

圈地大作战设计文档

李京老师班第 8 组

版本号	日期	简要叙述
V1	2020-5-20	初步设计设想
V2	2020-6-21	完善概要设计
V3	2020-8-15	完成详细设计

目录

圈地大作战设计文档.....	1
1 引言.....	4
1.1 编写目的.....	4
1.2 项目背景.....	4
1.3 项目主要贡献者.....	4
1.4 术语解释.....	5
2 需求概述与分析.....	7
2.1 项目综述.....	7
2.1.1 项目背景介绍.....	7
2.1.2 项目功能及目标概述.....	7
2.1.3 项目开发工具技术.....	7
2.1.4 项目逻辑关系.....	7
2.1.5 项目整体流程.....	8
2.2 开发与运行环境.....	8
2.2.1 开发环境的配置.....	8
2.2.2 运行与测试环境的配置.....	9
2.3 具体需求分析.....	9
2.3.1 游戏界面.....	9
2.3.2 游戏规则.....	9
3 总体设计.....	11
3.1 总体设计概述.....	11
3.2 前端设计概述.....	11
3.3 后端设计概述.....	17
3.3.1 基本术语与游戏规则详述.....	17
3.3.2 游戏规则设计与开发.....	18
3.4 前后端交互设计概述.....	19
3.4.1 前后端交互框架选择.....	19
3.4.2 前后端交互方式说明.....	19
4 界面设计.....	20
4.1 初始界面设计.....	20
4.2 人数选择界面设计.....	22
4.3 难度选择界面设计.....	23
4.4 游戏房间界面设计.....	24
4.5 游戏界面设计.....	27
4.6 排行榜界面设计.....	30
4.7 商城界面设计.....	32
4.8 设置界面设计.....	33
5 游戏规则与逻辑设计.....	35
5.1 初始化设计.....	35

5.2	玩家移动设计.....	37
5.3	圈地面积设计.....	39
5.3.1	圈地逻辑流程.....	39
5.3.2	数据结构说明.....	39
5.3.3	核心算法逻辑.....	40
5.3.4	示例说明.....	40
5.4	碰撞设计.....	40
5.4.1	整体的碰撞判断流程与分类.....	40
5.4.2	玩家碰到自己的 line.....	42
5.4.3	玩家碰到其他玩家的 line.....	42
5.4.4	玩家碰撞到地图的边界.....	43
5.4.5	玩家碰撞到其他玩家的保留区.....	44
5.5	复活设计.....	45
5.6	道具使用.....	46
6	前后端交互设计.....	48
6.1	随机匹配设计.....	48
6.1.1	机制实现.....	48
6.1.2	玩家在线匹配.....	48
6.2	创建房间与加入房间设计.....	50
6.2.1	初始化设计.....	50
6.2.2	创建房间.....	51
6.2.3	加入房间.....	51
6.2.4	离开房间.....	52
6.3	房间消息发送设计.....	52
6.4	帧同步设计.....	54

1 引言

1.1 编写目的

在需求分析阶段，项目组成员已经将用户对本应用的需求做出了详尽的阐述。本阶段在需求分析的基础上，对圈地大作战应用项目的**概要设计**与**详细设计**进行了简要的描述，主要包括前端各个游戏界面的实现与衔接、后端游戏逻辑的开发与设计，以及前后端交互的方式方法。

本文档主要面向项目经理、前后端开发人员以及测试调试人员，一方面便于项目开发团队成员内部对该项目有相对统一的认识和理解，另一方面也是为了更好地实现最终的应用项目。

同时，本设计文档也可作为项目管理、评审、跟踪及其他项目相关人员在项目实施工程中检查项目工作成果的依据。

1.2 项目背景

随着信息技术与科学技术的不断发展，人类对于休闲娱乐的需求越来越高，本项目名为圈地大作战，是一种为办公人士量身定制的一款基于微信小程序的游戏应用。它以简单的规则、流畅的界面给人以良好的用户体验。

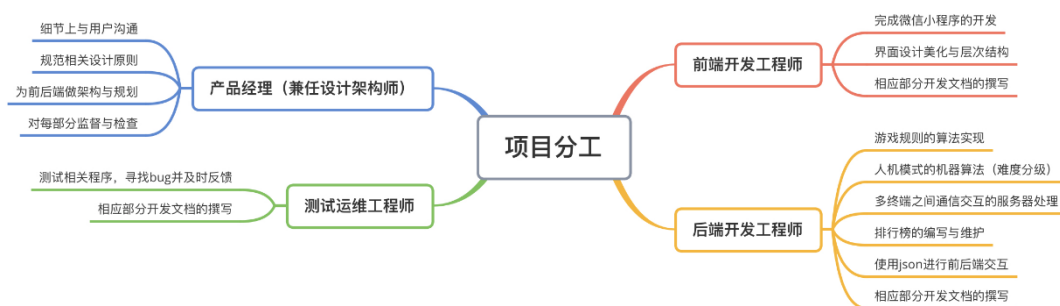
同时，作为中国科学技术大学软件工程课程的最终项目，本项目很好地适配了软件工程的起始、细化、构建、移交四个阶段，遵守了模型合适、方法合适、支持工程高质量、重视管理等基本原则，最终不仅做出了一款迎合用户需求的**优秀游戏**，也在 gitlab 内保留了应用开发时的各个版本，还编撰了包括需求分析、概要设计、详细设计等在内的一系列记录文档。

1.3 项目主要贡献者

项目的主要贡献者为：楼轶维、王家伟、芮轲、肖宇煊、桂陈幸康、张陶竞、杜艺帆、缪立君、侯懿文、常鑫鑫、叶里夏提·艾尔不力。（以上排名不分先后）小组成员的分工为（按学号顺序）：

姓名	学号	组内分工
芮轲	JL18110001	后端开发工程师
王家伟	PB17000023	后端开发工程师
桂陈幸康	PB17111576	前端开发工程师
侯懿文	PB17111592	后端开发工程师
杜艺帆	PB17111594	前端开发工程师
叶里夏提·艾尔不力	PB17111597	测试运维工程师
肖宇煊	PB17111601	后端开发工程师
楼轶维	PB17111611	产品经理
缪立君	PB17111635	后端开发工程师
张陶竞	PB17111637	前端开发工程师
常鑫鑫	PB17111649	后端开发工程师

组内分工与职能关系如下。

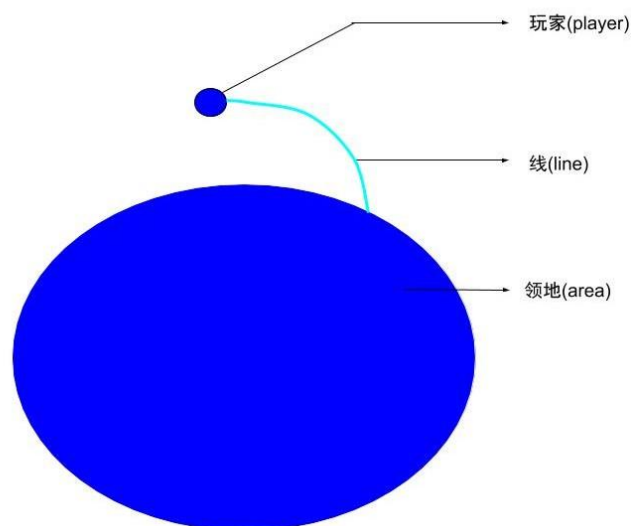


1.4 术语解释

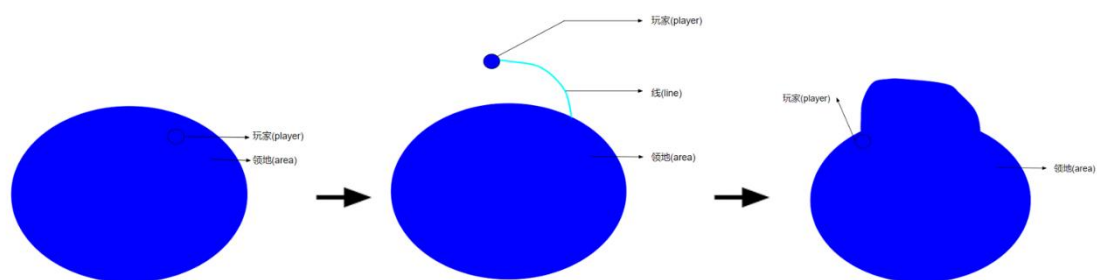
玩家(player): 在游戏内用一个圆表示, 不同玩家拥有不同颜色。

领地(area): 玩家所拥有的一块连通的地, 领地面积越大, 则得分越高。

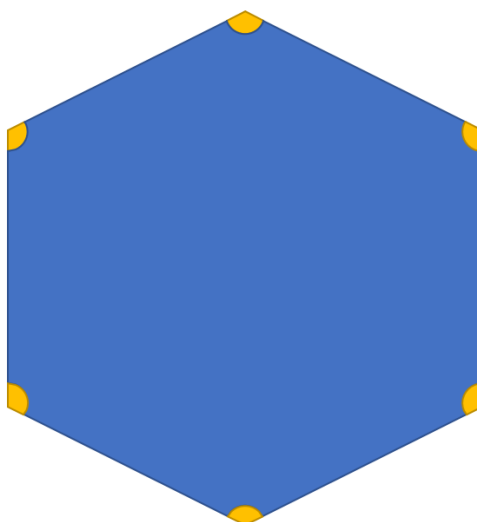
线(line): 玩家离开领地时从领地边界引出的一条线。



圈地: 玩家离开领地后又回到领地, 则线与原领地包围着的地归该玩家所有, 即该玩家完成了一次“圈地”。



保留区：每个玩家所拥有的一定面积的领地，称为保留区。游戏开始时玩家从保留地出发，保留区不会被其他玩家圈地。如下图所示黄色部分即为六个玩家各自的保留区。



死亡：该玩家及其所圈的领地消失，但保留地不变，当复活倒计时完毕或者玩家使用“立即复活”道具后该玩家复活，继续从该玩家所有的保留地出发进行圈地争夺。

击杀：当玩家触碰到敌方阵营玩家的线(line)时，会造成该敌方玩家的死亡，该过程称为玩家对敌方玩家的“击杀”。

2 需求概述与分析

2.1 项目综述

2.1.1 项目背景介绍

圈地大作战，是一款由中国科学技术大学本科生软件工程实训课程李京老师班第 8 组同学开发的基于微信小程序的娱乐游戏。它以简单的规则、流畅的界面给人以良好的用户体验，旨在让用户（玩家）工作学习之余能够减轻压力。

同时，它也作为本组同学在软件工程课程中所学的集中展示，很好地适配了软件工程的起始、细化、构建、移交四个阶段，遵守了模型合适、方法合适、支持工程高质量、重视管理等基本原则，不仅体现了本组同学协调互助、勇于探索的刻苦精神，也展现了学校和老师超前的教育理念与优秀的教育水平。

2.1.2 项目功能及目标概述

该游戏是一款玩法新颖，画面精美的休闲竞技类游戏。并以休闲娱乐放松为主要目标，在玩法、社交、个性化设置和游戏画质等方面都进行创新和升华，不仅符合年轻玩家社交和竞技需求，也满足休闲玩家需求，让用户享受娱乐与竞技的过程。

在游戏中玩家可以通过操作自己的角色扩展领地，并利用熟练的操作和技能击败与自己同场的对手，赢取最高排名。圈地大作战操作简单，上手快，而在简单之中又蕴含着丰富的竞技要素，让人不由自主的沉迷其中。游戏的公平性也非常出色，所有玩家在游戏开始时都处于同一起点，比拼的则是自己的策略与操作。

2.1.3 项目开发工具技术

项目开发使用的工具和技术主要有：Cocos Creator 游戏开发引擎、微信开发者工具等。

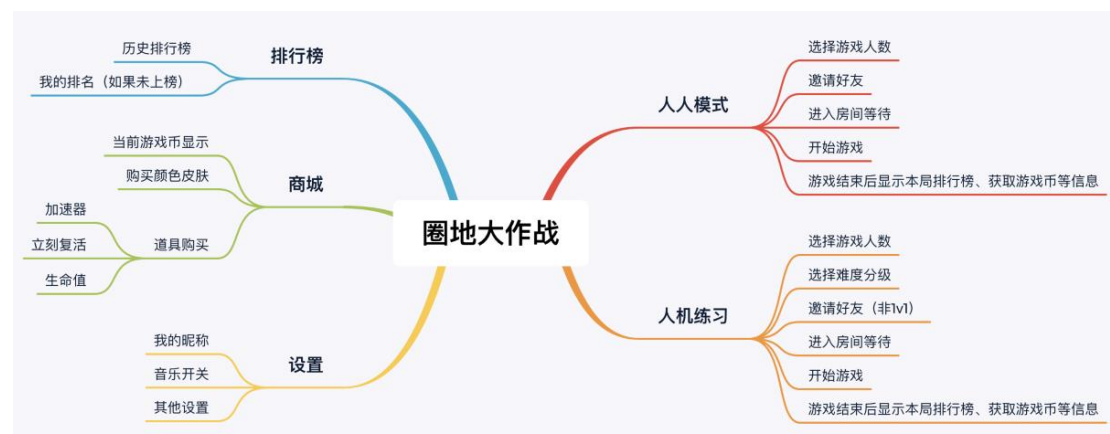
项目开发使用的语言包括 JavaScript、TypeScript 等。

2.1.4 项目逻辑关系

圈地大作战是基于微信小程序的一款联机游戏，每个用户都需要有体力值以开启游戏，每局游戏结束后会减少相应的体力值。如果体力值不够，则用户可以去商城中购买体力值，或等待一段时间（十分钟）系统自动刷新体力值后再行游戏。

每场游戏人数在 1 至 6 之间。游戏部分分为人人对抗和人机练习两个模式。在人人对抗模式中，需要两队玩家，每队 1 至 3 人。两队玩家进入游戏界面后按照游戏规则相互对抗，最终等比赛结束后再结算；在人机练习模式中，需要 1 至 3 个真人玩家组成一支队伍，系统会随机使用对应个数的人机作为对抗队伍。真

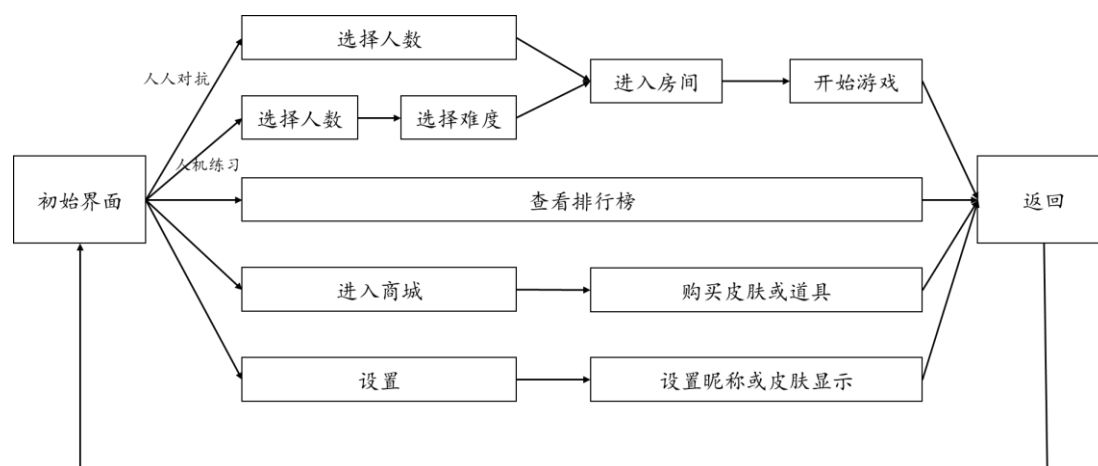
人队伍与人机队伍按照游戏规则相互对抗，最终等比赛结束后再结算。



除了人人对抗和人机练习两种模式之外，游戏还提供全局的排行榜和商城、设置等服务。在排行榜中，用户可以查看自己的全局排名。在商城中，用户可以使用游戏币购买皮肤或道具（例如加速器、立刻复活、体力值）。

