武汉大学计算机学院

本科生课程设计报告

BallGame 游戏总体设计与实现

专 业 名 称 ：计算机科学与技术

课 程 名 称 ：大型应用软件课程设计

团 队 名 称 ：啊对对对

指 导 教 师 ：马于涛

团 队 成 员 ：晋硕(组长)2018302110280

姜苏庭 2018302110208

廖梦恬 2018302110298

周华丽 2018302110167

钟桔爱 2018302110267

沈之雯 2018302110272

二〇二一年十一月

郑 重 声 明

本团队呈交的设计报告，是在指导老师的指导下，独立进行实验工作所取得的成果，所有数据、图片资料真实可靠。尽我所知，除文中已经注明引用的内容外，本设计报告不包含他人享有著作权的内容。对本设计报告做出贡献的其他个人和集体，均已在文中以明确的方式标明。本设计报告的知识产权归属于培养单位。

团队成员签名：晋硕、姜苏庭、廖梦恬、周华丽、钟桔爱、沈之雯

日期：2021 年 11 月 9 日星期二

摘　要

本次大型应用软件课程设计实验中，我们小组开发一个 Android 平台的战斗冒险类游戏 BallGame，以红球、灰球、黑球分别代表正能量、普通能量、负能量，以及它们之间的相互作用来模拟现实生活中人们受周围同伴情绪感染这一现象，使得用户在玩这款游戏的过程中解压。

本次实验我们小组主要是基于 Unity 游戏引擎，在 Visual Studio 平台上使用 C#语言开发一款 Android 端游戏。

本文档包括项目需求分析、系统设计说明、测试流程以及团队成员的个人心得体会等等。

关键词：Android 开发，Unity，集群算法，解压

目录

一、 项目信息与团队分工

1.1 项目地址

Github 地址：https://github.com/sakurajst/BallGame

1.2 团队分工

姜苏庭：

编码：完成地图的初始化、镜头的移动、摇杆控制主角的移动、灰球和黑球寻路的移动；

渲染：小球和地图的材质、主角和 boss 的环绕特效、主角合体的特效、画面色调的变化、屏幕后处理（HDR 和 bloom，ssao）。

晋硕：

编码：实现 UI 的实时更新，包括进度条、比例条、小地图；实现 UI 的自适配；

美术资源设计：负责所有 UI 素材的绘制，包括游戏界面的各种小图标、开始界面按钮以及 logo。

廖梦恬：

编码：完成游戏 123 阶段的过渡、游戏的胜利和失败、小球的生成和销毁、 合体的逻辑、小红球的集群算法移动逻辑、小球的碰撞逻辑、boss 的行为。

周华丽：

充当用户：提出游戏构想及需求；

文档：完成需求规格说明书的引言及系统需求概述部分；绘制类图并对其进行说明；整理和维护项目文档；制作 PPT；实验报告的第四部分项目实现；实验报告的整合与修改

钟桔爱：

充当用户：提出游戏构想及需求；

文档：完成需求规格说明书的运行环境规定、验收标准、迭代 I 阶段分工部分；绘制活动图及游戏 3 阶段的说明；实验报告的第五部分测试报告以

及实验报告的整合

沈之雯：

充当用户：提出游戏构想及需求；

文档：完成需求规格说明书的功能需求及非功能需求部分；绘制时序图

二、 需求规格说明书

2.1 引言

2.1.1 编写目的

本文档的目的是详细地描述 BallGame 游戏项目的功能性需求和非功能性需求，预期读者有用户、项目经理、开发人员以及测试人员。以便用户能够了解预期的功能和性能，并对整个需求进行讨论和协商；项目经理能够了解预期产品的功能，并据此进行系统设计和项目管理；开发人员能够对需求进行分析，并进行系统和功能设计，根据需求进行编码，实现并优化软件功能；测试人员能够根据本文档编写测试用例及测试文档，并对软件产品进行功能性和非功能性测试。

以下叙述将结合文字描述、UML 图等来描述 BallGame 的功能、性能、用户界面、运行环境、外部接口以及针对用户操作给出的各种响应。

2.1.2 项目介绍

（1）项目背景

都市中生活的人们总有或多或少的烦恼、压力、抑郁。而这款游戏模拟的正是一个充满正能量的你，身边充斥着负能量的人们，你需要接触他们，用你的正能量感染他们，他们也会用被你激发的正能量去鼓励下一个对生活苦恼着的人。有时总会有这样的一些人在你身边——浑浑噩噩、丧、自暴自弃。当你觉得你的力量不足以改变他们时，适当地远离他们能使你更加积极地面对生活。当你觉得你的力量足够影响他们，那就勇敢地靠近，唤醒他们沉睡的灵魂。

丧文化盛极一时，只有正能量能改变灰色的城市。当人群中积蓄的能量足够强大时，所谓“丧”自然会被感染，阴霾也会散去，城市也会恢复阳光灿烂的模样。所以我们计划开发一个 Android 平台的战斗冒险类游戏 BallGame，以红球、灰

球、黑球分别代表正能量、普通能量、负能量，以及它们之间的相互作用来模拟现实生活中人们受周围同伴情绪感染这一现象。

（2）项目说明

明确 BallGame 游戏中各项功能和非功能需求, 确定系统功能模块, 同时为系统设计人员提供设计依据, 也可供本项目的其他开发人员参阅. 本需求分析报告的目的是规范化本软件的编写, 旨在于提高软件开发过程中的能见度, 便于对软件开发过程中的控制与管理, 同时提出了 BallGame 游戏开发过程, 便于程序员与客户之间的交流与协作, 并作为工作成果的原始依据, 同时也表明了本软件的共性, 以期能够获得更大范围的应用.

（3）玩法介绍：

第一阶段，开始游戏，先初始化游戏地图，创建主角、小红球、小灰球和小黑球，这时各类球体都是处于自由状态，通过控制摇杆来控制主角的移动状态，引导碰撞灰球、躲避小黑球和黑球，指示 UI 上实时更新显示的红灰黑球比例。本阶段如果主角碰撞到小黑球，主角就会死亡，游戏失败。如果主角碰撞小灰球，小灰球变成小红球，当场上小红球达到一定数量，就可以按下合体键，第一次按下合体键，游戏就会进入第二阶段。

第二阶段开始时，主角处于合体状态， boss（大黑球）出现，但是合体后的主角体积依然小于 boss。地图上会随机出现道具（主角加速和主角冷却时间归零），主角移动碰到道具，就会即时的获得道具的功能。如果非合体的主角碰到 boss 就会死亡，游戏失败。如果合体的主角碰到 boss，主角会立刻解体，解体后的小红球会部分变成小黑球，小红球数量减少。当主角碰撞小灰球使得小红球数量增多且合体后的主角体积大于 boss 时进入第三阶段。

第三阶段，合体状态的主角体积大于 boss 的体积，主角可以直接碰撞 boss，而且每碰撞一次，boss 会失去一次生命，缩小一定体积，主角也解体。合体主角碰撞达到三次，boss 死亡，地图上的全部球体变红，游戏胜利。

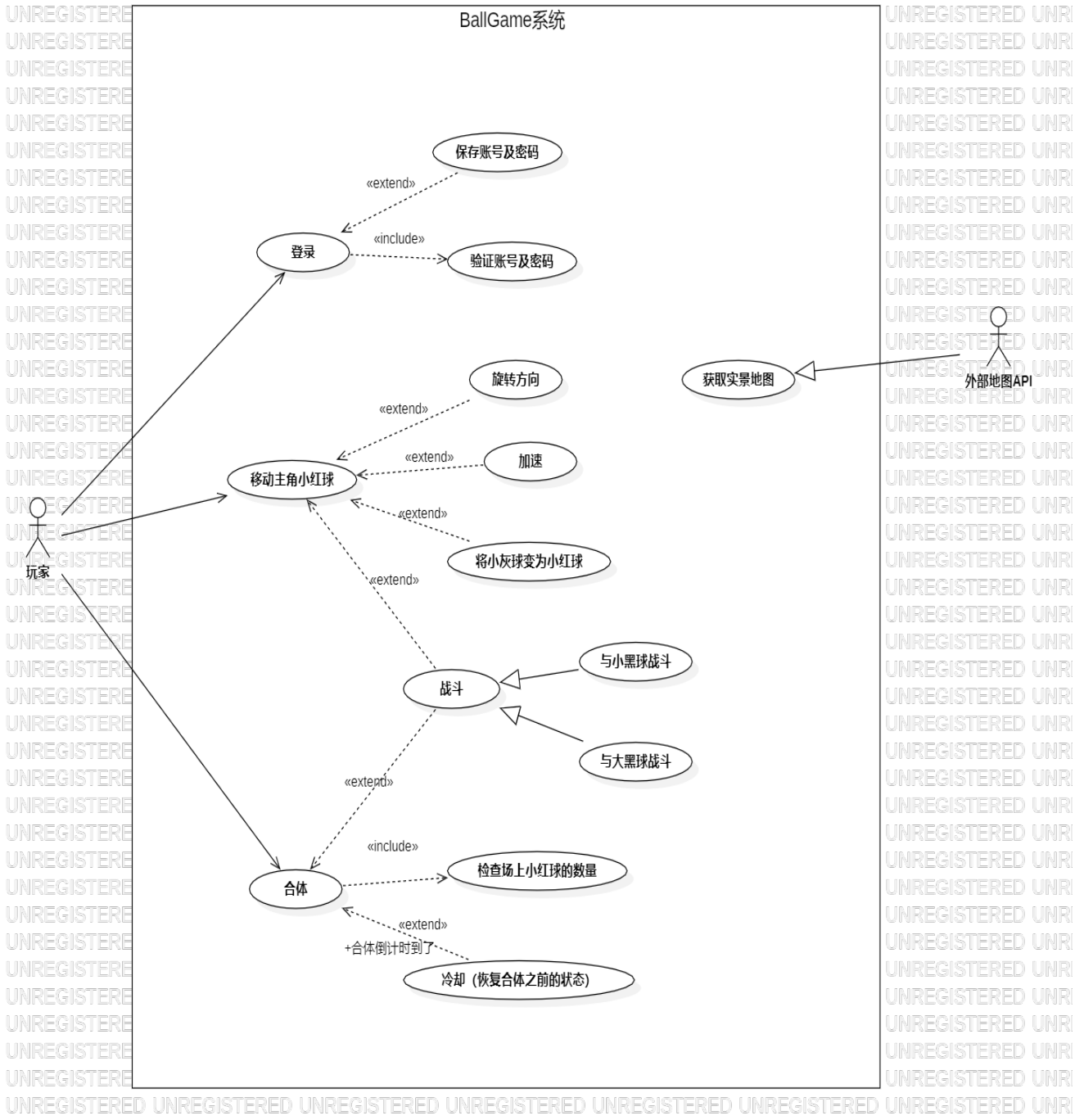
（4）名称定义

名称	定义
玩家	对主角小红球进行控制操作
主角小红球	游戏内的主角，比普通小红球多了一个合体的技能

小红球	除主角小红球外其他的普通小红球，可以将小灰球变为小红球，也会受小黑球影响变为小灰球或者小黑球
小灰球	中立角色，可以受小红球影响变为小红球
小黑球	攻击角色，可以将小红球变为小灰球或小黑球
大黑球	可以将其攻击范围内的小红球全部变为小灰球，将范围内的小灰球全部变为小黑球

2.2 系统需求概述

2.2.1 游戏系统的用例图



2.2.2 用例的概要描述

主要参与者	优先级	用例名称	用例概述
外部地图 API	高	获取实景地图	系统通过外部地图 API 获得玩家当前所处的实时地图场景并据此在游戏场景中进行重现

玩家	低	登录	玩家登录游戏账号，也可以在不登录的状态下直接游戏
玩家	中	移动主角小红球	玩家操作主角小红球的移动
玩家	高	旋转方向	旋转主角小红球的移动方向
玩家	高	加速	加速主角小红球的移动速度
玩家	中	将小灰球变为小红球	主角小红球将小灰球变为小红球
玩家	中	战斗	小红球与黑球战斗
玩家	中	与小黑球战斗	小红球会在小黑球的作用下，变为小灰球或者小黑球
玩家	高	与大黑球战斗	进入大黑球的攻击范围之后，主角小红球与大黑球进行战斗
玩家	高	合体	在场上小红球数量达到一定数量且玩家按下合体键后，主角小红球与普通小红球合体，提及增大，持续 n1 时间
	高	检查场上小红球的数量	判断场上小红球数量是否达到允许合体的预置数量
	高	冷却（恢复合体之前的状态）	在合体时间结束的时候，进入冷却阶段，即合体后的体积增大的红球重新分解冷却为原主角小红球和普通小红球

