AngryBirds（Seasons）系统设计文档

20373623 软件学院 周恩申

一、 系统概述

《AngryBirds(Seasons) 愤怒的小鸟(季节版)》是一款经典的休闲益智类游戏，其游戏机制以原版的愤怒的小鸟为基础，并加入了许多开发者自创的游戏玩法。开发者充分借鉴其设计游戏的思路，融入了魔法元素，增添了小鸟种类，改善了背景音乐和加载动画。玩家可以在本游戏中感受真实的物理引擎，体验弹弓发射摧毁目标的获胜感受以及强大魔法带来的视觉享受，在与猪猪大作战的同时，达到休闲益智的效果。

图 1 游戏图标

《AngryBirds(Seasons) 愤怒的小鸟(季节版)》开发环境为 Unity2D，编写脚本采用 C#语言，现版本代码量为 1600 行左右。

二、 运行环境

游戏运行 Windows 平台和 Mac 平台均可，不需要安装任何插件和其他库。如果需要打开脚本源码，需要 VS2019 中的 C#功能。

三、 总体设计

1. 界面设计

游戏的整体界面设计可以开始界面，游戏内容介绍（新手教程）界面，加载

界面，场景选关界面以及游戏界面五个界面。

开始界面中，有两个可以点击的按钮，一个名为“play”，另一个名为“点击获得更新信息”。其中“play”按钮位于屏幕中央，单击后可跳转至游戏内容介绍（新手教程）界面，而“点击获得更新信息”按钮位于界面左下方，点击后可以获得开发者在原版愤怒的小鸟的基础上新增的游戏玩法和内容简介。

图 2 开始界面

游戏内容介绍（新手教程）界面由一个选择是否开启新手教程板块和五个小的介绍组成。游戏内容介绍（新手教程）界面的右上角始终有一个橙色的叉型按钮（下文都以“跳过按钮”代替），点击后会进入加载游戏的界面。在该界面中的选择是否开启新手教程板块里有两个按钮，一个是位于屏幕正中央的“Start Now !”按钮，如果玩家是第一玩这个游戏，请务必点击该按钮进入介绍环节，否则将在后期面临对游戏机制不清楚的情况，如果玩家已经熟知游戏的规则及其内容，可以点击跳过按钮跳过该界面节省时间。进入到介绍板块之后，每一个小介绍包含图片信息，文字解释，右下方的 OK 按钮和跳过按钮，当玩家浏览完介绍的所有信息之后，点击 OK 按钮会进入到下一关介绍，直至游戏内容介绍（新手教程）界面结束。

