

Unity 开发：【Unity】AngryBirds (Seasons)

1. 技术框架与编程环境配置

1.1 Unity 概述及特色优势

1.1.1 Unity 介绍

作为软件工程专业的大学生，我们经常在休闲时间通过玩游戏来放松自己，缓解自己的学业压力，以及与朋友一起玩增进双方感情，但是对于游戏开发我们很难有更深层次的了解。我们在日常学习生活中总会听到“Unity”这个名词，那么在真正的游戏开发中，Unity 到底有什么作用呢？

对于游戏的开发，从语言的角度来说，一般有两种一种是脚本语言，例如 C#，JS，Boo 等，另一种是非脚本开发，就是用 C/C++ 和 python 来开发。但是如果采用脚本开发的话，单独的脚本语言是无法开发游戏的，需要结合游戏引擎。游戏引擎+脚本语言的方式，可以将游戏编程变为可视化，只需要添加对象，设置对象参数，再用脚本来实现代码就可以了，开发人员不用考虑内存的使用，因为游戏引擎会自动回收内存。

Unity 是由 Unity Technologies 开发的一个让玩家轻松创建诸如三维视频游戏，建筑可视化，实时三维动画等类型互动内容的多平台的综合型游戏开发工具，是一个全面整合的专业游戏引擎。其编写的程序可以发布游戏至 Windows，Mac，Wii，iPhone，WebGL(需要 HTML5)，Windows Phone 8 和 Android 平台。同时也可以利用 Unity web player 插件发布网页游戏，支持 Mac 和 Windows 的网页浏览，它的网页播放器也被 Mac 所支持。



图 1.1 Unity 图标

1.1.2 Unity 特色及其优势

Unity 是游戏开发领域最轻量级的游戏开发工具，入门简单，界面简单，安装，调试，发布都非常方便，语言采用 C# 或者 JS 作为脚本语言，学习成本低（.net 开发人员可以很容易转行过来），官方的文档相当完善，而且给出了相对的 demo。自己有自己的 Asset Store，社区活跃且有相当多的资源可供下载，开发效率高。

下面对 Unity 游戏开发引擎的特色进行阐述

1) 跨平台

游戏开发者可以通过不同的平台进行开发。游戏制作完成后，游戏无需任何修改即可直接一键发布到常用的主流平台上。

Unity 游戏可发布的平台包括 Windows、Linux、MacOS X、iOS、Android、Xbox360、PS3 以及 Web 等。跨平台开发可以为游戏开发者节省大量时间。

以往游戏开发中，开发者要考虑平台之间的差异，比如屏幕尺寸、操作方式、硬件条件等，这样会直接影响到开发进度，给开发者造成巨大的麻烦，Unity 几乎为开发者完美地解决了这一难题，将大幅度减少移植过程中不必要的麻烦。

2) 综合编译

Unity 的用户界面具备视觉化编辑、详细的属性编辑器和动态游戏预览特性。Unity 创新的可视化模式让游戏开发者能够轻松构建互动体验，当游戏运行时可以实时修改参数值，方便开发，为游戏开发节省大量时间。

3) 资源导入

项目可以自动导入资源，并根据资源的改动自动更新。Unity 支持几乎所有的主流的三维格式，如 3ds Max、Maya、Blender 等，贴图材质自动转换为 U3D 格式，并能和大部分相关应用程序协调工作。

4) 联网和一键部署

Unity 支持从单机应用到大型多人联网游戏的开发，同时只需一键即可完成作品的多平台开发和部署，让开发者的作品在多平台呈现。

5) 地形编辑器

Unity 内置强大的地形编辑系统，该系统可使游戏开发者实现游戏中任何复杂的地形，支持地形创建和树木与植被贴片，支持自动的地形 LOD、水面特效，尤其是低端硬件亦可流畅运行广阔茂盛的植被景观，能够方便地创建游戏场景中所用到的各种地形。

6) 物理特效

物理引擎是模拟牛顿力学模型的计算机程序，其中使用了质量、速度、摩擦力和空气阻力等变量。Unity 内置 NVIDIA 的 PhysX 物理引擎，游戏开发者可以用高效、逼真、生动的方式复原和模拟真实世界中的物理效果，例如碰撞检测、弹簧效果、布料效果、重力效果等。

7) 光影和着色器

Unity 提供了具有柔和阴影以及高度完善的烘焙效果的光影渲染系统，同时其着色器系统也具有易用性、灵活性、高性能。

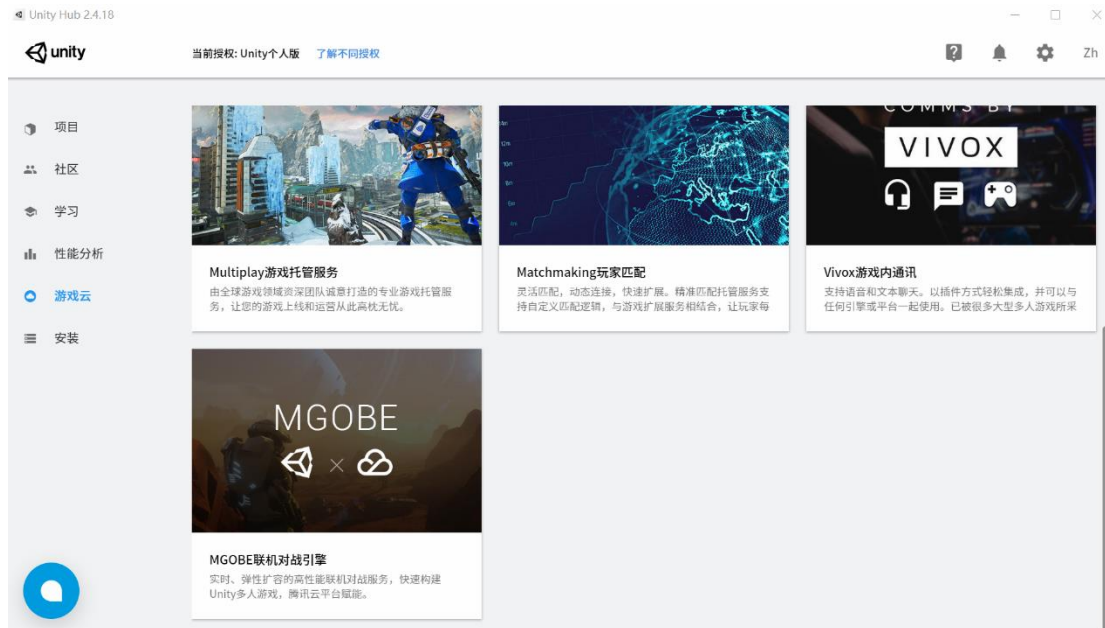


图 1.2 Unity 社区，学习服务

1.2 开发环境搭建

1.2.1 Unity Hub 环境搭建

- 下载 Unity Hub，方便管理自己个人的 Unity 项目。

下载地址：<https://unity.cn/releases>;

- 下载 Unity

Unity Hub 下载完毕之后，可以进入 Unity Hub 社区中，完成之后的 Unity 的安装，同时也可以 Unity Hub 社区与其他开发者讨论和技术交流。

安装教程：https://blog.csdn.net/Having_a_Bath/article/details/105592076;

1.2.2 Visual Studio 2019 环境搭建

- 下载 Visual Studio 2019 安装包

下载地址：<https://visualstudio.microsoft.com/zh-hans/vs/>;

在之后的 4.3 编程环境的介绍中会详解如何利用 Visual Studio 2019 搭建环境。

1.3 新建项目与 Unity 的使用

1.3.1 新建项目

首先点击 Unity Hub 的左栏中的项目界面，接着点击该界面右上角的蓝色的新建项目，此时会出现项目命名和项目的存放位置，以及选择 Unity 开发的内容和是否上传至 PlasticSCM (Unity 项目版本控制系统)，命名和项目位置根据自己的需求建立，Unity 开发的内容选择 Unity2D，同时如果希望上传可 PlasticSCM 可以选择同意，但是一般个人开发不建议上传，因为这样会使开发过程以及发布过程的流畅程度受到一定的影响。

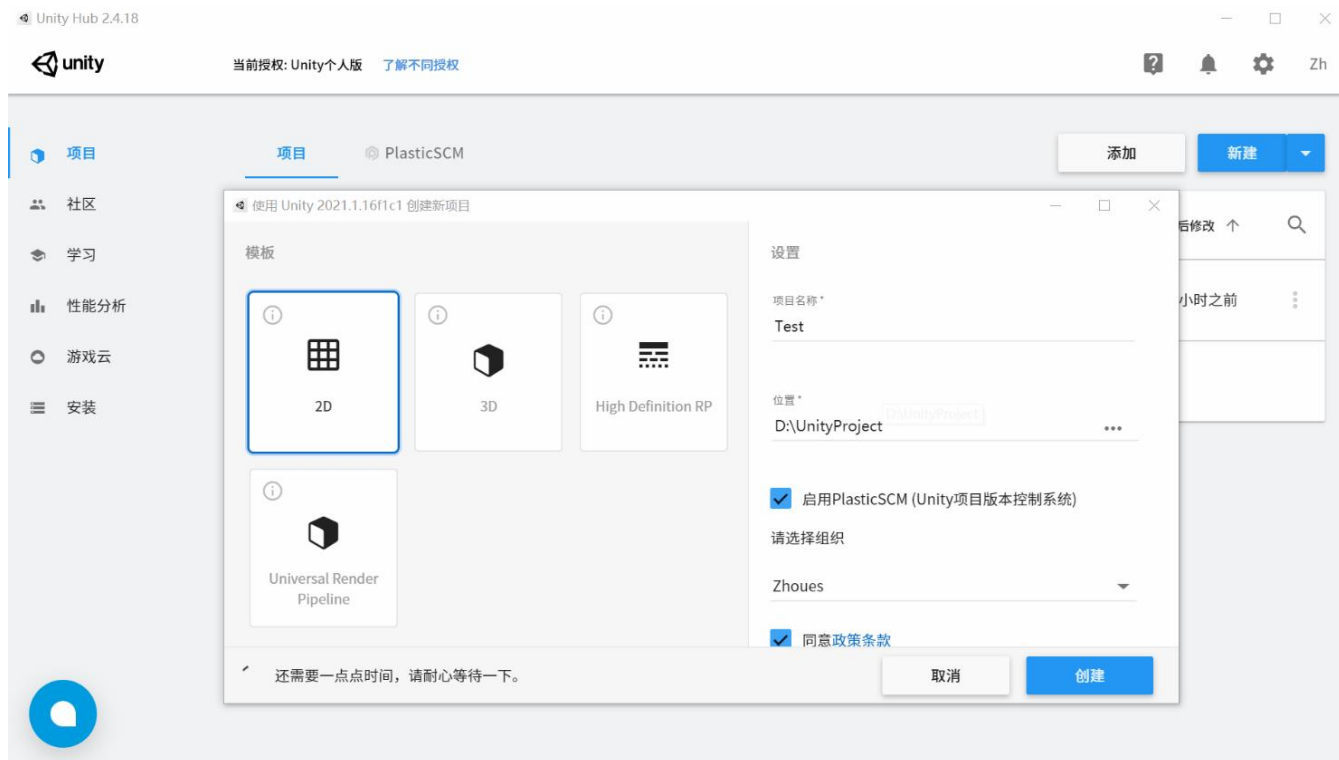


图 1.3 Unity 新建项目

然后点击 **Edit**→**Preferences**，在其中的 **External Tools** 的 **External Scripts Editor** 选择之前安装好的 **Visual Studio 2019**。