2.2.3 合体用例的详细描述

用况名	合体
描述	当场上小红球达到一定数量，主角可以选择按下合体键与其他小红球合体增大体积，若进入大黑球攻击范围就与之战斗
参与者	玩家
包含	检查小红球是否达到预置数量
扩展	如果合体后小红球进入大黑球攻击范围内，则进入战斗用况
泛化	无
前置条件	①玩家按下合体键； ②场上小红球达到所要求的数量
细节	（1）基本流：玩家按下合体键，系统检查场上小红球是否已经达到所要求的数量，如果满足要求，主角小红球与其他小红球合体，体积增大，该用况结束； （2）可选流：玩家按下合体键，但是场上小红球数量尚未满足要求，不执行合体操作 （3）可选流：场上小红球满足数量，玩家尚未按下合体键，不执行合体操作 （4）可选流：玩家按键时，红球已经是合体状态或处于上一次合体后的冷却状态，不执行合体操作
后置条件	合体成功，主角小红球体积增大
例外	① 玩家按下合体键且场上小红球数量足够，但系统异常无响应； ② 场上小红球数量不足；

	③ 场上小红球数量足够，但是玩家尚未按下合体键
限制	① 在红球处于合体状态或者合体成功后的冷却状态时按下合体键无效； ② 合体后的 n 秒内不碰撞会自动解体
注释	无

2.2.4 假设和依赖

- 运行游戏的设备需要联网以获得实景地图以提供游戏场景。
- 必须使用智能化的触摸屏的 Android 设备。

2.3 软件功能需求

（1）**游戏开始界面**

玩家进入游戏。

（2）**红球移动**

玩家通过控制手机轮盘控制红球移动方向。

（3）**拾取道具**

玩家控制红球移动到道具出现地点拾取道具。

（4）**使用道具**

玩家在拾取道具后即刻使用，获取加速或者等效果。

（5）**获取能量**

玩家控制红球移动到地图上灰球位置，使得灰球变成小红球围绕在红球周围，增加能量。

（6）**合体**

当能量达到要求后，玩家按合体键，控制的红球与周围小红球合成一个大红球。

（7）**战斗**

玩家在红球处于合体状态时并且体积比大黑球大时，能够向大黑球发起攻击。

触发条件	攻击效果
红球合体体积小于等于大黑球	红球合体解体，能量降低
红球合体体积大于大黑球	黑球生命减一

（8）**判定游戏胜利**

当场上无大黑球与小黑球时游戏胜利。

2.4 软件非功能需求

（1）**安全性**

游戏访问安全，数据加密。

（2）**响应速度**

及时响应，实时预览。

（3）**输入输出要求**

手机触屏输入。

（4）**故障处理要求**

游戏故障或者需要增加新的功能时，提前在游戏开始界面发布公告。

（5）**其他的特殊需求**

暂无特殊需求。

2.5 运行环境规定

运行平台：Android 平台

开发平台：Visual Studio 2019、Unity 2019.3.8

项目版本管理工具：Github

2.6 验收验证标准

2.6.1 界面验收标准

序号	界面名称	界面描述及预期效果

1	登录界面	显示 PLAY（开始游戏）框和 EXIT（退出游戏）框
2	说明界面	显示控制键，另外用文字说明控制键的用法；显示合体键，当合体键亮起时就可以按下进行球群合体；显示点击跳过框
3	游戏界面	显示游戏大地图和小地图，主角（大红球）、小红球、小灰球和小黑球的预开始状态，摇杆，合体按钮，游戏进度条以及比例条。随机出现的道具（主角加速工具和合体冷却时间归零工具）

2.6.2 功能验收标准

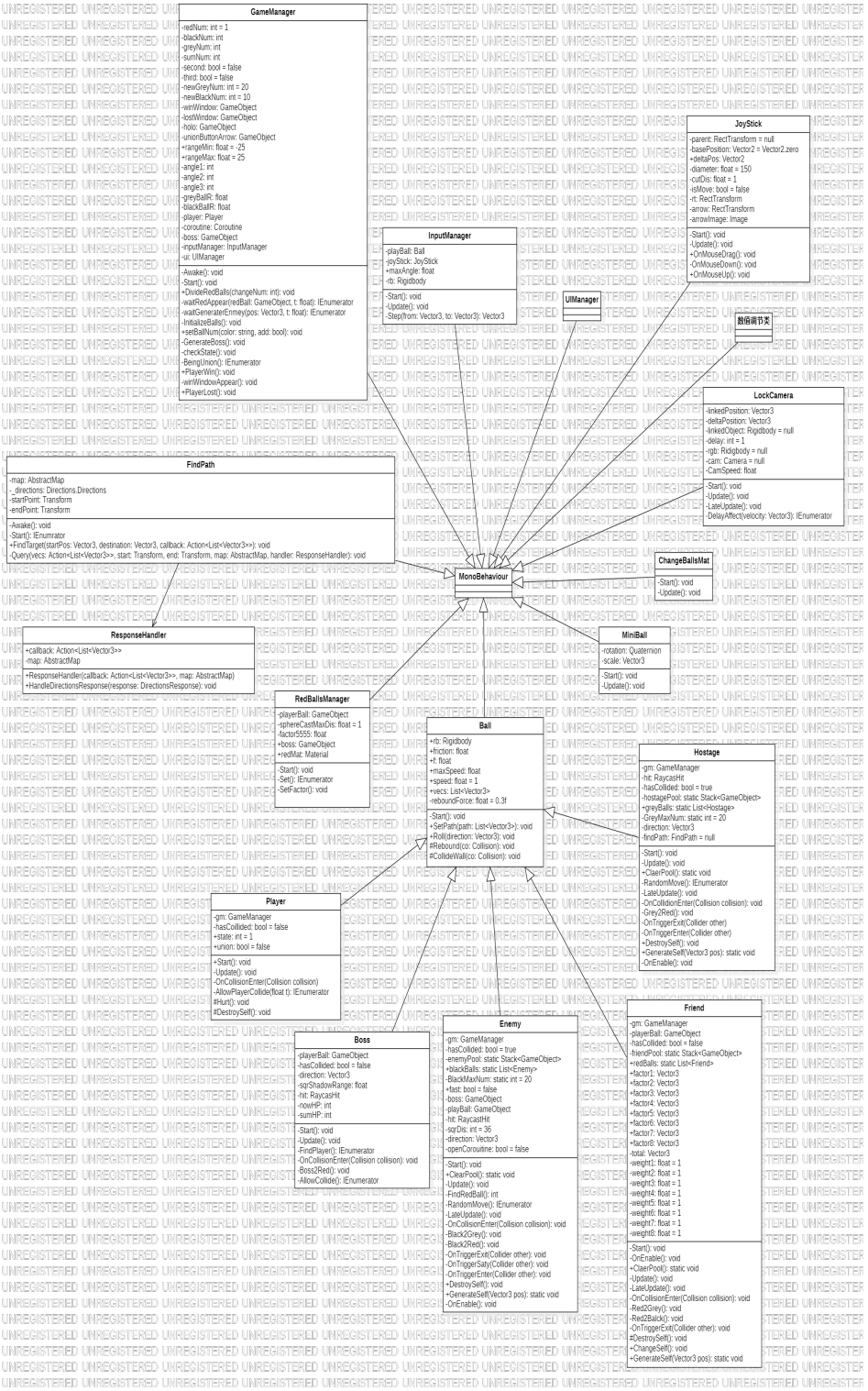
序号	功能名称	详细操作	预期效果
1	开始游戏	点击 PLAY 键	进入游戏界面
2	退出游戏	点击 EXIT 键	退出游戏
3	碰撞	控制摇杆	主角碰撞小灰球，使小灰球变小红球；第一阶段主角碰到小黑球就会死亡； 第二阶段合体的主角碰撞 BOSS，主角解体，部分小红球变小黑球，非合体的主角碰到 BOSS 就会死亡；当主角碰撞小灰球，使小灰球变成小红球，小红球数量达到合体时主角体积大于 BOSS 时进入第

			三阶段； 第三阶段，当合体主角碰撞 BOSS 数次达到 3 次，地图上全部的球变成红球
4	合体	按下合体键	主角周围的小红球全部进入主角内合体成一个更大的主角
5	拾取工具	控制摇杆	主角在摇杆的控制下，移动碰到道具，就可以获得道具的性能（主角加速工具或合体冷却时间归零工具）

三、 系统设计说明书

3.1 功能设计

3.1.1 类图



（1）**MonoBehaviour**

MonoBehaviour 是 Unity 中所有脚本的基类，可以让脚本作为场景物体的组件。

（2）**JoyStick**

- parent: 描述父体（摇杆的底座）的长宽高
- basePostion：摇杆的起始坐标
- deltaPos：要移动的二维向量
- diameter：摇杆移动的限定范围
- cutDis：每一帧允许移动的距离
- isMove：当前摇杆是否偏离了底座的中心，默认值为 false
- rt：摇杆自己的长宽高
- arrow：摇杆到边缘部分的箭头的长宽高
- arrowImage：箭头的图片资源
- Start()：场景刚开始运行时会执行，将某些变量初始化
- Update()：游戏每一帧都会执行，根据当前摇杆的偏移来计算主角小球加速度的方向和数值
- OnMouseDrag()：鼠标点住摇杆并拖拽的过程时执行，让鼠标拖拽的时候，根据计算得到摇杆正确的坐标并在 UI 上显示
- OnMouseDown()：鼠标在摇杆上按下时执行，令 isMove=true
- OnMouseUp()：鼠标在摇杆上按完弹起时执行，令 isMove=false

（3）**LockCamera**

- linkedPosition：相机每一帧的坐标参考
- deltaPosition：相机之间的坐标差值
- linkedObject：相机移动需要关联物体，指的是主角小球，初始值为 null
- delay：延迟 1s，小球移动了 1s，相机才会跟着移动
- rgb：刚体的组件
- cam：相机的组件
- CamSpeed:相机的速度
- Start()：初始化前面的变量

- LateUpdate()：调用协程函数 DelayAffect()
- DelayAffect(Vector3 velocity)：根据小球的移动状态，给相机一个加速度，让相机始终跟随小球移动

（4）**GameManager**

- newGreyNum：游戏开始，生成灰球的数量初始化为 20
- newBlackNum：游戏开始，生成黑球的数量初始化为 10
- rangeMin：生成小球时在主角附近的最小位置范围
- rangeMax：生成小球时在主角附近的最大位置范围
- greyBallR：灰球的半径
- blackBallR：黑球的半径
- Start()：调用 InitializeBalls()进行初始化
- InitializeBalls()：随机生成一批灰球和黑球
- PlayerLost()：游戏失败

（5）**InputManager**

- playerBall：主角小球
- joyStick：摇杆
- maxAngle：用于限定速度改变的角度，对速度方向和加速度方向做差值
- rb：自己的刚体组件
- Start()：初始化
- Update()：每一帧更新
- Step(Vector3 from, Vector3 to)：限定速度改变的角度，对速度方向和加速度方向做差值
- UIManager：负责处理 UI

（6）**MiniBall**

- Rotation
- Scale
- Start()：初始化

- Update()：每一帧进行更新

（7）**Ball**

- rb：自己的刚体组件
- friction：摩擦力系数
- f：速度的衰减值
- maxSpeed：最大速度
- speed：当前速度
- vecs：寻路每一步要走的方向
- reboundForce：小球相撞时，反弹的力度
- Start()：初始化
- SetPath(List<Vector3> path)：设置寻路的路径向量
- Roll(Vector3 direction)：输入为当前加速度的方向，功能为速度的一个平滑
- Rebound(Collision co)：小球碰撞反弹时修改小球的速度
- CollideWall(Collision co)：小球撞到墙体时修改小球的速度

