# 软件工程

# 项目上机报告

## 项目名称

模拟Helix jump

学号 1405150117 姓名 王智泉

学号 1405150131 姓名 陈佳伟

学号 1405150107 姓名 陈兆雷

日期2018.5.23

**第一章：软件需求（撰写人:）**

**1.1：项目简介**

1.1.1 项目背景

电脑游戏自1972 年由威尔克芳舍(Will-Crowther) 编写的一段简单的FORTRAN程序开始，已经历了数十年风风雨雨了。从最开始避开陷阱的简单地图，到今天即时战略、角色扮演、经营策略、休闲养成等各种类型的游戏;从2D到3D,游戏无论在技术上还是画面上都以惊人的速度不断突破。它的成长速度是如此之快，出乎任何人的意料。角色扮演网络游戏(Role Playing Games)，简称RPG,玩家需要扮演游戏中的位或者多位角色，在虚拟的世界中进行冒险，诞生于上个世纪六十年代的西方世界。角色根据不同的游戏情节和统计数据(例如力量。灵敏度，智力、魔法等)具有不同的能力，而这些属性会根据游戏规则在游戏情节中改变。

目前，游戏业已经以每 年超越200亿美元的产值成为 全球最大的娱乐事业，继美国、日本、韩国之后，越来越多国家也开始重视游戏产业的庞大商机，而以国家力量主导发展电子游戏产业。现在，北美的电子游戏产业(包括TV GAME和PC GAME)的收入已经接近了整个电影产业的收入，这就足以证明游戏行业潜力的巨大。

中国游戏行业起步相对国外较晚。1990年 10月，《轩辕剑》的初伐在台湾成功发行。1995年，大宇制作了中文武供游戏《仙剑奇侠传》这些游戏经推出便受到玩家大力欢迎，以其浓郁的中国风情与独到设计风格被奉为经典之作。据年会期间发布的《2009年中国游戏产业报告》显示，2009 年中国网络游戏实际销售收入为256.2 亿元，比2008年增长了39.4%，为相关产业带来的直接收入达555亿元。2009年，中国网络游戏用户数达到6587万，比2008年增加了33.46%，其中付费网络游戏用户数达到3715万， 比2008 年增加了22.1%。预计2014年网络游戏用户将达到123亿，网络游戏实际销售收入将达到508 亿元。目前，中国游戏产业已经成为新的经济增长点，作为六大支柱产业之一，极大地刺激了中国经济的增长。

3.2 2国内外在该方向的研究现状及分析

正是因为形成了如此庞大的产业经济和用户群体，那么对游戏设计行业的规范和合理化的引导是必不可少的。游戏界面设计是美和技术的结合，游戏已经被越来越多的人承认是种艺术形式， 并且称之为所谓的“第五类艺术”。 其最明显的特点是游戏开始结合了音乐、绘画、文学这三类艺术形式。早期的游戏设计是以功能第为指导原则的，完成和实现必要的功能是其终极目标。研究者们更多的是从工程技术和软件开发的角度来研究的，而单从界面设计的角度来研究游戏界面设计的还比较少，因此本课题具有可研究的空间和价值。

著名的游戏开发者比尔沃尔克(Bill-Volk) 曾经对游戏设计写下了一个等式“界面+产品要素=游戏”，强调在游戏设计中界面的重要性。提到界面，人们很容易将软件与之联系在一起，这是狭义概念的界面设计。从广义上讲，界面，又称用户界面(UI),是指人与物之间相互施加噪响的区域。设计的界面存在于人、物信息交流的一切领域，例如我们切菜的时候刀把手就是这个界面，开车的时候方向盘、仪表盘、后视镜就是这个界面，用电脑的时候显示屏、输入设备，就是这个界面。优秀的界面应该简单并且用户乐于使用，用户在游戏过程中能真正享受人机交流和人性化操作带来愉悦，其核心是自然良好的人机交互性。

交互设计(Interacion Design)作为一门关注交互体验的新学科在一十世纪八十年代产生，它由IDEO的一位创始人比尔莫格里奇在1984 年一次设计会议上首次提出。对计算机游戏而言，交互过程实际上是一个向计算机输入和向用户输出的过程，由于输入、输出的途径是多样的，因此交互的方式也是多样化的。从最初需要专业训练的命令语言用户界面，到用户只需确认的图形用户界面，到引入动态媒体和音频媒体的多媒体用户界面，再到目前通过整合来自多个通道的输入来捕捉用户意图的多通道用户界面，以及未来的虚拟现实技术，交互的媒介从软件界面、键盘、鼠标扩展到视线、语言、动作等，交互发展的趋势体现了对人的因素的不断重视，追求所谓“人机和谐”的多维信息空间和“基于自然交互方式的”的人机交互风格，理想的人机交互模式就是“用户自由”。现在，无论是硬件界面还是软件界面，游戏都朝着这一目标迈进。

腾讯公司的产品直以用户体验为核心要素，其CEO马化腾认为交互设计就是要Don'tmakeme think;符合用户习惯与预期;做适时的提醒;不强迫用户:选择最佳方案，操作便利。这个思想从更加实用的角度分析了产品界面的交互性设计原则。

网络游成与以往的游戏发展演变之路是不同的，已不再单单表现为游戏终端设备的差异，进而体现了游戏传输载体以及游戏参作方式的变革。纵观目前全球游戏产业，网络游戏走走向成熟和深化，“娱乐随时随地”正在梦想成真，技术融合带来游戏跨平台发展，通过网络、个人电脑、大型游戏机、家用游戏机、交互电视、手持终端设备都可以进行互动交流，不同设备可以通过网络运行相同游戏。从游戏界面的发展过程我们可以看出，游戏界面的形式已经变得丰富多样，而这些界面本身也在发生着交融，在基于各种界面的信息传播方式也得到了不同程度的发挥，在泛游戏化的背景下，同时代的其他媒介界面都大量的加入了游戏元素。

3.3 3研究的目的及意义

如今外国游戏已经在中国市场占有很重要的地位，使得中国的文化环境逐渐改变，民族文化已经不能吸引玩家，尤其近年来屡屡的事件和媒体舆论的影响，已经让网络游戏成为大众眼中毒害青少年的精神毒品。而宣传中国传统文化，弓导青少年健康游戏、快乐游戏应该是每个游戏工作者的责任和义务。

网络游戏作为一个新生的艺术，具有独立的美学范式和艺术结构。游戏交互设计的优劣，直接影响着玩家的游戏体验和感性判断。游戏界面作为人机交互的桥梁，其作用无可取代。

游戏玩家对游戏的直观印象，一个来自操作，另一个就是画面。游戏界面本身就是画面的一部分，其地位举足轻重。

界面设计应该是游戏设计中非常重要的一个环节，玩家与游戏系统的直接交互就是通过界面系统完成的。游戏的界面跟产品的外观和功能样，要能吸引玩家并且易用。在设计界面的过程中，要一一直注重易用性设计原则，并且充分考虑用户感受，随时调整和修改界面的设计细节。