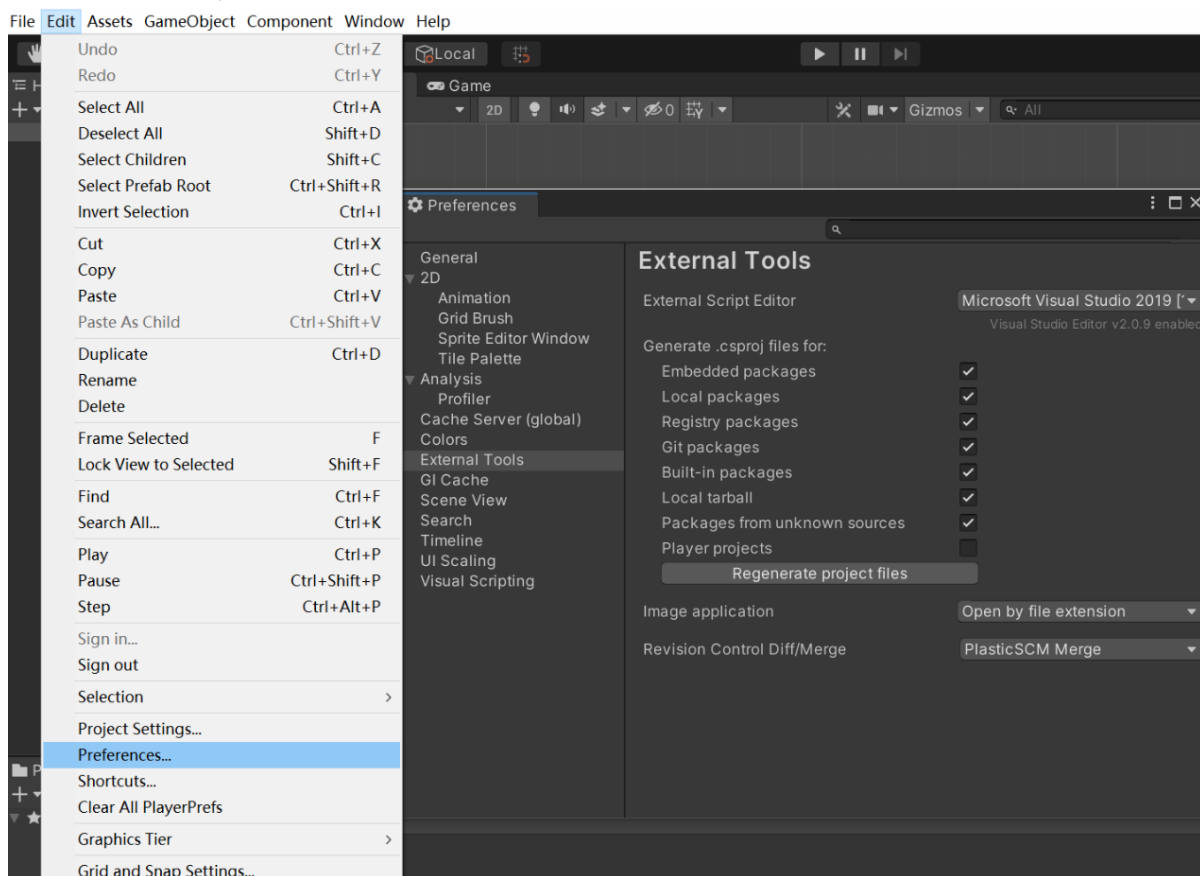


图 1.4 Unity 脚本编译器设置

1.3.2 界面分析

Unity 操作界面总共分为四个部分。第一个部分位于正中间，是设计的主要位置，其中有两个可视界面，一个是 **Scene**，主要用来给开发人员设计使用，一个是 **Game**，主要是用来展现设计完成之后的实际游戏界面。第二个部分左端，里面包含着这个场景中所有的设计元素，开发人员主要通过左端来添加以及删除开发的内容，其中默认包含了 **Main Camera**（主相机）。第三个部分位于右端 **Inspector**，主要是对左端的设计元素添加组件以及编写相应的脚本以控制其行为，Unity 有着丰富的组件，所以适当的使用可以让自己工作量大幅度减少。第四个部分位于下端，其主要包含整个项目的源文件及其辅助文件和素材，一般默认只包含 **Scenes**（场景），如果我们需要制作一款较为完备的游戏，我们至少还需要新建下列文件夹 **Animation**（动画），**Audio**（音乐），**Image**（素材图片），**Prefabs**（预制体），**Scripts**（脚本），**Resources**（关卡储备）。

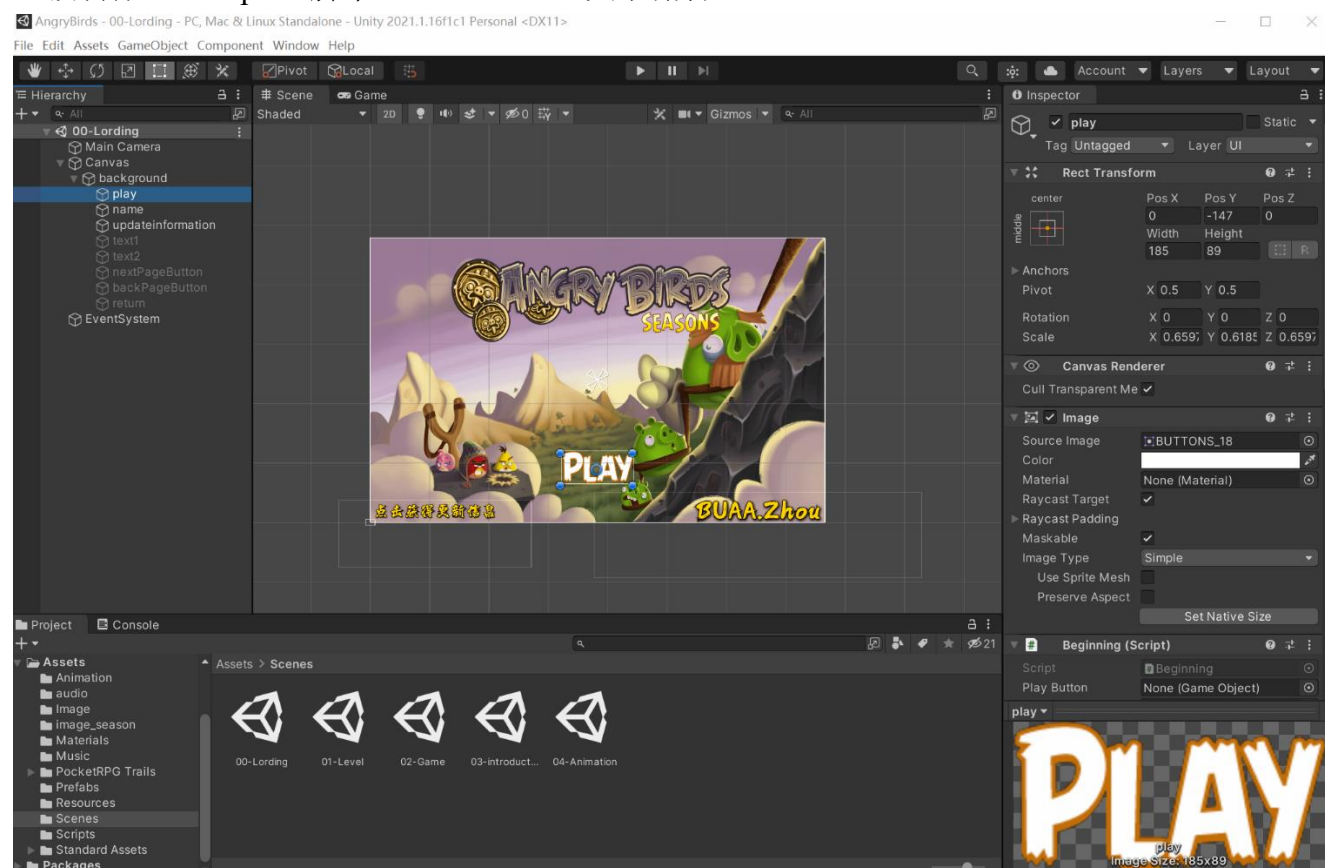


图 1.5 Unity 界面展示

1.3.3 图片导入以及图片裁剪

Unity 开发游戏在预备工作中需要准备好素材图片，由于 Unity 在功能上考虑了拖拽的便利，所以素材的导入可以直接将素材包拖入至下端的 **Image** 文件夹中，同时需要使用图片时，也可以从 **Image** 文件夹中拖入至屏幕正中间的 **Scene** 设计场景中，同时左边会出现该图片的选项，点击之后右端会出现组件及其脚本编辑的选项。

但是游戏的开发所用的素材往往不是每一个游戏物体都有单独游戏图片，而是把很多游戏图片集中在一起，此时我们需要对其进行裁剪。首先我们先在下端的素材包中选择该图片，此时右端的 **Inspector** 窗口中 **Sprite Mode** 窗口显示为

Single，此时我们把其调为 Multiple，然后点击右下方的 Sprite Editor，之后会跳出一个是否裁剪的窗口，点击 Apply 即可进入裁剪。在裁剪界面的顶部栏里有 Slice 选项，点击后自动裁剪，之后再点击右端的 Apply 应用裁剪，图片就裁剪完毕了。