（8）**Boss**

- playerBall：游戏玩家
- hasCollided：当前是不是在碰撞的状态，初始值是 false
- direction：每次加速度的方向
- sqrShadowRange：
- hit：发出射线后碰到的物体
- nowHP：boss 剩余的生命次数
- sumHP：boss 一共有的生命次数
- Start()：变量初始化
- Update()：每一帧进行更新，修改 direction，令 boss 移动
- FindPlayer()：每隔一段时间，跳到主角附近
- OnCollisionEnter(Collision collision)：小球碰撞后的逻辑，即小球碰到哪种类型的球以及碰撞之后变成什么颜色（销毁自己，生成新的小球）
- Boss2Red()：boss 变成红色

- AllowCollide()：设置 boss 当前是否处于碰撞状态

（9）**ChangeBallsMat**

挂在某一类小球的父体上，修改这一类小球的材质和颜色

（10）**Enemy**

- gm：GameManager 一个实例，初始化为 null
- hasCollided：当前是不是在碰撞的状态，初始值是 true
- enemyPool：小黑球的对象池，存放尚未使用（即尚未使用）的小黑球
- blackBalls：场景中已经使用的小黑球
- BlackMaxNum：小黑球的最大数量，初始值为 20
- fast：fast=true 时，表示进入 boss 范围内的小黑球范围会变快，初始值为 false
- boss：游戏中的 boss
- playerBall：游戏中的玩家
- hit：发出射线后碰到的物体
- sqrDis：是否寻找小红球的距离限制，小于 sqrDis 时，小黑球不再寻找小红球，直接碰撞小红球，初始值为 36
- direction：小球每一帧的加速度方向
- openCorouitine：是否开启了寻路的协程，初始值为 false
- Start()：初始化
- static void ClearPool()：清空对象池
- Update()：每一帧进行更新
- FindRedBall()：寻找小红球
- RandomMove()：随机寻路，把寻路方向作为自己的加速度
- LateUpdate()：设置 hasCollided 为 false
- OnCollisionEnter(Collision collision)：小球碰撞后的逻辑，即小球碰到哪种类型的球以及碰撞之后变成什么颜色（销毁自己，生成新的小球）
- Black2Grey()：黑球变灰
- Black2Red()：黑球变红
- OnTriggerExit(Collider other)：球体超出了以主角小球为中心的某个范围，调

用 DestroySelf()和 GenerateSelf()，将自己销毁并在范围内重新生成

- OnTriggerStay(Collider other)：小黑球进入了大黑球的范围内，小黑球的速度变快
- OnTriggerEnter(Collider other)：最后游戏要胜利的时候，所有小球变红
- DestroySelf()：销毁自己
- GenerateSelf(Vector3 pos)：生成新的球体
- OnEnable()：每次小球重新生成后修改其材质和颜色

（11）**Friend**

- gm：GameManager 一个实例，初始化为 null
- playerBall：游戏中的玩家
- hasCollided：当前是不是在碰撞的状态，初始值是 false
- friendPool：小红球的对象池，存放尚未使用（即尚未使用）的小红球
- redBalls：场景中已经使用的小红球
- factor1：任意红球趋向于远离周围红球
- factor2：任意红球趋向于靠近所有红球中心
- factor3：任意红球趋向于朝向红球群的平均方向
- factor4：任意红球趋向于接近灰球
- factor5：任意红球越远离主角，越想靠近主角；越靠近主角，越想远离主角
- factor6：任意红球靠近某面墙一定范围内时，越靠近墙，越趋向于远离墙
- factor7：随机进行微小扰动，任意红球拥有一个向左或向右的微小速度
- factor8：任意红球趋向于远离大黑球
- total：8 个 factor 加权求和后的最终结果
- weight1, weight2,weight3,weight4,weight5,weight6,weight7,weight8：每个 factor 分别对应对的权重，均初始化为 1
- Start()：场景刚加载时，脚本作为组件时调用，进行初始化
- OnEnable()：为该场景物体每次显示时调用，即每次小球重新生成后修改其材质和颜色
- static void ClearPool()：清空对象池
- Update()：每一帧进行更新

- LateUpdate()：设置 hasCollided 为 false， 每帧调用一次，在 Update()之后，帧循环的末尾
- OnCollisionEnter(Collision collision)：为该物体边缘刚刚碰撞到一个带有 Collsion 碰撞器的 3d 物体时触发，输入参数为对方的碰撞器，实现小球碰撞后的逻辑，即小球碰到哪种类型的球以及碰撞之后变成什么颜色（销毁自己，生成新的小球）
- Red2Grey()：红球变灰
- Red2Black()：红球变黑
- OnTriggerExit(Collider other)：为该物体刚好离开一个带有 Trigger 触发器的 3d 物体时触发，输入参数为对方的触发器。球体超出了以主角小球为中心的某个范围，调用 DestroySelf()和 GenerateSelf()，将自己销毁并在范围内重新生成
- DestroySelf()：销毁自己
- GenerateSelf(Vector3 pos)：生成新的球体

（12）**Hostage**

- gm：GameManager 一个实例，初始化为 null
- hit：发出射线后碰到的物体
- hasCollided：当前是不是在碰撞的状态，初始值是 true
- hostagePool：小灰球的对象池，存放尚未使用（即尚未使用）的小灰球
- greyBalls：场景中已经使用的小红球
- GreyMaxNum：小灰球的最大数量，初始值为 20
- direction：小球每一帧的加速度方向
- findPath：FindPath 的实例，初始化为 null
- Start()：初始化
- Update()：每一帧进行更新
- static void ClearPool()：清空对象池
- RandomMove()：随机寻路，把寻路方向作为自己的加速度
- LateUpdate()：设置 hasCollided 为 false
- OnCollisionEnter(Collision collision)：小球碰撞后的逻辑，即小球碰到哪种类

型的球以及碰撞之后变成什么颜色（销毁自己，生成新的小球）

- Grey2Red：灰球变红
- OnTriggerExit(Collider other)：球体超出了以主角小球为中心的某个范围，调用 DestroySelf()和 GenerateSelf()，将自己销毁并在范围内重新生成
- OnTriggerEnter(Collider other)：最后游戏要胜利的时候，所有小球变红
- DestroySelf()：销毁自己
- GenerateSelf(Vector3 pos)：生成新的球体
- OnEnable()：每次小球重新生成后修改其材质和颜色

（13）**FindPath**

- map：AbstractMap 的实例
- _directions：最终输出的寻路方向向量
- startPoint：开始点的坐标
- endPoint：终点的坐标
- Awake()：初始化
- Start()：初始化
- FindTarget(Vector3 startPos, Vector3 destination, Action<List<Vector3>> callback)：输入起始点和终点，调用 Query()函数返回最后的寻路数组
- Query(Action<List<Vector3>> vecs, Transform start, Transform end, AbstractMap map, ResponseHandler handler)：被 FindTarge()调用

（14）**ResponseHandler**

- callback：服务器给的数据
- map：AbstractMap 的实例
- ResponseHandler(Action<List<Vector3>> callback, AbstractMap map)：构造函数
- HandleDirectionsResponse(DirectionsResponse response)：用于回调，和服务器进行连接

（15）**Player**

- gm：GameManager 一个实例
- hasCollided：当前是不是在碰撞的状态，初始值是 false
- state：player 所处的大小状态，123 对应小中大，初始化为 1
- union：标志当前是否处于合体状态，初始值为 false
- Start()：初始化
- Update()：每一帧进行更新
- OnCollisionEnter(Collision collision)：小球碰撞后的逻辑，即小球碰到哪种类型的球以及碰撞之后变成什么颜色（销毁自己，生成新的小球）
- AllowPlayerCollide(float t)：设置 player 当前是否处于碰撞状态
- Hurt()：合体的红球进行解体
- DestroySelf()：游戏失败，销毁自己

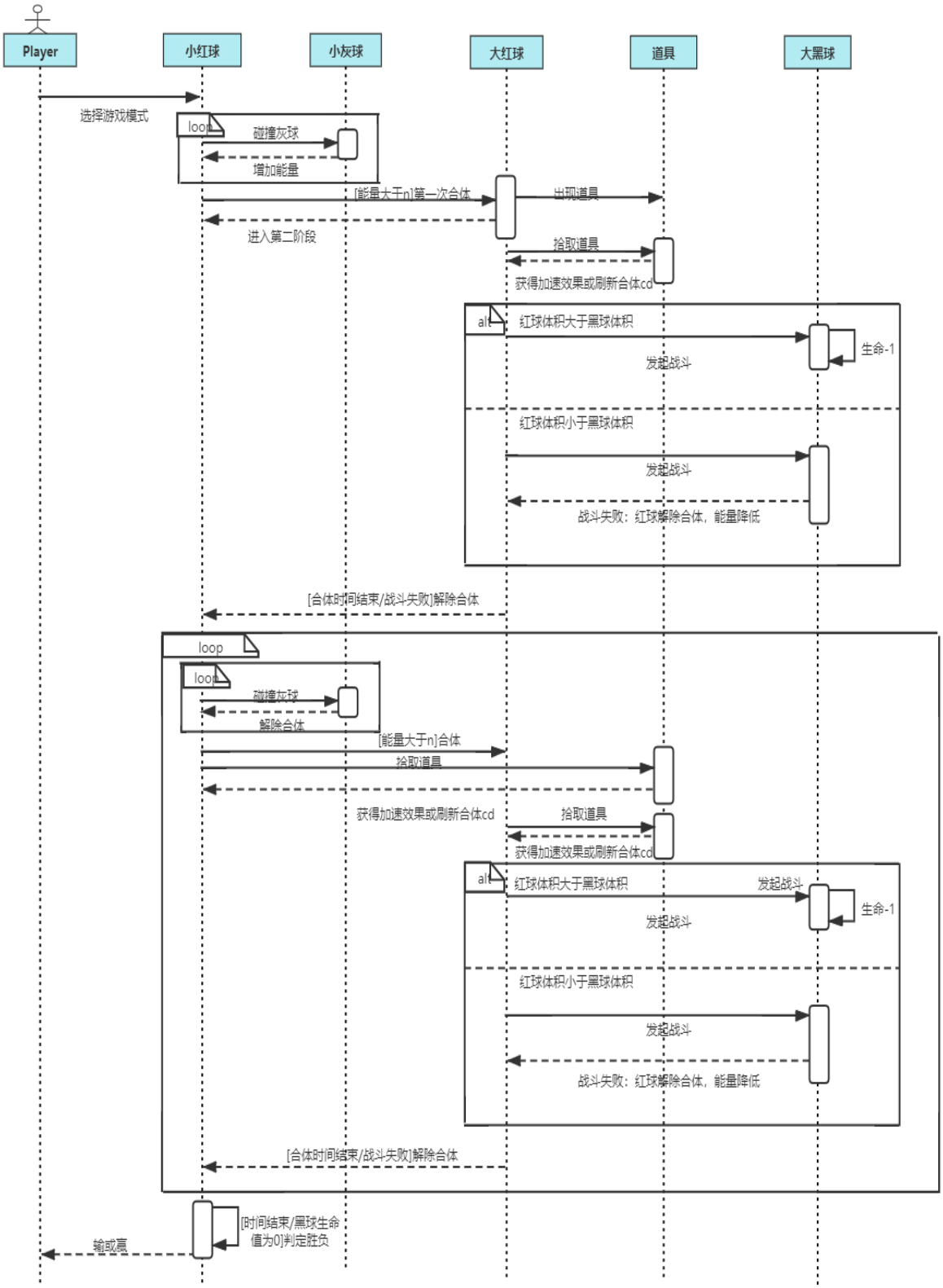
（16）**RedBallsManager**

- playerBall：游戏中的玩家
- sphereCastMaxDis：使用 Spherecast 找墙面，sphereCastMaxDis 是 Spherecast 射线的距离，初始化为 1
- factor5555：factor5 的一个附加权重
- boss：游戏中的 boss
- redMat：红球的材质
- Start()：调用协程 Set()函数
- Set()：每隔 setTimeGap 时间，调用一次 SetFactor()函数
- SetFactor()：计算 8 个 factor，factor1 趋向于远离周围红球速度归一化加起来；factor2 趋向于靠近所有红球中心；factor3 趋向于朝向红球群的平均方向，每帧都遍历所有红球‘；factor4 越靠近主角越趋向于接近灰球；factor5 越远离主角越趋向于接近主角与主角的距离；factor6 越靠近墙越越趋向于远离墙，Spherecast 找墙面,遇到墙，权重方向为墙法线方向；factor7 左右方向随机扰动；factor8 避大黑球。