游戏界 面既具有界面设计的一般规律，更有游戏这一特殊领域的个性设计原则。在界面的设计制作过程中，应在掌握基本游戏框架设定能力和对用户需求分析能力的基础上，了解并掌握游戏开发机制尤其是游戏界面的实现原理以及设计中常用的技术，例如:界面风格制定、美术需求文档制作、界面原型制作程序坐标图制作等技术。

如今外国游戏已经在中国市场占有很重要的地位，使得中国的文化环境逐渐改变，民族文化已经不能吸引|玩家，尤其近年来屡委的事件和媒体舆论的影响，已经让网络游戏成为大众眼中毒害青少年的精神毒品。而宣传中国传统文化，引导青少年健康游戏快乐游戏应该是每个游戏工作者的责任和义务。

1.1.2 开发技术和平台

**1.2：基于用例获取需求**

1.2.1确定参与者

游戏数据库系统，游戏玩家。

游戏玩家负责参与游戏的

1.2.2确定用例

与游戏玩家有关的用例

--参与游戏

--分享游戏成绩

与数据库系统有关的用例：

--添加用户数据

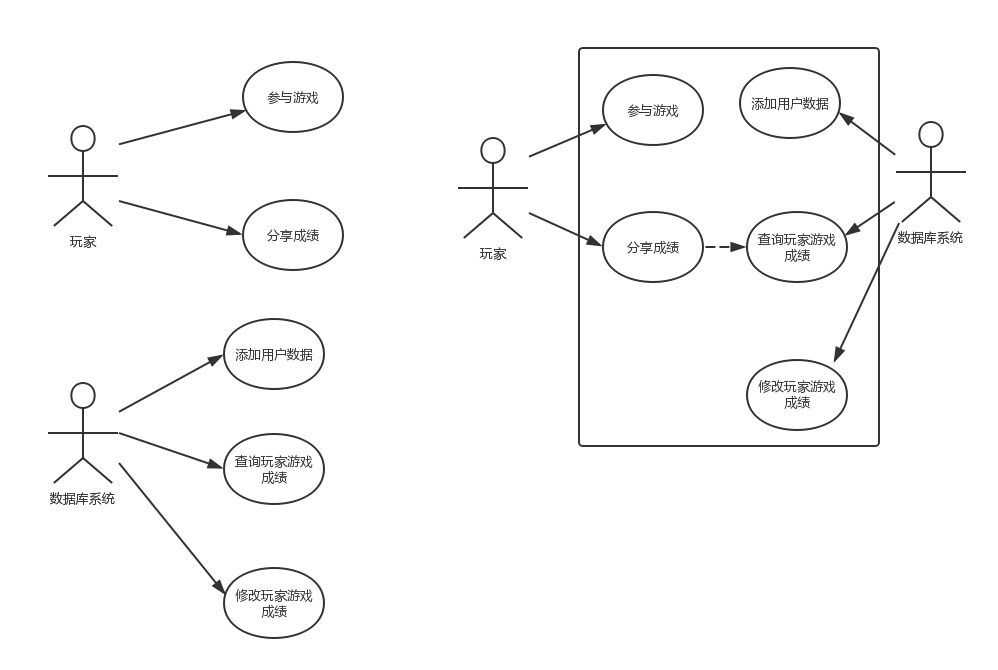
--查询玩家游戏成绩

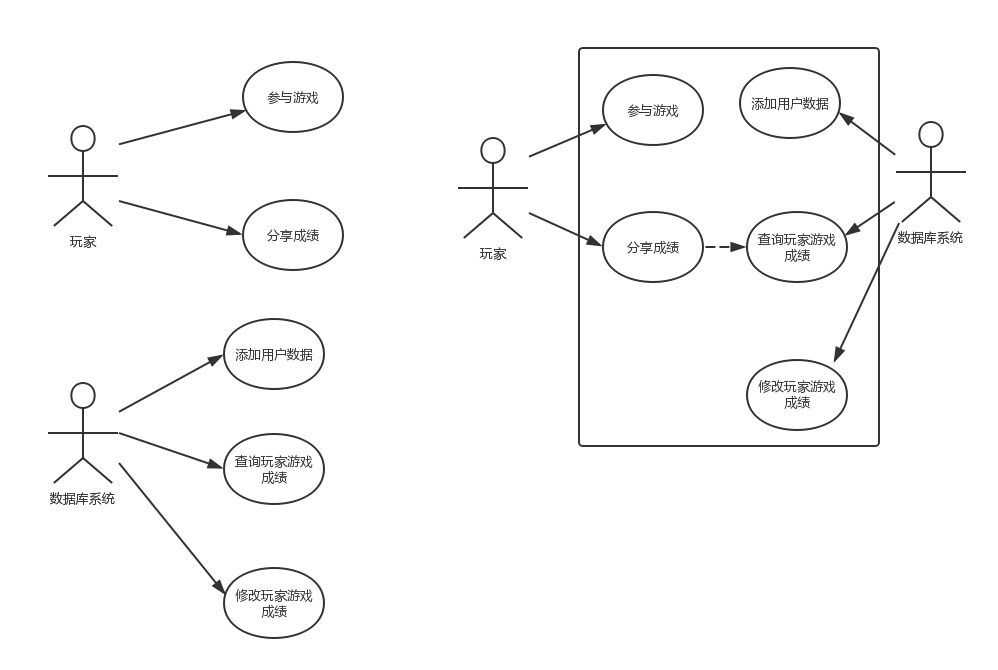
--修改玩家游戏成绩

--

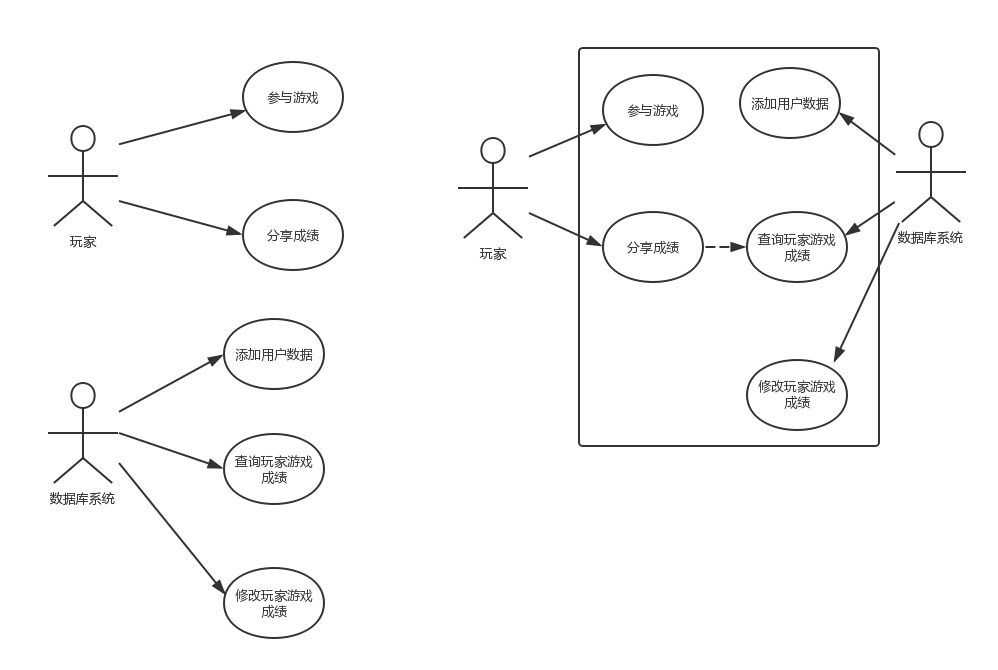
--

1.2.3构建用例模型





1.2.4用例模型的细化



1.2.5用例描述

参与游戏：玩家进行游戏过程

分享游戏成绩：玩家游戏结束后，可以将成绩分享至各个社交平台，显示前十名游戏成绩。

添加用户数据：新玩家参与游戏时自动添加玩家信息

查询游戏成绩：提供查询游戏成绩功能

修改玩家游戏成绩：

**1. 3：需求分析**

1.3.1识别边界类

3个开始界面 游戏界面 结束排行界面

1.3.2识别控制类

Transform rigidbody gamemanager collider renderer pillorcontroller ballcontroller CameraController NetworkCommunication UIManager

1.3.3识别实体类

Ball Plateform PieCell Pillor camera light PieCell AudioSource

**第二章 软件设计（撰写人:）**

**2.1： 总体设计**

2.1.1 体系结构设计

客户一服务器模式主要有以下几方面的优点：  
(1)**有利于实现资源共享。**网络中的资源具有分布不均匀性，各个不同结点之间的软硬件配置都存在很大差别。在C/S结构中的资源也是分布的，一般来说服务器在软硬件配置上或是数据资源分布上相对客户机而言都具有一定的优势，而且客户机与服务器具有一对多的关系和运行环境。用户不仅可存取在服务器和本地工作站上的资源，还可以享用其他工作站上的资源，实现了资源共享。  