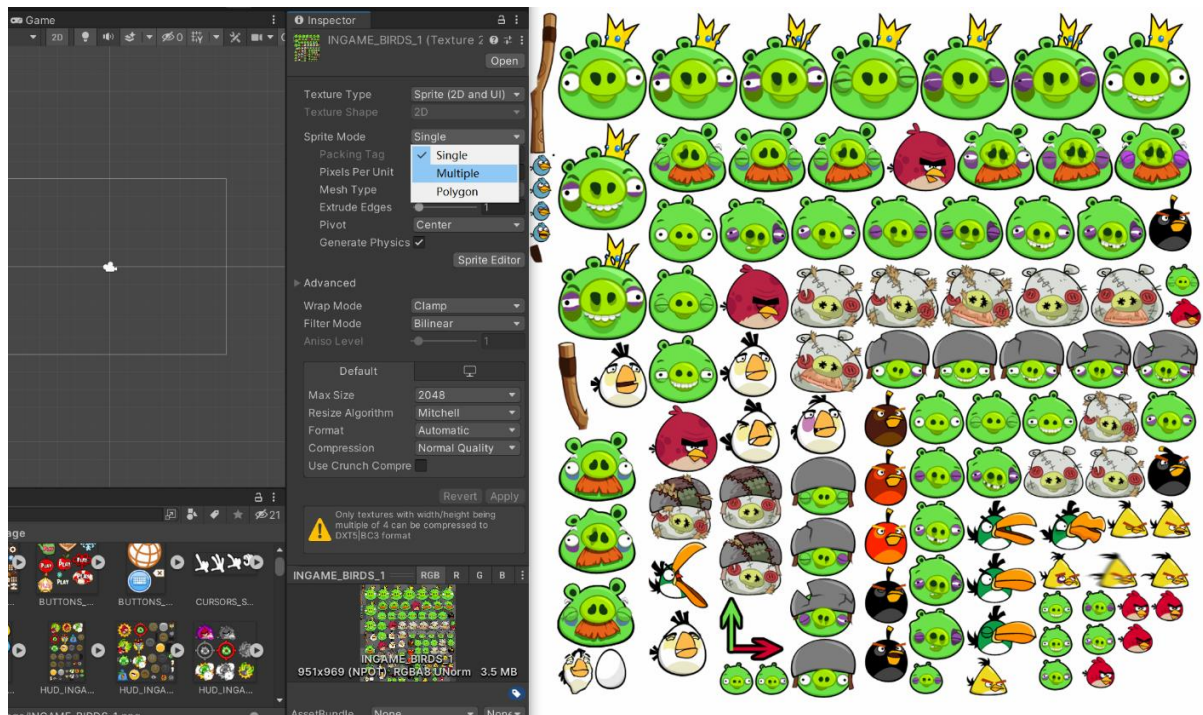


图 1.6 Unity 图片裁剪预备工作



图 1.6 Unity 图片裁剪

2. 案例简介

《AngryBirds 愤怒的小鸟》是由 Rovio 开发的一款休闲益智类游戏，于 2009 年 12 月首发于 iOS。游戏以小鸟报复偷走鸟蛋的肥猪为背景，讲述了小鸟与肥猪的一系列故事。游戏的玩法很简单，将弹弓上的小鸟弹出去，砸到绿色的肥猪，将肥猪全部砸到就能过关。鸟儿的弹出角度和力度由玩家的手指来控制，要注意考虑好力度和角度的综合计算，这样才能更准确的砸到肥猪。而被弹出的鸟儿会留下弹射轨迹，可供参考角度和力度的调整。另外每个关卡的分数越多，评价将会越高。



图 2.1 AngryBirds 宣传图画

本案例使用 Unity 开发，用 C# 语言编写脚本，实现了 Windows 平台和 Mac 平台的 AngryBirds 游戏开发。为了实现更好的视觉效果和听觉效果，该游戏采用了原本的愤怒的小鸟的原版图片素材以及原版的音乐素材。这款游戏的游戏规则如下：游戏中小鸟需要将弹弓上的小鸟弹出去，砸到绿色的肥猪，将肥猪全部砸到就能过关。鸟儿的弹出角度和力度由鼠标来控制，要注意考虑好力度和角度的综合计算，这样才能更准确的砸到肥猪。游戏中玩家需要消灭所有的肥猪才能胜利每局结束后，系统都会根据玩家的表现来评分。分数可以获得星星。当星星到达一定数量时，可以解锁新的场景。同时为了增添游戏乐趣，该游戏添加了许多新的小鸟种类，猪猪种类以及建筑物种类，同时也为每一个行为设置了自己独特的音效，使得玩家在体验游戏的时候更有氛围感。案例代码量 1600 行左右，初学者一周时间可以完成开发。

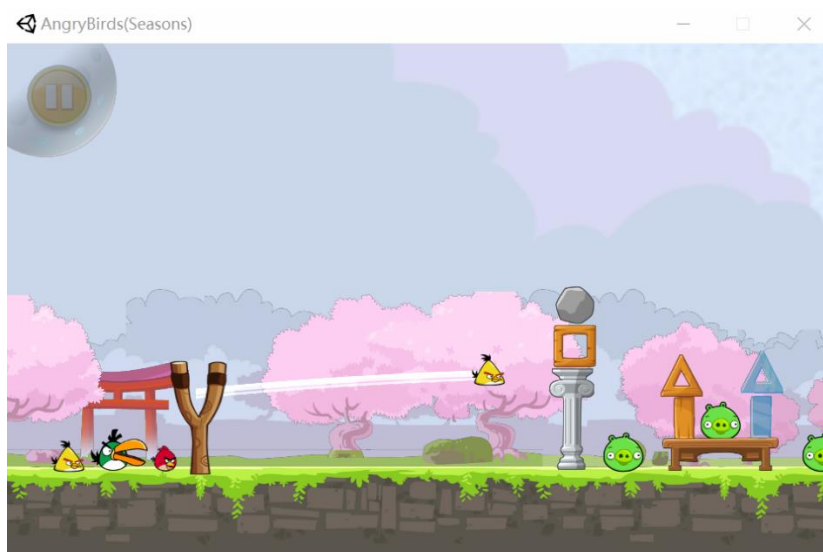


图 2.2 游戏实际运行效果

3. 需求分析

本案例目标使实现 Windows 平台和 Mac 平台的 AngryBirds (Seasons) 游戏开发。玩家运行游戏后会进入开始界面，可以选择按下“play”按钮开始新手教程，如果玩家对该游戏有一定的了解可以选择跳过教程，新手教程结束后会进入加载界面，加载界面结束后正式进入场景选择界面，此时有三个场景可供玩家选择，选择具体的场景之后可以选择相应的关卡，点击关卡的图标后可以进入游戏场景。

同样使复现经典游戏，AngryBirds 游戏的整体逻辑会比俄罗斯方块，贪吃蛇等小游戏更复杂一些，而且本案例要实现更多功能的小鸟，同时也要附加不同的音乐增加游戏的趣味性。

本案例讲解主要偏重于简单的 Unity2D 游戏开发，参考此案例你的作品可以实现 Unity2D 游戏的开发与发布，同时你也会了解到游戏是由哪些部分组成以及如何运行的。

3.1 游戏操作及界面需求

- 鼠标左键单击小鸟并按住，小鸟会发出已被选择的音效，同时鼠标的拖拽，小鸟与弹弓之间有皮绳相连。
- 鼠标左键松开后，小鸟会沿着弹弓提供弹力的方向飞行，同时小鸟会受重力的影响，飞行轨迹为一条抛物线。
- 如果小鸟有技能，则需要实现单击鼠标之后技能的释放。
- 小鸟碰撞猪或者建筑后会发出受伤音效，同时小鸟的图片也会更改成为受伤状态下的图片，从发射开始之后的 5s，飞出的小鸟销毁，同时播放销毁时的爆炸动画。
- 猪和建筑物在遭受小鸟撞击之后会发出对应的撞击音效，同时计算小鸟与猪的相对速度。如果相对速度位于受伤速度之上，死亡速度之间，猪和建筑物会触发受伤音效，同时更换图片为受伤图片。如果位于死亡速度之上，猪和建筑物会触发死亡音效，同时爆炸销毁并且会出现获得的得分。
- 每一关都会根据剩余的小鸟判定该关卡的星星数：如果没有剩余小鸟，获得

1 颗星，如果剩余 1 只小鸟，获得 2 颗星，如果剩余两只小鸟及以上，获得 3 颗星，每一关获得的最大星星数为 3 颗。在获胜界面中要出现关卡获得的星星数，同时要添加相应的烟花效果。

3.2 游戏物体需求

由于该游戏是由大量游戏物体叠加而成的，所以对于游戏物体的设定是非常重要的。

下面是小鸟的简介：

	普通小鸟，没有特殊技能。
	单击之后速度变为两倍。
	单击之后会爆炸摧毁周围物体。
	单击之后可以是飞行方向反向。
	没有特殊技能，但是体型巨大。
	单击之后会膨胀。




	单击之后速度变为原来的四倍。
	可以穿透任何物体，不受重力影响，单击之后会触发大范围的毁灭魔法。
	反重力，经常把自己包裹与黑暗之中。

表 3.1 小鸟简介

下面是猪的简介：

	普通小猪猪
	头盔猪
	猪老爷
	猪皇后

表 3.2 小鸟简介

3.3 游戏算法需求

《AngryBirds(Seasons) 愤怒的小鸟(季节版)》包含了原版愤怒的小鸟的全部功能，并增加了一些人性化的功能：

- 游戏界面中的暂停功能键单击之后游戏会进入暂停界面，同时屏幕亮度变暗，左侧会弹出三个按钮，从上到下分别是继续游戏，重新开始本关游戏，返回场景选择界面。这样可以使游戏达到随时随地玩的目的，同时也可以让玩家随意的选关和重玩。



图 3.1 暂停功能

- 获胜之后会出现获胜的动画，同时底部也会出三个按钮，从左到右分别是返回场景选择界面，重新开始本关游戏，下一关。这三个按钮也实现了玩家在通过一关之后可以连续游戏，使得玩家的体验感流畅顺滑。