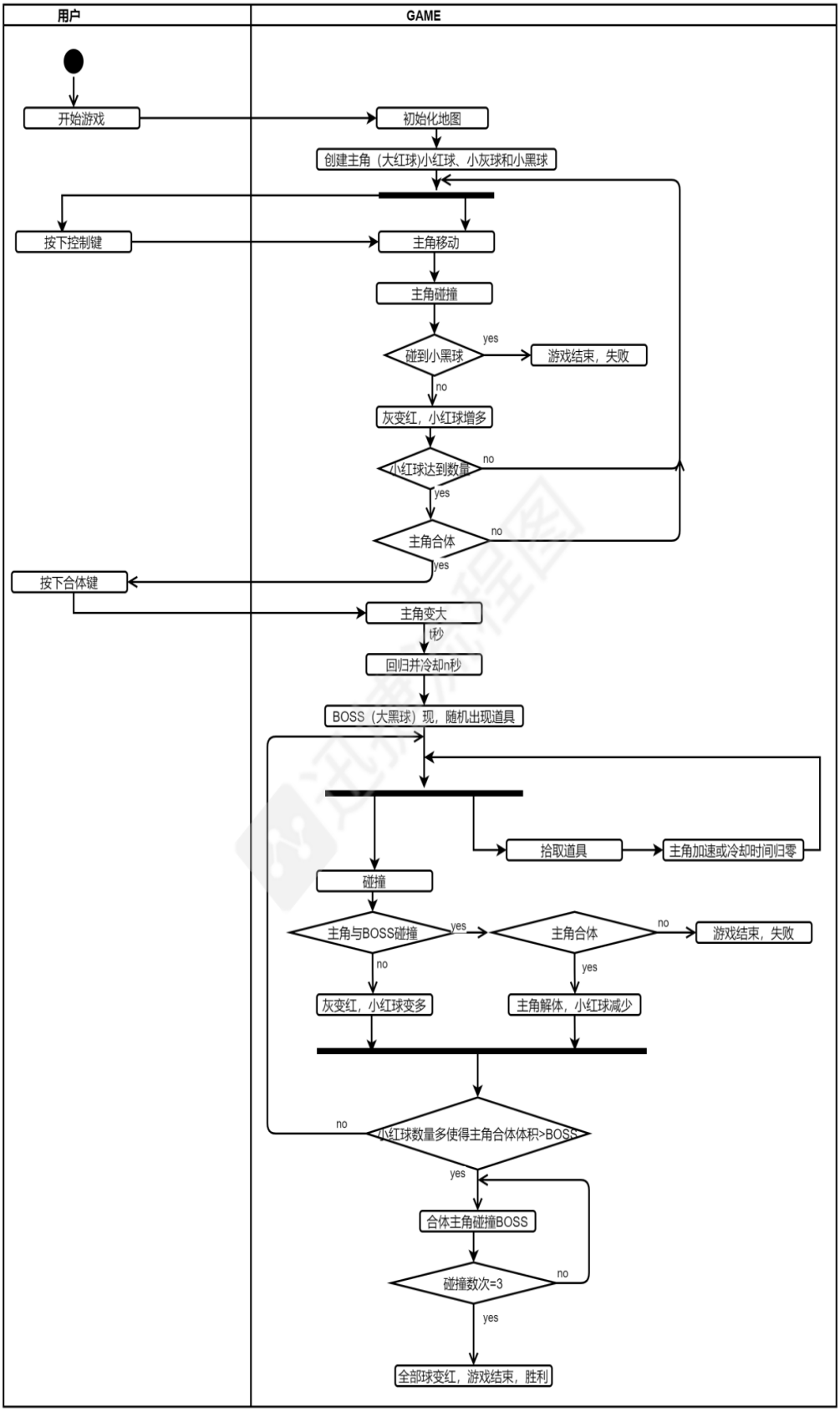
（17）**数值调节类**

对游戏中的数值进行调节

3.1.2 时序图


Player
小红球
小灰球
大红球
道具
大黑球
选择游戏模式
loop
碰撞灰球
增加能量
[能量大于n]第一次合体
出现道具
进入第二阶段
拾取道具
获得加速效果或刷新合体cd
alt
红球体积大于黑球体积
生命-1
发起战斗
红球体积小于黑球体积
发起战斗
战斗失败：红球解除合体，能量降低
[合体时间结束/战斗失败]解除合体
loop
loop
碰撞灰球
解除合体
[能量大于n]合体
拾取道具
获得加速效果或刷新合体cd
拾取道具
获得加速效果或刷新合体cd
alt
红球体积大于黑球体积
发起战斗
生命-1
发起战斗
红球体积小于黑球体积
发起战斗
战斗失败：红球解除合体，能量降低
[合体时间结束/战斗失败]解除合体
[时间结束/黑球生命值为0]判定胜负
输或赢

3.1.3 活动图


用户
GAME
开始游戏
初始化地图
创建主角（大红球)小红球、小灰球和小黑球
按下控制键
主角移动
主角碰撞
碰到小黑球
yes
游戏结束，失败
no
灰变红，小红球增多
小红球达到数量
no
yes
主角合体
no
yes
按下合体键
主角变大
t秒
回归并冷却n秒
BOSS（大黑球）现，随机出现道具
拾取道具
主角加速或冷却时间归零
碰撞
主角与BOSS碰撞
yes
主角合体
no
游戏结束，失败
no
yes
灰变红，小红球变多
主角解体，小红球减少
no
小红球数量多使得主角合体体积>BOSS
yes
合体主角碰撞BOSS
碰撞数次=3
no
yes
全部球变红，游戏结束，胜利

3.2 数据库设计

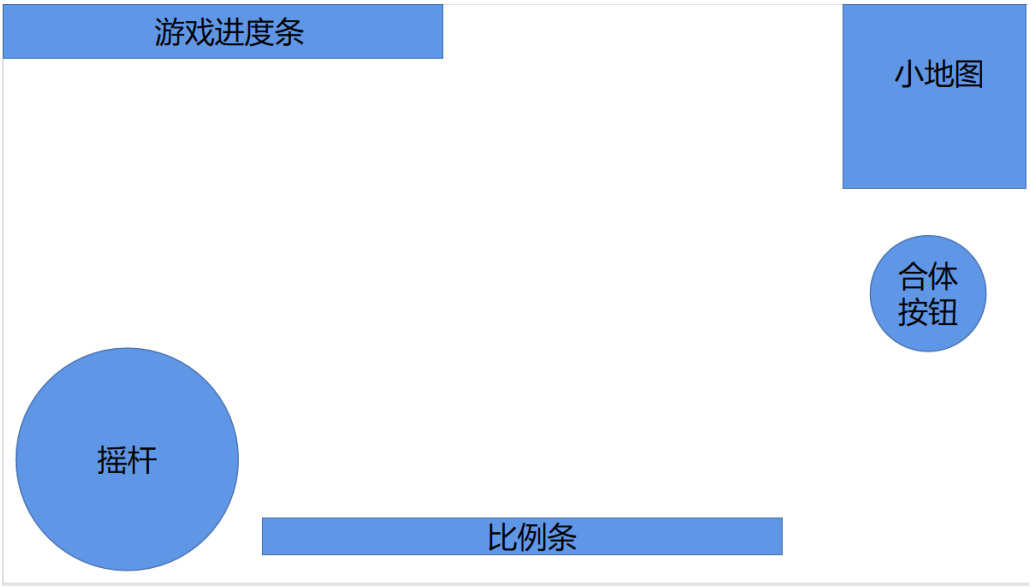
我们的项目没有使用数据库。

3.3 UI 设计

3.3.1 游戏 logo

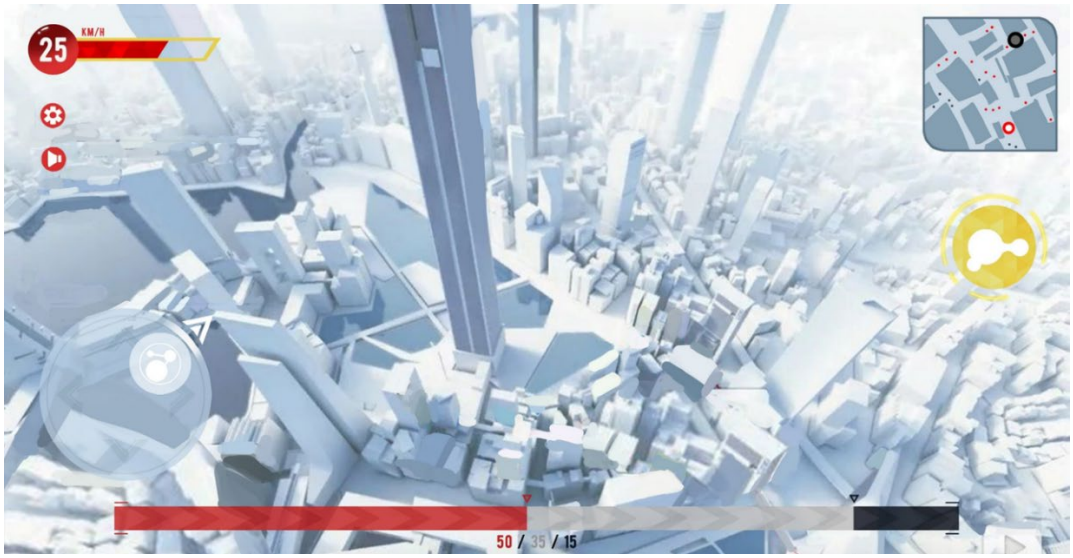
3.3.2 开始界面

3.3.3 游戏布局



- 窗口右上方角落放置小地图，展现当前区域黑、灰、红小球分布情况和大黑球位置。
- 上方状态栏展现主角还需要多少红色小球能体积增大进入下一阶段。
- 下方状态栏展现场景中所有黑、灰、红球比例。
- 右侧合体按钮当且仅当场上小红球数量达到预设值时有效
- 左下角摇杆被玩家用于控制主角的移动