(2)**有利于进程通信的同步**。分布式系统中的面临的一个重要的问题就是同步问题。在客户一服务器模型中，每一次通信由客户端进程发起请求，而服务器进程一直处于等待状态，以保证及时响应客户端发出的请求。当客户端发出请求后，服务器端响应客户端请求，并以此实现进程间的同步。  
(3)**可实现管理科学化和专业化**。系统中的资源分布在各服务器和工作站上，可以采用分层管理和专业化管理相结合的方式，用户有权去充分利用本部门、本领域的专业知识来参与管理，使得各级管理更加科学化和专业化。  
(4)**可快速进行信息处理**。由于在C/S结构中是一种基于点对点的运行环境，当一项任务提出请求处理时，可以在所有可能的服务器间均衡地分布该项任务的负载。这样，在客户端发出的请求可由多个服务器来并行进行处理，为每一项请求提供了极快的响应速度和较高的事务吞吐量。  
(5)**具有更好的可扩展性**。由于C/S是一种开放式的结构，因此可有效地保护原有的软、硬件资源。以前，在其他环境下积累的数据和软件均可在C/S中通过集成而保留使用，并且可以透明地访问多个异构的数据源和自由地选用不同厂家的数据应用开发工具，具有高度的灵活性;而以前的硬件也可完全继续使用，当在系统中增加硬件资源时，不会减弱系统的能力，同时客户机和服务器均可单独地升级，故具有极好的可扩展性。

客户一服务器双层体系结构存在以下几个局限性。  
(1)**缺乏有效的安全性**。由于客户端与服务器端直接相连，当在客户端存取一些敏感数据时，由于用户能够直接访问中心数据库，就可能造成敏感数据的修改或丢失。  
(2)**客户端负荷过重**。随着计算机处理的事务越来越复杂，客户端程序也日渐肥大。同时由于事务处理规则的变化，也需要随时更新客户端程序，就相应地增加了维护困难和工作量。  
(3)**服务器端工作效率低**。由于每个客户端都要直接连接到服务器以访问数据资源，这就使得服务器不得不因为客户端的访问建立连接而消耗大量本就十分紧张的服务器资源，从而造成服务器工作效率不高。  
(4)**容易造成网络阻塞**。正如前面所述，客户端的每次访问都要连接服务器，使得网络流量剧增，容易造成网络的阻塞。

Client3

Client2

Client1

计算机网络

database server

2.1.2 数据库设计

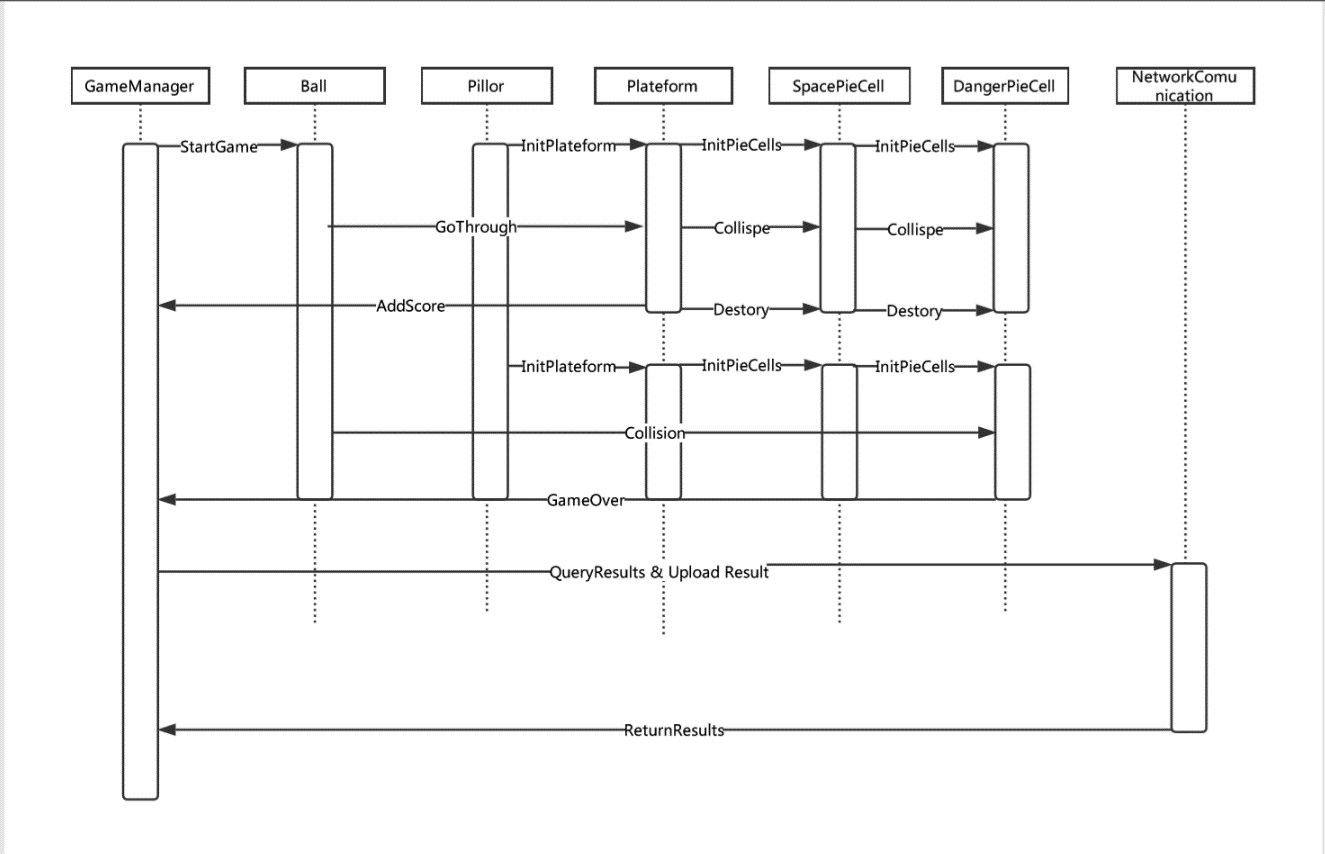
|  |  |  |
| --- | --- | --- |
| id |  |  |
| playername |  |  |
| playerscore |  |  |