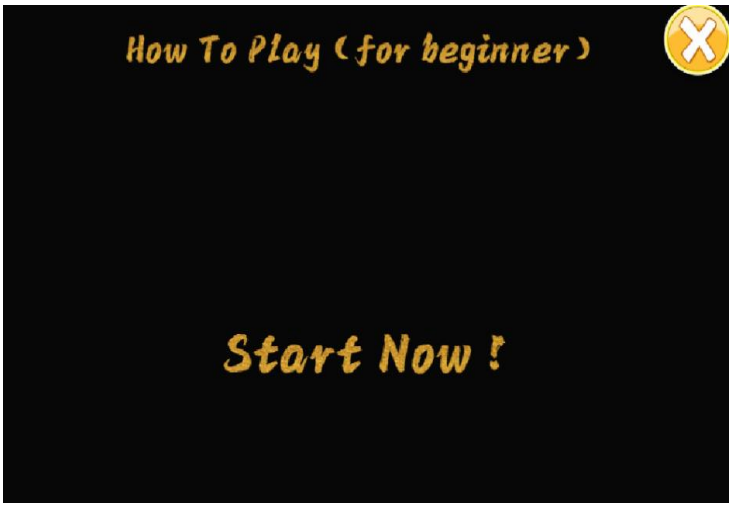


图 3 选择是否开启新手教程板块

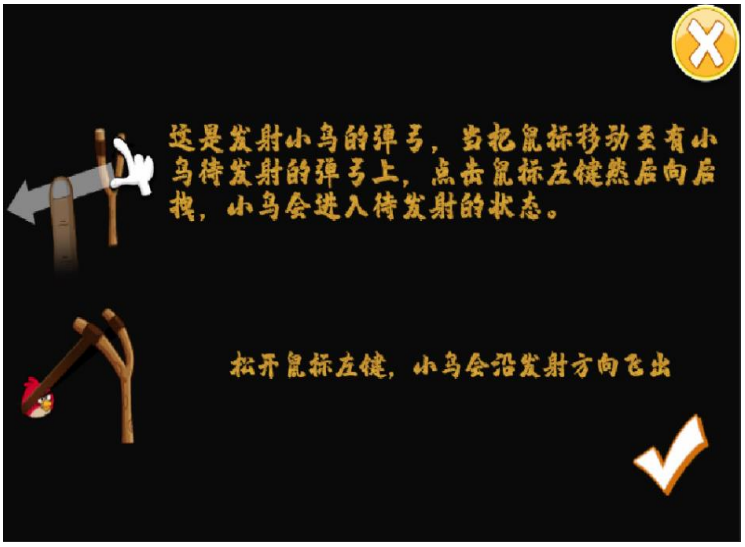


图 4 介绍板块

加载界面是由 20 张图片制作成的动画，持续 10 秒钟，动画播放完毕后自动进入场景选关界面。

图 5 加载界面

场景选关界面由两个板块组成，一个是场景板块，一个选关板块。场景板块由四个类似矩形的图标构成，点击之后可以进入该场景对应的选关板块，但是要注意的是，除了第一个场景是自动开启之外，其余的场景的开启都需要足够的星星数量。选关板块界面包含每个场景对应的关卡，同样的，每一个场景中起初之后第一关是自动开启的，其余的关卡开启都需要条件。同时选关界面的左下角由回退按钮，点击后会从选关板块返回至场景板块。

