目录

基于 Django 的多人在线游戏系统

软件工程 2003 杜睿

1 系统概述

选题：实现一个在线 2D 对战游戏。

1.1 背景描述

科技发展，为大众提供了丰富多彩的休闲方式。电子游戏，是当今青年一代放松娱乐的首选项。软件开发等技术愈发先进，游戏流畅度丰富度持续增加，人们愈发期待更具互动性、社交性的娱乐方式。单机游戏由于离线性，已不能完全满足玩家喜好；因此，多人在线游戏呈现日益增长趋势。

1.2 需求分析

目标 移动位置、释放技能等，击退敌方造成伤害；若击败除自己外剩余所有玩家，即可获得胜利。

模式 分为人机模式、联机模式。人机模式下，玩家与机器人进行对战，机器人随机移动位置并释放技能。联机模式下，玩家与真人在线匹配，当房间人数满足要求后，游戏方可开始。

操作 分为移动、技能、聊天。使用鼠标右键选择目的地，使自身以一定的速度向其移动；使用鼠标左键点击目的地，向指定位置释放技能；使用键盘上的 Q 键和 F 键来切换技能，分别对应火球、闪现技能；使用键盘上的 Enter 键和 Esc 键来打开或关闭聊天框，输入或发送信息。

技能 每名玩家用一个正圆表示。“火球”技能可以对其他玩家造成伤害，并使其半径缩减，当半径小于一定值时，判定该玩家去世。“闪现”技能可以让玩家瞬间移动到一定距离内的位置。火球技能和闪现技能都有冷却时间，在冷却时间内无法使用。

人数 除了显示每名玩家的头像和技能冷却时间，界面仍应显示当前房间准备就绪的玩家数量。

1.3 开发环境

云服务器 华为云耀云服务器，1 核 2GiB，Ubuntu 20.04 server 64 位

开发框架 Django 3.2.8，Python 3.8.10

数据库 Sqlite 3.31.1，Redis 4.5.1，Django-Redis 5.2.0，Channels-Redis 3.4.1

开发工具 Visual Studio Code 1.76.0，WebStorm 2022.3.2

版本控制 Git 2.25.1

2 系统设计

2.1 概要设计

2.1.1 用例图

用例图是一种软件工程中的行为建模技术，用于描述系统的功能需求和外部交互。

参与者 游戏有两种参与者，分别是玩家和管理员。玩家可以注册、登录、创建或加入房间、开始或加入游戏、操作角色、查看排行榜等。管理员可以登录、管理玩家账户、管理房间和游戏等。

用例 游戏有多个用例，分别是注册、登录、创建房间、加入房间、开始游戏、加入游戏、操作角色、查看排行榜、管理玩家账户、管理房间等。每个用例代表了一个完整的功能或业务流程。