2.1.3 识别设计类

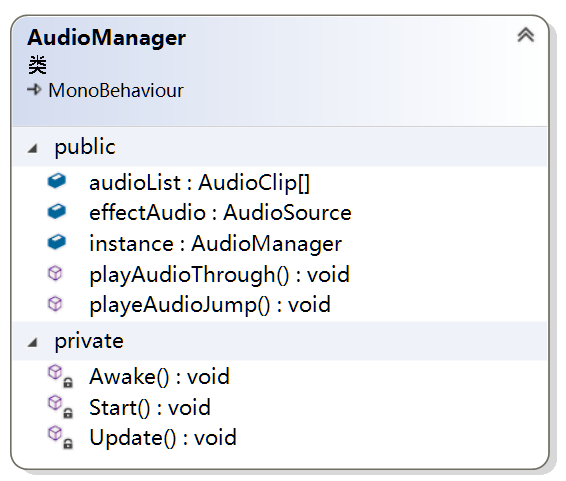
GameManager,Ball,Piller,Plateform,SpacePieCell,DangerPieCell,NetworkCommunication,UIManager,AudioManager

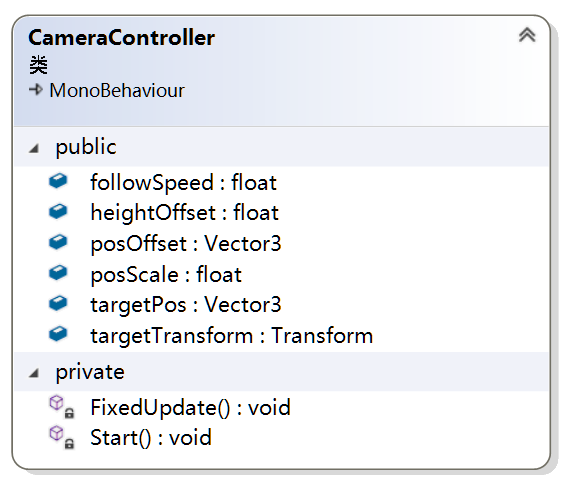
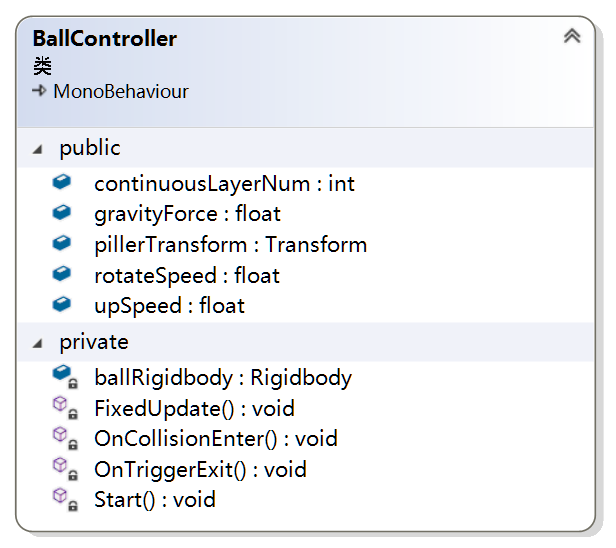
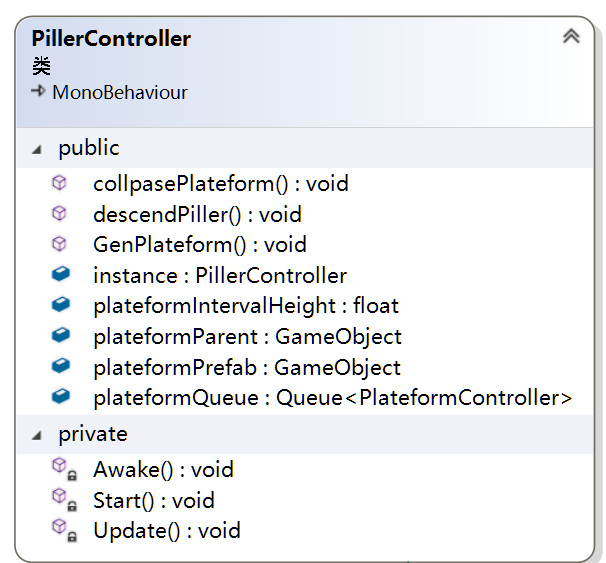
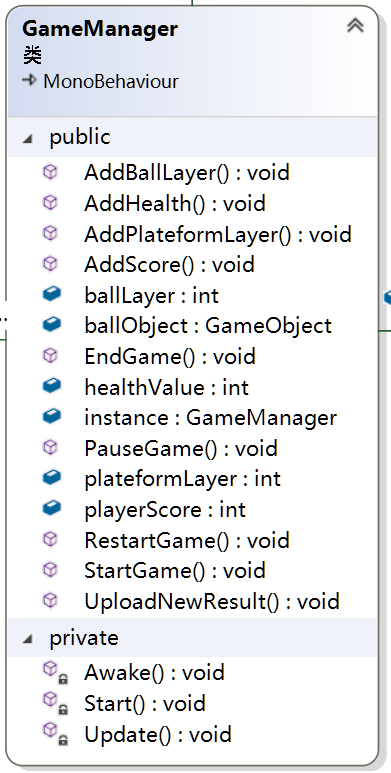
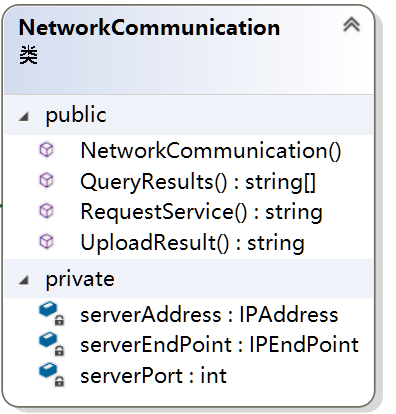
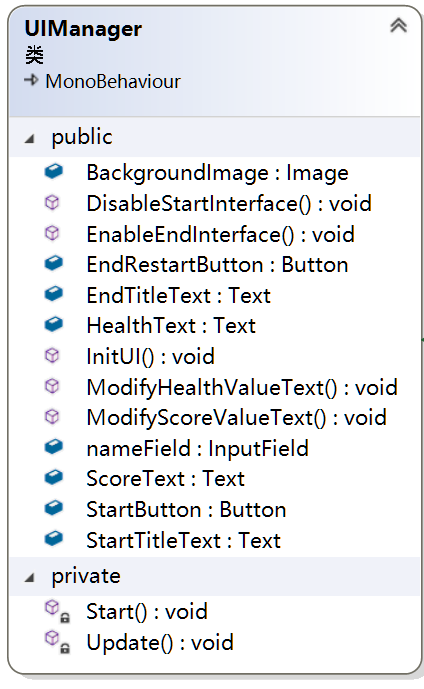
**2.2 详细设计**