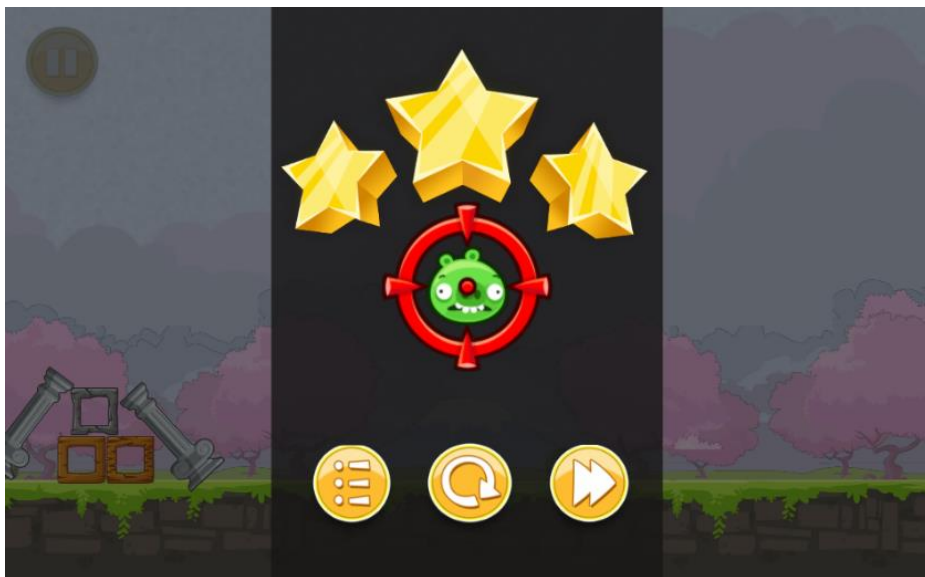


图 3.2 获胜界面

- 失败之后也会获得失败的动画，同时底部出现两个按钮，从左到右分别是返回场景选择界面，重新开始本关游戏。这样设计也使玩家实现了快速再次挑战的环境，加快了游戏的节奏。



图 3.3 失败界面

- 该游戏还给小鸟设置了拖尾效果，使小鸟在空中飞行的同时出现轨迹，满足感受真实物理引擎的需求，同时也使游戏界面更富有体验感。

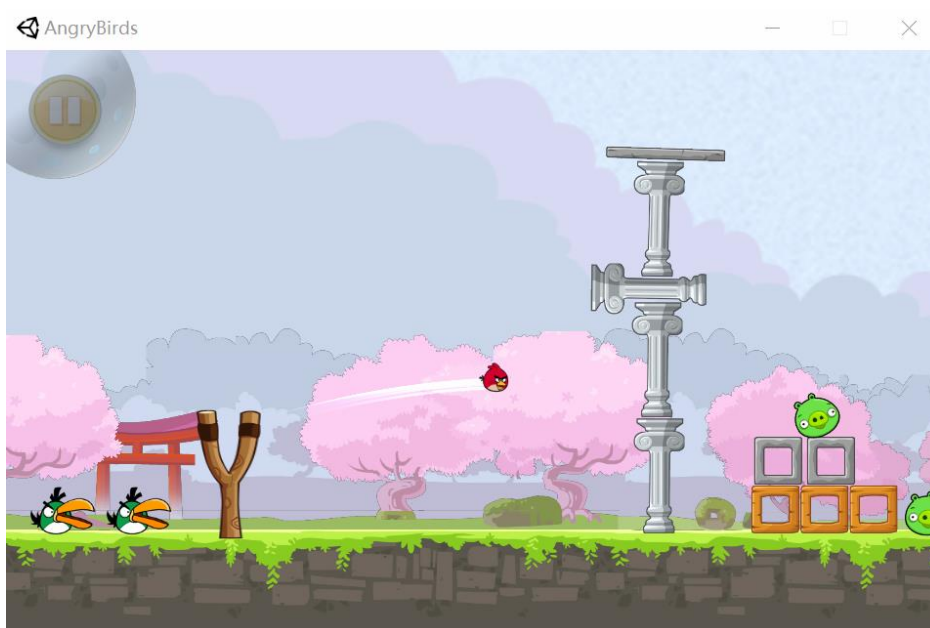


图 3.4 拖尾效果

- 该游戏在满足视觉要求的同时，还同时考虑到了听觉效果。开发者在每一个界面，每一个场景下都设置有独特的背景音乐，而且对于像小鸟的选择，小鸟的拖拽，小鸟的飞行，小鸟的受伤，猪的受伤，建筑物的破坏上都配由各自的音效，使得游戏更有氛围。
- 该游戏会帮助玩家记录自己每一关获得的历史最大星星数，同时也会在场景选关界面统计出该场景获得的所有星星数，帮助玩家迅速了解自己在该场景的进度，同时也为计算开启其他场景的星星数提供了便捷。

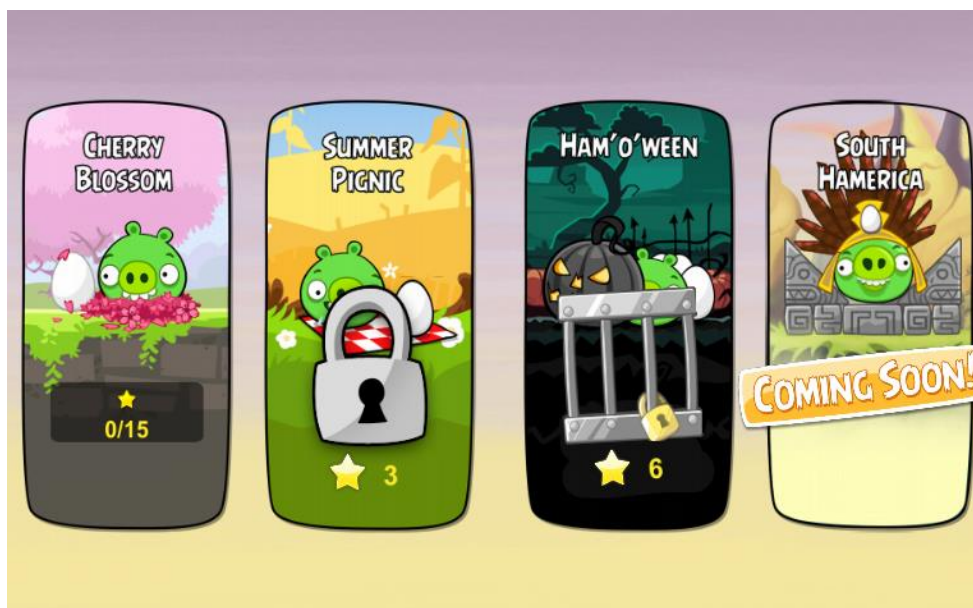


图 3.5 记录功能

4. 技术路线

4.1 需要具备的知识

- 面向对象的编程思想，C#语言的基础知识；
- 常见脚本书写环境的使用，比如 Visual Studio 2019、Visual Studio Code 等；
- 开发环境 Unity 的使用，本次开发技术需求较低，具备 Unity2D 的开发知识即可；
- 游戏开发相关简单逻辑知识；

4.2 参考资料

1. 入门 C#，可自行前往图书馆查阅相关入门资料，但对于小学期而言，更推荐边用边学的方式，可访问网上的资源，熟悉掌握 C#的基础语法
菜鸟教程：<https://www.runoob.com/csharp/csharp-tutorial.html>；
微软官方文档：<https://docs.microsoft.com/zh-cn/dotnet/csharp/>；
2. 入门 Unity，Unity 在开发游戏领域应用广泛，所以网上有很多相应的教程，但是由于小学期时间较短，所以更推荐看视频学习。
B 站教学视频：
<https://www.bilibili.com/video/BV12s411g7gU?from=search&seid=18373795598871350058>；
简书文字教程：<https://www.jianshu.com/p/c94a8f8fd0c9>；
3. 同学们应该都多少接触过电子游戏，或者是游戏爱好者，想必都了解《AngryBirds》的整体逻辑，在开发游戏之前建议是先尝试下原版的游戏，感受下一些具体细节，思考怎样的创新玩法可以让你的游戏与众不同。

4.3 编程环境推荐

开发环境：Visual Studio 2019

编程语言：C#/Unity

部署环境：WIN10（x64）

案例推荐使用集成开发环境（IDE）进行开发，如 Microsoft Visual Studio、Visual Studio Code 等。相比于文本编辑器，IDE 会集成更多开发软件会用到的功能，能大幅提高开发效率。

本节后续主要推荐了 Microsoft Visual Studio，并说明了安装过程。当然你也可以去选择 Visual Studio Code，其功能强大的插件库能胜任多种语言开发工作，缺点就是相对来说环境配置最为麻烦，你可以根据自己具体情况灵活选择最合适自己的 IDE，IDE 始终只是一个帮助开发的工具。

Visual Studio 优点是功能非常强大，性能好，配置简单，缺点可能就是体积大。下面进行 VS 的环境配置。

第一步在官网下载安装 VS。选择 Community 免费版本。下载好后一路下一步，在 Visual Studio Installer 中选择安装内容如 3.1 所示，位置不建议放到系统盘，因为后续使用还可能扩充。