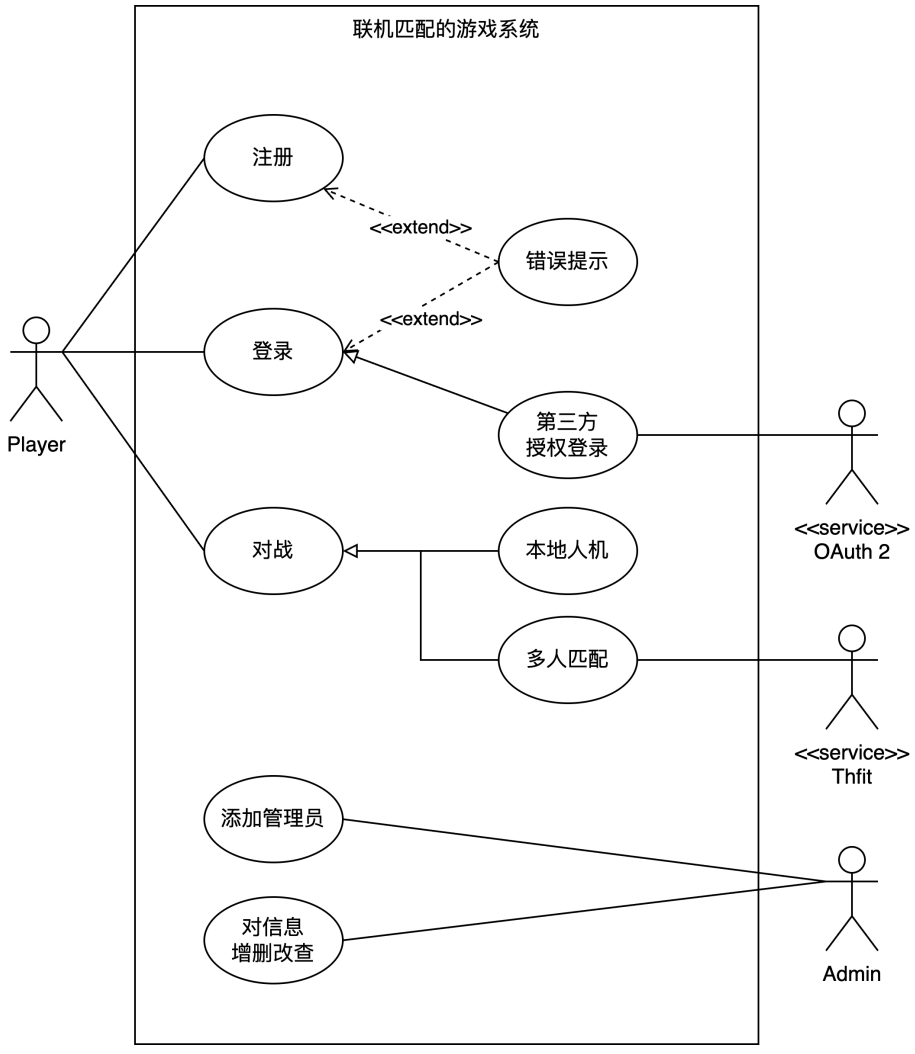

图 1: 分层用例图

2.1.2 系统构件图

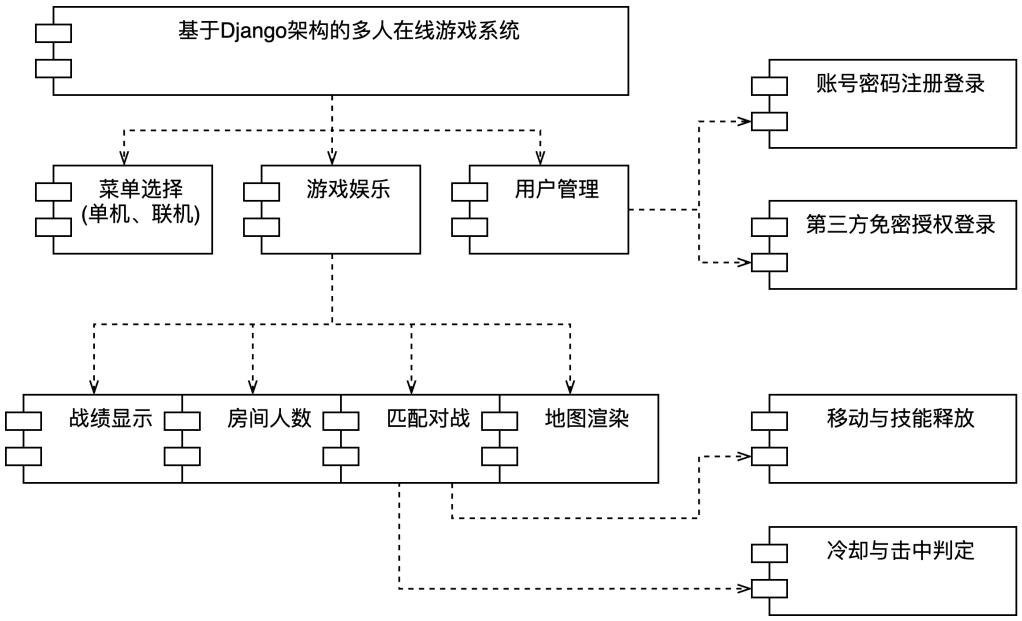

图 2: 系统构件图

2.1.3 操作原型图

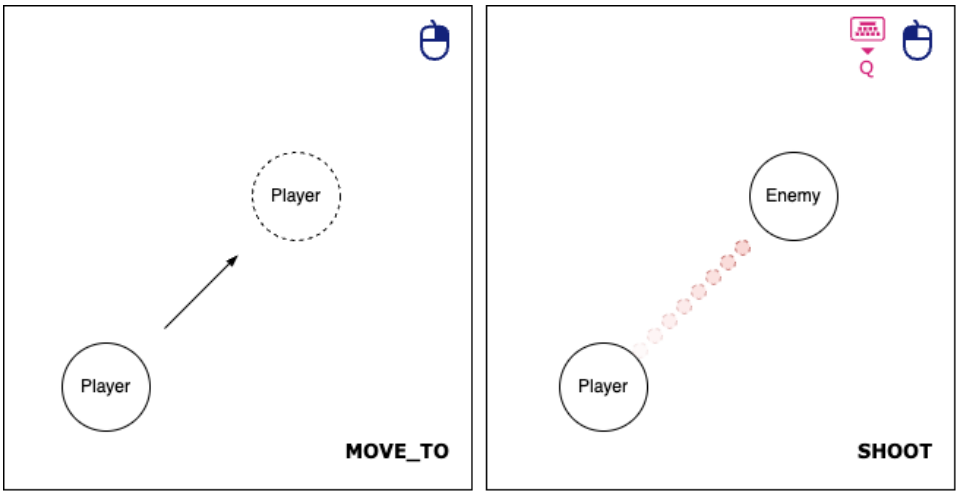

图 3: 操作原型图（移动、射击）

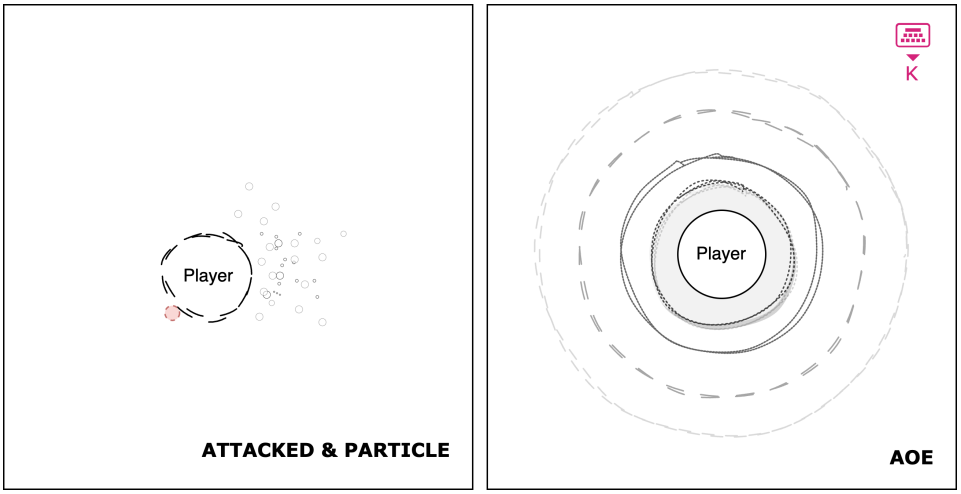

图 4: 操作原型图（粒子、技能）

2.1.4 数据库表

表 1: 玩家表

字段	类型	属性	备注
user	OneToOneField	on_delete=models.CASCADE	关联框架内置用户
photo	URLField	max_length=256, blank=True	头像，以渲染玩家
openid	CharField	default=””, max_length=50, blank=True, null=True	
score	IntegerField	default=1500	