3.3.4 预期效果



3.3.5 摇杆图片

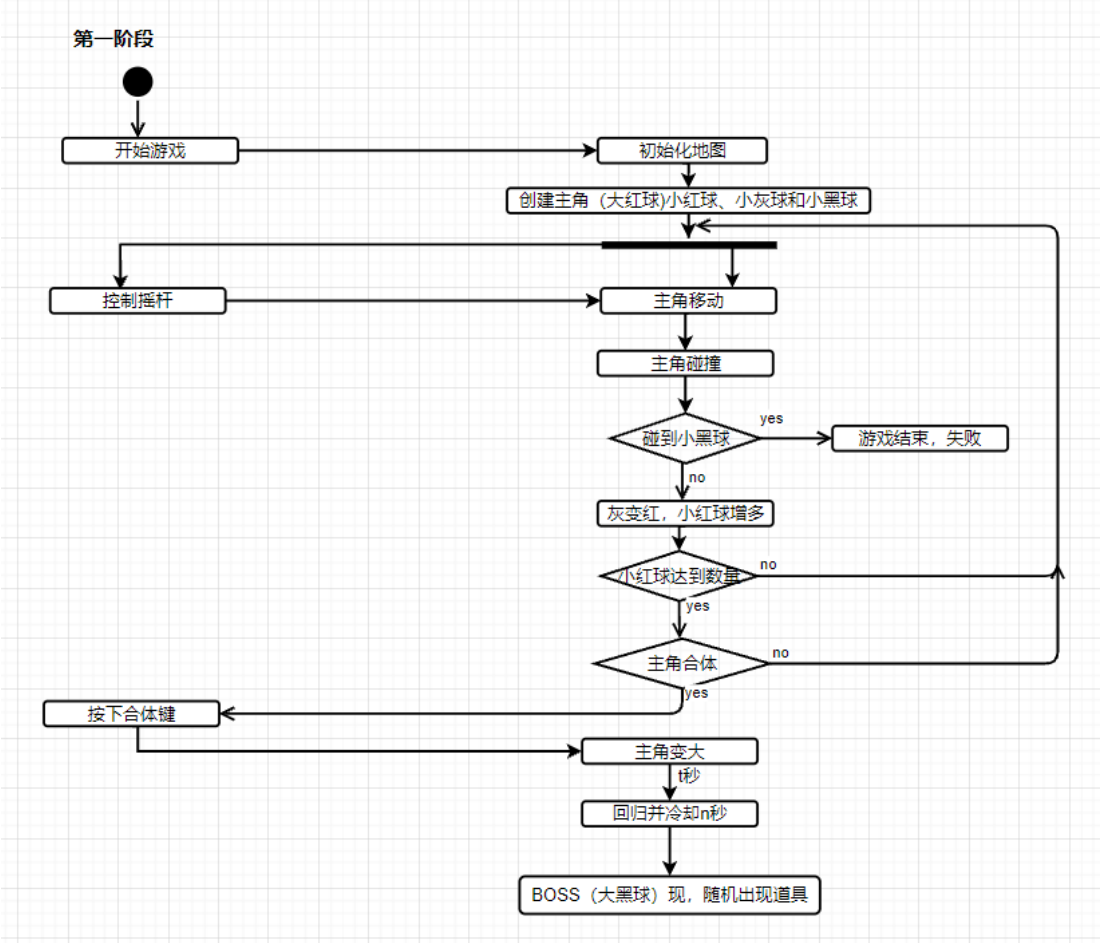
四、 项目实现

4.1 各种小球的逻辑

类型	移动逻辑	碰撞逻辑
主角	玩家通过摇杆进行控制	能将小灰球变成小红球，合体阶段的主角能将小黑球变为小红球；非黑体阶段的主角遇见小黑球，游戏以失败结束；合体阶段的主角体积大于 boss 时与 boss 碰撞，boss 失去一次生命，主角解体
小红球	集群算法	能将小灰球变为小红球
小灰球	随机移动	
小黑球	调用地图插件 API 实现寻路算法	能将小红球变为小黑球；遇见非合体阶段的主角时，游戏结束
Boss		boss 体积大于合体阶段的主角时与主角发生碰撞，主角解体，部分小红球变为小黑球

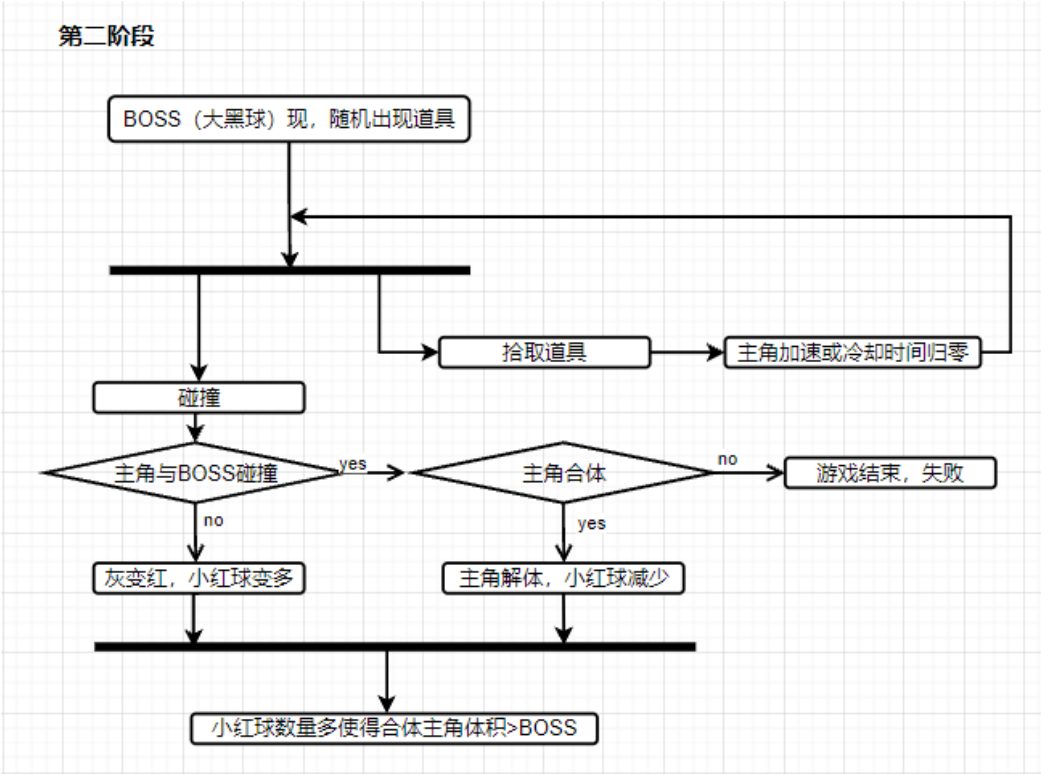
4.2 游戏的 3 个阶段

4.2.1 游戏阶段 I



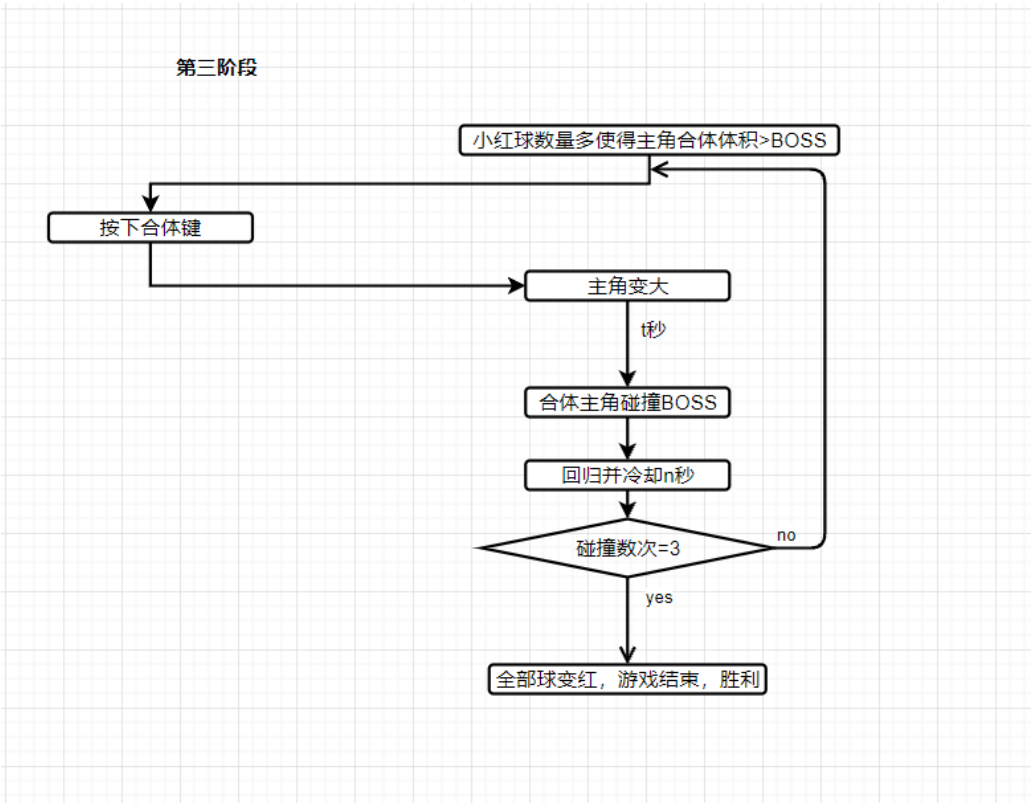
开始游戏，先初始化游戏地图，创建主角、小红球、小灰球和小黑球，这时各类球体都是处于自由状态，通过控制摇杆来控制主角的移动状态。本阶段如果主角碰撞到小黑球，主角就会死亡，游戏失败。如果主角碰撞小灰球，小灰球变成小红球，当场上小红球达到一定数量，就可以按下合体键，第一次按下合体键，游戏就会进入第二阶段。

4.2.2 游戏阶段 II



本阶段开始时，主角处于合体状态， boss（大黑球）出现，但是合体后的主角体积依然小于 boss。地图上会随机出现道具（主角加速和主角冷却时间归零），主角移动碰到道具，就会即时的获得道具的功能。如果非合体的主角碰到 boss 就会死亡，游戏失败。如果合体的主角碰到 boss，主角会立刻解体，解体后的小红球会部分变成小黑球，小红球数量减少。当主角碰撞小灰球使得小红球数量增多且合体后的主角体积大于 boss 时进入第三阶段。

4.2.3 游戏阶段 III



本阶段，合体状态的主角体积大于 boss 的体积，主角可以直接碰撞 boss，而且每碰撞一次，boss 会失去一次生命，缩小一定体积，主角也解体。合体主角碰撞达到三次，boss 死亡，地图上的全部球体变红，游戏胜利。

4.3 游戏的 3 个设计模式

4.3.1 单例模式

项目中，只有一个 GameManager 对象，作为组件挂在空物体 Manager 上，用于管理游戏的整体进程，控制每种实例的数量，调用 UIManager 的函数更改 UI。

4.3.2 对象池模式

场景中反复出现和消失的有三种球：小红球、小灰球、小黑球，对于这三种球，类里分别有一个静态私有池，用数据结构栈来保存隐藏了的场景物体。放弃单独地分配和释放对象，从固定的池中重用对象。

以小黑球（Enemy 类）为例：

```
private bool hasCollided = true;
private static Stack<GameObject> enemyPool = new Stack<GameObject>();
public static List<Enemy> blackBalls = new List<Enemy>();
private static int BlackMaxNum = 20;

public void DestroySelf()
{
    rb.velocity = Vector3.zero;
    //Debug.Log("销毁黑球");
    gm.SetBallNum("black", false);
    入栈，隐藏物体
    enemyPool.Push(gameObject);
    gameObject.SetActive(false);
    blackBalls.Remove(this);
}
```

球的销毁：实际为球从场景中隐藏，进入对象池（入栈）

```
public static void GenerateSelf(Vector3 pos)
{
    if (BlackMaxNum <= blackBalls.Count)
        return;
    //Debug.Log("生成黑球");
    GameObject.Find("Manager").GetComponent<GameManager>().SetBallNum("black",
    GameObject go;
    if (enemyPool.Count > 0)
    {
        go = enemyPool.Pop();
    }
    else
```

如果栈不为空，
出栈，显示物体

球的生成：如果栈不为空，则从对象池中出栈一个物体，并将其显示在场景中，如果栈空则重新生成该物体。

4.3.3 组件模式

单一实体横跨多个域。为了保持域之间相互隔离，每个域的代码都独立的放在自己的组件类中。实体本身可以简化为这些组件的容器。

Unity 引擎本身使用了三大设计模式：更新方法、游戏循环模式和组件模式，其中更新方法和游戏循环模式，应用在 unity 引擎的生命周期里。以 Friend 类为例：