2.1.5 项目整体流程

本项目按照如下流程进行。在初始界面，用户（玩家）可选择多种交互模式，包括人人对抗、人机练习两种模式的游戏，或是查看排行榜、商城购物、设置相关信息等，最终返回至初始界面。



2.2 开发与运行环境

2.2.1 开发环境的配置

本项目采用的开发环境采用

- 微信开发者工具（采用稳定版 Stable Build 1.03.2006090）
- Cocos Creator（v2.4.2，或 v2.3.1 以上版本）

2.2.2 运行与测试环境的配置

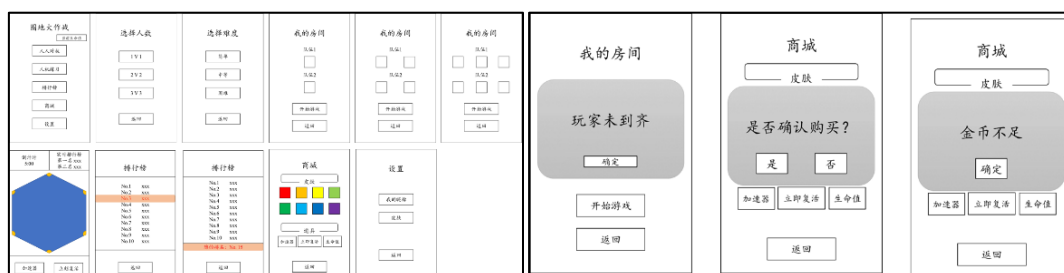
本项目采用的运行与测试环境采用

- 微信开发者工具（采用稳定版 Stable Build 1.03.2006090）
- Cocos Creator（v2.4.2，或 v2.3.1 以上版本）

2.3 具体需求分析

2.3.1 游戏界面

根据本组成员的思考与讨论，为完成相对完整的游戏实现，至少需要 11 个界面和 3 个弹窗，缩略图如下所示。



根据相对应的需求，有各种不同逻辑的界面转换，转换逻辑与 2.1.5 节中的项目整体流程类似，在此不做赘述。

每一个界面的具体需求，可以参考《需求分析-李京老师班第 8 组》。同时，这部分需求与简单的初步设计设想在 3.2 节前端设计概述中有较为详细的描述。

2.3.2 游戏规则

本项目作为一款新颖的游戏，具有较为简单的游戏规则与逻辑，具体可划分为初始化、圈地过程和击杀对手与复活三个部分，下面将对此进行叙述。

初始化：

从游戏房间界面进入游戏界面后，即开始游戏，每个玩家从六边形的相对保留区出发。如果是 1V1 模式，对战的两个玩家从上下两个保留区出发；如果是 2V2 模式，对战的四个玩家，一队从左边的两个保留区出发，另一队从右边的两个保留区出发；如果是 3V3 模式，一队的三个玩家从上面的三个保留区出发，另一队从下面的保留区出发。

保留区是每个玩家的私属区域，其边界对于其他玩家来说就如同游戏区域的边界，无法突破，只有保留区的主人才能回去。游戏是从这里开始，死亡后的复活也是从这里开始。

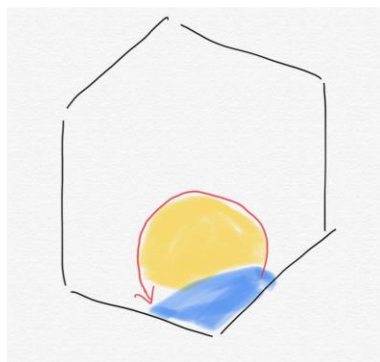
圈地过程：

玩家从保留区出发后，通过控制代表自己的方块，在游戏区域内向四周移动，

走出一条路径并留下痕迹（曲线）；当玩家走出的痕迹与原本圈到的区域有交点时，即玩家在游戏区域内圈了一块小区域，此时该玩家新增一块已圈之地。

玩家圈到的地，可以从未被圈过的地，也可以是对手或队友已经圈到的地，只要不是自己曾经圈到且仍属于自己的地，都可以算作新圈的地。

举个具体的例子，某玩家已经圈出了蓝色的区域，并按下图中红色路径行走，与曾圈过且仍属于自己的蓝色的区域构成了一个圈，则圈内的黄色部分记作新圈的地，标记为黄色。实际游戏中，蓝色和黄色以及红色的部分都使用同一种皮肤颜色标记，这里是为了区分说明，所以使用了不同的颜色。

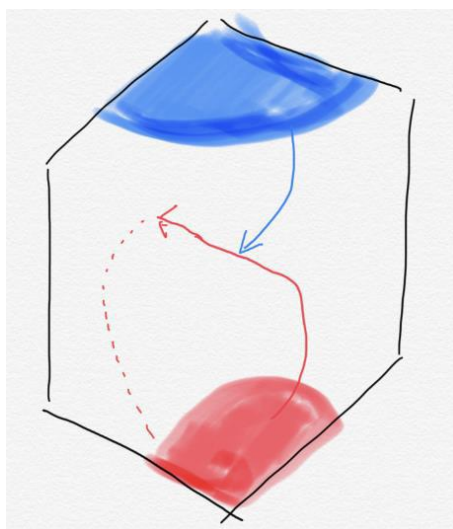


击杀对手与复活：

当玩家 A 在向外圈地时，走出的路径被其他敌方玩家 B 触碰到，则玩家 A 死亡，10 秒钟后玩家 A 复活，重新从玩家 A 的保留区出发，玩家 A 原本圈过的区域仍记作 A 的（但死亡期间被其他玩家圈了的区域就属于其他玩家）。

如果玩家 A 在死亡后立刻使用道具“立即复活”，则可以不等 10 秒的死亡时间直接复活。

举个具体的例子，红色区域是玩家 A 圈的，蓝色区域是玩家 B 圈的，当玩家 A 已经沿红色实线路径走出一段，并想向红色虚线路径围成一块较大的区域，此时蓝色玩家经过红色玩家走出的路径，即可消灭玩家 A，此时玩家 A 即刻进入自己的保留区等待 10 秒死亡时间后才能出来。如果玩家 A 有“立即复活”道具，可以使用，使用后直接从保留区出发继续游戏。



3 总体设计

3.1 总体设计概述

本项目的人员构成与项目划分类似，大致分为前端界面部分、后端游戏规则与逻辑部分、前后端交互实现部分。

前端界面部分主要负责游戏界面的设计与美工，不仅包括各个界面的设计以及界面间跳转逻辑的实现，也包括收集玩家的操作信息（包括玩家的模式选择、游戏过程中使用的道具、控制摇杆的方向等）。

后端游戏规则与逻辑部分主要负责将游戏规则以代码的形式体现出来，同时根据前端收集到的玩家操作信息进行应变，以达到对玩家位置、状态等信息的维护。

前后端交互实现部分主要负责通过借助腾讯云小游戏联机对战引擎和 Cocos Creator 以实现前后端信息与数据交互。

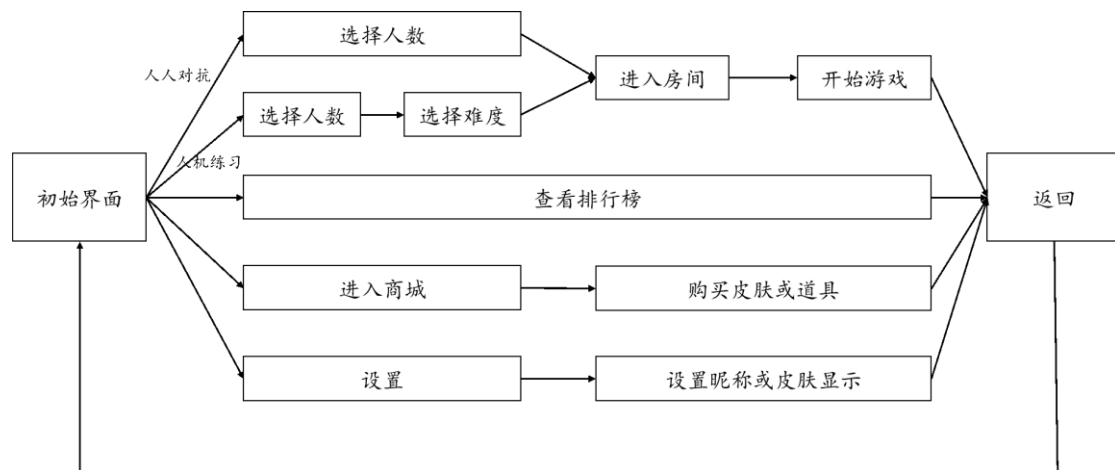
下面第 3 章将以此为序，简要介绍三部分的概要设计；其后的第 4、5、6 章将分别详尽地叙述这三部分的详细设计。

3.2 前端设计概述

圈地大作战的界面开发工具采用微信官方提供的“微信开发者工具”，并结合 Cocos Creator。在 Cocos Creator 用 Javascript 可以较简单地实现界面跳转相关的按钮。整个界面的布局，通过设置每个组件的坐标进行调整，在调整的同时，就可以看到界面显示的最终结果，操作方便。设计完成一部分界面之后，可以直接进行预览，并选择手机型号，查看最终结果显示的比例大小，调试方便。使用 Cocos Creator 的构建发布功能，可以直接构建并导入项目到微信开发者工具，之后可以生成二维码，在手机端预览。

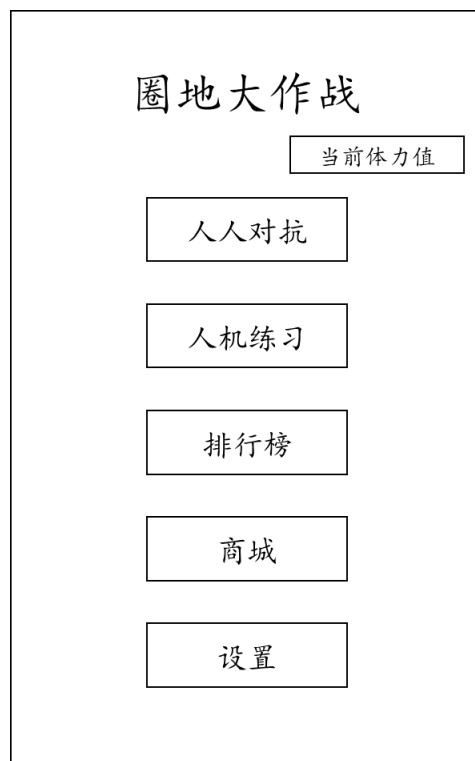
总体来看对界面设计开发，结合使用微信开发者工具和 Cocos Creator，有以下优点：简化界面开发过程，直观显示界面，方便设计；文件结构简单，图片素材、场景、声音素材、脚本文件等存放在不同文件夹，可以用用户自定义，其余文件结构由 Cocos Creator 管理；由于 Cocos Creator 用于开发游戏多于微信开发者工具，可以搜索到更多资料用于开发游戏参考。

圈地大作战计划共有 11 个界面和 3 个弹窗，所有的界面跳转流程图如下：



以下是具体界面的简易设计及介绍。

初始界面是用户点击进入应用之后所在的界面（受游戏邀请用户除外），主要实现菜单的功能。



如上图，初始界面包括应用名称、玩家当前体力值显示和 5 个按钮。玩家当前体力值是玩家能否开启游戏的判断依据，每次开启游戏，将对体力值进行扣除；体力值会随时间恢复；当前体力值不足时，无法开启游戏，玩家此时可以选择等待或者进入商城购买体力。通过点击初始界面的 5 个按钮，玩家对应会进入“人对抗”、“人机练习”、“排行榜”、“商城”和“设置”界面。

在初始界面点击“人对抗”按钮，玩家将进入选择人数界面，在此界面，玩家可以选择进行游戏的双方人数，共有“1 V 1”、“2 V 2”和“3 V 3”三个选择按钮。如下图所示。

选择人数

1 V 1

2 V 2

3 V 3

返回

在初始界面点击“人机练习”按钮，并选择完游戏难度，玩家也会进入选择人数界面，此时的选择人数界面和前一种情况相同。在选择人数界面，点击选择人数按钮之后会跳转至游戏房间界面。选择人数界面底部的返回按钮用于返回初始界面。

当玩家在初始界面点击“人机练习”按钮，会进入选择难度界面，在此界面对游戏难度进行选择。

选择难度

简单

中等

困难

返回

难度分为“简单”、“中等”和“困难”三种，对应三个选择按钮。点击此界面底部的继续按钮将前进至选择人数界面；如果需要重新选择难度，则只能从选择人数界面点击返回按钮，返回初始界面，再选择“人机练习”，进入难度选择界面。

在人数选择界面，点击任意一种人数模式按钮，将进入游戏房间界面，按照人数不同，有三种房间界面，区别仅为游戏人数不同。玩家均会以头像表示，创建房间的玩家进入此界面时，除玩家本人，其余头像位置均为加号“+”图案，点击加号，可以发送房间链接给好友，邀请好友进行游戏。好友加入游戏，加号变为显示加入的好友。当由邀请加入房间的玩家点击返回按钮，在房间创建者的房间界面，此玩家的头像变回加号，且此玩家返回至初始界面。当房间创建者点击返回按钮，房间被解散，此房间所有玩家返回至初始界面。



在游戏房间界面，玩家人数达到选择人数后，只有房间创建者可以点击“开始游戏”按钮，并跳转至游戏界面。玩家人数未达到选择人数，点击“开始游戏”按钮，弹出“玩家未到齐”的弹窗。

对于“返回”按钮，房间开创者（队伍 1 的玩家 1）按下，则解散整个房间，所有已进入房间的玩家都返回至初始界面；普通玩家（非队伍 1 的玩家 1）按下，则该玩家离开房间，返回初始界面，其他玩家依然处于房间内，待玩家到齐后可开始游戏，即进入游戏界面。

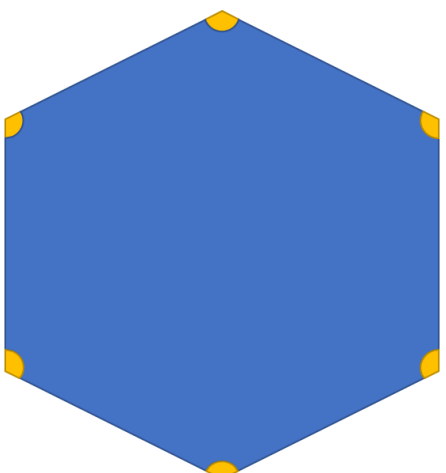


当进入游戏界面，游戏正在进行时，左上角显示倒计时时间，倒计时 5 分钟到后游戏自动结束，并结算每位玩家圈过的地。右上角显示实时排行榜，展示实时双方玩家分数及排名。