2.1.5 类图

见下页。

2.1.6 顺序图

见下页。

2.2 详细设计

2.2.1 应用架构

Django 是一个基于 Python 的 Web 框架，遵循 MVT（模型-视图-模板）架构模式：模型（Model）负责定义数据结构，视图（View）负责处理用户交互和业务逻辑，而模板（Template）负责呈现数据。这种分离使得开发人员能够更好地关注各自的领域，提高开发效率。

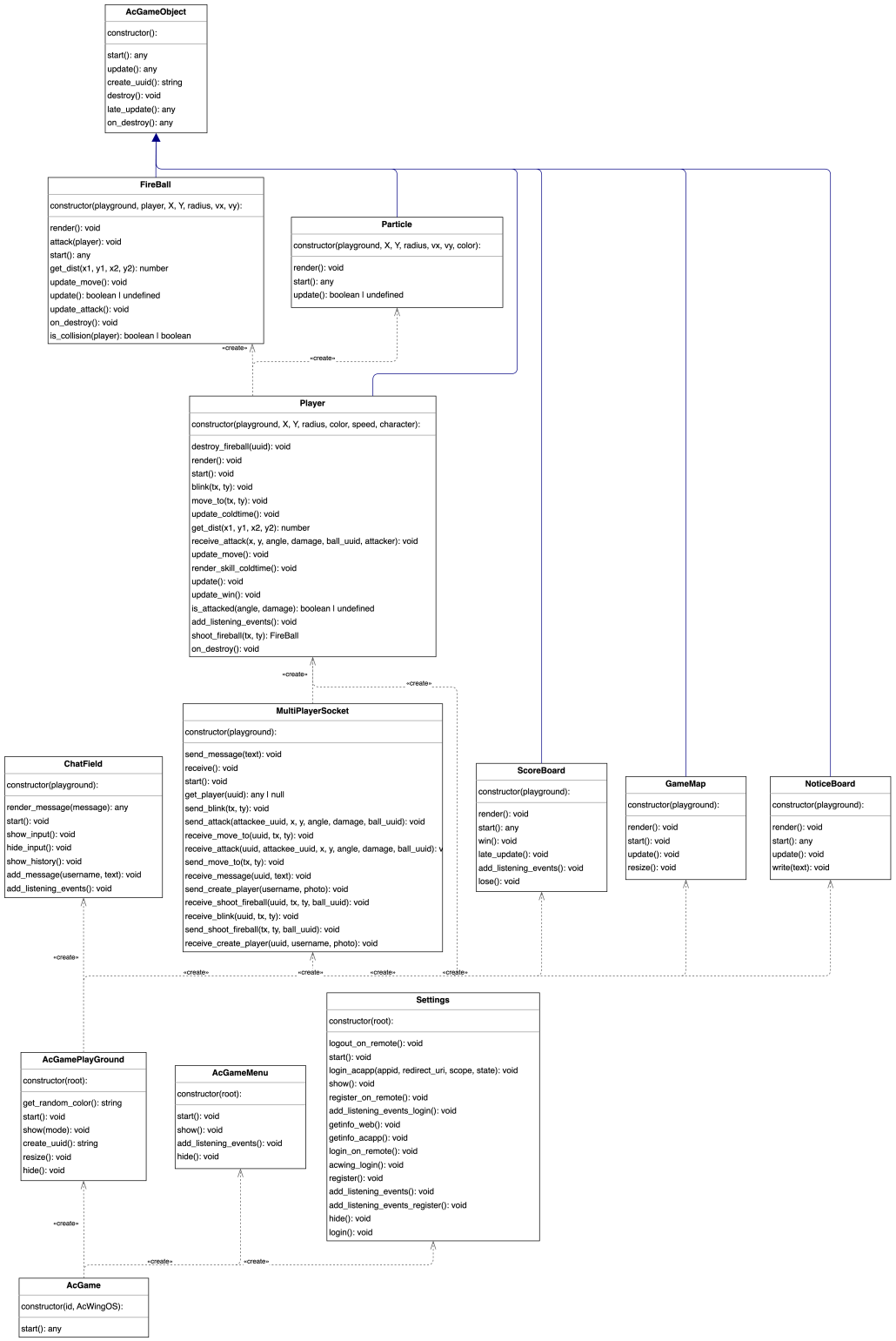

图 5: JavaScript 类图

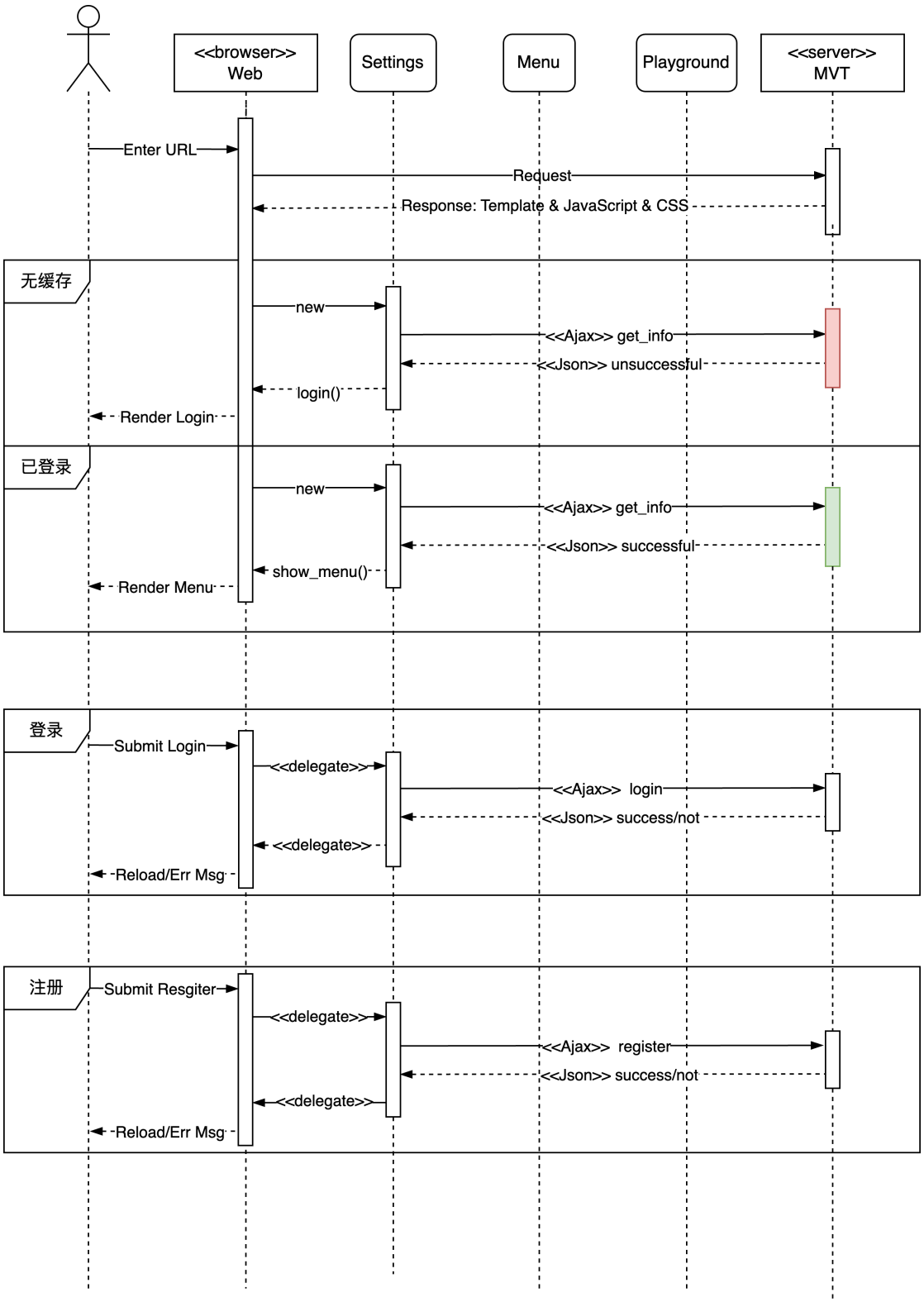

图 6: 顺序图（访问主页、注册登录）

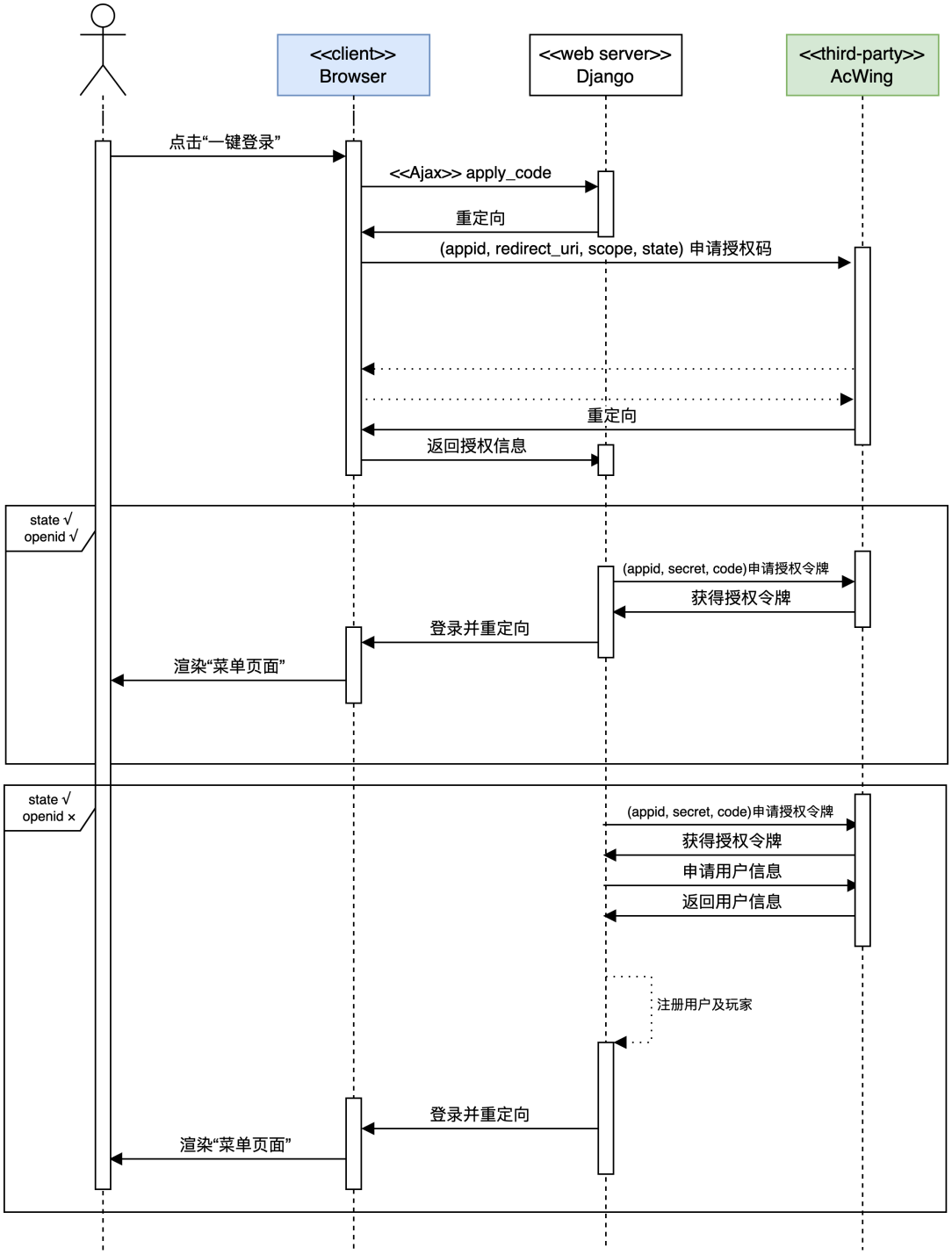