- Start()：场景刚加载时，脚本作为组件时调用，进行初始化
- OnEnable()：为该场景物体每次显示时调用，即每次小球重新生成后修改其材质和颜色
- static void ClearPool()：清空对象池
- Update()：每一帧进行更新
- LateUpdate()：设置hasCollided为false， 每帧调用一次，在Update()之后，帧循环的末尾
- OnCollisionEnter(Collision collision)：为该物体边缘刚刚碰撞到一个带有Collsion碰撞器的3d物体时触发，输入参数为对方的碰撞器，实现小球碰撞后的逻辑，即小球碰到哪种类型的球以及碰撞之后变成什么颜色（销毁自己，生成新的小球）
- Red2Grey()：红球变灰
- Red2Black()：红球变黑
- OnTriggerExit(Collider other)：为该物体刚好离开一个带有Trigger触发器的3d物体时触发，输入参数为对方的触发器。球体超出了以主角小球为中心的某个范围，调用DestroySelf()和GenerateSelf()，将自己销毁并在范围内重新生成
- DestroySelf()：销毁自己
- GenerateSelf(Vector3 pos)：生成新的球体

4.4 3 种寻路算法

下面这个脚本是 MapBox 插件对外的一个接口，调用 FindPath 类里的 FindTarget()函数，输入起点和终点，以数组的形式返回沿路需要经过的主要坐标。

```
namespace Mapbox.Examples
{
    public class FindPath : MonoBehaviour
    {
        [SerializeField]
        AbstractMap map;
        Directions.Directions _directions;

        [SerializeField]
        Transform startPoint;
        [SerializeField]
        Transform endPoint;

        void Awake()
        {
            _directions = MapboxAccess.Instance.Directions;
        }

        IEnumerator Start()
        {
            yield return new WaitForSeconds(1);
        }

        public void FindTarget(Vector3 startPos, Vector3 destination, Action<List<Vector3>> callback)
        {
            startPoint.position = startPos;
            endPoint.position = destination;
            ResponseHandler handler = new ResponseHandler(callback, map);
            Query(callback, startPoint, endPoint, map, handler);
        }

        private void Query(Action<List<Vector3>> vecs, Transform start, Transform end, AbstractMap map, ResponseHandler handler)
        {
            Vector2d[] wp = new Vector2d[2];
            wp[0] = start.GetGeoPosition(map.CenterMercator, map.WorldRelativeScale);
            wp[1] = end.GetGeoPosition(map.CenterMercator, map.WorldRelativeScale);
            DirectionResource _directionResource = new DirectionResource(wp, RoutingProfile.Walking);
            _directionResource.Steps = true;

            _directions.Query(_directionResource, handler.HandleDirectionsResponse);
        }
    }


    class ResponseHandler
    {
        public Action<List<Vector3>> callback;
        AbstractMap map;

        public ResponseHandler(Action<List<Vector3>> callback, AbstractMap map)
        {
            this.map = map;
            this.callback = callback;
        }

        public void HandleDirectionsResponse(DirectionsResponse response)
        {
            if (null == response.Routes || response.Routes.Count < 1)
            {
                return;
            }

            List<Vector3> dat = new List<Vector3>();
            foreach (var point in response.Routes[0].Geometry)
            {
                dat.Add(Conversions.GeoToWorldPosition(point.x, point.y, map.CenterMercator, map.WorldRelativeScale).ToVector3xz());
            }
            callback(dat);
        }
    }
}
```

4.4.1 小红球：集群算法

- **主要目的**

  让所有小红球围绕在主角身边，跟随主角移动，同时聚集成一群

- **代码实现**

```
51      /// <summary>
52      /// 红球的集群算法
53      /// TODO:遇到凹形墙会卡死
54      /// </summary>
55      void SetFactor()
56      {
57          List<Friend> reds = Friend.redBalls;
58          List<Hostage> greys = Hostage.greyBalls;
59          // Debug.Log(reds.Count);
60          //因素123清零
61          for (int i = 0; i < reds.Count; i++)
62          {
63              reds[i].factor1 = Vector3.zero;
64              reds[i].factor2 = Vector3.zero;
65              reds[i].factor3 = Vector3.zero;
66              //reds[i].factor8 = Vector3.zero;
67          }
68
69          //因素1计算，遍历n(n-1)/2
70          //算因素2 因素3的总值
71          //因素45
72          Vector3 factors2 = Vector3.zero;
73          Vector3 factors3 = Vector3.zero;
74          for (int i = 0; i < reds.Count; i++)
75          {
76              for (int j = i + 1; j < reds.Count; j++)
77              {
78                  Vector3 delta = reds[i].transform.position - reds[j].transform.position;
79                  float dis = 1 / (delta.magnitude);
80                  reds[i].factor1 += dis * delta.normalized;
81                  reds[j].factor1 -= dis * delta.normalized;
82              }
83
84              //算factor4，对于每一个红球，找最近的灰球
85              float minDis = 100;
86              int minIndex = -1;
87              for (int j = 0; j < greys.Count; j++)
88              {
89                  if ((reds[i].transform.position - greys[j].transform.position).sqrMagnitude < minDis)
90                  {
91                      minDis = (reds[i].transform.position - greys[j].transform.position).sqrMagnitude;
92                      minIndex = j;
93                  }
94              }
```

```
95          // Debug.Log(minIndex);
96          reds[i].factor5 = playerBall.transform.position - reds[i].transform.position;
97          reds[i].factor5 -= (playerBall.transform.position - reds[i].transform.position).normalized * factor5555;
98          if (greys.Count <= 0 || minIndex == -1)
99              reds[i].factor4 = Vector3.zero;
100         else if (minIndex >= 0)
101         {
102             Vector3 v = (greys[minIndex].transform.position - reds[i].transform.position);
103             reds[i].factor4 = v.normalized * (1 / v.magnitude);
104             reds[i].factor5 = Vector3.zero;
105         }
106         factors2 += reds[i].transform.position;
107         factors3 += reds[i].GetComponent<Rigidbody>().velocity.normalized;
108     }
109     factors2 /= reds.Count;
110     factors3 /= reds.Count;
111
112     //算出因素1的均值
113     //因素2 3 赋值
114     float r = 1;
115     RaycastHit hit;
116     for (int i = 0; i < reds.Count; i++)
117     {
118         reds[i].factor1 /= reds.Count;
119         reds[i].factor2 = factors2 - reds[i].transform.position;
120         reds[i].factor3 = factors3;
121
122         //因素6
123         if (Physics.SphereCast(reds[i].transform.position, r, reds[i].GetComponent<Rigidbody>().velocity, out hit, sphereCastMaxDis, 1
124         {
125             reds[i].factor6 += hit.normal * reds[i].rb.velocity.magnitude * 0.1f + Vector3.Cross(hit.normal, Vector3.up).normalized;
126         }
127         else
128         {
129             reds[i].factor6 *= 0.5f;
130         }
131
```

- **代码说明**

  每帧实时计算 8 个 factor，根据 8 个 factor 得到当前帧加速度的方向：

  - factor1 趋向于远离周围红球速度归一化加起来；
  - factor2 趋向于靠近所有红球中心；
  - factor3 趋向于朝向红球群的平均方向，每帧都遍历所有红球；
  - factor4 越靠近主角越趋向于接近灰球；
  - factor5 越远离主角越趋向于接近主角与主角的距离；
  - factor6 越靠近墙越越趋向于远离墙，Spherecast 找墙面,遇到墙，权重方向为墙法线方向；
  - factor7 左右方向随机扰动；
  - factor8 避大黑球。

4.4.2 小灰球：随机移动

- **主要目的**

  在离主角很远的时候使用寻路算法来接近主角，离主角距离在一定范围内时，开始随机移动，同时也要考虑避开墙等障碍物

- **代码实现**

```
38      void Update()
39      {
40          if (Physics.SphereCast(transform.position, 1, rb.velocity, out hit, 5, 1 << 10))
41              direction += hit.normal * rb.velocity.magnitude + Vector3.Cross(hit.normal, Vector3.up).normalized;
42
43          direction += Vector3.Cross(Vector3.up, rb.velocity).normalized * Random.Range(-1f, 1f);
44
45          direction = direction.normalized;
46          Roll(direction);
47      }
48
49      public static void ClearPool()
50      {
51          hostagePool.Clear();
52      }
53
54      IEnumerator RandomMove()
55      {
56          while (true)
57          {
58              Vector3 pos = new Vector3(Random.Range(-50, 50), 0, Random.Range(-50, 50));
59              findPath.FindTarget(transform.position, pos, SetPath);
60
61              yield return new WaitForSeconds(4);
62
63              foreach (Vector3 v in vecs)
64              {
65                  while (true)
66                  {
67                      direction += (v - transform.position).normalized;
68                      if ((v - transform.position).sqrMagnitude <= 1)
69                          break;
70                      yield return 0;
71                  }
72
73                  yield return 0;
74              }
75
76              yield return 0;
77          }
78      }
79
```

- **代码说明**

每帧修改加速度的方向：

- 发出球形射线，如果检测到墙，就给加速度加一个沿着墙的切线方向加速度的分量
- 将上方向的向量与速度方向叉乘并归一化，再乘上一个大小位于[-1,1]区间的随机浮点数，作为左右随机扰动的分量
- 同时参考寻路函数 RandomMove 得到的方向，即每隔一段时间从该函数返回值的数组里依次采取数组元素作为加速度的分量

4.4.3 小黑球：调用地图插件 API 实现寻路算法

- **主要目的**

通过寻路算法不断地逼近主角的位置

- 代码实现

```
    void Update()
    {
        if (Physics.SphereCast(transform.position, 1, rb.velocity, out hit, 5, 1 << 10))
            direction += hit.normal * rb.velocity.magnitude * 0.8f + Vector3.Cross(hit.normal, Vector3.up).normalized;

        direction += Vector3.Cross(Vector3.up, rb.velocity).normalized * Random.Range(-1f, 1f);


        if (boss != null && boss.activeSelf && (boss.transform.position - transform.position).sqrMagnitude < 25)
        {
            direction += (boss.transform.position - transform.position).normalized * 2;
        }

        if (fast)
        {
            direction += (playerBall.transform.position - transform.position).normalized * 2;
        }
        else
        {
            int index = FindRedBall();
            if (index >= 0)
            {
                if ((Friend.redBalls[index].transform.position - transform.position).sqrMagnitude < (playerBall.transform.position - t
                    direction = (Friend.redBalls[index].transform.position - transform.position).normalized;
                else
                    direction = (playerBall.transform.position - transform.position).normalized;
                StopCoroutine(RandomMove());
                openCorouitine = false;
            }
            else
            {
                if (!openCorouitine)
                {
                    StartCoroutine(RandomMove());
                    openCorouitine = true;
                }

            }
        }

        direction = direction.normalized;
        Roll(direction);
    }
```

```
    int FindRedBall()
    {
        for (int i = 0; i < Friend.redBalls.Count; i++)
        {
            if ((Friend.redBalls[i].transform.position - transform.position).sqrMagnitude < sqrDis)
            {
                return i;
            }
        }
        return -1;
    }

    IEnumerator RandomMove()
    {
        while (true)
        {
            Vector3 pos = new Vector3(Random.Range(-50, 50), 0, Random.Range(-50, 50));
            GameObject.Find("Manager").GetComponent<FindPath>().FindTarget(transform.position, pos, SetPath);
            yield return new WaitForSeconds(4);
            foreach (Vector3 v in vecs)
            {
                while (true)
                {
                    direction += (v - transform.position).normalized;
                    if ((v - transform.position).sqrMagnitude <= 1)
                        break;
                    yield return 0;
                }

                yield return 0;
            }

            yield return 0;
        }
    }

```

- **代码说明**

  每帧修改加速度的方向：

  - 发出球形射线，如果检测到墙，就给加速度加一个沿着墙的切线方向加速度的分量
  - 将上方向的向量与速度方向叉乘并归一化，再乘上一个大小位于[-1,1]区间的随机浮点数，作为左右随机扰动的分量
  - 如果一定范围内，有 boss，就朝 boss 方向移动，即加一个 boss 方向的加速度分量
  - 如果小黑球已经在 boss 的 buff 范围内，就加一个朝向主角的加速度分量，否则，如果一定范围内，找到了任意小红球，则朝它移动，即加一个朝着该小红球的加速度分量，不然则使用寻路算法以靠近小红球

五、 测试报告

5.1 编写目的

本测试报告为 BallGame 游戏项目的测试报告, 目的在于总结测试阶段的测试以及分析测试结果, 描述系统是否符合需求. 预期参考人员包括用户、测试人员、开发人员、项目管理者和需要阅读本报告的高层经理。