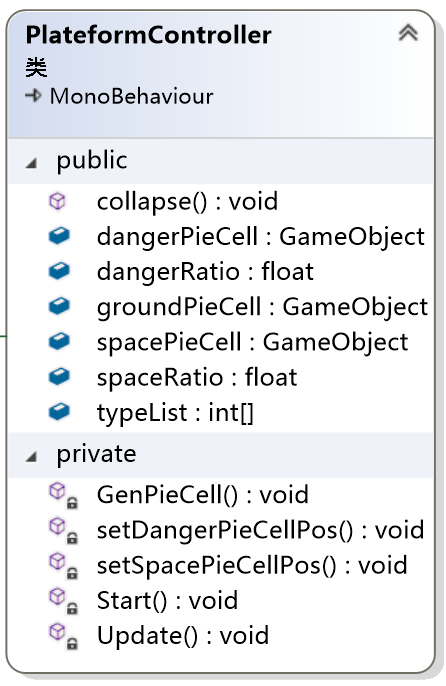
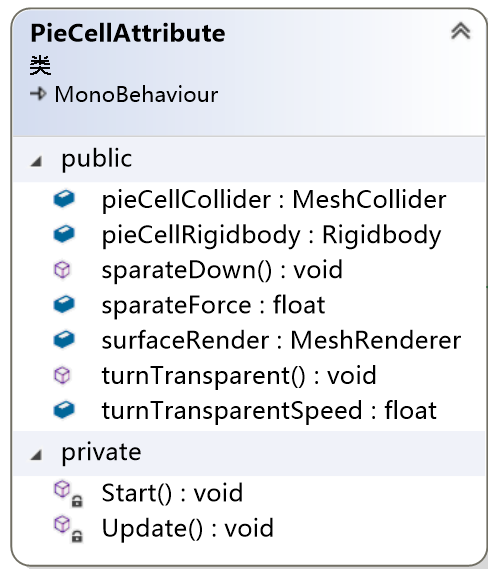
2.2.1 识别设计类的方法