图 7: 顺序图（第三方授权登录 OAuth2.0）

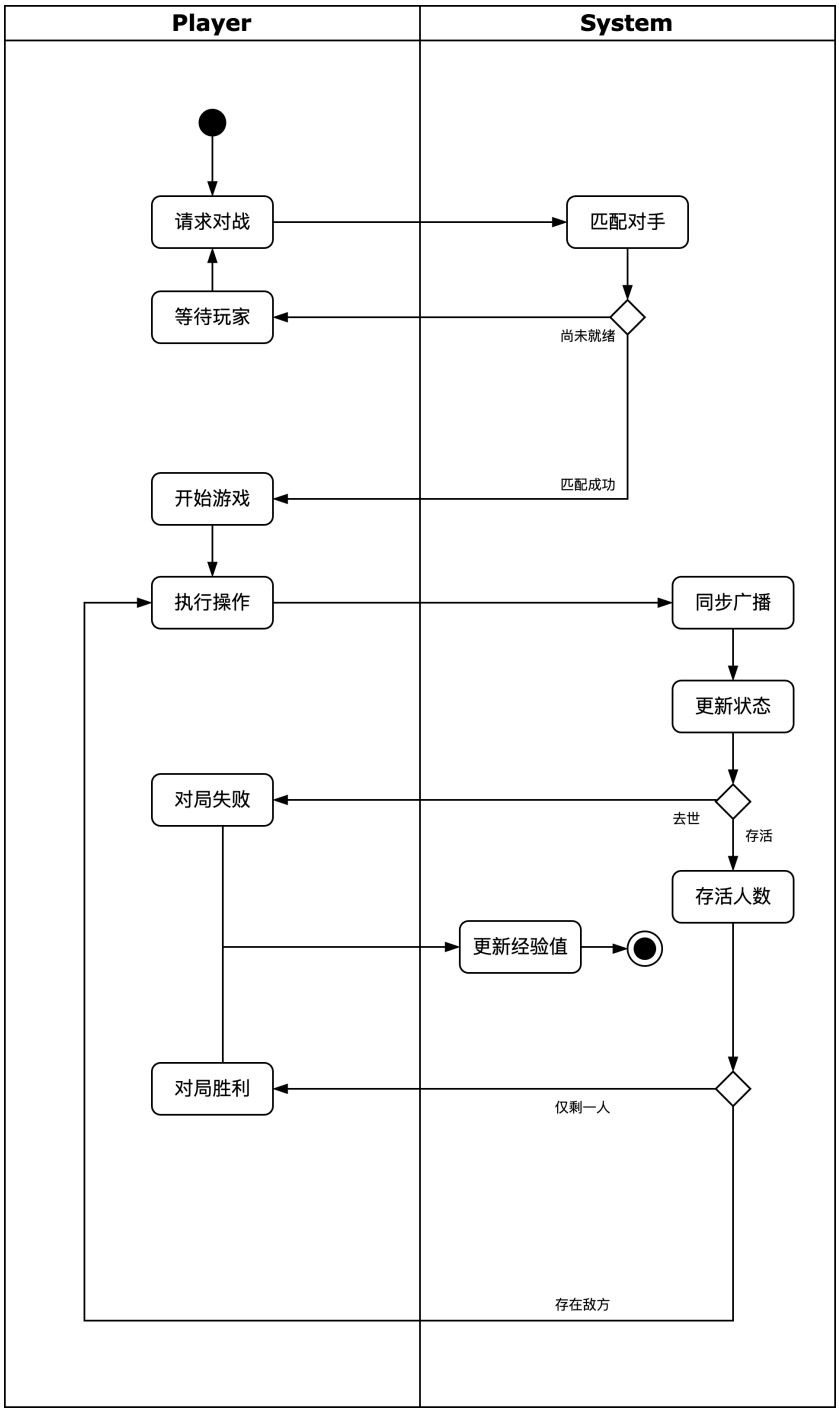
Player
System
请求对战
匹配对手
等待玩家
尚未就绪
匹配成功
开始游戏
执行操作
同步广播
更新状态
对局失败
去世
存活
存活人数
更新经验值
对局胜利
仅剩一人
存在敌方