图 4.1 安装模块选择

安装下载完毕后，可能会提示重启电脑，重启后打开，稍作等待会让你选择界面风格，自行选择即可。

5. 程序设计

5.1 程序功能模块

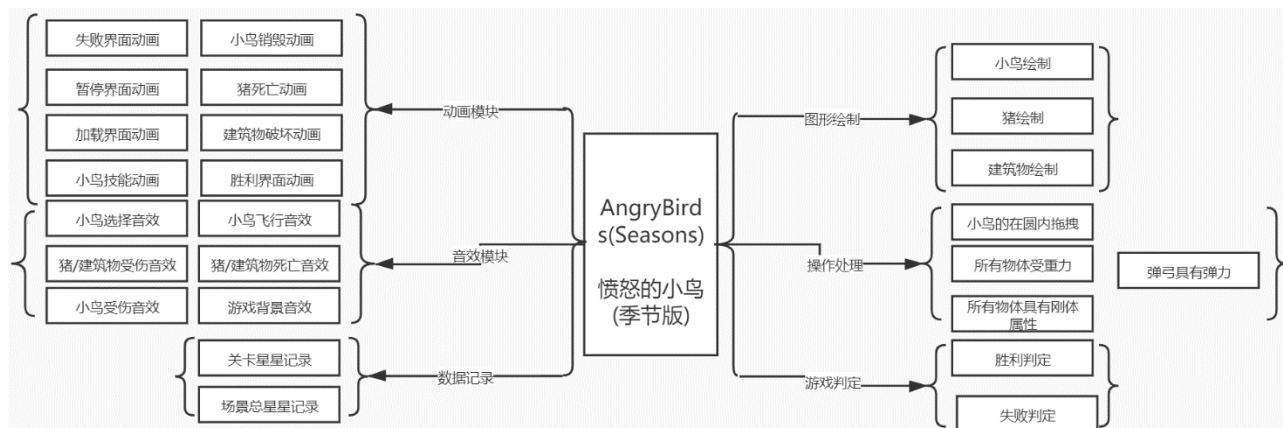


图 5.1 功能模块图

《AngryBirds(Seasons) 愤怒的小鸟(季节版)》的整体功能如上表所示。

5.2 程序整体逻辑

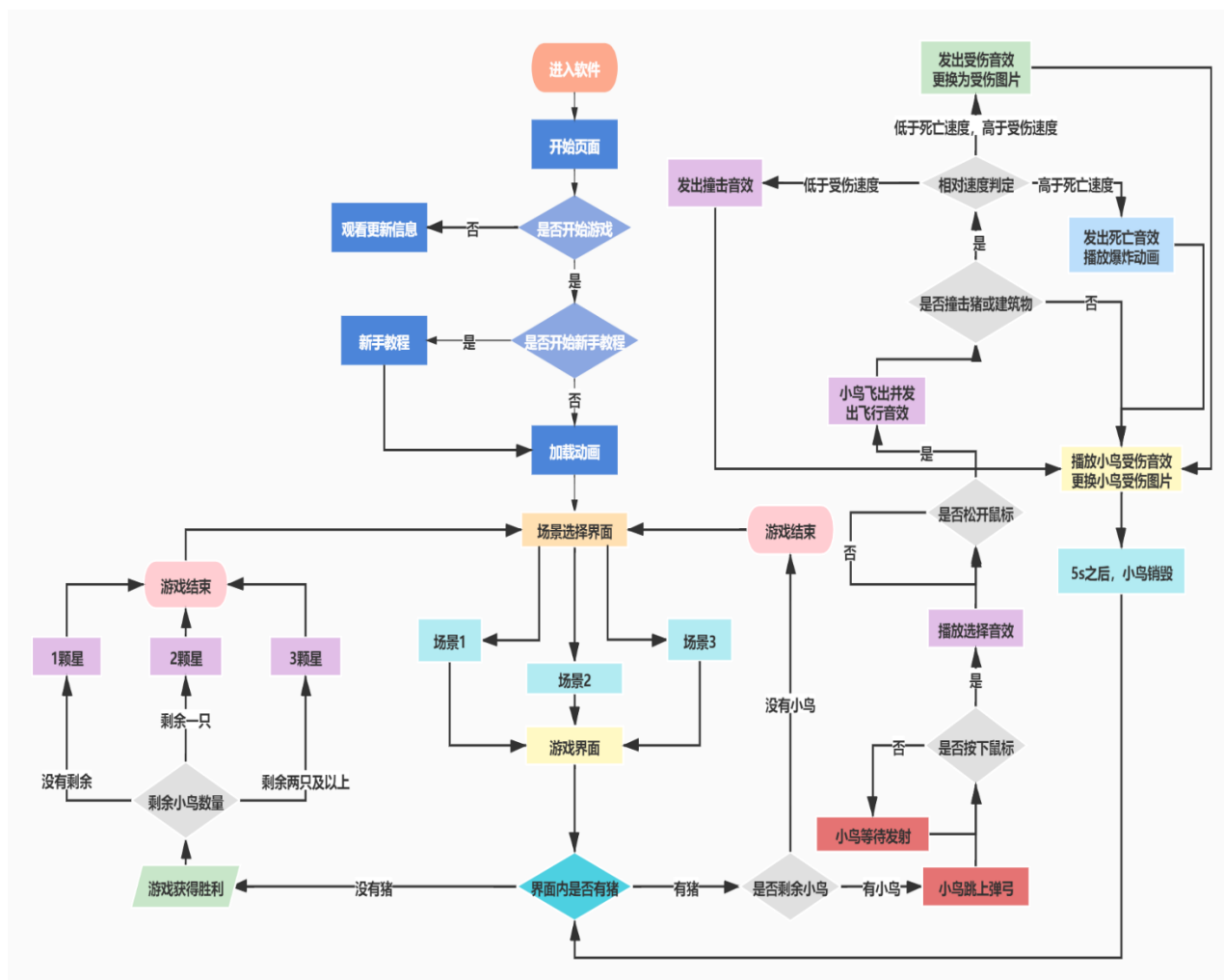


图 5.2 程序流程图

5.3 核心算法流程

5.3.1 GameManager.cs 文件解析

作为整个游戏的管理文件，GameManager.cs 是每一个 Unity 程序都必备的，它包含了整个游戏的初始化，场景调换，数据存储，功能键设置以及游戏逻辑的体现。

下面是该文件的变量申明：

```

public List<Bird> birds;    //用列表的形式表示小鸟的集合
public List<Pig> pig;      //用列表的形式表示猪的集合
public static GameManager _instance;    //当前的游戏管理对象

private Vector3 originPos;    //初始小鸟的位置

public GameObject win;        //获取面板的输赢状况
public GameObject lose;

public GameObject[] stars;    //从面板中获取星星集合

private int starNum = 0;    //记录每一关的得到星星数（方便之后数据的存储）

private int totalNum = 15;    //该场景下所有的关卡数

```

图 5.3 GameManager.cs 变量申明

下面是游戏的初始化：

```

private void Awake()    //初始化
{
    _instance = this;    //读取游戏管理对象
    if (birds.Count > 0)    //如果有小鸟
    {
        originPos = birds[0].transform.position;    //初始位置为第一只小鸟的位置
    }
}

@ Unity 消息 | 0 个引用
private void Start()
{
    Initialized();    //初始化
}

2 个引用
private void Initialized()    //初始化
{
    for (int i = 0; i < birds.Count; i++)    //遍历小鸟集合
    {
        if (i == 0)    //如果是第一只小鸟
        {
            birds[i].transform.position = originPos;    //把小鸟放置在初始位置，使其更柔和
            birds[i].enabled = true;    //该小鸟可用
            birds[i].sp.enabled = true;    //该小鸟的物理效果开启（sp是在Bird脚本里的弹动与摆动的物理效果）
            birds[i].canMove = true;    //小鸟可以被鼠标点击
        }
        else
        {
            birds[i].enabled = false;    //该小鸟不可用
            birds[i].sp.enabled = false;    //该小鸟的物理效果关闭
        }
    }
}

```

图 5.3 GameManager.cs 初始化

如图 5.3 所示，GameManager.cs 的初始化与游戏逻辑密不可分，游戏内部的初始化操作实现可以在如上的 Initialized()函数中可见，小鸟的物理组件的开启与关闭都在该函数中得到调控，同时也为调整待发射小鸟的位置，即从地面跳转至弹弓上做了准备。

下面是具体游戏逻辑实现：

```
public void NextBird() //判断游戏逻辑（如果已经获胜了，就不再需要进行Initialized()的初始化操作了，此时剩余的小鸟都在地面上）
{
    if (pig.Count > 0)
    {
        if (birds.Count > 0) //准备初始化下一只小鸟
        {
            Initialized(); //初始化
        }
        else //输了
        {
            lose.SetActive(true); //启动输了的画面
        }
    }
    else //猪已经全部消灭，获得胜利
    {
        win.SetActive(true); //启动赢了的画面
    }
}

1 个引用
public void ShowStars() //胜利时出现星星
{
    StartCoroutine("show"); //开启协程
}

0 个引用
IEnumerator show()
{
    for (; starNum < birds.Count + 1; starNum++)
    {
        if (starNum >= stars.Length)
            break; //如果超过三个星星就直接跳出循环
        else
        {
            yield return new WaitForSeconds(0.2f); //等待0.2s
            stars[starNum].SetActive(true); //开启星星特效
        }
    }
}
```

图 5.4 GameManager.cs 游戏逻辑

如图 5.4 所示，GameManager.cs 中游戏逻辑中的 NextBird()函数用来控制游戏的输赢，同时也与初始化 Initialized()函数联合完成小鸟从预备到发射的连贯操作。同时游戏逻辑函数 NextBird()函数与胜利和失败的界面动画也进行了连接，图片上的 ShowStars()函数为胜利界面上星星显示的特效开启函数。这样使游戏从开始到结束流畅，符合基本游戏顺序。

下图是游戏功能键的设置：


```

public void StartTOPlay()
{
    SceneManager.LoadScene(1); //File中Build Setting中对于01-Level场景的模块序号为1
}
0 个引用
public void Replay() //如果按下重玩按钮
{
    SavaData(); //保存数据
    Time.timeScale = 1;
    SceneManager.LoadScene(2); //File中Build Setting中对于02-Game场景的模块序号为2
}

0 个引用
public void Home()
{
    SavaData(); //保存数据
    Time.timeScale = 1;
    SceneManager.LoadScene(1); //File中Build Setting中对于01-Level场景的模块序号为1
}

0 个引用
public void NextLevel() //如果按下下一关的按钮
{
    SavaData();
    Time.timeScale = 1;
    string levelNum = PlayerPrefs.GetString("nowLevel");
    //去掉字符串里带level的字符，即得到当前是第几关
    levelNum = levelNum.Replace("level", "");
    //关卡数加一 这里还要判断一下当前i是否大于当前地图里边最大的关卡数
    int i = int.Parse(levelNum) + 1;
    levelNum = "level" + i.ToString();
    PlayerPrefs.SetString("nowLevel", levelNum);
    SceneManager.LoadScene(2);
}

```

图 5.5 GameManager.cs 功能键设置

如图 5.5 所示，GameManager.cs 中包含了所有场景的功能键的实现，如果游戏的某一个按钮需要添加功能，只需要添加 Button()组件和 On Click()组件，然后在 On Click()中选择添加 GameManager.cs 中实现该功能的函数即可。具体添加组件可见下图