中间大部分为游戏地图，玩家在此对方块进行移动，从而完成圈地的过程。主游戏区是六边形，每个玩家从各自黄色的保留区出发，按照游戏规则向外圈地。

下部分显示游戏道具，包括加速器（增加玩家移动速度 30%，且持续 10 秒）和立即复活（“死亡”玩家不用等待 10 秒，立即“复活”回到游戏）。游戏时间倒计时结束，结束游戏，并显示本局排行、获得奖励等信息的清算弹窗，点击弹窗的返回按钮，玩家回到初始界面。

本局排行榜如果刷新历史（全局）排行榜的最好成绩，则更新历史（全局）排行榜。游戏币可在商城中使用，用于购买皮肤或道具等。在该弹窗按下“返回”按钮后，所有玩家进入初始界面。

倒计时 5:00	实时排行榜 第一名 xxx 第二名 xxx
	
<div>加速器</div> <div>立刻复活</div>	

倒计时 5:00	实时排行榜 第一名 xxx 第二名 xxx
<div>游戏结束</div> <div>本局排行榜</div> <div>本局获得游戏币：20金币</div> <div>返回</div>	
<div>加速器</div> <div>立刻复活</div>	

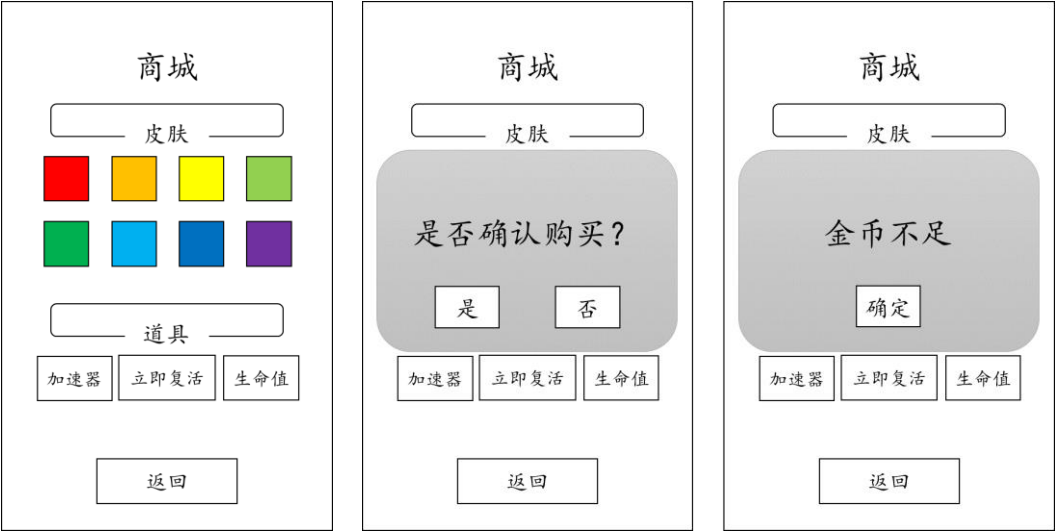
在初始界面点击“排行榜”按钮，会跳转至排行榜界面。在排行榜界面，显示全局分数最高的 10 名玩家。如果当前玩家在前 10 名之内，则高亮显示该玩家的分数；如果当前玩家不在前 10 名之内，则在最底部一行单独显示该玩家排名。界面的返回按钮用于返回初始界面。

排行榜	
No.1	xxx
No.2	xxx
No.3	xxx
No.4	xxx
No.5	xxx
No.6	xxx
No.7	xxx
No.8	xxx
No.9	xxx
No.10	xxx
返回	

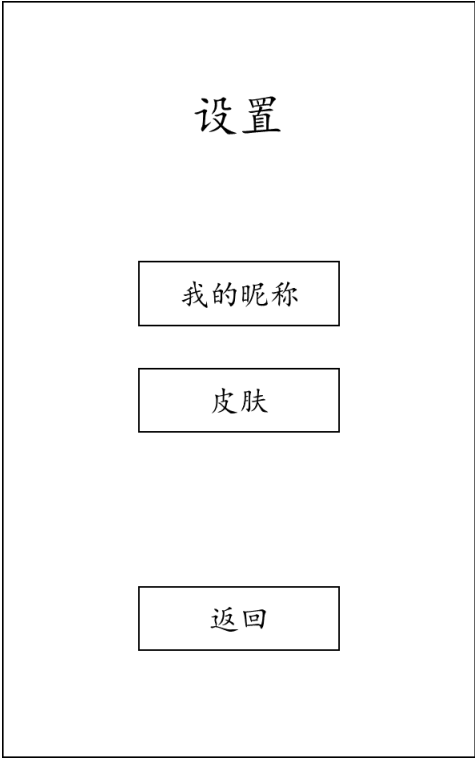
排行榜	
No.1	xxx
No.2	xxx
No.3	xxx
No.4	xxx
No.5	xxx
No.6	xxx
No.7	xxx
No.8	xxx
No.9	xxx
No.10	xxx
您的排名：No. 15	
返回	

在初始界面点击“商城”按钮，会跳转至商城界面。在商城可以利用游戏获得奖励虚拟货币购买游戏皮肤和道具。在购买时，会弹出确认购买的弹窗，弹窗

内点击“是”，确认购买。如果虚拟币不足，则会在确认购买后，弹出“金币不足”的弹窗，显示未成功购买。点击最下方的返回按钮，返回至初始界面。



在初始界面点击“设置”按钮，跳转至设置界面。在设置界面，玩家可以对自己的昵称和游戏时的皮肤进行设置。皮肤部分显示所有已购买皮肤。



3.3 后端设计概述

3.3.1 基本术语与游戏规则详述

基本术语参见 1.4 节术语解释，重点包括玩家(player)、领地(area)、线(line)、

圈地、保留地、死亡、击杀等。

游戏规则大致可分为初始化、圈地过程、击杀对手与复活等三个部分，下面依次介绍。

初始化：

从游戏房间界面进入游戏界面后，即开始游戏，每个玩家从六边形的相对保留区出发。如果是 1V1 模式，对战的两个玩家从上下两个保留区出发；如果是 2V2 模式，对战的四个玩家，一队从左边的两个保留区出发，另一队从右边的两个保留区出发；如果是 3V3 模式，一队的三个玩家从上面的三个保留区出发，另一队从下面的保留区出发。

保留区是每个玩家的私属区域，其边界对于其他玩家来说就如同游戏区域的边界，无法突破，只有保留区的主人才能回去。游戏是从这里开始，死亡后的复活也是从这里开始。

圈地过程：

玩家从保留区出发后，通过控制代表自己的方块，在游戏区域内向四周移动，走出一条路径并留下痕迹（曲线）；当玩家走出的痕迹与原本圈到的区域有交点时，即玩家在游戏区域内圈了一块小区域，此时该玩家新增一块已圈之地。

玩家圈到的地，可以是从未被圈过的地，也可以是对手或队友已经圈到的地，只要不是自己曾经圈到且仍属于自己的地，都可以算作新圈的地。

击杀对手与复活：

玩家从保留区出发后，通过控制代表自己的方块，在游戏区域内向四周移动，走出一条路径并留下痕迹（曲线）；当玩家走出的痕迹与原本圈到的区域有交点时，即玩家在游戏区域内圈了一块小区域，此时该玩家新增一块已圈之地。玩家圈到的地，可以是从未被圈过的地，也可以是对手或队友已经圈到的地，只要不是自己曾经圈到且仍属于自己的地，都可以算作新圈的地。

3.3.2 游戏规则设计与开发

开发语言选择：

经过项目成员线上讨论，决定使用 **TypeScript** 作为项目后端使用的语言，用于完成游戏规则的设计与开发，主要原因有：

- 项目使用的 Cocos Creator 游戏引擎支持 TypeScript；
- TypeScript 是 JavaScript 的超集，JavaScript 代码可以与 TypeScript 一同工作，同时也可以使用编译器将 TypeScript 代码编译为 JavaScript 代码；
- TypeScript 引入了通过类型注解提供编译时的静态类型检查，并且数据要求带有明确的类型，使得代码更容易维护；
- TypeScript 引入了 JavaScript 中没有的“类”概念，并且还引入了模块的概念，可以把声明、数据、函数和类封装在模块中

游戏规则开发说明：

编写一个 `Game` 类，在每局游戏开始时实例化一个 `game` 对象，其中保存了本局游戏相关的信息，包括各个玩家的信息。同时，`Game` 类还提供一些方法，以供在前后端交互时调用。前端获取了各个玩家的操作，将其作为参数传递给 `game` 对象，调用相应的方法。在每次调用后，`game` 都会根据参数更新各个玩家和 `game` 的属性，完成移动、圈地等操作，并且更新地图信息，最后将执行结果返回前端。

3.4 前后端交互设计概述

3.4.1 前后端交互框架选择

经过项目成员线上会议讨论，决定选择 Cocos Creator 游戏引擎作为前后端交互的框架，主要原因有：

- Cocos Creator 是一个完整的游戏开发解决方案，包含了轻量高效的跨平台游戏引擎，以及能让用户更快速开发游戏所需要的各种图形界面工具；
- Cocos Creator 编辑器提供面向设计和开发的两种工作流，提供简单顺畅的分工合作方式，也集成了前后端交互的方式和接口；
- Cocos Creator 目前支持发布游戏到 Web、iOS、Android、各类"小游戏"、PC 客户端等平台，真正实现一次开发，全平台运行。

3.4.2 前后端交互方式说明

Cocos Creator 实现游戏逻辑的运行与交互是通过为场景中的节点挂载各种内置组件和自定义脚本组件。包括从最基本的动画播放、按钮响应，到驱动整个游戏逻辑的主循环脚本和玩家角色的控制。几乎所有游戏逻辑功能都是通过挂载脚本到场景中的节点来实现的。

在 Cocos Creator 中组件脚本用于前后端交互的模块主要是“`properties`”和“`update`”以及一些预置的事件接口，`properties` 功能是为前端的节点设计属性，比如移动速度、移动方式等等属性用于前端的显示，`update` 模块在场景加载后就会每帧调用一次，我们一般把需要经常计算或及时更新的逻辑内容放在这里。这样后端通过设计游戏规则和逻辑，监听全局事件或者进行每帧的更新来更新组件的属性，即可达到前后端交互的功能。

4 界面设计

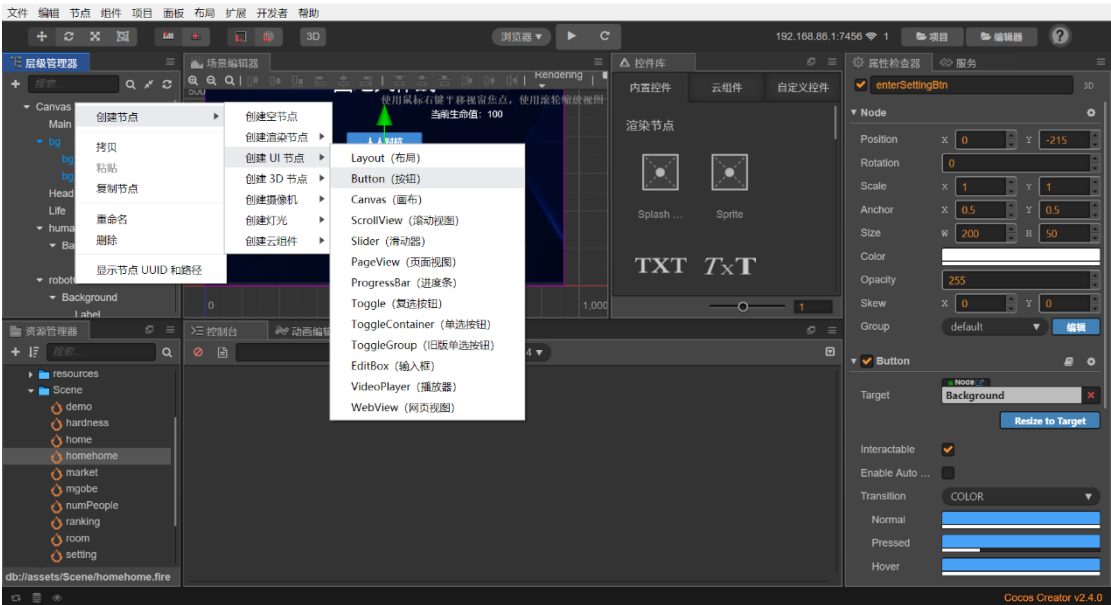
4.1 初始界面设计

在引入了游戏联机对战引擎之后，我们对游戏的所有界面进行了重新设计和调整。最终初始界面设计如下：



界面顶部正中间显示为小游戏名称，右侧为玩家剩余的生命值，生命值用于创建房间和加入游戏。界面正中间显示小游戏所有功能的入口按钮，点击对应按钮实现跳转界面功能。

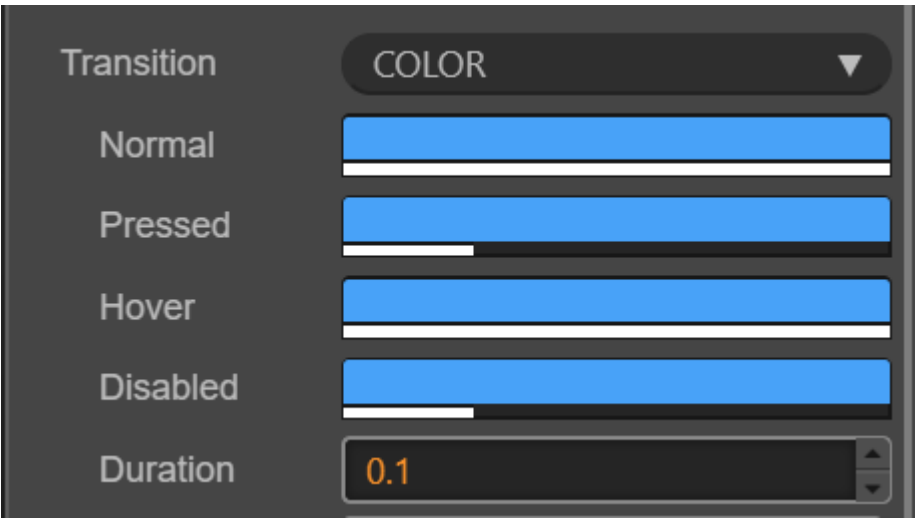
在实现初始界面时，使用 Cocos Creator，编辑器版本为 Creator 2.4.0。小游戏名称和生命值显示为 Label 类型组件，剩余组件为按钮类型。增加按钮类型组件做法如下图，增加 Label 类型组件方法类似。



增加组件后，对组件的具体参数在属性检查器中设置，得到最终的效果。
对于需要实现界面跳转的按钮组件，还需要增加脚本。以设置按钮为例，添加在此按钮的 js 脚本主要内容如下：

```
1. changeScene() {  
2.     cc.director.loadScene("setting"); //setting 为设置界面的名称  
3. }
```

点击按钮的动画在如下部分设置：



其中 Transition 用于选择按钮状态变化的过度方式，此处 COLOR 指按下按钮之后仅发生颜色变化，Normal 表示按钮在不按动情况下的颜色，Pressed 为按下按钮颜色的变化，图中颜色代表选择的颜色，颜色下方白色或黑白条形指透明度，全白为不透明。