图 8: 顺序图（对战过程）

2.2.2 接口协议

表 2: **HTTP API**

路由	响应	备注
/	HTML	网页内容、样式和交互逻辑
settings/getinfo/	JSON	{结果，用户名，头像}
settings/login/	JSON	{结果}
settings/logout/	JSON	{结果}
settings/register/	JSON	{结果}
settings/acwing/web/apply_code/	JSON	{结果}
settings/acwing/web/receive_code/	Redirect	重定向至主页
settings/acwing/acapp/apply_code/	JSON	{结果，appid，redirect_uri，scope，state}
settings/acwing/acapp/receive_code/	JSON	{结果，用户名，头像}

本系统实现了一个简易的游戏引擎，以实现逐帧重绘渲染。
同时，联机对战中，使用 WebSockets 协议建立双向连接，实现客户端和服务器之间双向和同步数据交换。关键事件触发时，如创建玩家、移动位置、射击火球、闪现、击中，系统将进行房间内玩家操作同步，保证游戏的实时性和交互性。

表 3: **WebSocket Handler**

函数	功能
connect	接受 WebSocket 连接
disconnect	当 WebSocket 连接断开时，从房间组中移除它
receive	解析消息数据，并根据事件类型调用相应的方法
create_player	向匹配服务器发送玩家的信息，并加入一个房间组
group_send_event	向所有房间组成员发送事件数据
move_to	向房间组发送移动事件数据
shoot_fireball	向房间组发送射击事件数据
attack	向房间组发送击中事件数据，并更新玩家的血量和分数
blink	向房间组发送闪现事件数据
message	向房间组发送聊天事件数据

3 系统实现

3.1 关键技术

- HTML5、CSS3 和 JavaScript 是 Web 开发的基础技术。HTML5 用于定义网页的结构和内容，CSS3 用于控制网页的布局和样式，而 JavaScript 用于添加交互性和动态效果。
- JQuery 是一个流行的 JavaScript 库，简化了 JavaScript 编程。
- Ajax 允许 Web 应用程序在不刷新整个页面的情况下与服务器进行数据交互。
- SQLite 是一种轻量级的关系数据库管理系统，被广泛用于嵌入式设备和移动应用程序中。
- Redis 是一种高性能的内存数据存储系统，常用于缓存和消息队列等场景。
- OAuth 2.0 授权协议，允许第三方应用访问用户在其他服务提供商处存储的受保护资源。
- Thrift 是一种跨语言的微服务开发框架，提供了一种高效的二进制协议来传输数据。
- Linux Shell 是一种命令行界面，允许用户与操作系统进行交互。

3.2 核心部分

下面只列出部分典型代码，全部代码详见：https://github.com/DURUII/toy-acapp-django。

3.2.1 游戏引擎

以基类为例。

```
// 存储游戏对象实例
let AC_GAME_OBJECTS = []

/**
 * 简易游戏引擎，
 * 提供生命周期接口，
 * 以便子类（如地图、玩家、火球、粒子）实现逐帧渲染。
 */
class AcGameObject {
    constructor() {
        AC_GAME_OBJECTS.push(this);

        this.has_called_start = false;
        this.timedelta = 0;

```

```
        this.uuid = this.create_uuid();
    }

    create_uuid() {
        let res = "";
        for (let i = 0; i < 8; i++) {
            let x = parseInt(Math.floor(Math.random() * 10));
            res = res + x;
        }
        return res;
    }

    start() {}

    update() {}

    late_update() {}

    on_destroy() {}

    destroy() {
        this.on_destroy();
        for (let i = 0; i < AC_GAME_OBJECTS.length; i++) {
            if (AC_GAME_OBJECTS[i] === this) {
                AC_GAME_OBJECTS.splice(i, 1);
                break;
            }
        }
        // Garbage Collection
    }
}


let last_timestamp;
let AC_GAME_ANIMATION = function (timestamp) {
    for (let i = 0; i < AC_GAME_OBJECTS.length; i++) {
        let obj = AC_GAME_OBJECTS[i];
        if (!obj.has_called_start) {
```

```
            obj.start();
            obj.has_called_start = true;
        } else {
            // 单位（毫秒）
            obj.timedelta = timestamp - last_timestamp;
            obj.update();
        }
    }

    for (let i = 0; i < AC_GAME_OBJECTS.length; i++) {
        let obj = AC_GAME_OBJECTS[i];
        obj.late_update();
    }

    last_timestamp = timestamp;

    // 递归调用
    requestAnimationFrame(AC_GAME_ANIMATION);
}

// 每帧中遍历所有的游戏对象，并调用它们的 start、update 和 late_update 方法。
requestAnimationFrame(AC_GAME_ANIMATION);
```

3.2.2 登录模块

以登录为例。

```
login_on_remote() {
    let outer = this;

    let username = this.$login_username.val();
    let password = this.$login_password.val();
    this.$login_error_message.empty();

    $.ajax({
        url: "https://app4986.acapp.acwing.com.cn/settings/login/",
        type: "GET",
        data: {
```

```
            username: username,
            password: password,
        },
        success: function (resp) {
            if (resp.result === "success") {
                location.reload();
            } else {
                outer.$login_error_message.html(resp.result);
            }
        }
    });
}
```