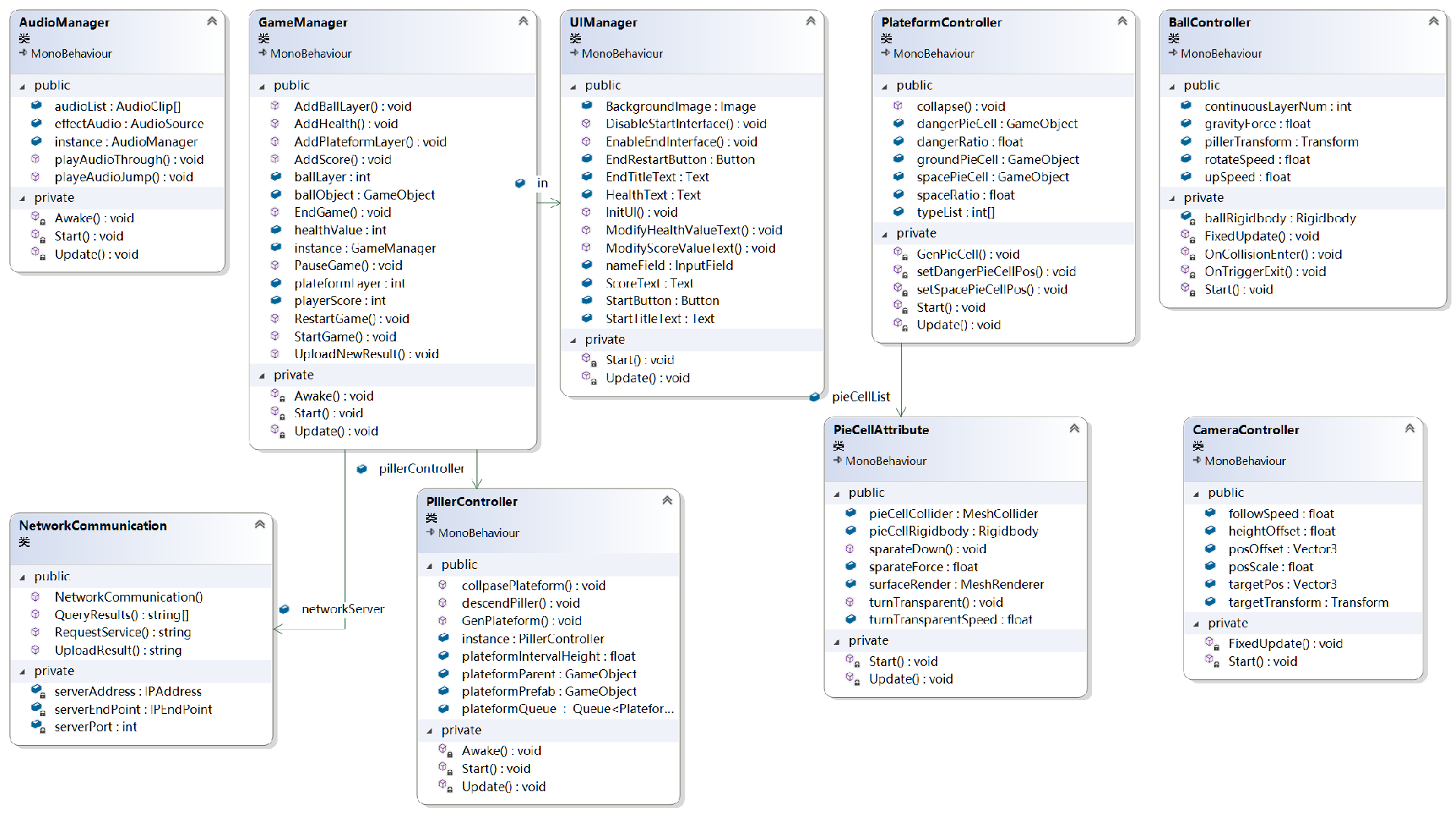
2.2.2 识别设计类的属性







2.2.3 设计模型



**第三章 软件实现（撰写人:）**

**3.1 界面实现**

**3.2 关键代码**

**AudioManager.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class AudioManager : MonoBehaviour {

public static AudioManager instance;

public AudioSource effectAudio;

public AudioClip [] audioList;

private void Awake() {

instance = this;

}

// Use this for initialization

void Start () {

}

// Update is called once per frame

void Update () {

}

public void playeAudioJump() {

float clipPitch = Random.Range(0.8f, 1.2f);

effectAudio.clip = audioList[0];

effectAudio.volume = 0.3f;

effectAudio.pitch = clipPitch;

effectAudio.Play();

}

public void playAudioThrough() {

float clipPitch = Random.Range(0.8f, 1.2f);

effectAudio.clip = audioList[1];

effectAudio.pitch = clipPitch;

effectAudio.volume = 1f;

effectAudio.Play();

}

}

**BallControlller.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class BallController : MonoBehaviour {

public float upSpeed;

public float gravityForce;

public float rotateSpeed;

public int continuousLayerNum;

public Transform pillerTransform;

private Rigidbody ballRigidbody;

// Use this for initialization

void Start () {

ballRigidbody = this.GetComponent<Rigidbody>();

}

// Update is called once per frame

void FixedUpdate () {

ballRigidbody.AddForce(new Vector3(0, -1 \* gravityForce, 0));

float rotateDir = 0;

if (Input.GetKey(KeyCode.LeftArrow)) {

rotateDir = 1;

} else if (Input.GetKey(KeyCode.RightArrow)) {

rotateDir = -1;

}

this.transform.RotateAround(pillerTransform.position, Vector3.up, rotateDir \* rotateSpeed \* Time.deltaTime);

}

private void OnCollisionEnter(Collision collision) {

if (collision.collider.tag == "Ground" && this.transform.position.y >= collision.transform.position.y) {

ballRigidbody.velocity = new Vector3(0, upSpeed, 0);

AudioManager.instance.playeAudioJump();

if (continuousLayerNum >= 3) {

GameManager.instance.AddScore(10);

PillerController.instance.GenPlateform();

PillerController.instance.collpasePlateform();

PillerController.instance.descendPiller();

}

continuousLayerNum = 0;

} else if (collision.collider.tag == "Danger") {

ballRigidbody.velocity = new Vector3(0, upSpeed, 0);

AudioManager.instance.playeAudioJump();

if (continuousLayerNum >= 3) {

GameManager.instance.AddScore(10);

PillerController.instance.GenPlateform();

PillerController.instance.collpasePlateform();

PillerController.instance.descendPiller();

continuousLayerNum = 0;

return;

}

GameManager.instance.AddHealth(-1);

if (GameManager.instance.healthValue <= 0) {

GameManager.instance.EndGame();

}

}

}

private void OnTriggerExit(Collider other) {

if (other.tag == "Space" && this.transform.position.y < other.transform.position.y) {

GameManager.instance.ballLayer += 1;

GameManager.instance.AddScore();

AudioManager.instance.playAudioThrough();

PillerController.instance.GenPlateform();

PillerController.instance.collpasePlateform();

PillerController.instance.descendPiller();

continuousLayerNum++;

}

}

}

**CameraController.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class CameraController : MonoBehaviour {

public Transform targetTransform;

public Vector3 posOffset;

public float heightOffset;

public float posScale;

public Vector3 targetPos;

public float followSpeed;

// Use this for initialization

void Start () {

posOffset = this.transform.position - targetTransform.position;

posScale = this.transform.position.z / targetTransform.position.z;

heightOffset = posOffset.y;

}

// Update is called once per frame

void FixedUpdate () {

//targetPos = new Vector3(posScale \* targetTransform.position.x,GameManager.instance.ballLayer \* -10, posScale \* targetTransform.position.z);

//this.transform.position = Vector3.Lerp(this.transform.position, targetPos, followSpeed \* Time.deltaTime);

targetPos = new Vector3(posScale \* targetTransform.position.x, targetTransform.position.y + heightOffset, posScale \* targetTransform.position.z);

this.transform.position = targetPos;

this.transform.LookAt(GameManager.instance.ballObject.transform);

//Quaternion tmpQuater = Quaternion.identity;

//tmpQuater.x = 60;

//tmpQuater.y = targetTransform.rotation.y;

//this.transform.rotation = tmpQuater;

}

}

**GameManager.cs**

using System;

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.SceneManagement;

public class GameManager : MonoBehaviour {

static public GameManager instance;

public int playerScore;

public int plateformLayer;

public int ballLayer;

public int healthValue;

public GameObject ballObject;

public PillerController pillerController;

public NetworkCommunication networkServer;

public UIManager interfaceManager;

private void Awake() {

instance = this;

playerScore = 0;

plateformLayer = 0;

ballLayer = 0;

healthValue = 2;

networkServer = new NetworkCommunication("127.0.0.1", 8829);

interfaceManager.InitUI();

}

// Use this for initialization

void Start() {

for (int i = 0; i < 10; ++i) {

pillerController.GenPlateform();

}

PauseGame();

}

// Update is called once per frame

void Update() {

}

public void AddPlateformLayer() {

plateformLayer++;

}

public void AddBallLayer() {

ballLayer++;

}

public void AddScore(int \_s = 1) {

playerScore += \_s;

interfaceManager.ModifyScoreValueText(playerScore);

}

public void AddHealth(int \_h = 1) {

healthValue += \_h;

interfaceManager.ModifyHealthValueText(healthValue);

}

public void PauseGame() {

Time.timeScale = 0;

}

public void StartGame() {

Time.timeScale = 1;

interfaceManager.DisableStartInterface();

AddHealth(0);

AddScore(0);