为使界面统一整齐，对所有组件的字体大小、组件大小进行一定规定，初始界面的所有组件具体参数总结如下表：

组件名称	参数(Position 原点为画布中心)
背景	Size: 960(W), 640(H)
标题 Head	Font Size: 55; Font Family: Arial; Position: x=0, y=225

按钮 Button	Size: 200(W), 50(H); Background Color: #48A2F8; Position: 最上部的按钮为 x=0, y=65, 之后按钮之间间隔为 y 方向 70; Font Family: PingFangSC-Regular, sans-serif
生命值 Label	Font Size: 25, Font Family: Arial; Position: x=220, y=140

4.2 人数选择界面设计



人数选择界面组成如上图所示，最上方一排中间为界面标题，指示界面主要功能；左侧为返回按钮，用户可以返回初始界面修改之前的选项；界面正中是 1v1，2v2，3v3 三个按钮，用户必须且只能选择其中一项来确定游戏的联机人数，选择完毕后，点击下方的继续按钮，提交选择结果并跳转到下一界面。

返回按钮附加的 ts 脚本主要内容如下：

```
1. start () {
2.     //backScene 为要返回的界面，backBtn 为返回按钮
3.     if (!!this.backScene) {
4.         this.backBtn.on(cc.Node.EventType.TOUCH_START, () => cc.director.loadScene(this.backScene));
5.         //点击返回按钮，触发 loadScene，返回初始界面
6.     }
7. }
```

继续按钮则附加一个前往游戏界面的 js 脚本，主要内容如下：

```
1. changeScene() {
2.     cc.director.loadScene("game");
3. }
```

通过加载 game 界面，实现跳转。

对此界面所有组件的规范化参数统计如下：

组件名称	参数
背景	Size: 960(W), 640(H)

标题 Head	Font Size: 55; Font Family: Arial; Position: x=0, y=225
选项按钮 Button	Size: 200(W), 60(H); Background Color: #48A2F8; Position: 最上部的按钮为 x=0, y=65, 之后按钮之间间隔为 y 方向 85; Font Family: PingFangSC-Regular, sans-serif
继续按钮 Button	Size: 200(W), 60(H); Background Color: #006CFA; Font Family: PingFangSC-Regular, sans-serif
返回按钮 Sprite	箭头组件: Size: 20(W), 20(H); Color: #BBBBBB; 文字组件: Font Size: 25; Font Family: PingFangSC-Regular, sans-serif; Color: #BBBBBB

4.3 难度选择界面设计

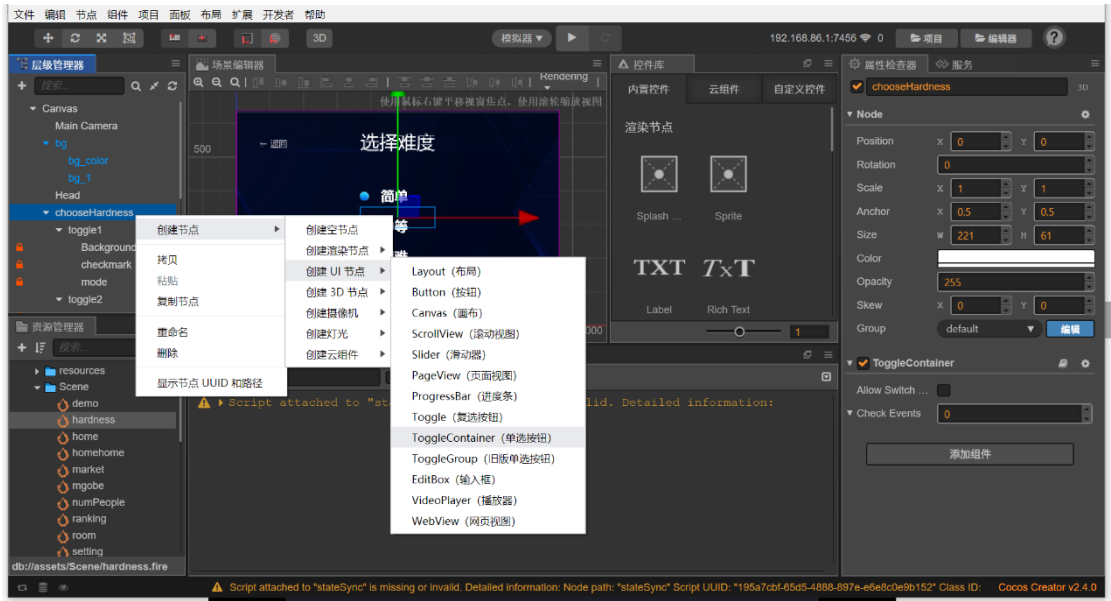


难度选择界面组成如上图所示，最上方一排中间为界面标题，指示界面主要功能——用户在选择人机对抗游戏模式后，可在此界面自定义游戏难度，后端会根据用户选择调整算法，使用户获得不同的游戏体验；左侧为返回按钮，用户可以返回初始界面修改之前的选项；界面正中是 *简单*，*中等*，*困难* 组成的三选一按钮组，用户在三者中择其一，选择完毕后，点击下方的 *继续* 按钮，提交选择结果并跳转到下一界面。

难度选择界面的继续按钮附加的 js 脚本实现了跳转至人数选择界面的功能（只有人机练习模式才有难度选择）：

```
1. changeScene() {
2.   cc.director.loadScene("numPeople");
3.   //numPeople 为人数选择界面的名称
4. }
```

难度选择的按钮由一个三选项的单选按钮 `ToggleContainer` 实现，添加 `ToggleContainer` 的方法如下：



由于默认的蓝色选中按钮最终显示会有所偏移，因此将所有蓝色选中按钮位置设置为 $x=-1, y=-1$ （坐标原点为按钮的灰色背景的中心）。

对此界面所有组件的规范化参数统计如下：

组件名称	参数
背景	Size: 960(W), 640(H)
标题 Head	Font Size: 55; Font Family: Arial; Position: $x=0, y=225$
单选按钮 ToggleContainer	按钮: Size: 32(W), 32(H); 文字: Font Size: 40; Font Family: Arial; 第一个按钮位置为 $x=-100, y=65$ （原点为整个界面背景的中心），按钮间距为 y 方向 90
继续按钮 Button	Size: 200(W), 60(H); Background Color: #006CFA; Font Family: PingFangSC-Regular, sans-serif
返回按钮 Sprite	箭头组件: Size: 20(W), 20(H); Color: #BBBBBB; 文字组件: Font Size: 25; Font Family: PingFangSC-Regular, sans-serif; Color: #BBBBBB

4.4 游戏房间界面设计



游戏房间界面组成如上图所示，最上方一排中间为界面标题，左侧为返回按钮，用户可以返回初始界面修改之前的选项。

界面正中的 *初始化* 按钮，首次进入房间，用户点击该按钮进行 SDK 初始化，初始化后该按钮失焦，不再相应用户点击，即用户只需初始化 SDK 一次。

```

1. // 初始化按钮相应设置
2. initView() {
3.     if (!Util.isInited()) {
4.         this.setEnableButtons(false);
5.     } else {
6.         this.initButton.interactable = false;
7.         this.setEnableButtons(true);
8.     }
9. }

```

初始化后 *创建房间*，*随机匹配*，*加入房间* 按钮由失焦状态获取焦点，开始相应用户点击事件。

```

1. // 切换*创建房间*，*随机匹配*，*加入房间*按钮显示状态
2. setEnableButtons(isEnabled: boolean) {
3.     this.createRoomNode.setEnable(isEnabled);
4.     this.matchNode.setEnable(isEnabled);
5.     this.joinRoomNode.setEnable(isEnabled);
6. }

```

用户点击 *创建房间* 按钮，后端接受点击事件，SDK 创建房间，用户随后进入房间，即跳转到游戏界面。

```

1. // 相应创建房间按钮点击事件，调用 SDK 创建房间
2. onCreateRoomNodeClick() {
3.     if (!Util.isInited()) {
4.         return Util.appendLog("请先初始化 SDK");
5.     }
6.     this.setEnableButtons(true);

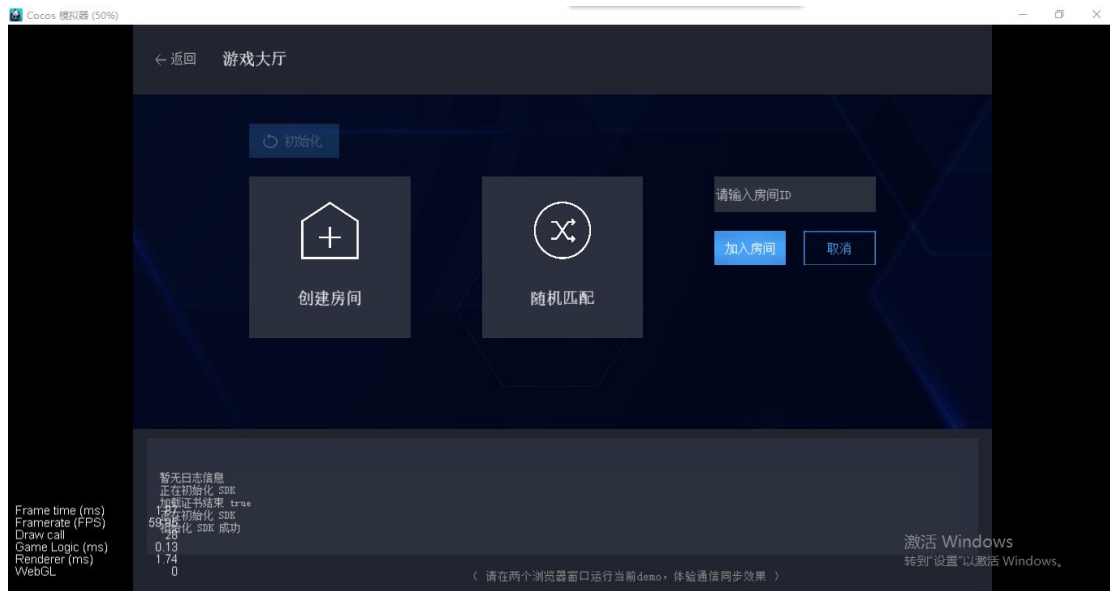
```

```

7.      !this.lockSubmit && this.createRoom(); // creatRoom:SDK 创建房间
      函数
8.      }

```

用户点击 **加入房间** 按钮，该按钮状态切换，用户需要在弹出的输入框中填写想加入的房间号，点击 **加入房间** 按钮进行提交，只有提交正确的房间号方可加入房间，点击 **取消** 用户可以放弃加入房间操作，界面切换为原界面，按钮状态切换后界面如下：



```

1. // 切换*加入房间*按钮显示状态
2. onJoinRoomNodeClick() {
3.     if (!Util.isInited()) {
4.         return Util.appendLog("请先初始化 SDK");
5.     }
6.     this.setEnableButtons(true);
7.     this.joinRoomNode.showInput(); // 显示隐藏的输入框，加入房间和
      取消按钮
8. }

```

用户向后端提交房间号，调用 SDK 加入房间：

```

1. this.joinRoomNode.onSubmit = (roomId: string) => !this.lockSubmit &&
      this.joinRoom(roomId);

```

用户点击 **随机匹配** 按钮，分为两种条件进行响应，一是使用默认 `gameId`，则直接使用默认匹配 `Code` 发起匹配，调用 SDK 随机匹配；二是使用自定义 `gameId`，则需要输入匹配 `Code`，此处会弹出输入框，提交和取消按钮，该按钮的切换方式同 **加入房间** 按钮，在此不赘述；

```

1. // 两种随机匹配条件下的响应方式
2. onMatchNodeClick() {
3.     .....
4.
5.     this.setEnableButtons(true);
6.

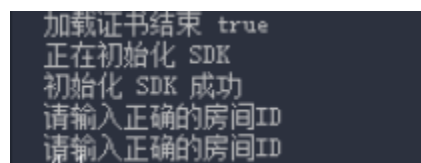
```

```

7.      if (global.gameId === configs.gameId) {
8.          // 如果使用默认 gameId, 则直接使用默认匹配 Code 发起匹配
9.          !this.lockSubmit && this.matchPlayers(configs.matchCode);
10.     } else {
11.         // 如果使用自定义 gameId, 则需要输入匹配 Code
12.         this.matchNode.showInput();
13.     }
14. }

```

最下方的输出框用于显示日志, 提示操作完成或操作错误, 如用户首次点击了初始化按钮, 会显示“正在初始化”, 完成后会显示“初始化完成”; 又如用户申请加入房间输入错误的房间 ID, 会提示“请输入正确的房间 ID”, 如下图所示:



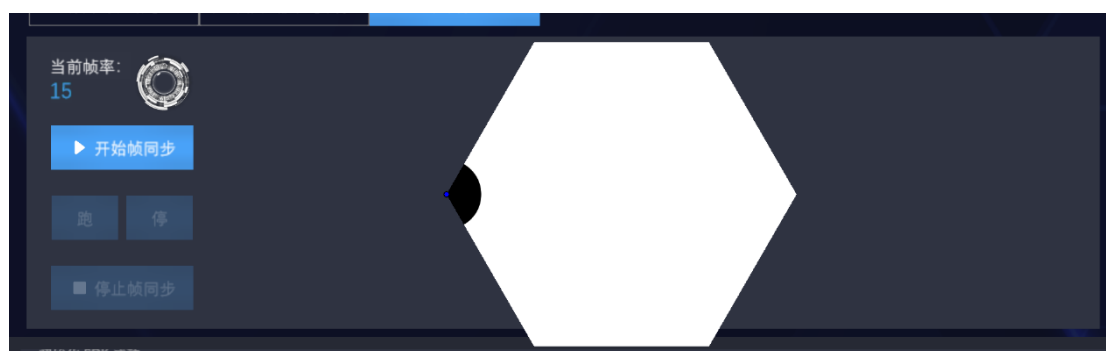
```

1. // 显示日志
2. ....
3. if (Util.isInited()) {
4.     return Util.appendLog("SDK 已经初始化, 无需重复操作");
5. }
6. ....
7. Util.appendLog("正在初始化 SDK");
8. ....
9. Util.appendLog("请在首页填入正确的 gameId、secretKey、url");
10. ....

