Effect”，选择添加“Duck Volume”；选中 SFX，在 Inspector 面板中点击“Add Effect”，选择添加“Send”，在 Send 属性中，设置 Receive 为“Music\Duck Volume”，设置 Send Level 为 0db(full)；重新选中 Music，在 Duck Volume 属性中设置 Threshold 为-46，设置 Ratio 为 250，设置 Attack Time 为 0。

三、最终游戏效果

该游戏为一款 PC 端的 3D 的单机多人射击类游戏，每个玩家操作不同的坦克互相射击，射击时可以进行蓄力，蓄力时间越长，子弹伤害越大。

游戏为回合制，每个回合只有留在场上的最后 1 辆坦克获得胜利，游戏胜利条件为赢得 5 轮游戏的胜利，所有的坦克中，谁先获得 5 轮胜利，谁就是游戏最后的赢家。以 PC 上两位玩家为例，每位玩家的玩法为：

（1）玩家 1：

① W、S、A、D 控制坦克的移动，分别表示前后左右；

② 空格键控制子弹的发射；

③ 长按空格实现蓄力。

（2）玩家 2：

① 方向键的上下左右分别控制坦克的上下左右移动；

② Enter 键控制子弹的发射；

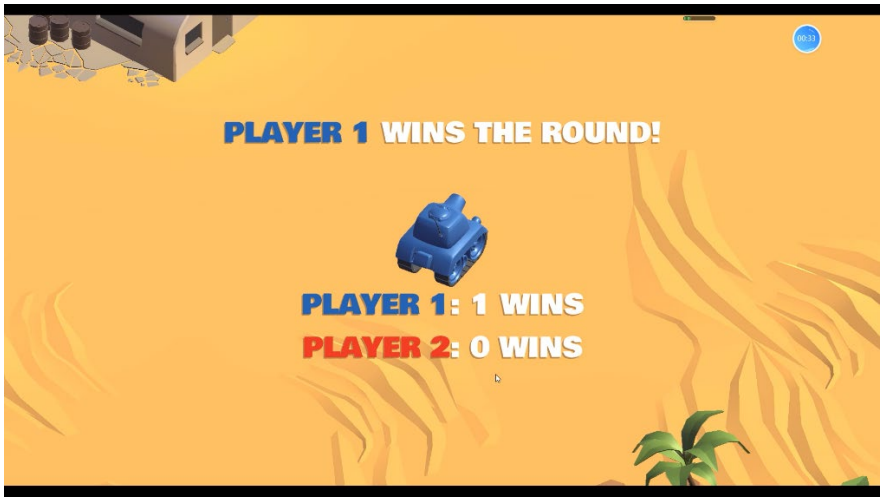
③ 长按 Enter 实现蓄力。

游戏演示视频见附件：Tanks 演示视频.mp4。

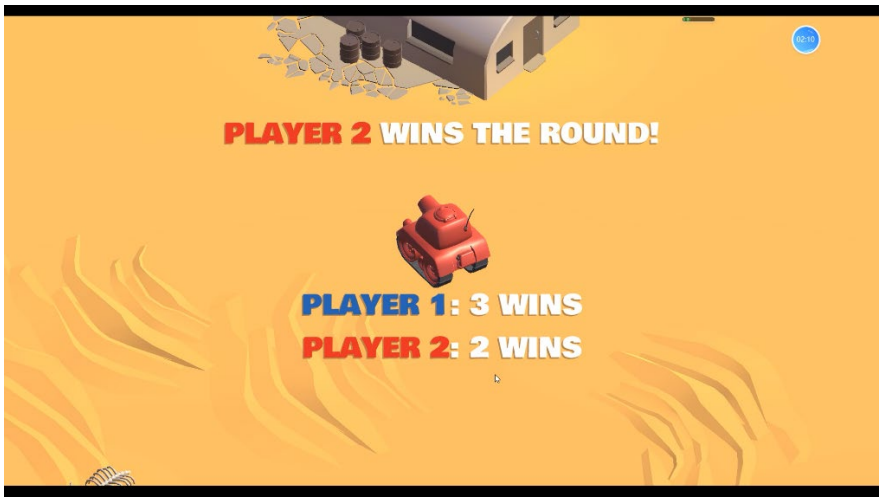
以下为主要内容截图。

（1）开始游戏

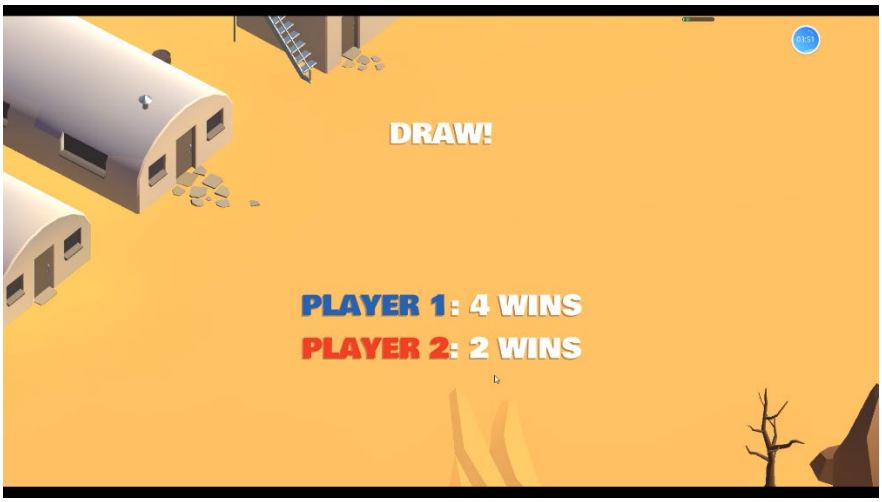
（1）玩家 1 获胜



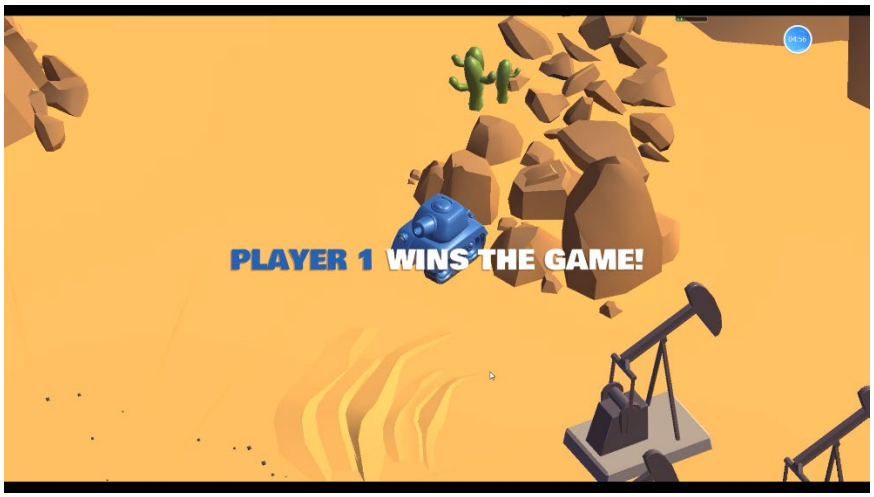
（3）玩家 2 获胜



（4）平局



（5）玩家 1 赢得游戏



四、游戏制作遇到的问题及解决方法

本实验的成功之处在于该游戏从游戏开始到游戏结束具有完整的游戏过程，包括回合的开始与结束，玩家获得胜利的轮数，以及玩家平局的处理，并且最终通过胜利回合数最先达到 5 来作为游戏胜利的条件。在游戏视觉效果方面，通过粒子系统实现了坦克移动的扬尘、坦克血量耗尽时的爆炸特效、子弹落地与击中时的特效，增加了游戏的视觉体验。同时，通过脚本控制 Camera 随着坦克的移动而跟随移动，并且随着坦克间距离的远近来缩放及放大镜头显示，在游戏的过程中能够增加用户的体验效果。