3.3 边缘部分

3.3.1 同步系统

以射击为例。

```
send_shoot_fireball(tx, ty, ball_uuid) {
    let outer = this;
    this.ws.send(JSON.stringify({
        "event": "shoot_fireball",
        "uuid": outer.uuid,
        "tx": tx,
        "ty": ty,
        "ball_uuid": ball_uuid
    }
    ));
}


receive_shoot_fireball(uuid, tx, ty, ball_uuid) {
    let player = this.get_player(uuid);
    if (player) {
        let fireball = player.shoot_fireball(tx, ty);
        fireball.uuid = ball_uuid;
    }
}
```

3.3.2 匹配系统

使用 Thrift 进行房间匹配。

```
queue = Queue()


class Player:
    def __init__(self, score, uuid, username, photo, channel_name):
        self.score = score
        self.uuid = uuid
        self.username = username
        self.photo = photo
        self.channel_name = channel_name
        self.waiting_time = 0


class Pool:
    def __init__(self):
        self.players = []

    def add_player(self, player):
        self.players.append(player)

    def increase_waiting_time(self):
        for player in self.players:
            player.waiting_time += 1

    def check_match(self, a: Player, b: Player):
        dt = abs(a.score - b.score)
        a_max_dif = a.waiting_time * 50
        b_max_dif = b.waiting_time * 50
        return dt <= a_max_dif and dt <= b_max_dif

    def match_success(self, ps):
        a, b, c = ps[0], ps[1], ps[2]
        print(f"Match Success: {a.username} {b.username} {c.username}")
        room_name = f"room-{a.uuid}-{b.uuid}-{c.uuid}"
        players = []
```

```
36
37          for p in ps:
38              async_to_sync(channel_layer.group_add)(room_name, p.channel_name)
39
40              players.append(
41                  {
42                      "uuid": p.uuid,
43                      "username": p.username,
44                      "photo": p.photo,
45                      "hp": 100,
46                  }
47              )
48
49          cache.set(room_name, players, 3600)
50
51          for p in ps:
52              async_to_sync(channel_layer.group_send)(
53                  room_name,
54                  {
55                      "type": "group_send_event",
56                      "event": "create_player",
57                      "uuid": p.uuid,
58                      "username": p.username,
59                      "photo": p.photo,
60                  },
61              )
62
63      def match(self):
64          while len(self.players) >= 3:
65              self.players = sorted(self.players, key=lambda p: p.score)
66              flag = False
67              for i in range(len(self.players) - 2):
68                  a, b, c = self.players[i], self.players[i + 1], self.players[i + 2]
69
70                  if (
71                      self.check_match(a, b)
72                      and self.check_match(a, c)
73                      and self.check_match(c, b)
```

```
                ):
                    self.match_success([a, b, c])
                    self.players = self.players[:i] + self.players[i + 3 :]
                    flag = True
                    break

            if not flag:
                break

        self.increase_waiting_time()


def get_player_from_queue():
    try:
        return queue.get_nowait()
    except:
        return None


def worker():
    pool = Pool()
    while True:
        player = get_player_from_queue()
        if player:
            pool.add_player(player)
        else:
            pool.match()
            sleep(1)


class MatchHandler:
    def add_player(self, score, uuid, username, photo, channel_name):
        console.print(f"add player {username} - {score}")
        player = Player(score, uuid, username, photo, channel_name)
        queue.put(player)
        return 0
```

```python
if __name__ == "__main__":
    handler = MatchHandler()
    processor = Match.Processor(handler)
    transport = TSocket.TServerSocket(host="127.0.0.1", port=9090)
    tfactory = TTransport.TBufferedTransportFactory()
    pfactory = TBinaryProtocol.TBinaryProtocolFactory()

    server = TServer.TThreadedServer(processor, transport, tfactory, pfactory)

    Thread(target=worker, daemon=True).start()

    print("Starting the server...")
    server.serve()
    print("done.")
```