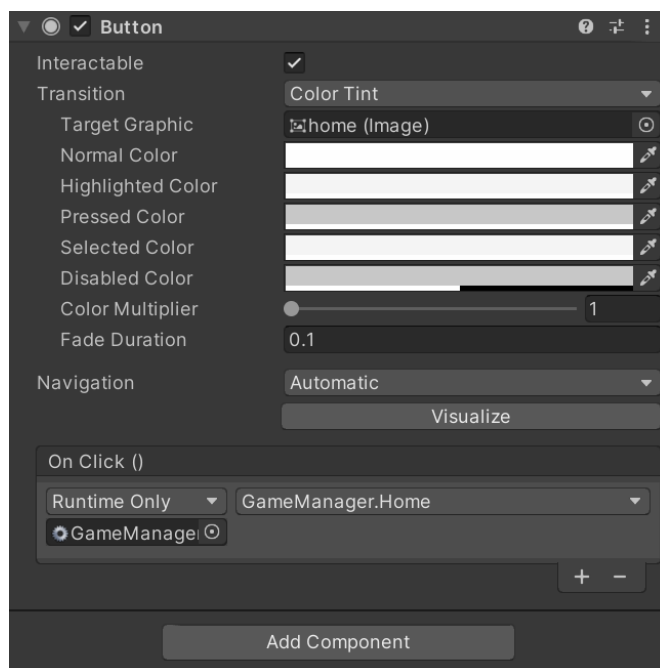


图 5.6 按钮添加 GameManager.cs 功能键函数

下图是游戏数据存储：

```
public void SavaData()
{
    //PlayerPrefs.SetInt("string",int num)          通过键值对存储该关卡的星星数量
    /*
     * 使用本地持久化类PlayerPrefs完成Unity整个游戏的数据存储
     * 很有技巧的数据存储
     * 避免的外部数据库的使用
     */
    if (starNum > PlayerPrefs.GetInt(PlayerPrefs.GetString("nowLevel"))) //如果获得的星星大于历史记录
    {
        PlayerPrefs.SetInt(PlayerPrefs.GetString("nowLevel"), starNum); //获取nowLevel对应的level + 该关卡的序号，存下该关卡的获得星星数
    }
    //存储所有的星星个数
    int sum = 0;
    for (int i = 1; i <= totalNum; i++)
    {
        sum += PlayerPrefs.GetInt("level" + i.ToString()); //累加该场景目前所有星星数（由于显示text中的分子）
    }

    PlayerPrefs.SetInt("totalNum", sum); //通过键值对存储该场景totalNum个关卡所获得的所有星星数
    //print(PlayerPrefs.GetInt("totalNum"));
}
```

图 5.7 GameManager.cs 游戏数据存储

相较于其他的许多软件需要外部数据库存储该软件内的数据，Unity 提供了一个本地持久化类 **PlayerPrefs** 来实现游戏的数据存储，这样可以使游戏与数据更紧密，也会防止数据的外泄以及软件加载过慢的问题。此处的 **SavaData()** 函数用于所有的除了暂停和继续的功能键处，这样可以确保游戏的进度和数据不会因此丢失。

5.3.2 Bird.cs 文件解析

小鸟是整个游戏的中心物体，所以对小鸟的所有行为控制，除了 **GameManager.cs** 中对其初始化之外，其余的行为都需要控制，因此 **Bird.cs** 文件的代码相较于其他脚本而言显得更为复杂和多。

下图是鼠标控制小鸟行为的部分：

如下图 5.8 所示，由于小鸟的很多物理组件在初始化的时候被开启或者关闭，但是这些只是预备操作，真正的判断是当玩家开始通过鼠标点击小鸟之后。物理动力学的开启与关闭以及小鸟飞出 (**Fly()** 函数控制) 之后不再受鼠标控制，还有小鸟被鼠标点击之后弹弓的划线模拟弓带 (**Line()** 函数控制) 都需要 **OnMouseDown()** 和 **OnMouseUp()** 来控制。同时音效的开启与关闭也离不开鼠标控制这个板块。

```

private void Awake()           //初始化
{
    sp = GetComponent<SpringJoint2D>(); //获取当前弹动与摆动的物理效果
    rg = GetComponent<Rigidbody2D>();   //获取当前刚体的物理运动状态
    myTrail = GetComponent<TestMyTrail>(); //获取当前拖尾状态
    render = GetComponent<SpriteRenderer>(); //获取当前小鸟的状态的图片
}

☉ Unity 消息 | 0 个引用
private void OnMouseDown()    //鼠标按下
{
    if(canMove)
    {
        Audioplay(select);      //开启选择音效
        isClick = true;
        rg.isKinematic = true;  //开启物理动力学
    }
}

☉ Unity 消息 | 0 个引用
private void OnMouseUp()      //鼠标抬起
{
    if (canMove)
    {
        isClick = false;
        rg.isKinematic = false; //关闭物理动力学
        Invoke("Fly", 0.1f);    //延迟0.1s, 给予足够多的时间进行物理计算

        //禁用划线组件
        right.enabled = false;  //弹弓右子中心的划线效果消失
        left.enabled = false;   //弹弓左子中心的划线效果消失

        canMove = false;       //飞出去的小鸟不再被鼠标控制
    }
}

```

```

void Fly()
{
    isFly = true;           //在飞行途中
    isReleased = true;      //已经被释放, 此时与弹弓没有联系
    Audioplay(fly);         //开启飞行音效
    myTrail.StartTrails();  //开启拖尾效果
    sp.enabled = false;     //弹动与摆动的物理效果消失
    Invoke("Next", 5);      //小鸟飞出5s之后, 分别依次执行移除列表中的小鸟, 销毁小鸟, 出现小鸟消失特效
}

1 个引用
void Line() //划线操作 (两点确定一条直线)
{
    //启用划线组件
    right.enabled = true;    //弹弓右子中心的划线效果开启
    left.enabled = true;     //弹弓左子中心的划线效果开启

    right.SetPosition(0, rightPos.position); //两点确定一条直线
    right.SetPosition(1, transform.position);

    left.SetPosition(0, leftPos.position);    //两点确定一条直线
    left.SetPosition(1, transform.position);
}

```

图 5.8 Bird.cs 鼠标控制

下图是相机追随小鸟移动的部分：

```
private void Update()
{
    if (EventSystem.current.IsPointerOverGameObject()) //是否在点击UI界面（解决暂停后点击会触发小鸟技能的bug）
        return;
    if (isClick) //鼠标一直按下,进行位置跟随
    {
        transform.position = Camera.main.ScreenToWorldPoint(Input.mousePosition); //坐标系转换

        //transform.position += new Vector3(0, 0, 10);
        transform.position += new Vector3(0, 0, -Camera.main.transform.position.z); //小鸟图层深度的改变

        if (Vector3.Distance(transform.position, rightPos.position) > maxDis) //进行位置限定
        {
            Vector3 pos = (transform.position - rightPos.position).normalized; //获得从弹弓子中心指向小鸟的单位向量
            pos *= maxDis; //最大长度向量
            transform.position = pos + rightPos.position; //限制小鸟的活动范围
        }

        Line(); //开始划线,出现皮筋的效果
    }

    //相机跟随
    float posX = transform.position.x; //获取到待飞小鸟的一维x轴的位置
    float posY = transform.position.y; //获取到待飞小鸟的一维y轴的位置

    //Lerp( , , )通过向量插值实现主相机平滑地跟随,第一个点是当前目标点,第二个是目的地,第三个为当前运动速率(平滑度 * 时间间隔)
    //Mathf.Clamp(value, min, max) 把value限制在min和max之间 相机x的范围

    Camera.main.transform.position = Vector3.Lerp(Camera.main.transform.position, new Vector3(Mathf.Clamp(posX, 0, 40), Mathf.Clamp(posY, 0, 15),
        Camera.main.transform.position.z), smooth * Time.deltaTime);

    if (isFly) //如果处于飞行状态
    {
        if (Input.GetMouseButtonDown(0)) //此时单击鼠标
        {
            ShowSkill(); //触发小鸟技能
        }
    }
}
```

图 5.9 Bird.cs 相机移动

由于 Unity 的 Game 展示屏幕十分狭窄，如果我们把小鸟以及猪和建筑物全部放置于一个画布的范围之内，会使得玩家感觉拥挤从而有不适感。因此我们需要拓展游戏的范围，但是如果只是简单的拓展场景，玩家并不会看到，因此我们需要将主相机捆绑在小鸟上，这样就可以是 Game 展示屏幕中呈现我们设置的扩充场景，使得游戏不在拥挤，更为开阔，增加玩家游戏体验。

5.3.3 Pig.cs 文件解析

猪和建筑物作为整个游戏仅次于小鸟的游戏物体，其行为的处理也十分重要。如果我们抛开游戏的胜利条件即游戏场景内部不存在其他猪的话，猪与建筑物其实采用的脚本都是一致的，比如说受伤的图片更改，受伤的音效播放，死亡的爆炸效果，死亡的得分效果等等。把猪与建筑物高度统一，可以使游戏代码量减少，也可以使游戏编写时更有层次与逻辑感。

下图是碰撞检测和死亡处理部分：

对于猪和建筑物的受伤和死亡的判断，我们没有采用存储生命值的方式来判断猪和建筑物的状态，因为一旦猪和建筑物的数量很多，需要存储的数据很大，而且每一次小鸟的碰撞，都需要进行全部的状态数据更新，这样会使原本小量级的游戏因此变得卡顿，严重影响到了玩家的游戏体验。所以我们可以采用相对速度的方式来实现猪和建筑物的受伤和死亡的判断，如果相对速度小于受伤的临界速度，猪和建筑物只会触发碰撞音效；如果相对速度位于受伤的临界速度与死亡

的临界速度之间，猪和建筑物会受伤，同时触发相应音效；如果相对速度大于死亡临界速度，猪和建筑物销毁，同时触发相应的爆炸动画和音效。由于 Unity 中有 `collision.relativeVelocity()` 碰撞组件可供我们选择，所以采用这种方式很容易实现。

此时有一个细节问题需要我们考虑，相对速度具体到底是指碰撞前小鸟与猪和建筑物的相对速度（即小鸟的速度减去猪和建筑物的速度），还是碰撞前后猪和建筑物的相对速度（即前后速度相减）。此时我们采用物理来解决这个问题，使用动量守恒：设小鸟质量为 m ，速度为 v ，猪质量为 M ，碰撞后小鸟的速度为：