```

4.5 游戏界面设计

根据前面人数选择界面的选择结果, 进入游戏界面时, 会相应地在六边形地图中产生玩家, 每个玩家由控制杆进行操作, 可以在游戏中使用之前购买的道具。当玩家仅有一人时, 游戏界面摇杆和玩家示意图如下:



游戏界面中的控制摇杆由一个底层图片组件和一个可移动顶层组件构成。摇杆整体效果图如下:



玩家移动时，拖动中间的顶层组件，即可控制移动方向：



如上图所示，此时玩家移动方向为右上方向。

实现摇杆功能的脚本绑定在摇杆的底层图片上，主要部分代码及解释如下：

```
1. onTouchStart(event) {  
2.     //触摸发生，event.getLocation()用于获取触摸坐标  
3.     let pos = this.node.convertToNodeSpaceAR(event.getLocation());  
4.     //convertToNodeSpaceAR 用于将获得的世界坐标转换成底层组件上的坐标  
5.     this.joyStickBtn.setPosition(pos);  
6. },
```

```
1. onTouchMove(event) {  
2.     //joyStickBtn 为顶层可移动组件  
3.     //通过 event.getDelta() 获取手指位移距离(向量)，加在之前 joyStickBtn 的位置，得到新的位置  
4.     let posDelta = event.getDelta();  
5.     this.joyStickBtn.setPosition(this.joyStickBtn.position.add(posDelta));  
6.     // get direction  
7.     this.dir = this.joyStickBtn.position.normalize();  
8. },
```

```
1. onTouchEnd(event) {  
2.     //触摸结束，将 joyStickBtn 恢复到初始位置  
3.     this.joyStickBtn.setPosition(cc.v2(0, 0));  
4. },
```

```
1. update(dt) {  
2.     //此部分代码实现将 joyStickBtn 始终限制在底层组件中  
3.     //利用三角形相似，计算缩放比例 ratio  
4.     let len = this.joyStickBtn.position.mag();  
5.     let maxLen = this.node.width / 2;  
6.     let ratio = len / maxLen;
```

```

7.
8.      //当触摸移动超出底层组件范围，将其按比例限制在底层组件中
9.      if (ratio > 1) {
10.         this.joyStickBtn.setPosition(this.joyStickBtn.position.div(ratio));
11.      }
12.      ...
13.  },

```

游戏界面中倒计时显示示意图如下：



倒计时显示游戏剩余时间，从玩家进入游戏界面后开始倒计时，通过在游戏界面附加 js 脚本实现，主要代码如下：

```

1.  function countingDown() {
2.      //此处为倒计时对时间的计算，最小单位为 100 毫秒
3.      if(hundred_milisecc === 0 && second === 0 && minute === 0 && hour
        === 0) {
4.          counting_flag = 1;
5.          pause.style.display = "none";
6.          clearInterval(starting_counting);
7.      } else if(hundred_milisecc === 0 && second === 0 && minute === 0 &
        & hour !== 0) {
8.          hour--;
9.          minute = 59;
10.         second = 59;
11.         hundred_milisecc = 10;
12.     } else if(hundred_milisecc === 0 && second === 0 && minute !== 0) {
13.         minute--;
14.         second = 59;
15.         hundred_milisecc = 10;
16.     } else if(hundred_milisecc === 0 && second !== 0){
17.         second--;
18.         hundred_milisecc = 10;
19.     } else {
20.         hundred_milisecc--;
21.     }

```

游戏界面显示的道具按钮，使玩家可以在游戏中使用之前在商店购买的道具，示意图如下：



点击 *加速器* 或 *立即复活* 按钮，在玩家的信息中，对点击项的数量减一，同时
在游戏界面显示相应的结果，即玩家移动速度加快或等待复活时间结束。

对游戏界面部分组件参数总结如下：

组件名称	参数
摇杆 Button	底层图片：Size: 50(W), 50(H); 顶层可移动组件：Size: 20(W), 20(H)
时间显示 Label	Font Size: 25; Font Family: PingFangSC-Regular, sans-serif
道具按钮 Button	Size: 200(W), 50(H); Background Color: #48A2F8; Font Family: PingFangSC-Regular, sans-serif

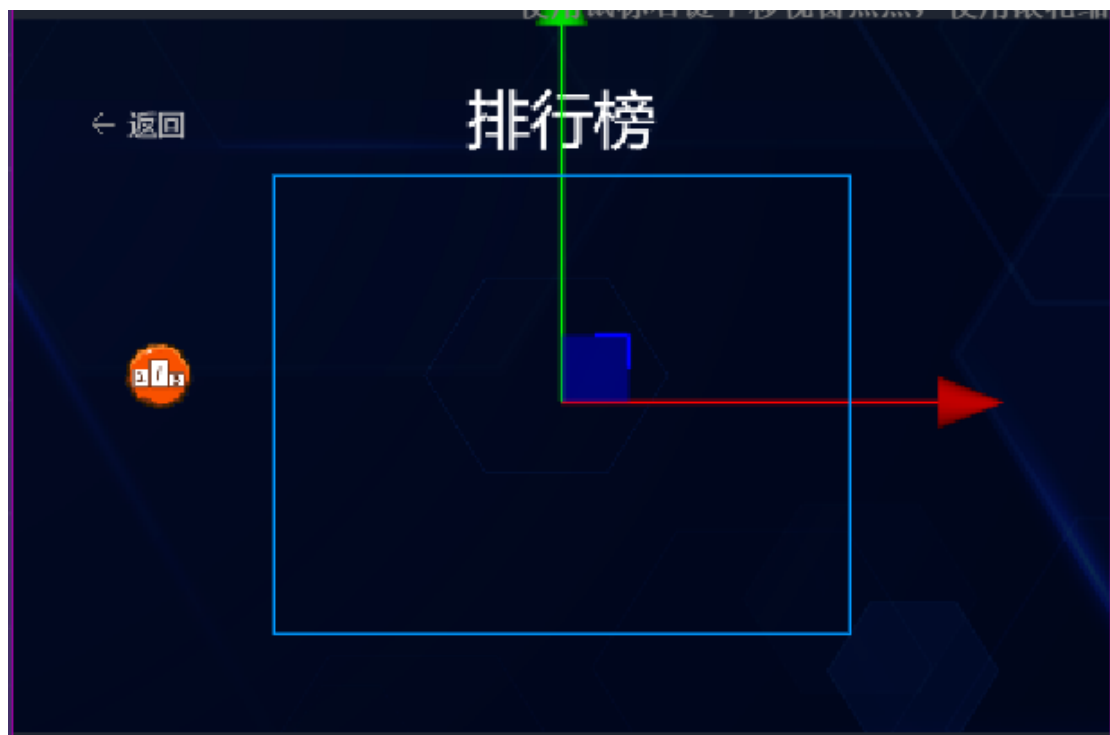
由于玩家的初始保留区和六边形地图可能随玩家人数增多而变化，因此在上
表未列出其参数。

4.6 排行榜界面设计

排行榜界面效果图设计如下：



上图中还没有显示玩家排行榜结果，为了表示直观，下图中蓝色框的部分为
最终显示玩家排行榜的地方。



点击左侧的按钮显示排行榜，其中最低行显示的是当前登录玩家的排行，上面部分为全局排行。

排行榜显示的主要代码如下：

```
1. showRanks() {  
2.   if (typeof wx === 'undefined') {  
3.     return;  
4.   }  
5.  
6.   if (!this.wxSubContextView.active) {  
7.     // 设置排行榜容器可见  
8.     this.wxSubContextView.active = true;  
9.  
10.    // 测试排行榜显示时，采用随机数当做玩家每局结算时的分数  
11.    let score = Math.round(Math.random() * 10);  
12.  
13.    // 发送结算分数到开放域  
14.    wx.getOpenDataContext().postMessage({  
15.      message: score  
16.    });  
17.  }  
18.  ...
```

此界面返回按钮用于返回至初始界面，返回按钮展示如下：



返回按钮由一个箭头图片组件和“返回”字样文字组件构成，添加在此按钮上的 ts 脚本主要内容如下：

```
1. start () {  
2.     //backScene 为要返回的界面，backBtn 为返回按钮  
3.     if (!!this.backScene) {  
4.         this.backBtn.on(cc.Node.EventType.TOUCH_START, () => cc.director.loadScene(this.backScene));  
5.         //点击返回按钮，触发 loadScene，返回初始界面  
6.     }  
7. }
```

4.7 商城界面设计

简单的商店界面如下，上半区为游戏可购买的皮肤，购买完成之后，点击单选框即可替换游戏中圈地的颜色，下半区为可购买的道具，加速器可让玩家移动速度增加 30%，持续时间 10 秒；立即复活则能让因游戏规则而“死亡”的玩家立即复活进入游戏，无需等待 10 秒。生命则可以增加一次体力值，即增加一次开始游戏的机会。



单选框使用的是 cocos creator 中的 toggle 组件，包括颜色方块，汉字，数字和单选框，选中时动画效果如下：选中之后整体放大。



道具使用的是 button 组件，包括按钮及价格，选中时也是整体放大。

返回使用的是 button 组件，并挂载了 returnHome 的脚本文件：

```
1. start () {  
2.   if (!!this.backScene) {  
3.     this.backBtn.on(cc.Node.EventType.TOUCH_START,()=>cc.director.loadScene(this.backScene));  
4.   }  
5. },  
6. ...
```

4.8 设置界面设计

设置界面可以改变自己游戏内的昵称，可以设置音乐的开关，以及其他的设置。

输入昵称使用的是 editbox 组件，包括输入框和预设文字。

控制音乐开关则使用 button 组件，点击一次之后可以打开/关闭音乐。



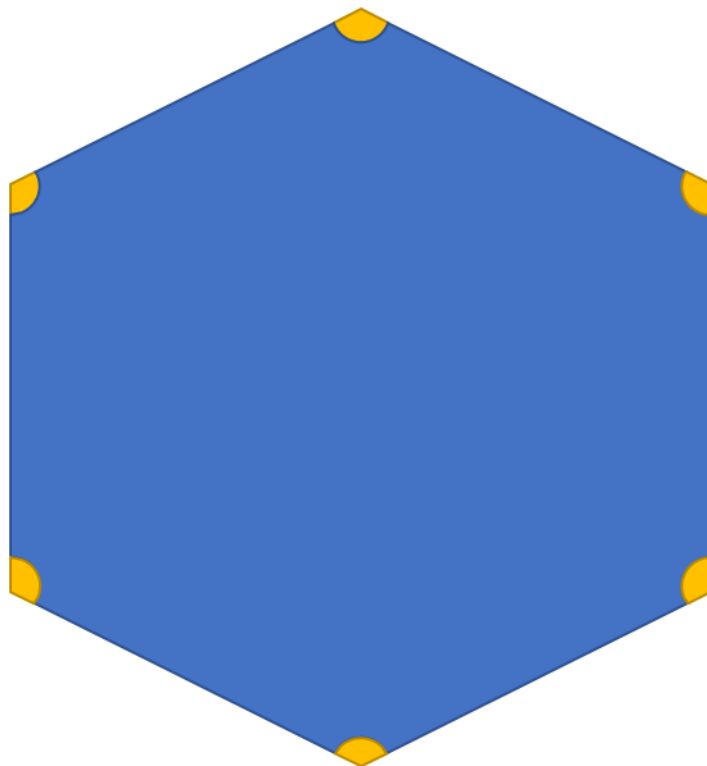
返回使用的是 button 组件，并挂载了 returnHome 的脚本文件：

```
1. start () {  
2.     if (!!this.backScene) {  
3.         this.backBtn.on(cc.Node.EventType.TOUCH_START,()=>cc.director.lo  
         adScene(this.backScene));  
4.     }  
5. },  
6. ...
```

5 游戏规则与逻辑设计

5.1 初始化设计

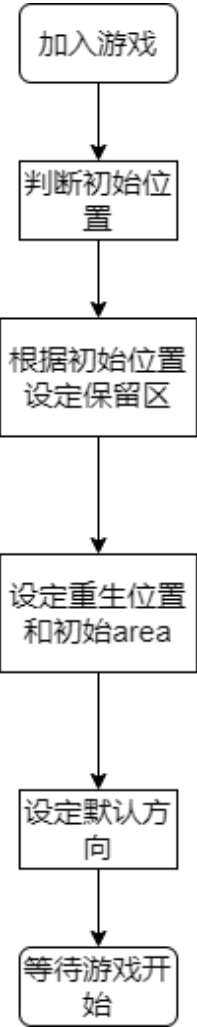
游戏地图为正六边形，每个玩家从各自的保留区出发，按照游戏规则向外圈地。



在游戏开始时：

- 如果是 1V1 模式，则两玩家分别从上下两个保留区出发；
- 如果是 2V2 模式，则四个玩家分别从左右四个保留区出发；
- 如果是 3V3 模式，则六个玩家分别从六个保留区出发。

在此之前，需要对各个玩家的初始信息进行设定，包括重生位置以及扇形的保留区等。流程图如下所示：



因此，设计了 `initArea([x, y]: [number, number], deltaDegree: number)` 方法，其输入参数包括：

参数	类型	说明
<code>[x, y]</code>	<code>[number, number]</code>	玩家初始位置
<code>deltaDegree</code>	<code>number</code>	根据初始位置计算出的保留区偏转角度

无返回值，作用是设定玩家的重生位置和保留区，并且初始化玩家位置及初始 `area`。

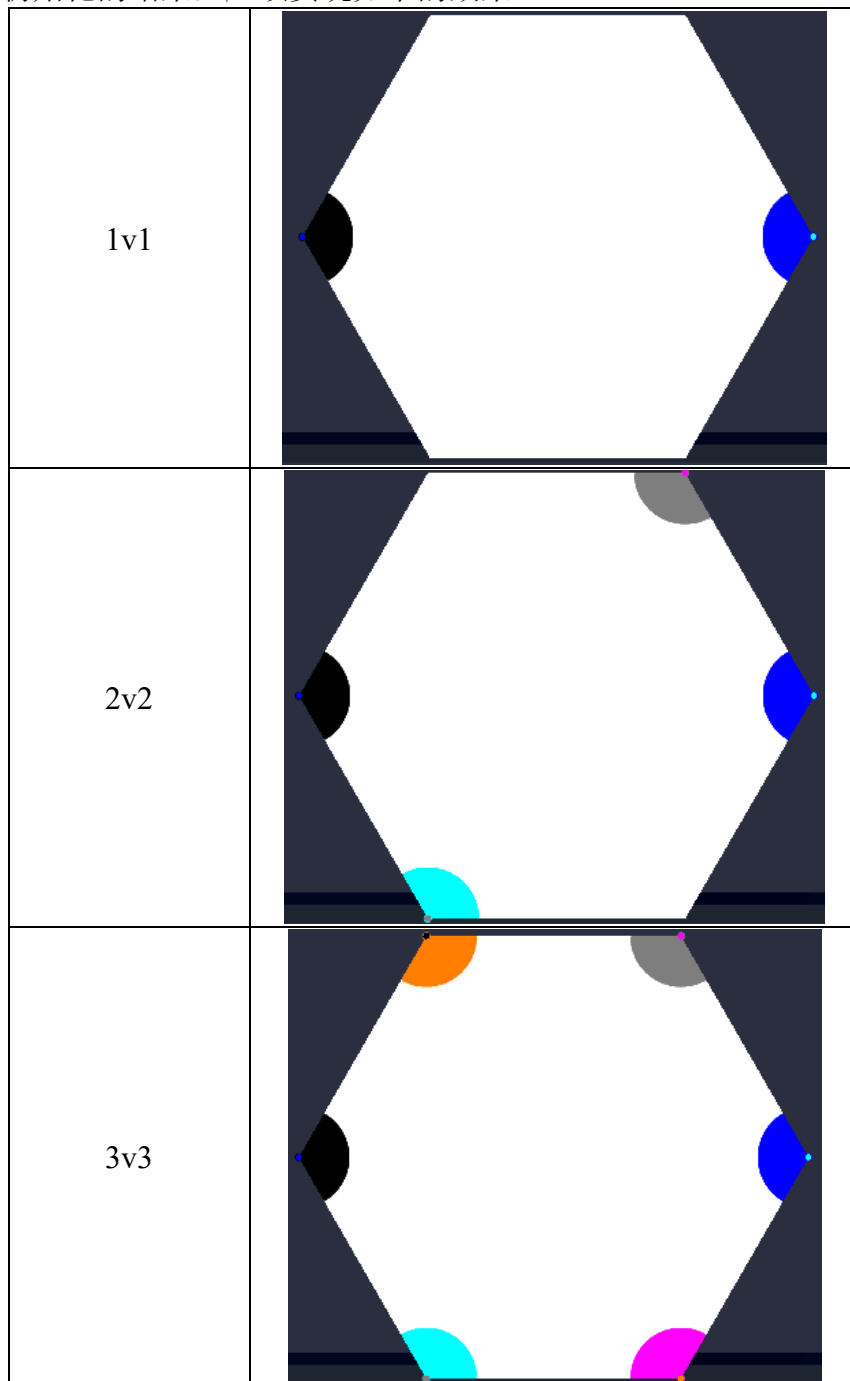
在这个函数中，核心算法是玩家保留区的计算，只需要生成一个圆心角为 $\frac{2}{3}\pi$ 的扇形即可。

```
1. private initArea([x, y]: [number, number], deltaDegree: number): void {
2.   根据保留区半径计算需要添加的坐标数 numOfPoints
3.   for (let i = 0; i <= numOfPoints; i += 1) {
4.     计算相对圆心的坐标[tempX, tempY]
5.     this._reservedArea.push([x + tempX, y + tempY])
6.   }
7.   this._reservedArea.push([x, y])
}
```