}

public void EndGame() {

Time.timeScale = 0;

interfaceManager.EnableEndInterface();

String[] results = networkServer.QueryResults();

for(int i =0;i < results.Length-1;++i) {

interfaceManager.EndTitleText.text += ((i+1).ToString() + " : " + results[i] + '\n');

}

}

public void RestartGame() {

UploadNewResult();

SceneManager.LoadScene(0);

}

public void UploadNewResult() {

string playerName = interfaceManager.nameField.text;

if(playerName != "") {

networkServer.UploadResult(playerName, playerScore);

}

}

}

**NetworkCommunications.cs**

using System;

using System.Collections;

using System.Collections.Generic;

using System.Net;

using System.Net.Sockets;

using System.Text;

using UnityEngine;

public class NetworkCommunication{

IPAddress serverAddress;

int serverPort;

IPEndPoint serverEndPoint;

public NetworkCommunication(String \_ip, int \_port) {

serverPort = \_port;

serverAddress = IPAddress.Parse(\_ip);

serverEndPoint = new IPEndPoint(serverAddress, serverPort);

}

public String RequestService(String \_r) {

try {

Socket clientRequestSocket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

clientRequestSocket.Connect(serverEndPoint);

clientRequestSocket.Send(Encoding.UTF8.GetBytes(\_r));

byte[] buffer = new byte[1024 \* 1024];

int msgLen = clientRequestSocket.Receive(buffer);

String recMsg = Encoding.UTF8.GetString(buffer, 0, msgLen);

clientRequestSocket.Close();

return recMsg;

} catch (Exception e) {

Console.WriteLine("Exception : " + e);

return e.ToString();

}

}

public String [] QueryResults() {

String msg = "QueryResults";

String[] msg\_para = RequestService(msg).Split(':');

String[] results = new String[2];

if (msg\_para.Length > 1) {

results = msg\_para[1].Split('|');

for (int i = 0; i < results.Length; ++i) {

results[i] = results[i].Replace("(", "").Replace(")", "").Replace("'", "");

}

}

return results;

}

public String UploadResult(String \_n,int \_s) {

Debug.Log("Upload Result");

String msg = "UploadResult:" + \_n + ":" + \_s;

return RequestService(msg);

}

}

**PieCellAttribute.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class PieCellAttribute : MonoBehaviour {

public float turnTransparentSpeed;

public float sparateForce;

public MeshRenderer surfaceRender;

public MeshCollider pieCellCollider;

public Rigidbody pieCellRigidbody;

// Use this for initialization

void Start () {

surfaceRender = this.GetComponentInChildren<MeshRenderer>();

pieCellRigidbody = this.GetComponent<Rigidbody>();

pieCellCollider = this.GetComponent<MeshCollider>();

pieCellRigidbody.isKinematic = true;

}

}

// Update is called once per frame

void Update () {

}

public void turnTransparent() {

surfaceRender.material.color = Color.Lerp(surfaceRender.material.color, new Color(0, 0, 0, 0), 10 \* Time.deltaTime);

public void sparateDown() {

Vector3 tmpOffset = this.transform.position - GameManager.instance.ballObject.transform.position;

tmpOffset.y = 2;

pieCellRigidbody.isKinematic = false;

pieCellRigidbody.AddForce(sparateForce \* tmpOffset);

//pieCellCollider.isTrigger = true;

if (this.GetComponent<MeshCollider>().tag == "Space") {

Destroy(this.gameObject);

} else {

Destroy(this.gameObject, 0.5f);

}

this.GetComponent<MeshCollider>().tag = "PieCell";

}

}

**PillerController.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class PillerController : MonoBehaviour {

public static PillerController instance;

public float plateformIntervalHeight;

public GameObject plateformPrefab;

public GameObject plateformParent;

public Queue<PlateformController> plateformQueue = new Queue<PlateformController>();

private void Awake() {

instance = this;

}

// Use this for initialization

void Start() {

}

// Update is called once per frame

void Update() {

}

public void GenPlateform() {

Vector3 tmpPos = new Vector3(0, -1 \* plateformIntervalHeight \* GameManager.instance.plateformLayer, 0);

GameObject tmpPlateform = GameObject.Instantiate(plateformPrefab, tmpPos, Quaternion.identity);

tmpPlateform.transform.SetParent(GameManager.instance.transform);

plateformQueue.Enqueue(tmpPlateform.GetComponent<PlateformController>());

GameManager.instance.AddPlateformLayer();

}

public void collpasePlateform() {

PlateformController tmpPlate = plateformQueue.Dequeue();

tmpPlate.collapse();

}

public void descendPiller() {

Vector3 tmpPos = this.transform.position;

tmpPos.y -= plateformIntervalHeight;

this.transform.position = tmpPos;

}

}

**PlateformController.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class PlateformController : MonoBehaviour {

public PieCellAttribute[] pieCellList = new PieCellAttribute[12];

public int[] typeList = new int[12];

public GameObject spacePieCell;

public GameObject groundPieCell;

public GameObject dangerPieCell;

public float spaceRatio;

public float dangerRatio;

// Use this for initialization

void Start () {

spaceRatio = (float)(10 - Mathf.Max(Mathf.Log10(GameManager.instance.plateformLayer))) / (5 + GameManager.instance.plateformLayer);

dangerRatio = Mathf.Min((Mathf.Log10(GameManager.instance.plateformLayer)), 0.5f);

GenPieCell();

}

// Update is called once per frame

void Update () {

}

void setSpacePieCellPos() {

//Random Space Position

int spaceCounter = 0;

for(int tmp\_i = 0; tmp\_i < typeList.Length;++tmp\_i) {

if (Random.Range(0f, 1f) < spaceRatio) {

typeList[tmp\_i] = 1;

++spaceCounter;

}

}

if (spaceCounter == 0) {

int tmpN = Random.Range(0, 12);

typeList[tmpN] = 1;

}

}

void setDangerPieCellPos() {

//Random Danger Position

bool isDanger = Random.Range(0f, 1f) < dangerRatio;

if (isDanger) {

for (int tmp\_i = 0; tmp\_i < typeList.Length; ++tmp\_i) {

if (typeList[tmp\_i]==0 && Random.Range(0f,1f) < dangerRatio) {

typeList[tmp\_i] = 2;

}

}

}

}

void GenPieCell() {

setSpacePieCellPos();

setDangerPieCellPos();

for (int i = 0; i < 12; ++i) {

GameObject tmpPieCell =null;

if (typeList[i] == 0) {

tmpPieCell = GameObject.Instantiate(groundPieCell, this.transform.position, Quaternion.Euler(90, i \* 30, 0));

} else if (typeList[i] == 1) {

tmpPieCell = GameObject.Instantiate(spacePieCell, this.transform.position, Quaternion.Euler(90, i \* 30, 0));