5.2 目标

（1）界面验收：登录界面上可显示 PLAY 框和 EXIT 框；游戏界面能够显示游戏大地图和小地图，主角（大红球）、小红球、小灰球和小黑球的预开始状态，摇杆，合体按钮，游戏进度条以及比例条。随机出现的道具（主角加速工具和合体冷却时间归零工具）

（2）功能验收：各个键实现对应的功能。例如控制摇杆可使得主角碰撞小灰球，使小灰球变小红球；第一阶段主角碰到小黑球就会死亡；第二阶段合体的主角碰撞 BOSS，主角解体，部分小红球变小黑球，非合体的主角碰到 BOSS 就会死亡；当主角碰撞小灰球，使小灰球变成小红球，小红球数量达到合体时主角体积大于 BOSS 时进入第三阶段；第三阶段，当合体主角碰撞 BOSS 数次达到 3 次，地图上全部的球变成红球；按下合体键使得主角周围的小红球全部进入主角内合体成一个更大的主角。主角捡到道具的前后状态

5.3 运行环境

Android 平台

5.4 测试演示

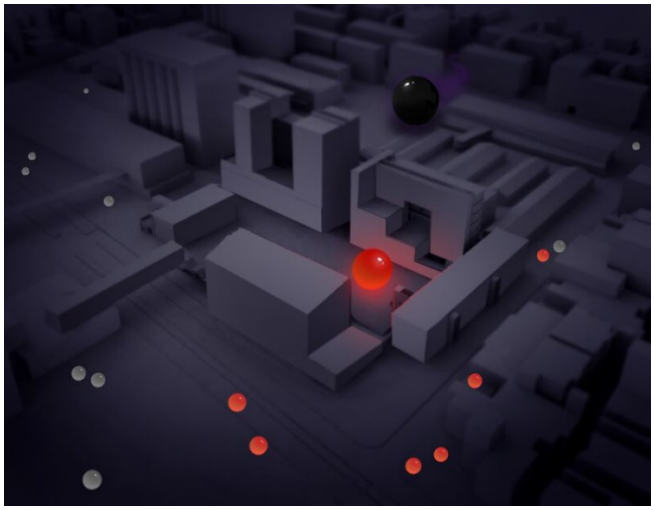
5.4.1 开始界面测试

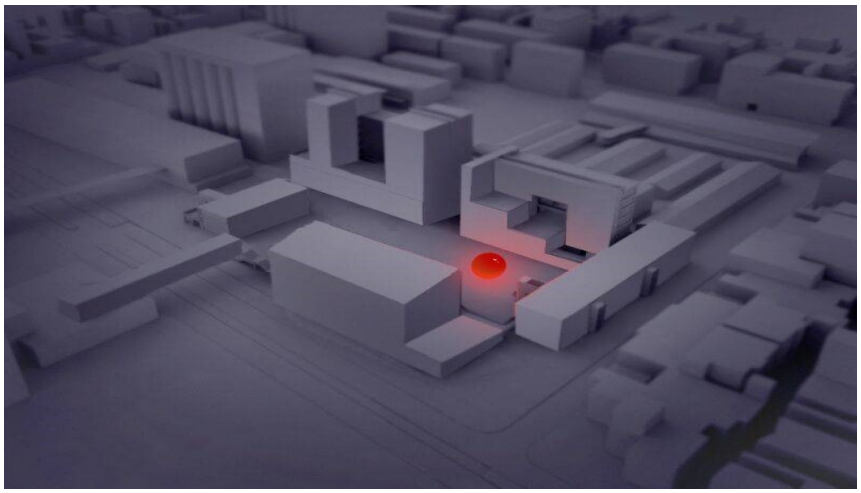
开始界面、点击 PALY 后进入游戏预状态

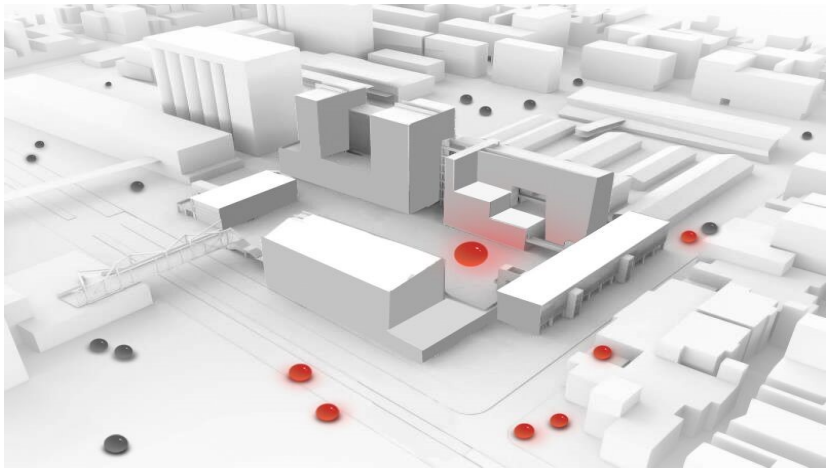
5.4.2 游戏执行测试

游戏初期，地图色调为深灰色/黑色，场景光线较暗，加体积雾，体现阴翳效果。当 boss 大黑球出现时，场景突然变暗，随着正能量的继续增加慢慢变亮。游戏胜利时，所有灰球慢慢变为红球，地图色调由灰色渐亮为白色。