$\frac{(M-m)*v}{(M+m)}$ ，猪的速度为 $\frac{2mv}{(M+m)}$ ，因此比较小鸟与猪碰撞前的相对速度和比较碰撞后猪的速度的变化量大小其实一样，差了一个常数倍数，都可以判断猪的状态。

```
private void OnCollisionEnter2D(Collision2D collision)//碰撞检测
{
    //print(collision.relativeVelocity.magnitude); //显示相对速度，合理选取受伤和死亡的取值范围
    //collision.relativeVelocity（相对速度）是向量，需要转换为标量
    if(collision.gameObject.tag == "Player") //如果小鸟碰撞的是猪或者建筑物
    {
        Audioplay(birdCollision); //播放小鸟碰撞时使用的音乐组件
        collision.transform.GetComponent<Bird>().Hurt(); //更新小鸟受伤图片
    }

    if (collision.relativeVelocity.magnitude > maxSpeed)//碰撞相对速度大于最大速度，直接死亡
    {
        Dead(); //猪死亡的效果处理
    }
    else if (collision.relativeVelocity.magnitude > minSpeed && collision.relativeVelocity.magnitude < maxSpeed) //相对速度位于最大和最小速度之间为受伤状态
    {
        render.sprite = hurt; //图片更新为受伤图片
        Audioplay(hurtClip); //播放猪或者建筑物碰撞时使用的音乐组件
    }
}

2 个引用
public void Dead()
{
    isDead = true; //已经死亡
    if(isPig)
    {
        GameManager._instance.pig.Remove(this); //如果是猪，从列表中移除它
    }
    Destroy(gameObject); //猪或建筑物死亡后销毁
    Instantiate(boom, transform.position, Quaternion.identity); //产生爆炸效果

    GameObject go = Instantiate(score, transform.position + new Vector3(0,0.5f,0), Quaternion.identity); //产生得分效果
    Destroy(go,2); //销毁得分特效

    Audioplay(dead); //播放猪死亡或者建筑毁坏使用的音乐组件
}
```

图 5.10 Pig.cs 碰撞检测和死亡处理

5.3.4 LevelSelect.cs 文件解析

除了游戏场景之外，最重要的就是场景选择和关卡选择界面了，因为这涉及到游戏外部逻辑的实现。这里我们可以采用一个经典算法——前驱法，该算法原理：如果某一个关卡它的前面一关是开启的，且星星数量大于 0，说明该关卡可以被选择，考虑边界情况，如果这个关卡没有前驱，且自己没有星星数量，说明这个是第一关，那么直接开启即可。

下图是场景与关卡开启部分：

```

private void Start()
{
    if (transform.parent.GetChild(0).name == gameObject.name) //如果是第一关
    {
        isSelect = true; //该关卡可以被选择
    }
    else
    {
        //前驱算法开启下一关
        int beforeNum = int.Parse(gameObject.name) - 1; //获得前一关的关卡名
        if (PlayerPrefs.GetInt("level" + beforeNum.ToString()) > 0) //使用前驱算法, 判断是否满足开启条件
        {
            isSelect = true; //可选择
        }
    }

    if (isSelect)
    {
        image.overrideSprite = levelBG; //替换为可选择后的关卡图片
        transform.Find("num").gameObject.SetActive(true); //激活num对应的关卡图片

        //通过字符串拼接获得关卡名字, 然后获得该关卡对应的星星数量
        int count = PlayerPrefs.GetInt("level" + gameObject.name);
        //
        if (count > 0) //如果该关卡星星数大于零
        {
            for (int i = 0; i < count; i++) //遍历, 同时让相等数量的星星显示在选择关卡界面
            {
                stars[i].SetActive(true); //显示星星
            }
        }
    }
}

```

图 5.11 LevelSelect.cs 场景与关卡开启

5.3.5 技能小鸟 cs 文件解析

由于部分小鸟有特殊技能, 所以对于特殊的小鸟都必须具备其特有的脚本, 但是这并不意味着每一种特殊的小鸟的脚本都需要单独编写, 由于小红鸟具有最一般的性质, 其他的小鸟的属性都是在其基础上进行修改的, 所以我们可以完全继承 Bird.cs 中的所有属性和方法, 然后对其单独的某一个板块进行重写即可。

Bird.cs 中可进行重写的函数只有一个, 叫做 ShowSkill(), 这是一个虚方法, 方便进行重载和继承, 下图是位于 Bird.cs 中 ShowSkill() 的声明:

```

11 个引用
public virtual void ShowSkill() //小鸟各种不同的技能(虚方法, 方便重载继承)
{
    isFly = false; //飞行过程结束, 不可再实现小鸟技能
}

```

图 5.12 Bird.cs 中 ShowSkill() 的声明

5.3.5.1 YellowBird.cs 和 PinkBird.cs 文件解析

黄色小鸟 (单击之后速度变为 2 倍) 和粉色小鸟 (单击之后速度变为 4 倍) 的 ShowSkill() 方法重写很简单, 其实现采用刚体组件里的 velocity 方法即可修改

速度的具体数值。下图是黄色小鸟的 ShowSkill()方法重写，粉丝小鸟把 2 改成 4 即可。

```
public class YellowBird : Bird           //继承一般红鸟的脚本
{
    7 个引用
    public override void ShowSkill()     //重写技能板块
    {
        base.ShowSkill();               //继承基类
        rg.velocity *= 2;                //改变子类，使其速度变为原来的两倍
    }
}
```

图 5.12 YellowBird.cs 中 ShowSkill()的重写

5.3.5.2 GreenBird.cs 文件解析

绿色小鸟（单击之后速度反向）如果按照原版游戏实现困难，按照原游戏，点击之后应该是加速度朝上的匀减速运动，之后再做加速度向上的加速运动，画出一个类圆弧的形状。而我们只需要让其速度的 x 轴分量反向，y 轴分量清零即可简易实现反向的功能，下图是绿色小鸟的 ShowSkill()方法重写。

```
public class GreenBird : Bird           //继承一般红鸟的脚本
{
    7 个引用
    public override void ShowSkill()     //重写技能板块
    {
        base.ShowSkill();               //继承基类
        Vector3 speed = rg.velocity;     //修改速度
        speed.x *= -0.5f;                 //x轴反向
        //y轴变为0（按照原游戏，应该是加速度朝上的匀减速运动，之后再做加速度向上的加速运动，画出一个类圆弧的形状）
        speed.y = 0;
        rg.velocity = speed;
    }
}
```

图 5.13 GreenBird.cs 中 ShowSkill()的重写

5.3.5.3 OrangeBird.cs 文件解析

橙色小鸟（单击之后会膨胀）实现也很简单，在点击鼠标之后，修改原小鸟的碰撞范围即可，小鸟一般会添加两个组件，一个是 Circle Collider 2D，用来实现圆形区域的碰撞，另一个是 Rigid Body 2D，用来模拟刚体的物理效果。因此我们可以直接修改 Circle Collider 2D 中的碰撞圆形的半径即可。下图是橙色小鸟的 ShowSkill()方法重写。

```

public class OrangeBird : Bird //继承一般红鸟的脚本
{
    private CircleCollider2D circle; //未膨胀前的圆形碰撞范围
    public Sprite clickOrange; //点击鼠标膨胀后的图片
    private SpriteRenderer render_orange;
    7 个引用
    public override void ShowSkill() //重写技能板块
    {
        base.ShowSkill(); //继承基类
        circle = GetComponent<CircleCollider2D>(); //获取未膨胀前的圆形碰撞范围
        render_orange = GetComponent<SpriteRenderer>(); //获取未膨胀前的图片信息
        circle.radius += 1; //膨胀后圆形半径增加
        render_orange.sprite = clickOrange; //更换图片
    }
}

```

图 5.14 OrangeBird.cs 中 ShowSkill() 的重写

5.3.5.4 BlackBird.cs 文件解析

相比于其它小鸟的技能实现，黑色小鸟的爆炸效果实现比较困难。现给出其设计思路：从 5.3.5.3 的解析中我们了解到 Circle Collider 2D 组件，其作用在于规定一个圆形碰撞范围，这里我们需要两个 Circle Collider 2D 组件，一个用于小鸟自身的碰撞，另一个用于确定爆炸范围，注意该 Circle Collider 2D 中的 Is Trigger 选项要搭上勾，因为这个是触发检测。

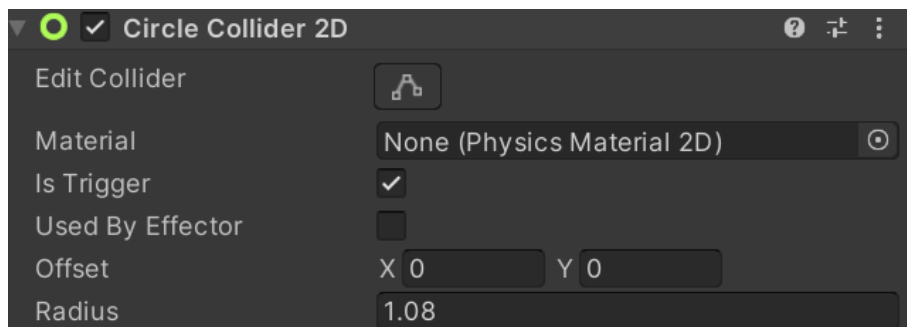


图 5.15 Circle Collider 2D 的注意事项

所以我们可以让小鸟飞行的时候，设置一个集合，这个集合里面存放着以黑色小鸟为圆心的圆中可爆炸销毁猪和建筑物（有 Enemy 的标签），当我们点击鼠标触发技能的时候，销毁集合里面的物体即可，销毁的时候调用 Pig.cs 里面的 Dead()方法。下图是黑色小鸟的 ShowSkill()方法重写。


```

public class BlackBird : Bird //继承一般红鸟的脚本
{
    public List<Pig> blocks = new List<Pig>(); //这个集合里存放以黑炮为圆心的圆中可爆炸销毁猪和建筑物（有Enemy的标签）
    Ⓢ Unity 消息|0 个引用
    private void OnTriggerEnter2D(Collider2D collision) //进入圆圈里的触发区域
    {
        if (collision.gameObject.tag == "Enemy") //如果标签是Enemy
        {
            blocks.Add(collision.gameObject.GetComponent<Pig>());
        }
    }
    Ⓢ Unity 消息|0 个引用
    private void OnTriggerExit2D(Collider2D collision) //离开圆圈里的触发区域
    {
        if (collision.gameObject.tag == "Enemy") //如果标签是Enemy
        {
            if (collision.GetComponent<Pig>().isDead == false) //之前没有触发Dead方法
                blocks.Remove(collision.gameObject.GetComponent<Pig>());
        }
    }

    7 个引用
    public override void ShowSkill() //重写技能板块
    {
        base.ShowSkill(); //继承基类
        if (blocks.Count > 0 && blocks != null) //如果存在障碍物
        {
            for (int i = 0; i < blocks.Count; i++) //爆炸清除障碍物
            {
                blocks[i].Dead(); //爆炸清除障碍物
            }
        }
        OnClear(); //处理后事
    }
}