} else if (typeList[i] == 2) {

tmpPieCell = GameObject.Instantiate(dangerPieCell, this.transform.position, Quaternion.Euler(90, i \* 30, 0));

}

tmpPieCell.transform.SetParent(this.transform);

pieCellList[i] = tmpPieCell.GetComponent<PieCellAttribute>();

}

}

public void collapse() {

for (int i = 0;i < 12;++i) {

pieCellList[i].sparateDown();

}

Destroy(this.gameObject, 0.5f);

}

}

**UIManager.cs**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class UIManager : MonoBehaviour {

public Image BackgroundImage;

public Button StartButton;

public Text StartTitleText;

public Text EndTitleText;

public Button EndRestartButton;

public Text ScoreText;

public Text HealthText;

public InputField nameField;

// Use this for initialization

void Start () {

}

// Update is called once per frame

void Update () {

}

public void InitUI() {

BackgroundImage.gameObject.SetActive(true);

StartButton.gameObject.SetActive(true);

StartTitleText.gameObject.SetActive(true);

ScoreText.gameObject.SetActive(false);

HealthText.gameObject.SetActive(false);

EndRestartButton.gameObject.SetActive(false);

EndTitleText.gameObject.SetActive(false);

nameField.gameObject.SetActive(false);

}

public void DisableStartInterface() {

BackgroundImage.gameObject.SetActive(false);

StartButton.gameObject.SetActive(false);

StartTitleText.gameObject.SetActive(false);

ScoreText.gameObject.SetActive(true);

HealthText.gameObject.SetActive(true);

}

public void EnableEndInterface() {

BackgroundImage.gameObject.SetActive(true);

EndRestartButton.gameObject.SetActive(true);

EndTitleText.gameObject.SetActive(true);

nameField.gameObject.SetActive(true);

}

public void ModifyHealthValueText(int \_num) {

HealthText.text = "Health : " + \_num.ToString();

}

public void ModifyScoreValueText(int \_num) {

ScoreText.text = "Score : " + \_num.ToString();

}

}

**第四章 软件测试（撰写人:）**

**4.1 白盒测试**

（利用基本路径法设计测试用例，测试3.2关键代码中至少三个主要方法）

白盒测试（white-box testing）又称透明盒测试（glass box testing）、结构测试（structural testing）等，软件测试的主要方法之一，也称结构测试、逻辑驱动测试或基于程序本身的测试。测试应用程序的内部结构或运作，而不是测试应用程序的功能（即黑盒测试）。在白盒测试时，以编程语言的角度来设计测试案例。测试者输入数据验证数据流在程序中的流动路径，并确定适当的输出，类似测试电路中的节点。测试者了解待测试程序的内部结构、算法等信息，这是从程序设计者的角度对程序进行的测试。

白盒测试可以应用于单元测试（unit testing）、集成测试（integration testing）和系统的软件测试流程，可测试在集成过程中每一单元之间的路径，或者主系统跟子系统中的测试。尽管这种测试的方法可以发现许多的错误或问题，它可能无法检测未使用部分的规范。

白盒测试设计技术包括以下代码覆盖标准：

控制流测试

数据流测试

分支测试

语句覆盖

判定覆盖

修正条件/判定覆盖

主要路径测试

路径测试

优点

白盒测试是当今使用的两个最大的测试方法之一。 它有几大优势：

在测试过程中掌握有关源代码的知识是有益的。

通过揭示隐藏的错误进行的代码优化，可以消除可能存在的缺陷。

开发人员需仔细地进行接下来的开发，白盒测试可以为程序员提供反馈。

从源代码层面测试提供了可追溯性，简化了将来软件改动带来的测试改动。

白盒测试容易实现自动化。

关于测试的停止时间，白盒测试给出了明确的工程学上的规定。

缺点

尽管白盒测试具有很大的优势，它并不完美，并包含一些缺点：

白盒测试复杂，因为测试员必须有编程知识，算得上是一个程序员。根据测试层面的复杂性，白盒测试需要知识水平更高的程序员。

在某些情况下，要测试程序中的所有可能情况是不现实的，因此会有一些未被测试的情况。

白盒测试着眼于以存的软件，可能无法发现遗漏的功能。

**4.2 黑盒测试**

（利用等价类划分法设计测试用例，测试系统三个功能）

黑盒测试，软件测试的主要方法之一，也可以称为功能测试、数据驱动测试或基于规格说明的测试。测试者不了解程序的内部情况，不需具备应用程序的代码、内部结构和编程语言的专门知识。只知道程序的输入、输出和系统的功能，这是从用户的角度针对软件界面、功能及外部结构进行测试，而不考虑程序内部逻辑结构。测试案例是依应用系统应该做的功能，照规范、规格或要求等设计。测试者选择有效输入和无效输入来验证是否正确的输出。

此测试方法可适合大部分的软件测试，如集成测试（integration testing）以及系统测试（system testing）。

黑盒测试的优点有 ：

　　　　　　　1) 比较简单，不需要了解程序的内部的代码及实现

2) 与软件的内部实现无关

3) 从用户的角度出发，能很容易的知道用户会用到哪些功能，会遇到哪些问题

4) 基于软件开发文档，所以也能知道软件实现了文档中的哪些功能

5) 在做软件自动化测试时较为方便

缺点 ：

　　　　　　　1) 不可能覆盖所有的代码， 覆盖率较低，大概只能达到总代码量的30%

2) 自动化测试的复用性较低

**4.2.1** 测试用例说明

* 对于DangerPie的生成几率的范围是[0~0.5]
  + 其有效等价类为 [0~0.5]
  + 其无效等价类为[-∞,0] (0.5~+∞)
  + 测试用例输入值[0,0.1,0.2,0.5][-1,-100,0.51,0.6,1,100]
* 对于SpacePie的生成几率的范围是[0~0.98]
  + 其有效等价类为[0~0.98]
  + 其无效等价类为[-∞,0] (0.98~+∞)
  + 测试用例输入值[0,0.1,0.2,0.98][-1,-100,0.99,1,100]
* 对于球类碰撞函数的输入值为Collision
  + 其有效等价类为 Ground,Danger,Space
  + 其无效等价类为Pillor
  + 测试用例输入 Ground,Danger,Space,Pillor
* 对于用户键盘输入的输入值为[-1,1]
  + 其有效等价类为[-1,1]
  + 其无效等价类为其无效等价类为[-∞,-1] [1~+∞)
  + 测试用例输入[-1,-0,8,-0,1,0,0.1,0.8,1][-1.1,-10,-100,1.1,10,100]
* 测试上传数据的函数,输入值为[String,int]
  + 其有效等价类为[任意字符串,[0,100]]或空
  + 其无效等价类为[任意字符串,[-∞,0)]
  + 测试用例输入[czlsez,0],[czlwez,1],[czldsb,100],[wzlwez,-1],[wzlsez,-100]