图 6 场景板块

图 7 选关板块

游戏界面由两个部分组成，一个是左半部分的可操作界面，一个是右半部分的观赏界面。可操作界面由上半部分的暂停功能键很下半部分的弹弓组成，

观赏界面由猪猪搭建的城堡组成，有关暂停功能键下文会有详细的介绍。

图 8 游戏界面

2. 功能设计

《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：

（1） 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。

（2） 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。

（3） 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。

(4) 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。

(5) 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。

(6) 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。

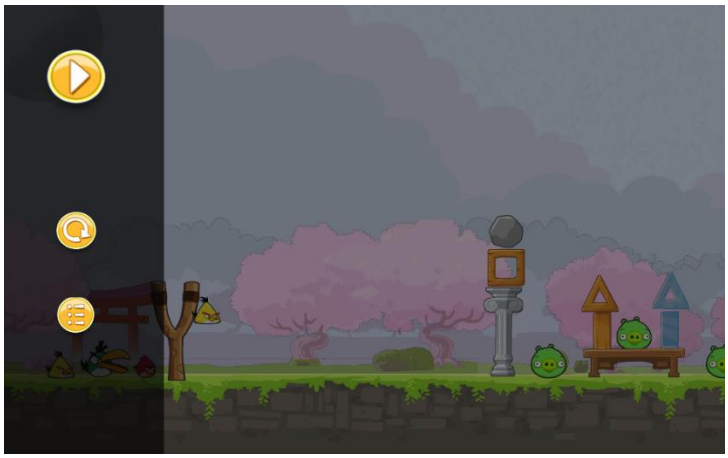


图 9 暂停界面

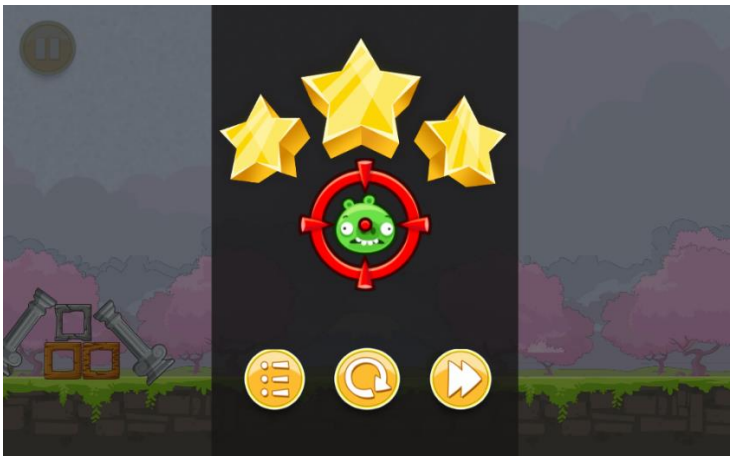


图 10 获胜界面

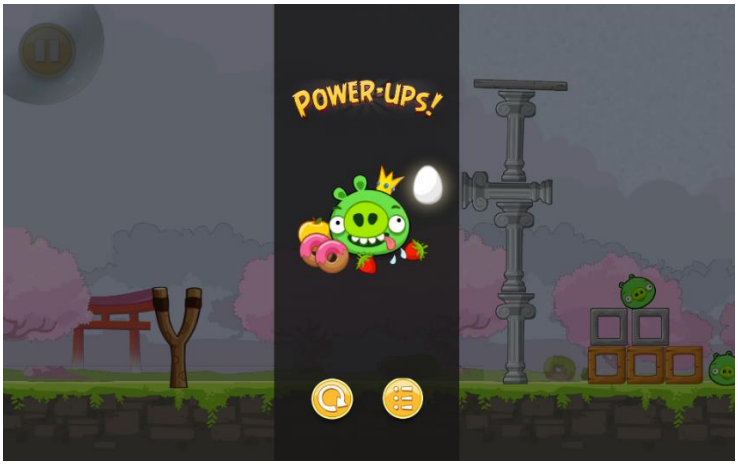


图 11 失败界面

四、 关键算法设计

由于 Unity 包含丰富的物理引擎和强大的函数库，所以对于关键算法的设计更多的在于对游戏逻辑的理解。

1. 小鸟的行为处理

小鸟的行为包括很多，比如鼠标点击后可以进入拖拽状态，鼠标松开后按照物理抛物线的形式发射，在发射的过程中相机需要跟随小鸟移动，小鸟碰撞的物体之后需要进行销毁，同时播放销毁动画和销毁音效。

```
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x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，第三给为当前运动速率(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);
```

图 12 相机跟随代码展示

2. 猪的行为处理

猪的行为包括很多，首先使碰撞检测，碰撞之后判断是受伤状态还是死亡状态，受伤之后的更换图片，播放音效，以及死亡之后播放爆炸动画。

```
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是使用的音乐组件
}
```

图 13 猪的碰撞检测和死亡处理代码展示

3. 场景与关卡的是否可选择以及通关后场景上的星星显示

该算法原理：采用前驱法，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，同时考虑边界情况，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。

```
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 13 前驱算法代码展示

4. 功能键的设置

功能键包括重玩，下一关，回到选关场景三个按钮，其实现的方式是通过进行数据的存储，控制动画的播放，利用数据库中的内容进行场景的跳转。

```
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 14 功能键代码展示

5. 游戏初始化和逻辑的设置

愤怒的小鸟（季节版）对于游戏的逻辑有很高的要求，因为小鸟和猪及建筑物上的物理组件丰富多样，需要进行实时的更新以达到衔接流畅的目的，同时游戏的获胜与失败和游戏胜利后播放的胜利动画也需要利用当时游戏的数据进行判断，所以该板块贯穿整个游戏，十分重要。

```
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
        }
    }
}
2 个引用
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
    }
}
```