4 运行情况

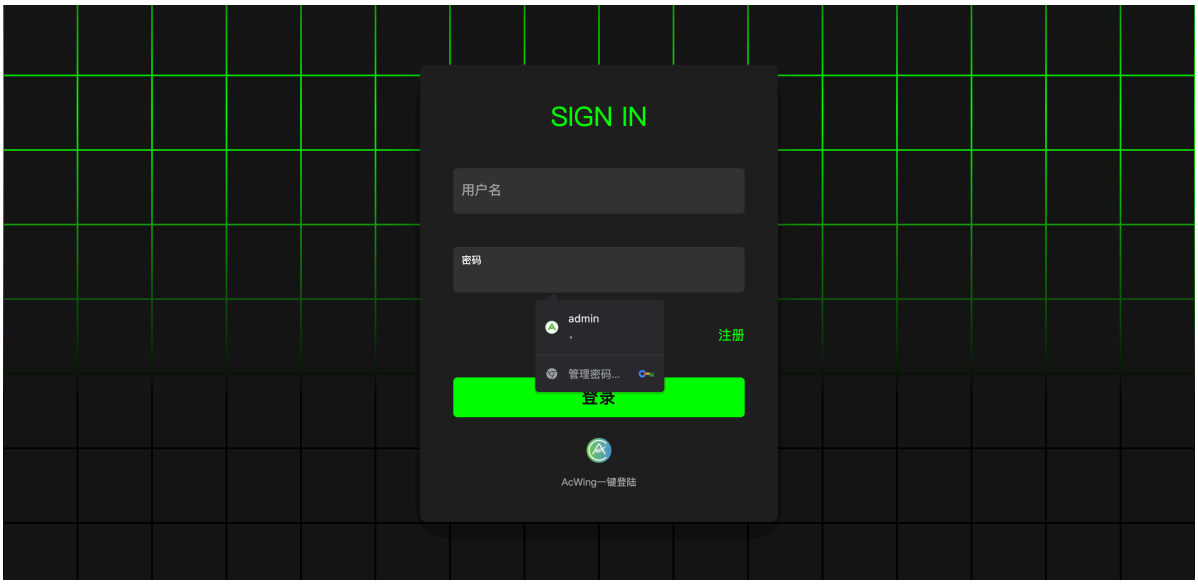

图 9: 注册/登录

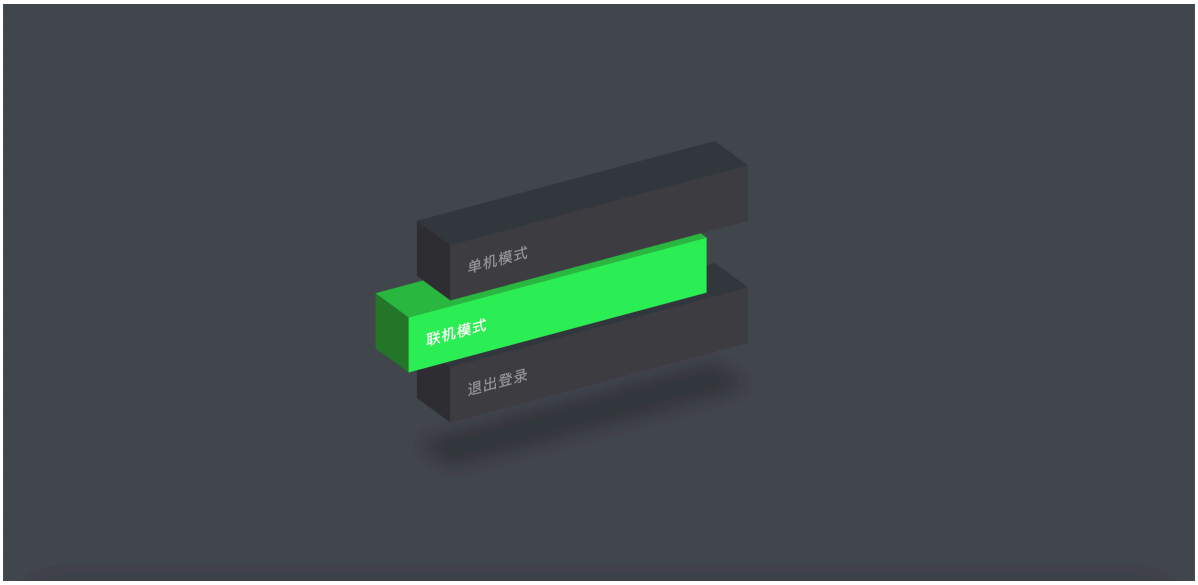

图 10: 菜单界面

图 11: 对局结果（以失败为例）

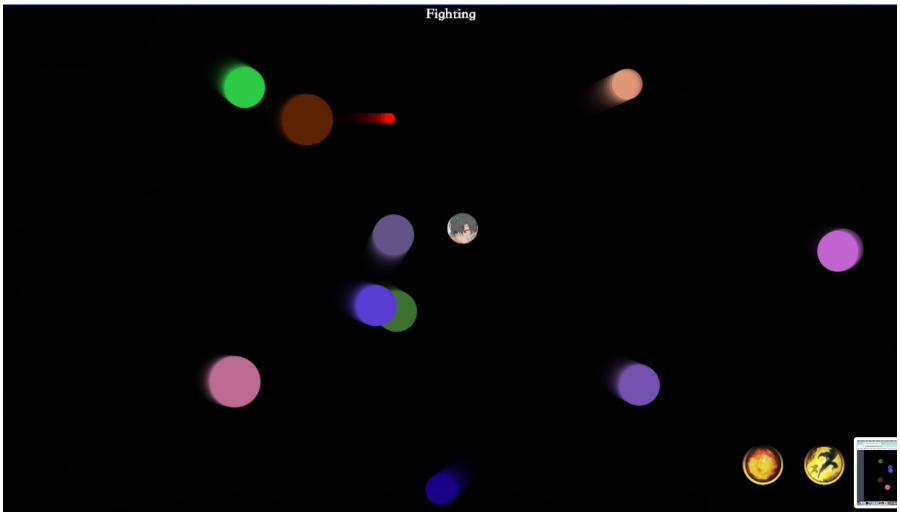

图 12: 对局界面（右下角展示技能冷却）

(a) 粒子特效 (b) 击中判定

图 13: 游戏细腻程度展示

5 课设总结

本项目是一个多人在线对战游戏，主要使用 Django 框架和 JavaScript 语言开发，综合运用 Thrift、Ajax、SQLite、Redis、OAuth2，Linux SSH、Shell、tmux 等技术。

于我而言，本项目是具有里程碑式的纪念意义的，不仅因为这是第一次在云服务器环境下开发应用，并使用 Nginx 部署至云端使他人可以通过域名访问，也是因为这是第一次接触游戏引擎，了解击中判定、游戏引擎等基本概念。

本游戏的亮点有技能冷却时间、房间匹配机制，拖影和粒子特效，人机对战、联机对战与双向通信，第三方免密授权登录等；当然，本游戏系统还较为简易，具有继续改进、打磨的空间。