另外，使用 Audio Source 组件增加了坦克移动、发射蓄力、发射子弹的音效，并且，通过使用 Audio Mixer 来实现了音频的混合，是的音频在坦克执行不同的操作时可以调整各音频的播放音量大小，增加了游戏的听觉体验。

同时，在界面上增加了 UI 显示玩家获得的分数以及每轮游戏的赢家，以及最终获胜的玩家，这样玩家可以随时根据游戏的这些 UI 提示感知游戏进度，游戏的上手程度更好。

不足之处在于未提供给玩家选择游戏人数，在现实中，其实很多游戏都会给玩家选择玩家数的功能，但是在本游戏中，默认为双人游戏，玩家数虽然在脚本中是可以通过变量改变的，但是本游戏在 Inspector 面板中将其固定为 2 了。并且，如果玩家选择单人游戏，应该为玩家提供一个电脑虚拟玩家作为另一个玩家，即人机模式，这样可以让玩家在只有一个人的时候也可以玩该游戏，可以起到增加游戏用户数量的作用。第三个不足之处在于当有一位玩家获得游戏胜利即游戏结束时，应该给用户提供退出游戏或继续游戏的选项，而在本游戏中会重新开始新的一局游戏，有待改进。

五、总结与体会

通过本次课程设计，我们参照视频开发了这一款有趣的游戏，游戏的开发过程有趣又具有挑战性。在这次课设中也收获了很多，学会了如何使用 Unity 的各个面板进行游戏场景的搭建，以及使用 Unity 的各种小工具和快捷操作来进行便捷操作。同时，由于之前未接触过 C#编程，通过这次实验，我也学习了以下 C#的语法，以及 C#面向对象的编程技术，并且学会了如何通过 Inspector 面板为脚本的 public 变量赋值，以及通过[HideInItnspector]修饰来隐藏 public 变量在 Inspector 面板的显示，这样可以让脚本间可以互相使用值，同时可以避免用户在 Inspector 面板对其进行非预期的修改。

第三个就是，学会了如何在脚本中动态生成预制体，并且如何通过激活与取消激活来隐藏游戏对象，以此来提高游戏性能，因为对一些需要频繁在游戏场景中出现又消失的物体，如果使用生成又销毁的方式会造成大量消耗内存，而 Unity 本身又是比较吃性能的编辑器。还有一点收获是，学习到了很多关于 Audio 的操作。比如，如何通过脚本来控制同一个 Audio Source 组件在不同的条件下播放不同的音频(Audio Clips)；以及如何通过 Audio Mixer 来进行不同音频间的适配，让不同的音频可以在不同的场景中混合播放，而不是单纯的叠加播放，这样可能会产生有时候听不清需要听到的音频的现象，降低用户体验性。