8. }

在 Player 的构造函数中，根据 Game 传入的玩家的初始位置，计算出对应的 deltaDegree，然后调用 initArea 即可。

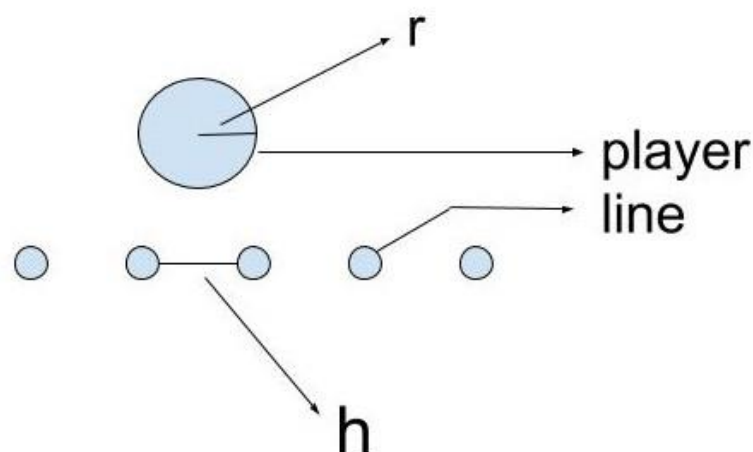
对于初始化的结果，应该实现如下的效果：



5.2 玩家移动设计

游戏动画实现思路为关键帧动画，玩家移动也并非连续移动，而是陆续地到达一些离散的点。相应的，玩家的 line 和 area 存储的点序列也为一些离散的点。

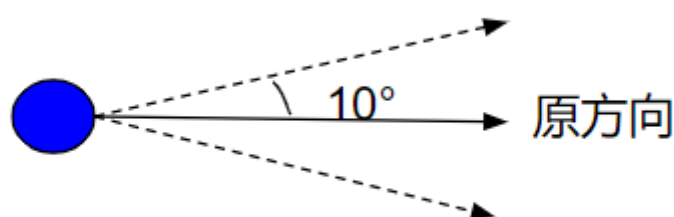
那么存在的一个问题是，当玩家移动并“切割”到其他玩家或自己的 line 时，是否会从点序列的“缝隙”中穿过，以及当玩家本应触碰到自己或其他玩家的 area 边界时，是否存在直接从边界穿过的可能性。



如上图所示，事实上玩家并非一个没有面积的点，而是一个圆，半径为 r 。玩家的 line 由一系列离散点组成，相邻点之间的距离为 h ，那么只要满足了 $h < 2r$ 的条件，line 和 area 的边界对于玩家来说便是不可穿越的了。

实际上，一般来说，帧动画中上一帧与下一帧之间，场景不会发生太大变化，玩家移动的距离也相当有限，因为只有这样才能使动画效果较为流畅，所以 h 一般是远小于 r 的，在实际测试中也印证了这一点。

此外，在玩家(player)类中 direction 表示玩家下一步移动方向，其数据结构为 [number, number]，是一个二维向量。实际运动过程中，玩家具有一定的速度，其方向不可大幅度任意改变，否则在视觉效果上会与日常物理规律有较大出入，故玩家转向需限制在一定幅度以内。具体来说，可将玩家下一步移动方向向量转换为角度，联系上一个方向的角度，差值不可超过 10 度。

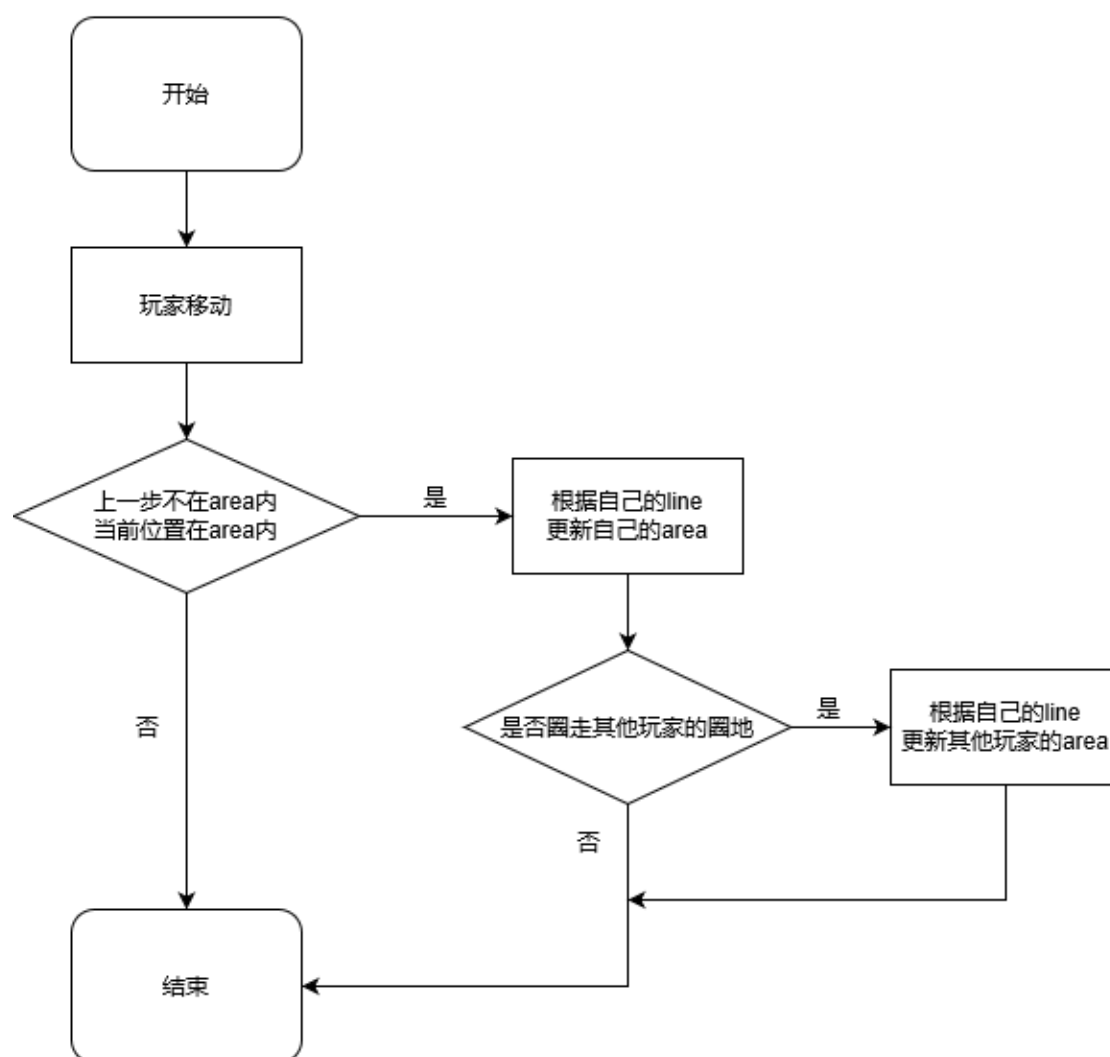


如图所示，玩家移动时，下一步方向的范围在两个虚线箭头内。

5.3 圈地面积设计

5.3.1 圈地逻辑流程

圈地逻辑流程



5.3.2 数据结构说明

玩家所圈地 area 是一块二维平面上的区域，把所圈地看做一个多边形，使用连续的点序列依次表示多边形的顶点，来表示这个多边形为玩家的圈地。

玩家的线 line 是玩家离开 area 后，还未回到 area 中的行走路径，使用每一帧位置的连续点序列表示 line。

5.3.3 核心算法逻辑

函数接口为：private updateArea(line: [number, number][]): void

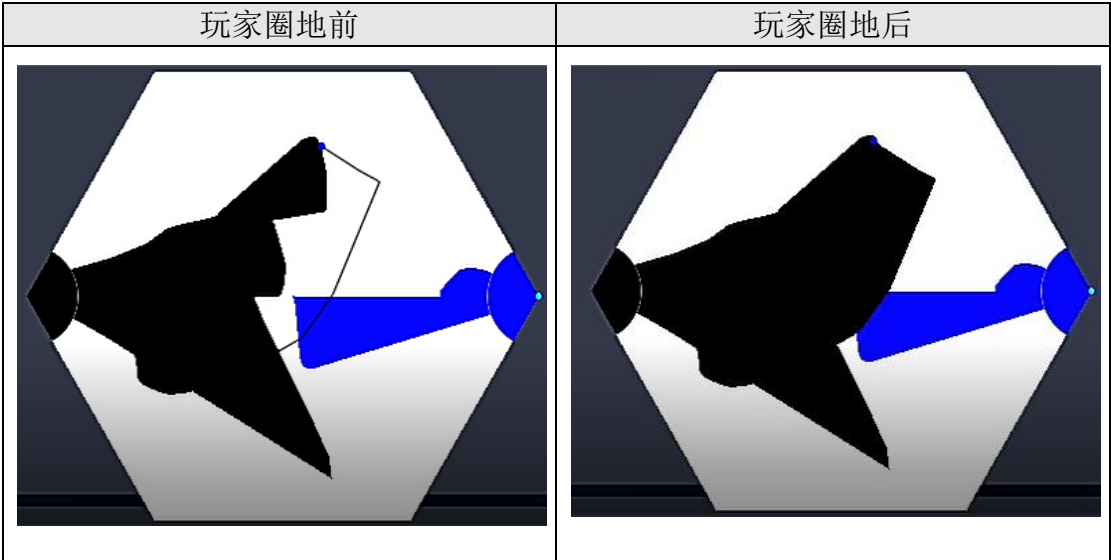
参数	类型	说明
line	[number, number] []	玩家自己圈地时生成的 line 或 被其他玩家圈走地时切割圈地部分的 line

函数功能：根据输入的 line 完成对于 area 的更新。

<div><div>1. private updateArea(line: [number, number][]): void {</div><div>2. 根据 line 中的首尾两点，找到 area 边界上对应的这两点距离最近的两个点 start，end</div><div>3. area 被 start 和 end 分为两段点序列，分别与 line 的点序列形成两个闭合的序列，记为 area1，area2</div><div>4. 选择与出生地连通的 area 作为 this._area</div><div>5. //对于被其他玩家圈走地的情况</div><div>6. if（被圈地后当前位置与出生地不连通）{this.die()}</div><div>7. }</div></div>
--

5.3.4 示例说明

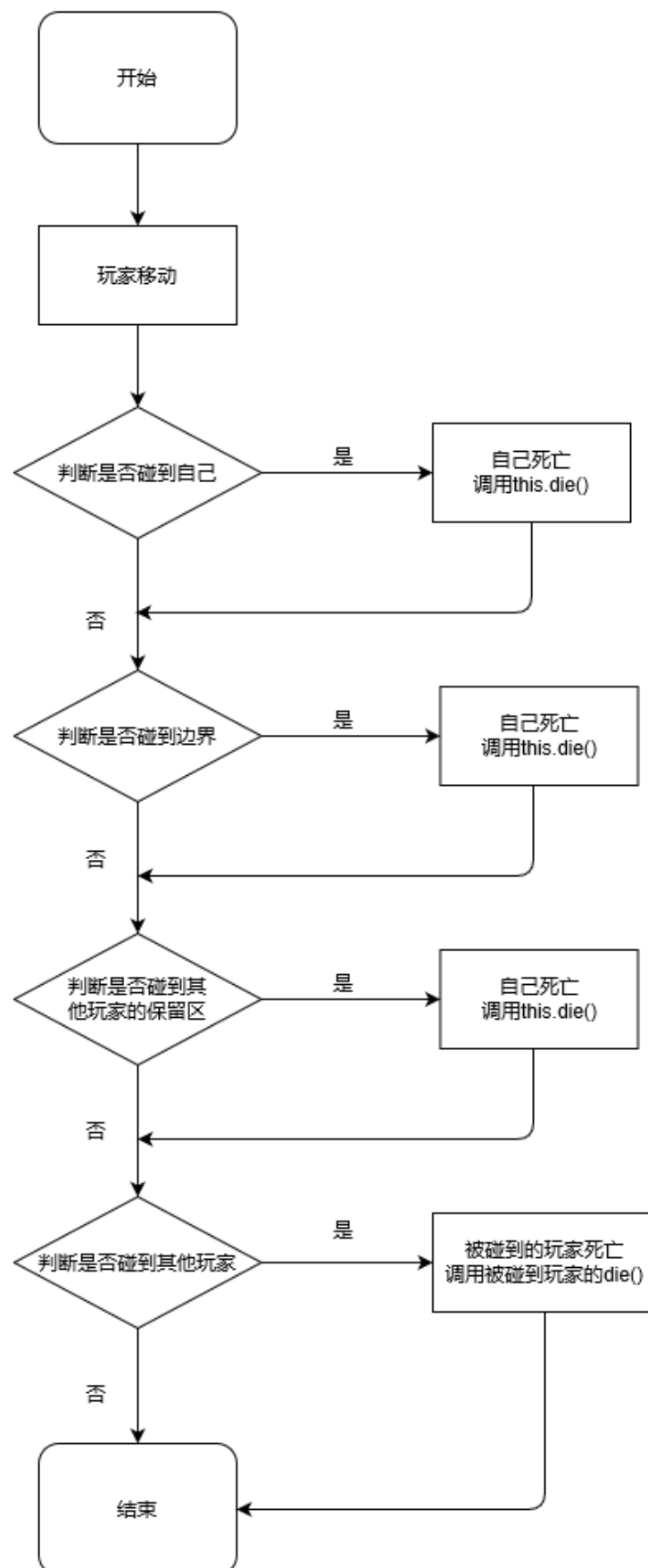
下图中黑色玩家完成圈地后，更新了自己 area，同时圈走了蓝色玩家的部分圈地，更新蓝色玩家的 area。