图 15 数据存储代码展示

五、 数据库设计

由于 Unity 有着自己独特的本地持久化类 PlayPrefs，所以我们可以借助他来完成整个游戏的数据存储。

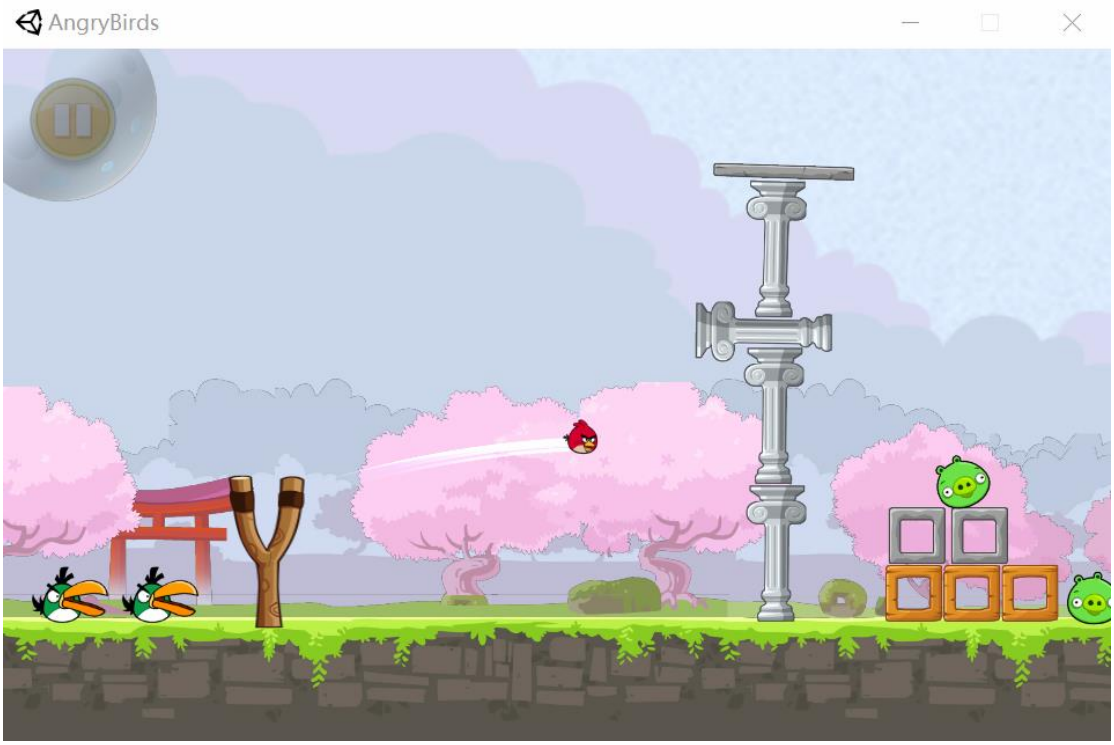
```
public void SavaData()
{
    //PlayerPrefs.SetInt("string",int num)              通过键值对存储该关卡的星星数量
    /*
     * 使用本地持久化类PlayerPrefs完成Unity整个游戏的数据存储
     * 很有技巧的数据存储
     * 避免的外部数据库的使用
     */
    if (starNum > PlayerPrefs.GetInt(PlayerPrefs.GetString("nowLevel")))    //如果获得的星星大于历史记录
    {
        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 16 数据存储代码展示

六、 运行效果展示

由于之前已经展示完所有非游戏过程的所以运行效果，所以该板块只展示游戏过程的运行效果。



七、 心得与展望

1. 心得：（1）熟悉并掌握了一部分的 Unity 开发，对于 Unity 有了更深的理解（2）增加了练习 C# 书写脚本的经验，下次用 C# 书写更有自信（3）从零基础开始，逐步摸索并初次体验小软件开发的过程，这种体验与经历使在未来的学习生涯中不可或缺。
2. 展望：就本游戏而言，期待开发出更多的功能，比如小鸟射出的预计轨迹，小鸟射出后轨迹的保留，以及有可以放大或者缩小的游戏界面的按钮等等，提升游戏的可玩性；就学习生活而言，希望将来开发出更有价值的作品。