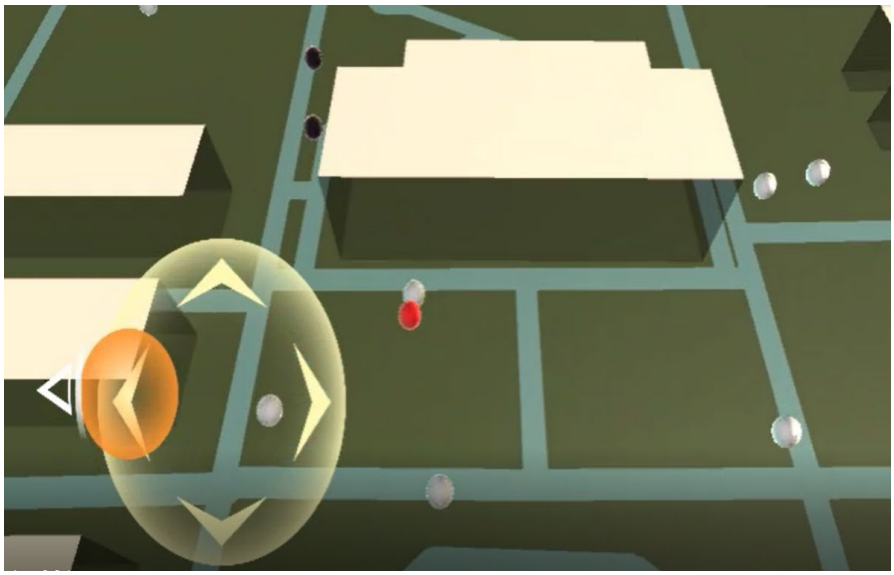
（1）游戏阶段Ⅰ、阶段Ⅱ、阶段Ⅲ色调：

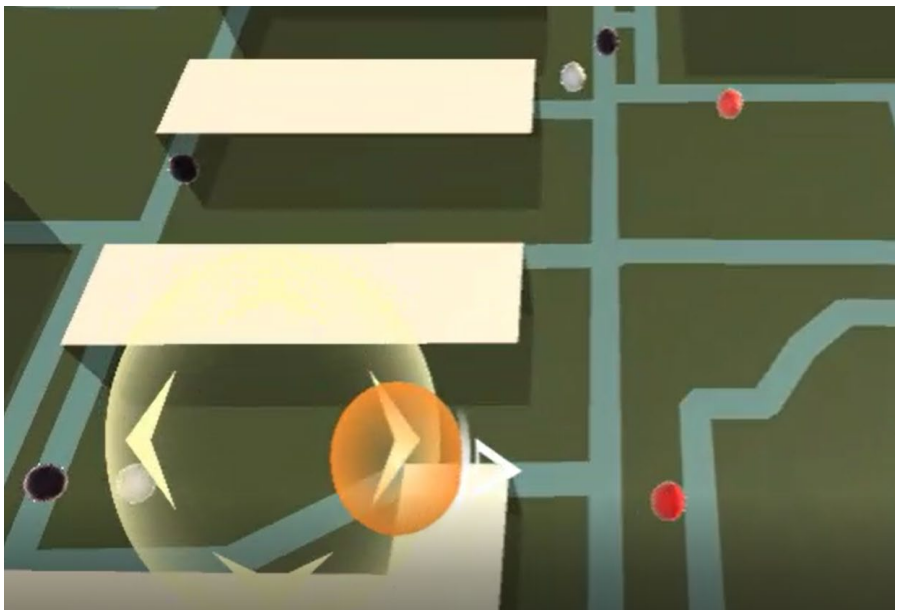




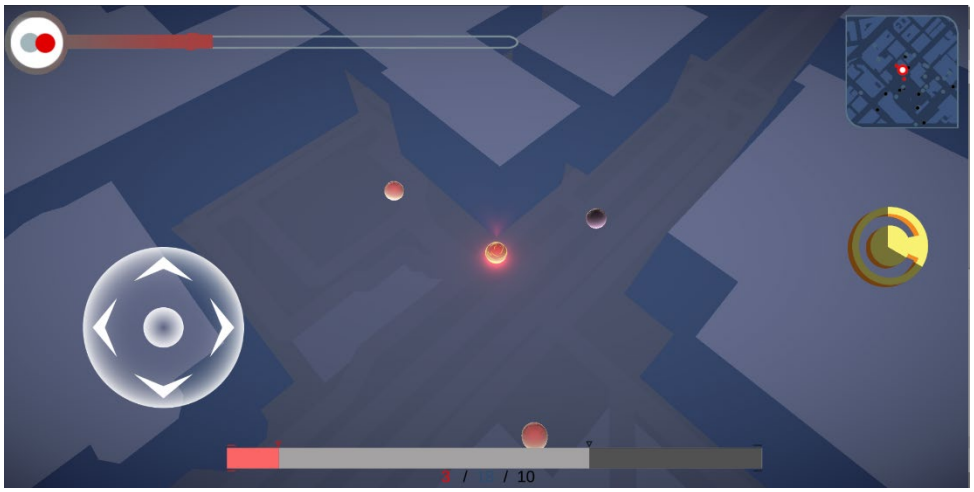


（2）小球之间碰撞改变颜色的测试，如主角碰撞小灰球，使得小灰球变小红球：

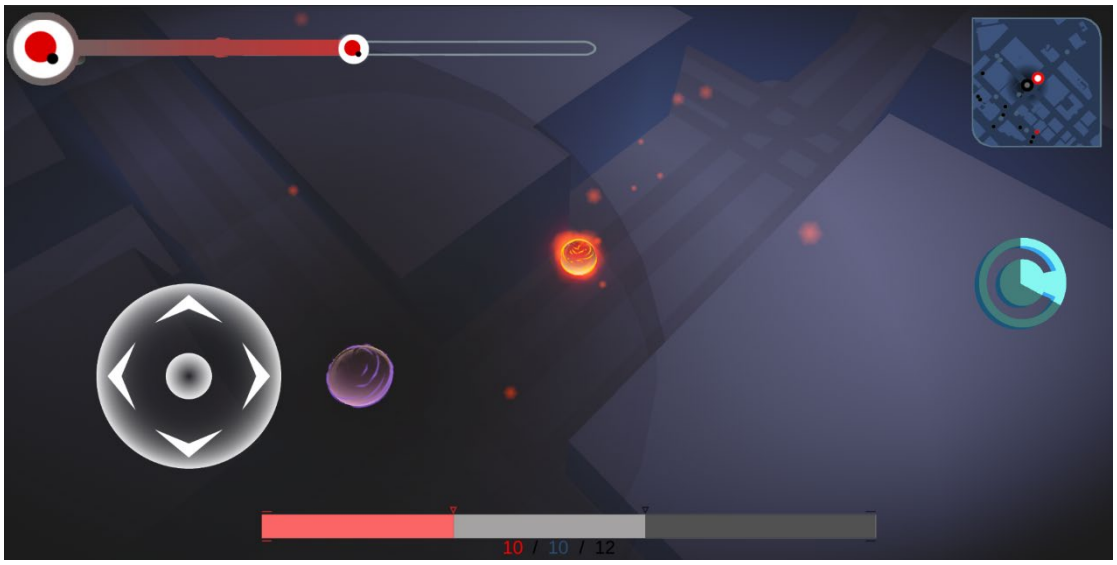




（3）主角合体前、合体时以及合体后的测试：



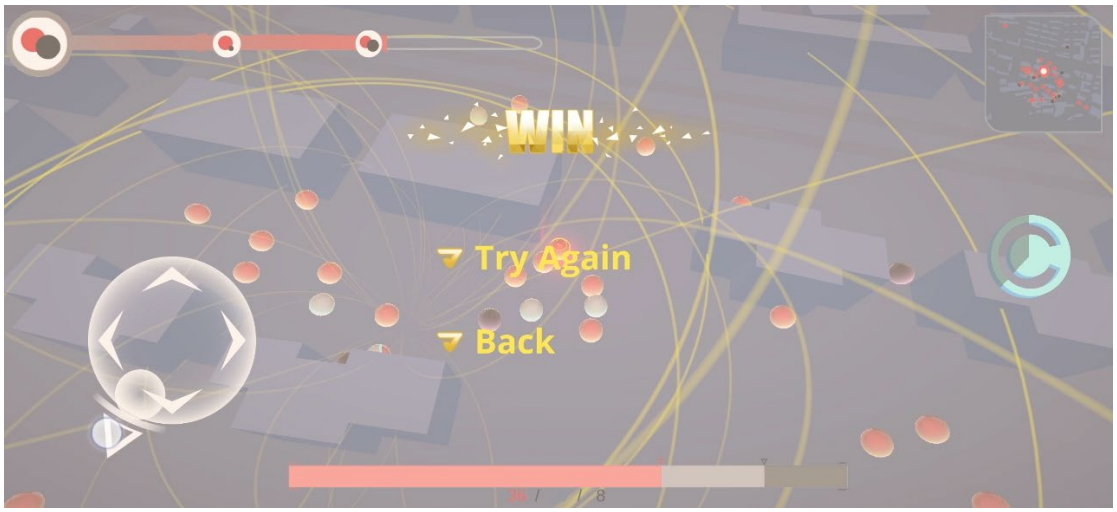
（4）主角和 boss 碰撞战斗测试：



5.4.3 游戏结束测试

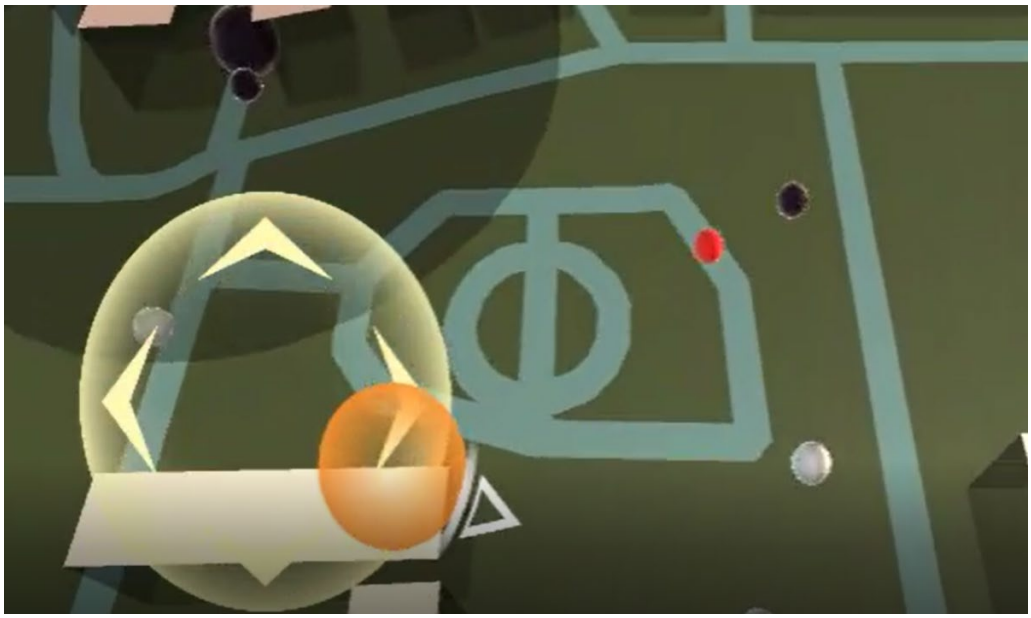
（1）游戏胜利测试：

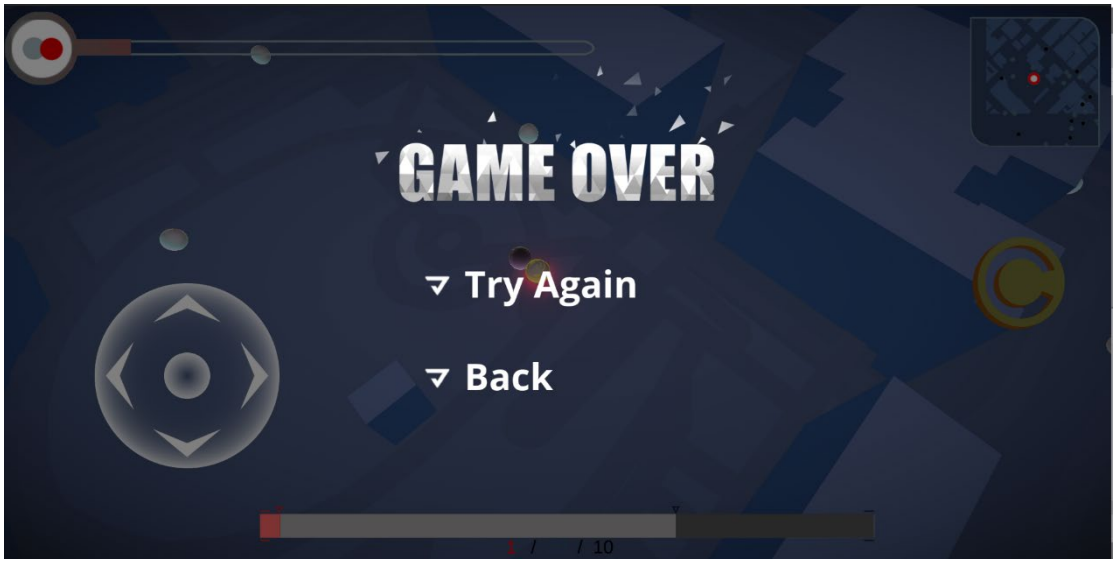
全场所有小球变为红色时，游戏胜利。



（2）游戏失败测试：

例如，主角在第一阶段碰到小黑球，主角死亡，游戏失败。





六、 个人心得体会

晋硕：

通过这次项目，我深入了解了 unity 编程开发的整个流程和基于 unity 进行游戏编程的各个实现环节，学习了视图界面的搭建、脚本的编写以及尝试了各种控件的使用，由于 UI 设计的需要也学习了 Adobe Illustrator 的使用，实现并完成了矢量图的绘制工作。

姜苏庭：

这次项目自己首先使用 mapbox 插件完成了地图的初始化，并借用插件 api 实现了小球的寻路功能；然后参考其他滚球类游戏，调整出合适的镜头和手感，同时完成了摇杆实现主角移动。

此外，除了编写部分逻辑代码，自己还负责了大部分渲染和特效。综合使用 unity 引擎的粒子系统、c#脚本和 shader 制作了特效和物体材质，渲染上的突破点主要是把 unity 工程从默认的渲染管线升级成了 urp 渲染管线，对图形学和着色器语言的运用有所提升。

廖梦恬：

掌握了 unity 游戏引擎的基础知识，对游戏的物理、渲染、UI 有了更深的理解。增强了 C#的理解和编写能力。经过团队合作后增强了协作和版本控制的能力，对一个软件的开发过程从需求到开发到测试有了更深的体会。

周华丽：

在本次课程设计中，我从用户的角度提出软件构想，并与负责开发工作的队友及时沟通、确认项目需求，同时负责对类图的绘制及整个游戏的实现说明，既充当了甲方的角色，又有了乙方的体验，明白项目开发过程中甲乙双方沟通的重要性。在整个项目开发中，通过修改、维护项目文档及最后实验报告的编写，不仅仅进一步加深了自己对本项目的理解，也让我明白整个项目的开发是一个迭代、循序渐进的过程，规范每一个过程的项目记录对后续的修改或者开发是非常重要的。

钟桔爱：

在这次实验中，我觉得最重要的是团队协作。对一个要完成完整的项目来说，团队协作极其重要，其要求每一个人要明确自己的分工，在一定时间内完成自己

的工作，最后才会有效整合在一起，效率才会很高。与此同时，学会团队沟通才能更加有效地解决在实验过程中的问题。及时沟通也是极其重要的的，不及时沟通会容易导致项目需求偏差，累积问题，最终导致需要修改的工程量会很大。

在本次项目中，我主要负责充当用户：提出游戏构想及需求；文档：完成需求规格说明书的运行环境规定、验收标准、迭代Ⅰ阶段分工部分；绘制活动图及游戏 3 阶段的说明；实验报告的第五部分测试报告以及实验报告的整合。对项目文档开发有了更深层次的了解。

沈之雯：

在本次实验中，我主要作为用户和参与文档的编写。遇到了很多的问题首先因为没有参与对代码的编写导致编写文档的过程和实际实现存在问题，其次作为用户提出的需求的切实可操作性缺少考虑。但是在团队协作的过程中慢慢解决的这些问题，不断和团队调整自己提出需求的逻辑可行性和可实现性，在编程人员的分析和对代码的注释之中，理清楚了文档编写的逻辑和文档内容衔接的流畅性。在完成实验后通过不断反思和学习，理解了关于 unity 的具体操作和操作的代码流程和文档编写应有的模板和贴合实验项目进行文档编写的技能。

七、 参考文献

[1] 普莱斯曼.软件工程：实践者的研究方法(原书第8版).北京：机械工业出版社，2016

[2] metro 项目：https://hot-dry-noodles.github.io/round-1.html

八、 团队成员在项目中的角色和具体贡献

姓名	角色	可验证贡献
晋硕	开发工程师	**编码：**实现 UI 的实时更新，包括进度条、比例条、小地图；实现 UI 的自适配； **美术资源设计：**负责所有 UI 素材的绘制，包括游戏界面的各种小图标、开始界面按钮以及 logo。
姜苏庭	开发工程师	**编码：**完成地图的初始化、镜头的移动、摇杆控制主角的移动、灰球和黑球寻路的移动； **渲染：**小球和地图的材质、主角和 boss 的环绕特效、主角合体的特效、画面色调的变化、屏幕后处理（HDR 和 bloom，ssao）。
廖梦恬	开发工程师	**编码：**完成游戏 123 阶段的过渡、游戏的胜利和失败、小球的生成和销毁、 合体的逻辑、小红球的集群算法移动逻辑、小球的碰撞逻辑、boss 的行为。
周华丽	用户、 测试工程师	**充当用户：**提出游戏构想及需求； **文档：**完成需求规格说明书的引言及系统需求概述部分；绘制类图并对其进行说明；整理和维护项目文档；制作 PPT；实验报告的第四部分项目实现；实验报告的整合与修改
		充当用户：提出游戏构想及需求；

钟桔爱	用户、 测试工程师	**文档：**完成需求规格说明书的运行环境规定、验收标准、迭代I阶段分工部分；绘制活动图及游戏 3 阶段的说明；实验报告的第五部分测试报告以及实验报告的整合
沈之雯	用户、 测试工程师	**充当用户：**提出游戏构想及需求； **文档：**完成需求规格说明书的功能需求及非功能需求部分；绘制时序图

教师评语评分

评阅人：

年　　月　　日