5.4 碰撞设计

5.4.1 整体的碰撞判断流程与分类

碰撞判断流程



碰撞一共分为四种类型的碰撞：

- 玩家碰到自己的 line
- 玩家碰到其他玩家的 line
- 玩家碰撞到地图的边界
- 玩家碰撞到其他玩家的保留区

下面对于这四种碰撞的判断与设计分别进行说明。

5.4.2 玩家碰到自己的 line

功能说明：
完成了对于玩家碰撞自己的 line 的判断。如果一个玩家碰到了自己正在圈地的 line，那么玩家就会死亡。

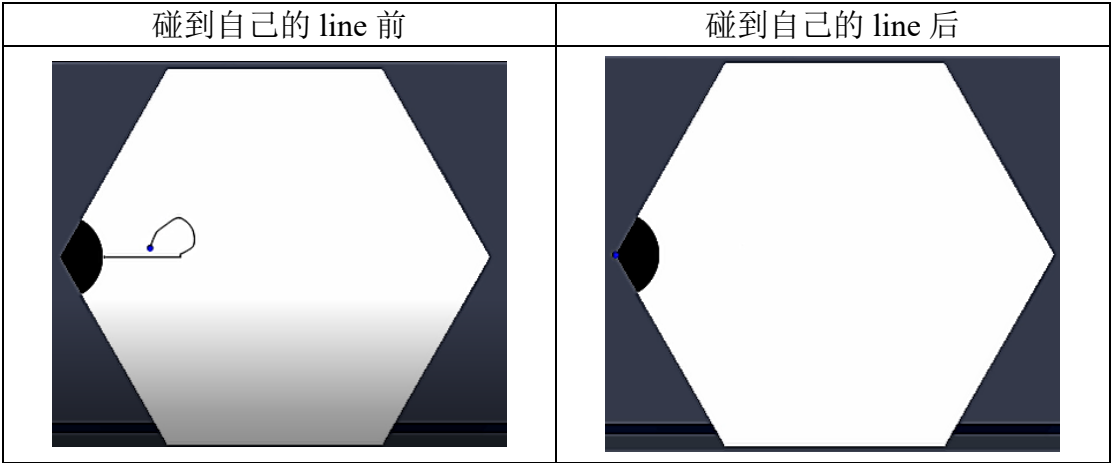
算法简述：
核心判断逻辑为：

```
1. if(this.distance(this._line[i], this._position) < this._pointRadius) {
2.     this.die()
3. }
```

即如果玩家当前的位置 `this._position` 和与他自己正在圈地的 `line` 中的某一个点 `this._line[i]` 的距离小于碰撞半径 `this._pointRadius`，则说明自己切割到了自己的 `line`，那么玩家就会死亡。

这里为了避免玩家当前的位置与刚刚生成不久的 `line` 的距离过近，所以在判断时选取了 `line` 中 `i < this._line.length - (this._pointRadius / this._speed * 6)` 的点序列进行碰撞判断，从而避免了玩家在进行小范围移动时被检测为碰到自己的 `line` 序列而死亡。

示例说明：
碰到自己后，自己的 line 消失，回到出发地。



5.4.3 玩家碰到其他玩家的 line

功能说明：

完成了对于玩家之间碰撞的判断，如果某玩家走出的路径 `line` 被其他玩家碰到，那么该玩家则会死亡。

接口设计：

碰撞判断设计为 `Player` 类中的一个 `method`，该接口的输入为一个玩家的当前位置坐标，无返回值。接口中完成了对于碰撞的判断，如果当前玩家被碰撞，则调用当前玩家的 `die()`。

接口形式为 `public checkCollision(p: [number, number]): void`。

算法简述：

每当一玩家（如玩家 A）移动之后，调用其他玩家（包括玩家 B, C 等等）的 `checkCollision()`，传入的参数为该玩家（如玩家 A）当前移动之后所在的位置。

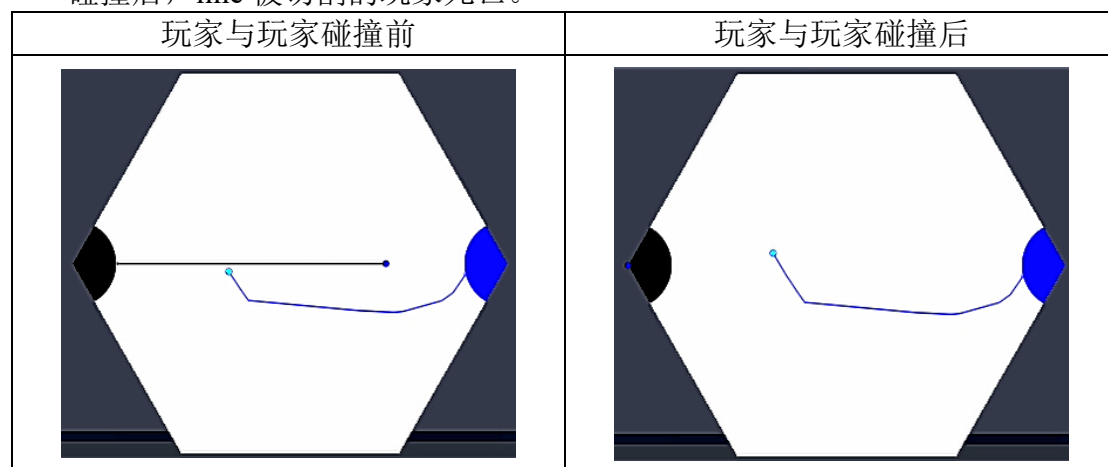
`checkCollision()` 中遍历了当前玩家 `_line` 中的点序列，核心判断逻辑为

```
1. if(this.distance(this._line[i], p) < this._pointRadius) {  
2.   this.die()  
3. }
```

即如果满足该条件，说明传入的 `p` 的坐标和当前玩家的 `_line` 中的点序列的某点的距离已经小于了点的碰撞半径，认为当前玩家被碰撞，当前玩家死亡。

示例说明：

碰撞后，`line` 被切割的玩家死亡。



5.4.4 玩家碰撞到地图的边界

功能说明：

完成了玩家碰撞边界的判断。

地图为六边形的边界，玩家只能在地图内进行圈地，当一名玩家碰撞到地图的边界时，玩家就会死亡。

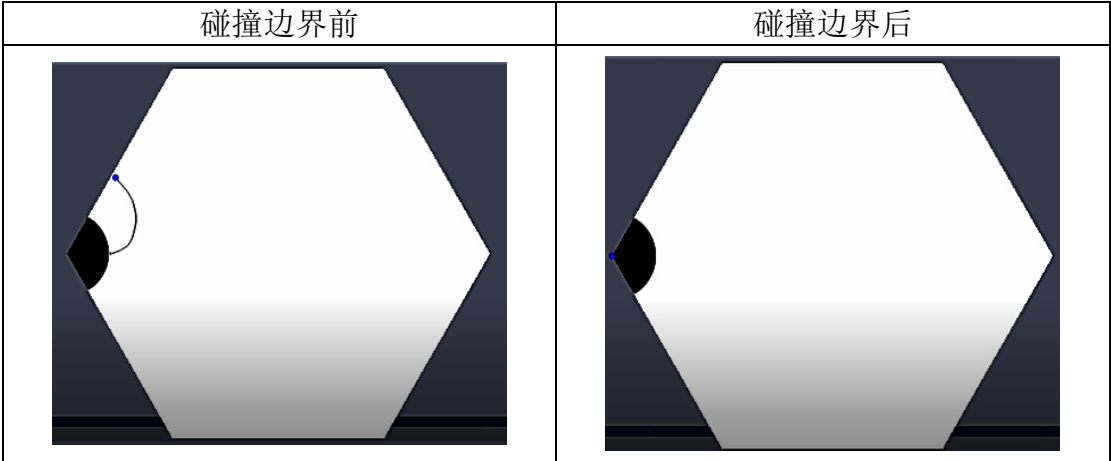
算法简述：

核心逻辑为：

```
1. if(!this.inPolygon(border, this._position)) {
2.     this.die()
3. }
```

通过判断玩家当前位置是否在六边形地图中来判断边界碰撞。如果玩家当前位置不在六边形边界内，则可以认为玩家从上一个位置到当前位置的移动过程中碰到了边界，玩家死亡。

示例说明：
碰到边界后，玩家死亡。



5.4.5 玩家碰撞到其他玩家的保留区

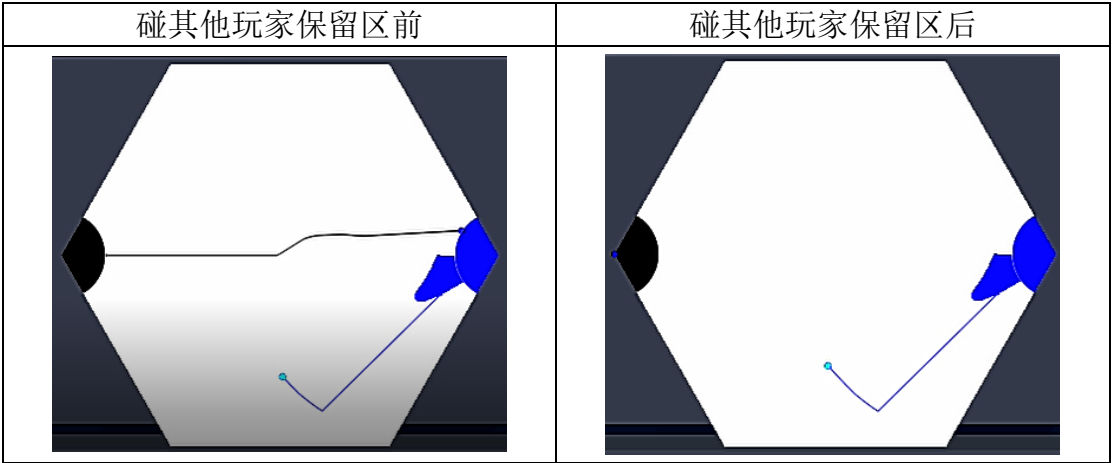
功能说明：
完成了玩家碰撞到其他玩家的保留区的判断。
每个玩家的保留区为一个扇形区域，该区域为对应玩家独占的区域，如果玩家碰到了其他玩家的保留区，则判定为死亡。

算法简述：
核心逻辑为：

```
1. if(this.inPolygon(p.reservedArea, this._position)) {
2.     this.die()
3. }
```

即如果当前玩家的位置 `this._position` 在其他玩家的保留区 `p.reservedArea` 内，则当前玩家死亡。

示例说明：
当碰到其他玩家保留区时，玩家死亡。



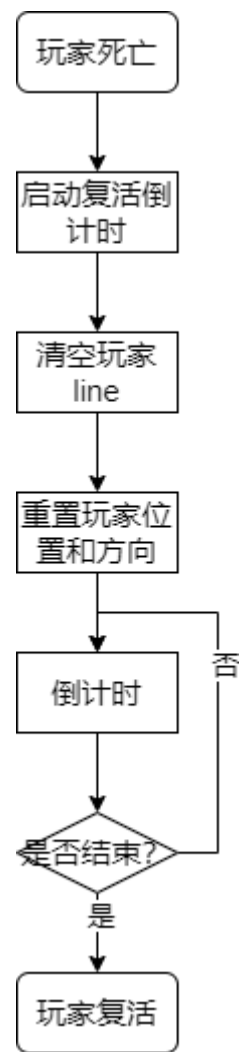
5.5 复活设计

造成死亡的情况有：

- 被其他玩家击杀
- 玩家触碰到自己的线(line)
- 玩家触碰到地图边界
- 玩家的领地在被其他玩家圈了一部分后变成两块或多块领地，导致玩家与保留地被隔开

当玩家被判定死亡后，需要将玩家现有的 line 清空，启动复活倒计时，当复活后，需要将玩家的位置重置。

流程图如下所示：



根据需求，设计以下三个方法：

方法	输入参数	返回值	作用
die()	无	无	标记玩家状态为死亡，清空玩家 line，并且重置复活倒计时
respawnCountDown()	无	无	对重生进行倒计时
respwan()	无	无	标记玩家状态为非死亡，并且重置玩家位置及方向信息

当玩家被判定死亡后，调用该玩家的 die()。在游戏过程中，如果玩家处于死亡状态，则调用玩家的 respawnCountDown()。在 respawnCountDown()中，如果玩家的倒计时变为 0，则调用玩家的 respwan()。

5.6 道具使用

使用道具：

- 立即复活：死亡状态下可使用，使用后可以立即复活。

- 加速：存活状态下使用，使用后该玩家移动速度提升。

立即复活在玩家死亡的时候可以使用，在使用时，直接将玩家的死亡倒计时清零，实现立即复活的效果。

加速在玩家未死亡的时候可以使用，在使用后，增加玩家的速度，并且启动一个倒计时。当倒计时为零的时候，道具持续时间结束，将玩家的速度恢复未正常值。

6 前后端交互设计

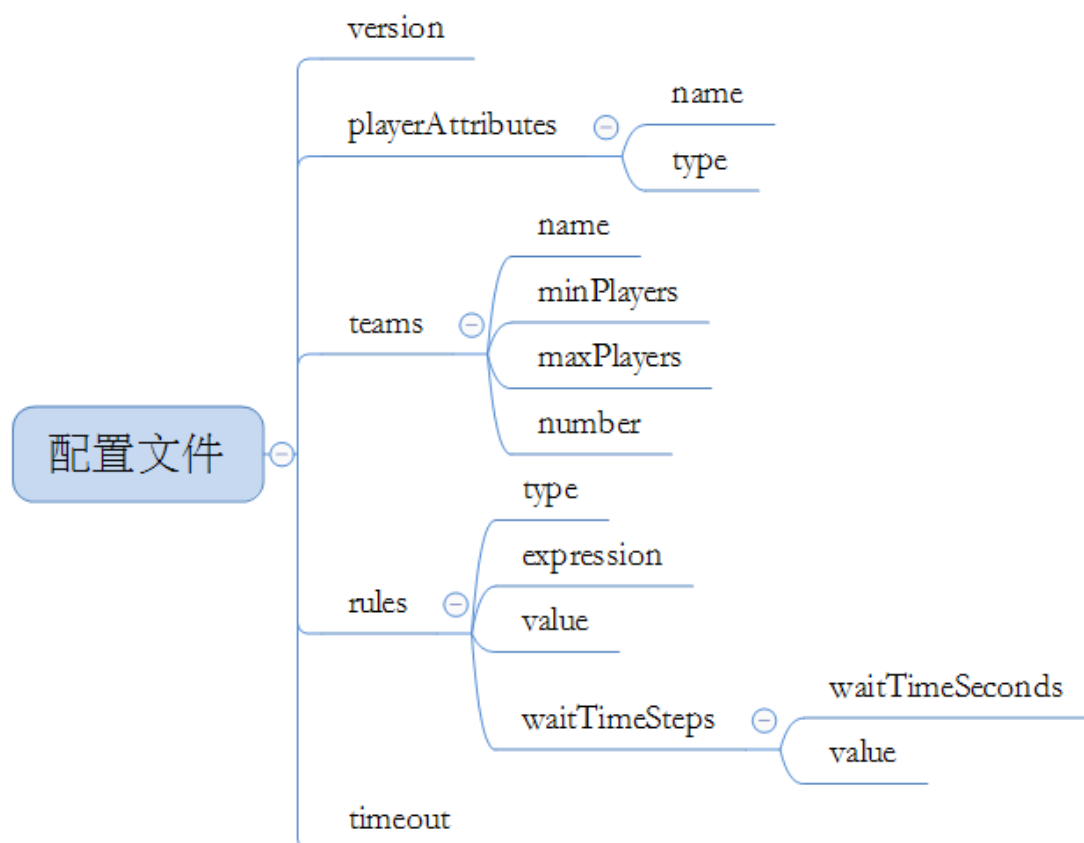
6.1 随机匹配设计

6.1.1 机制实现

使用腾讯游戏联机对战引擎实现自定义逻辑的实时服务器构建随机匹配机制。该实时服务器主要包含两部分：

- 房间的系列操作会自动触发到自定义服务逻辑；前端请求到自定义服务逻辑；
- 当玩家加房成功后，可以使用客户端 SDK 中的 `sendToGameSvr` 方法直接与游戏服务器通信，实现游戏服务端拓展逻辑，如保存玩家数据，游戏状态同步等。