```

图 5.16 BlackBird.cs 中 ShowSkill()的重写

5.3.6 拖尾效果解析

拖尾效果可以使游戏的视觉感觉更好，同时描绘出来的弧线也可以使玩家体验最真实的物理引擎。由于 Unity 的技术交流频繁，而且众多开发人员对于同一个问题提供了模板性的解决方案，所以对于 Unity 中常见的拖尾效果，我们直接引用模板组件即可。

模板使用连接：<https://blog.csdn.net/akofl314/article/details/37603559>;

6. 测试与部署

6.1 软件测试

软件测试贯穿软件生命周期的全过程，鉴于项目代码结构并不复杂，测试时主要对本游戏中的每个基本功能，包括开始界面等五个界面的衔接，每一个功能键的实现，以及游戏场景内的小鸟与猪和建筑物等进行单元测试，多是对单个函数的单元测试以及程序整体运行流程的正常推进。要确保考虑到各种边界情况，保证涵盖所有的可能性，在测试时可以使用黑盒测试或白盒测试等方法验证代码正确性。完成度较高后也需要进行实机测试，自己玩一下才会发现更多细节问题。同

时在已有框架基础上的程序开发过程中，保持良好的代码习惯，包括注重代码可读性、可维护性、可复用性和添加注释等方式，都能减轻代码修改和测试过程中调试 bug 的难度。

6.2 软件部署和发布

案例项目属于 Unity 程序项目，结构和功能较为简单，属于初学者常使用的编码修正模型，尚未涉及到较大规模的团队软件开发。Unity 游戏的发布有一套固定的流畅：

首先点击 File → Build Settings

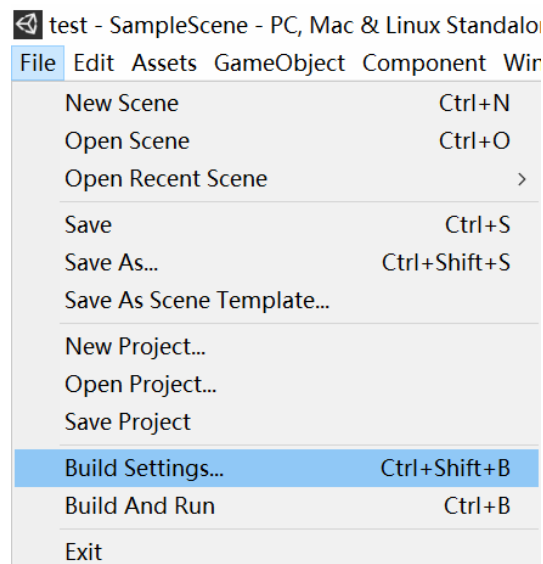


图 6.1 Unity 进入 Build Settings 界面

下图为 Build Settings 界面，其中左下角 Player Settings 为之后点击处，而右下角的 Build 和 Build And Run 则是游戏正式发布。

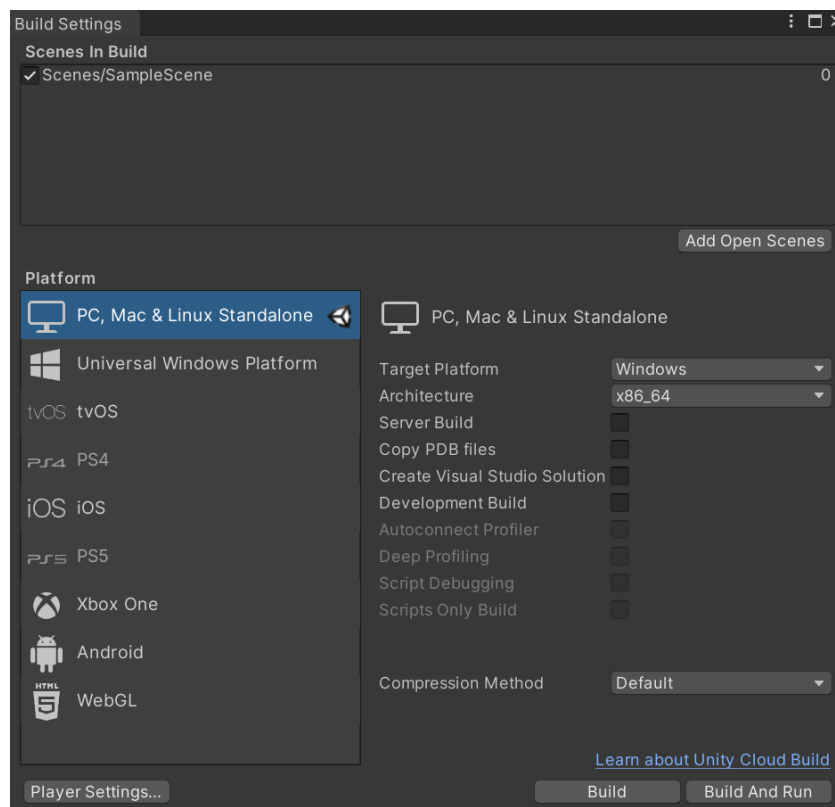


图 6.1 Build Settings 界面

我们点开左下角的 Player Settings 界面,其中 Product Name 为该项目的名称,Version 为版本序号(该游戏第一发布 Version 为 1.0),Default Icon 为该游戏的图标,Default Cursor 为该游戏中鼠标的图标,修改完毕后关闭回到 Build Settings。

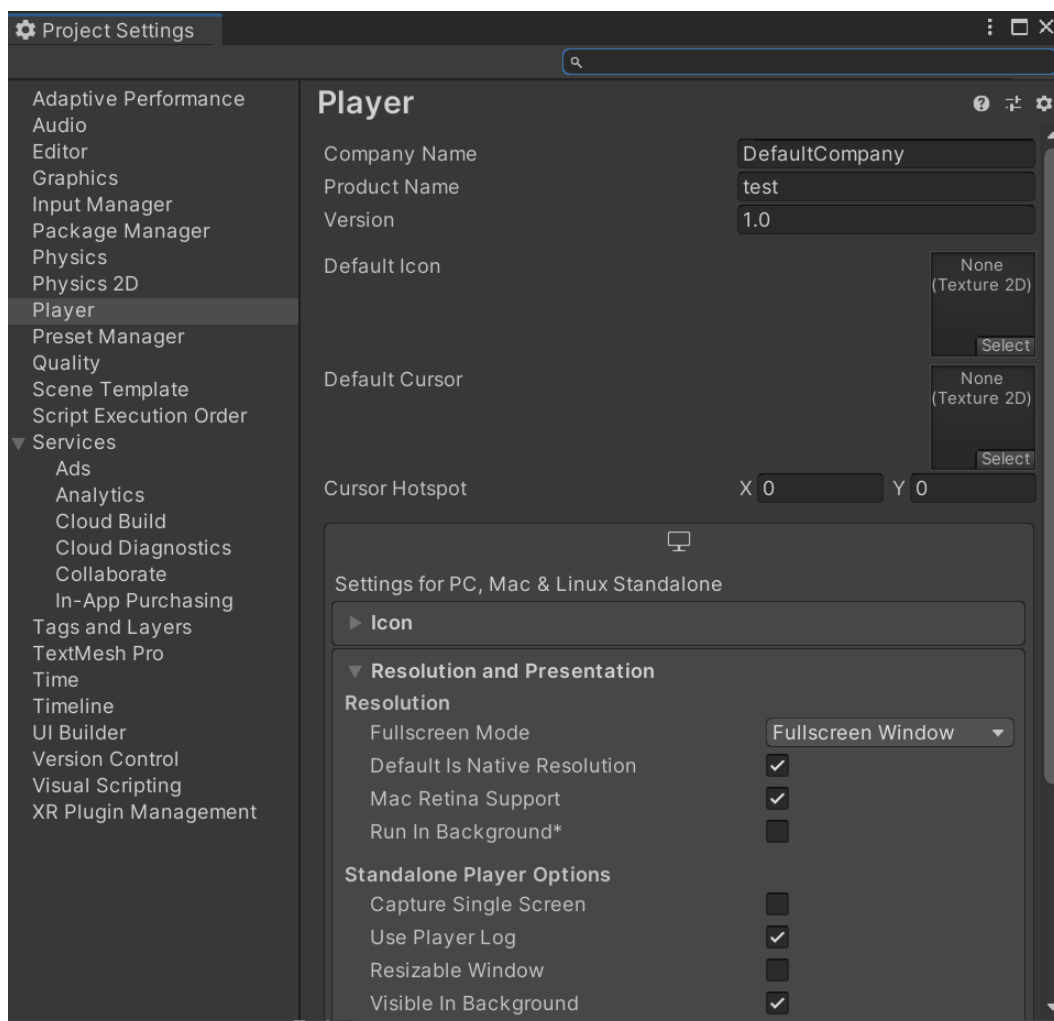


图 6.2 Player Settings 界面

回到 Build Settings 之后点击 Build 和 Build And Run 即可完成游戏正式发布,其发布的内容一般包括五个文件,其中一个为 exe 可执行文件。

AngryBirds(Seasons)_Data	2021/8/30 21:55	文件夹	
MonoBleedingEdge	2021/8/30 21:55	文件夹	
AngryBirds(Seasons)	2021/7/31 8:31	应用程序	639 KB
UnityCrashHandler64	2021/7/31 8:32	应用程序	1,101 KB
UnityPlayer.dll	2021/7/31 8:32	应用程序扩展	27,392 KB

图 6.3 发布内容

但是注意,通过这种方式获得的游戏软件其内部数据已经封装完毕,从外部很难直接修改其内部数据,同时之前准备的素材图片和素材音乐等均会转化为相应的数据存储形式(比如.dll 文件),先要在发布之后进行修改比较困难,所以如果发现有问题建议在 Unity Hub 找到原项目,在 Unity 上对未发布的原项目进行修改,然后选择重新发布。