新建匹配是通过编写匹配描述和相应的匹配规则集，我们可预览相应的 JSON 配置。新建匹配后，在页面就会生成相应的匹配 `code`。此外我们还可通过引擎提供的实时服务器页面来修改，删除或克隆匹配及匹配规则集。



6.1.2 玩家在线匹配

玩家在线匹配的方式为玩家创建并加入房间，支持自定义匹配规则。

首先创建匹配，调用客户端 API，接着配置游戏信息并定义玩家在线匹配参数，然后实例化 `Room` 对象并初始化 `Listener`，给房间添加监听，最后调用玩家

在线匹配 API, Room 接收广播回调。

玩家在线匹配 API 为:

```
1. room.matchPlayers(matchPlayersPara, event => {
2.   if (event.code === 0) {
3.     console.log("匹配成功", room.roomInfo);
4.   } else {
5.     console.log("匹配失败", event.code);
6.   }
7. });
```

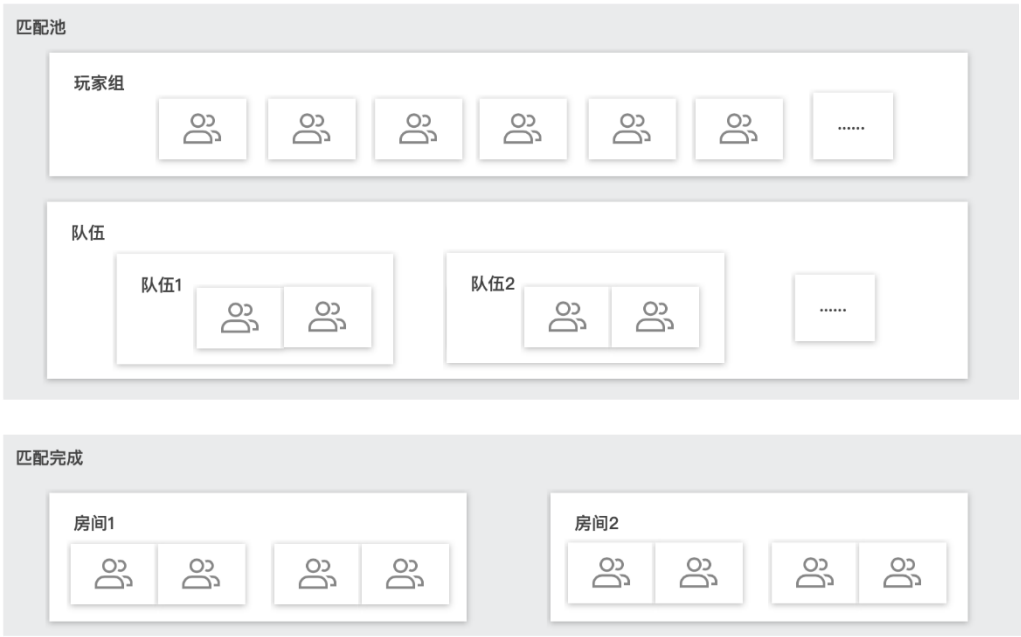
广播回调 API 为 onJoinRoom 和 onLeaveRoom:

```
1. // 广播: 房间有新玩家加入
2. room.onJoinRoom = event => console.log("  新  玩  家  加  入
   ", event.data.joinPlayerId);
3. // 广播: 房间有玩家退出
4. room.onLeaveRoom = event => console.log("  玩  家  退  出
   ", event.data.leavePlayerId);
```

搜索合适的玩家组成游戏对局, 具体匹配过程包括:

- 发起匹配的玩家组, 会进入同一个匹配池。匹配池中的玩家组将经历两个阶段: 等待被匹配成队伍、等待被匹配进房间;
- 服务器会根据开发者配置的规则, 在现有的匹配池中尽可能为玩家搜索符合条件的匹配对象组成队伍, 再将符合条件的队伍组成房间, 搜索过程中的两个阶段对开发者是不可见。由于不断有玩家进入匹配池, 也不断有玩家超时, 匹配池中的玩家组和队伍都是动态的。所以为了匹配效率, 匹配结果并不总是全局最优解。

匹配过程如下所示。



6.2 创建房间与加入房间设计

6.2.1 初始化设计

该设计主要通过游戏联机对战引擎的 `GameServer.IGameServer` 接口实现。也就是实时服务器接口，该接口提供了接收客户端消息、监听房间广播相关接口。

我们在创建房间之前，首先通过 `onInitGameData` 初始化游戏数据。回调参数为 `args`，类型为 `{room:IRoomInfo}`。

`IRoomInfo` 定义如下：

字段名	类型	描述
id	number	房间 ID
name	string	房间名称
type	string	房间类型
createType	CreateType	创建房间方式
maxPlayers	number	房间最大玩家数量
owner	string	房主 ID
isPrivate	boolean	是否私有
customProperties	string	房间自定义属性
playerList	IPlayerInfo[]	玩家列表
teamList	ITeamInfo[]	团队属性
frameSyncState	FrameSyncState	房间状态
frameRate	number	帧率
routeId	string	路由 ID
createTime	number	房间创建时的时间戳,秒

startGameTime	number	开始帧同步时的时间戳,秒
---------------	--------	--------------

6.2.2 创建房间

创建房间广播回调接口 onCreateRoom，该接口参数类型为 ActionArgs<ICreateRoomBst>，定义如下：

字段名	类型	描述
roomInfo	IRoomInfo	房间信息

创建房间后，创建者自动成为房主，拥有解散房间的权利。



若房主离开房间，服务器会指定下一个房主。

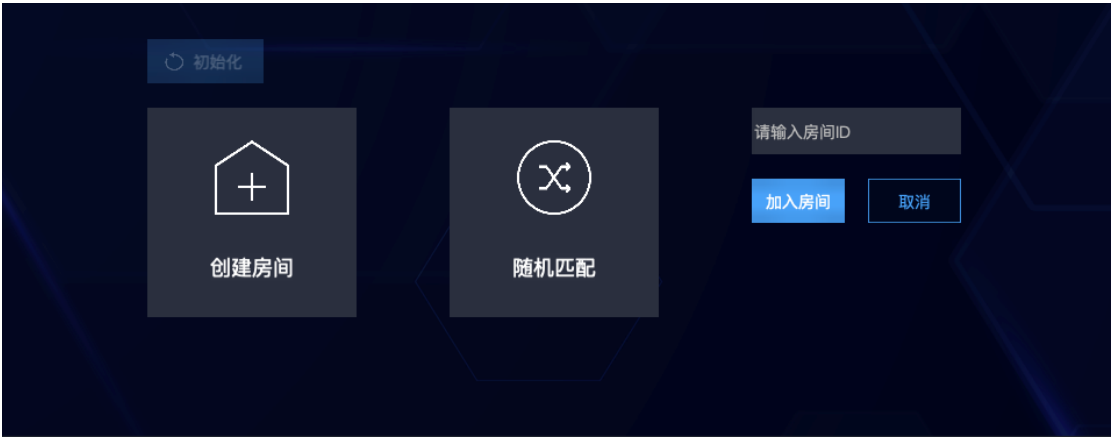
房间中展示房间号，当前玩家 ID 以及房间在线人数。

6.2.3 加入房间

玩家加房广播回调接口 onJoinRoom，该接口参数类型为 ActionArgs<IJoinRoomBst>，定义如下：

字段名	类型	描述
roomInfo	IRoomInfo	房间信息
joinPlayerId	string	加房玩家 ID

客户端可以通过调用该接口加入指定房间，IRoomInfo 中的 id 段为加入指定房间的房间号。指定的房间号必须属于由 onCreateRoom 接口创建的房间。



6.2.4 离开房间

玩家退房广播回调接口 onLeaveRoom，该接口参数类型为 ActionArgs <ILeaveRoomBst>，定义如下：

字段名	类型	描述
roomInfo	IRoomInfo	房间信息
leavePlayerId	string	退房玩家 ID

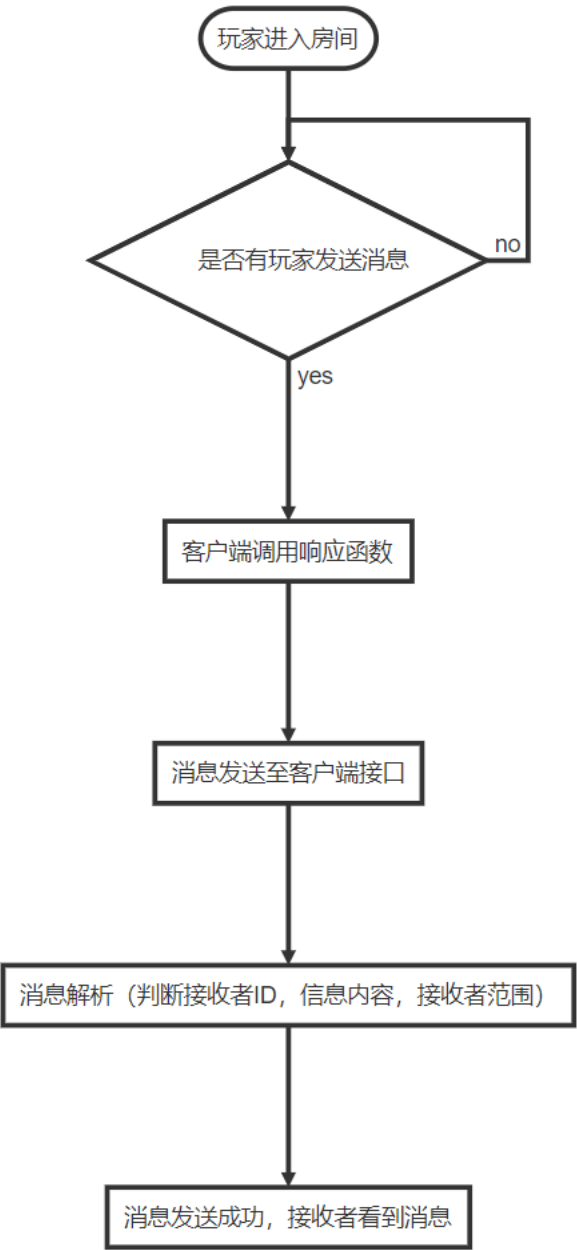
玩家离开房间会返回退房玩家 ID leavePlayerId，如果房主离开房间，服务器会指定下一个房主，并获取其玩家 ID。

6.3 房间消息发送设计

功能说明：

- 在玩家进入到同一个房间时，有消息列表显示消息；
- 玩家可以通过发送消息栏进行消息发送。

交互逻辑图：



接口设计：

房间消息发送主要包括 `sendToClient` 和 `onRecvFromClient`。其中 `sendToClient` 是发送消息至客户端的接口，`onRecvFromClient` 是收到房间内其他玩家消息广播回调的接口。

以下是 `sendToClient` 接口的参数及解释。

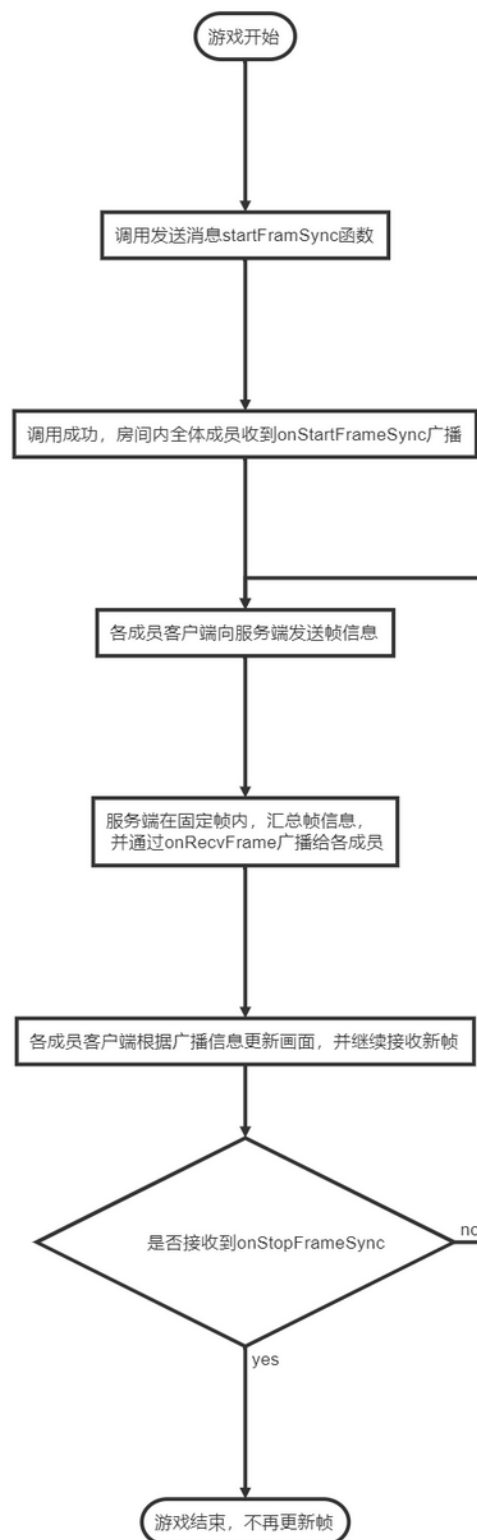
参数	描述	对象
SendToClientPara	发送消息参数	recvPlayerList: 接收消息玩家 ID
		msg: 消息内容
		recvType: 消息接收者（全部玩家、除自己外的其他玩家、房间中部分玩家）
callback	响应回调函数，调用结果异步返回，调用成功后所指定的接收消息的玩家将收到信息	

6.4 帧同步设计

功能说明：

使得多玩家操作画面帧达到同步，即各玩家游戏画面一致。

交互逻辑设计流程图：



接口设计：

名称	描述	参数与响应回调函数
startFrameSync	开始帧同步	预留参数 para 响应回调函数 callback
onStartFrameSync	开始帧同步广播回调；表示房间开始帧同步，收到该广播后，将持续接收到 onRecvFrame 广播	event: 回调参数，只含有服务端发送过来的 data 字段
stopFrameSync	停止帧同步	预留参数 para 响应回调函数 callback
onStopFrameSync	停止帧同步广播回调接口，表示房间结束帧同步，收到该广播后将不再接收到 onRecvFrame 广播	event: 回调参数，只含有服务端发送过来的 data 字段
sendFrame	发送帧同步数据，只有房间处于“已开始帧同步”状态才能调用该接口，后台将集合全部玩家的帧数据，并以一定时间间隔（房间帧率）通过`onRecvFrame`广播给各客户端。调用结果将在`callback`中异步返回	发送帧同步数据参数 sendFramePara；帧数据 data：为普通 object，内容自定义 响应回调函数 callback
onRecvFrame	房间帧消息广播回调，表示收到一个帧 Frame，并且是由多个帧组成的，即一帧时间内房间内所有玩家向服务器发送帧消息的集合	event: 回调参数，只含有服务端发送过来的 data 字段