南京大学计算机系
比赛对抗平台基本设计（仅供参考）

胡求 < huqiu00@163.com >

目录

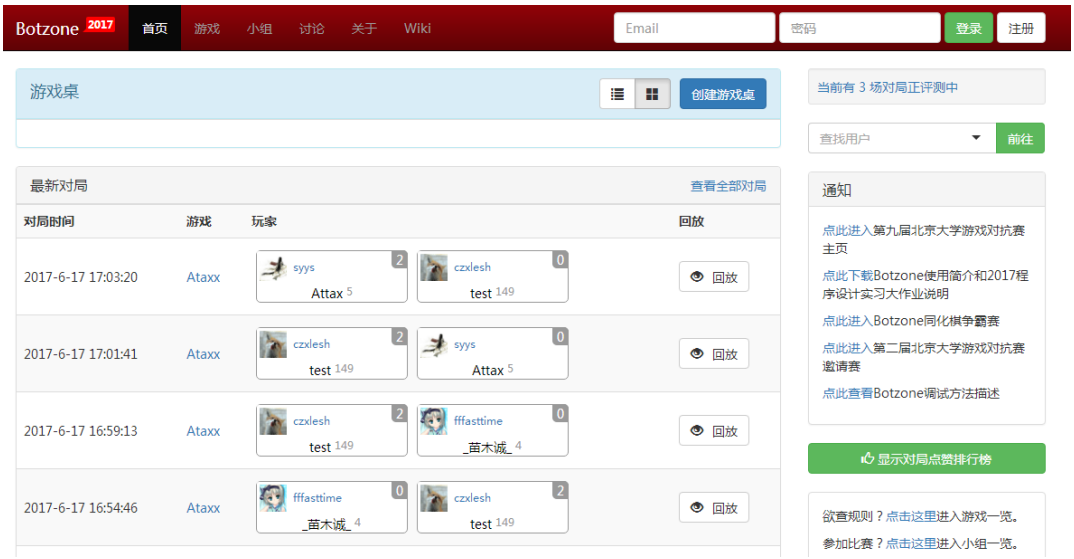
一、前言

一直以来，《程序设计基础实验》课程都包含一个比赛性质的，常常是在某个游戏规则下，学生们设计 AI 程序来实现互相对抗，最终按比赛成绩分出高下，这种形式往往比较有趣，吸引力强，能够极大调动学生热情，所以受到很大欢迎。最近两年举办的就有 2016 年中国象棋暗棋比赛，和 2017 年的花样五子棋比赛，都获得了巨大的成功。但是，每年一旦采取新的比赛项目，就需要助教重新写一套比赛评判，对抗平台，以及相应的客户端程序，大大消耗了助教和老师的时间，需要做很多重复的工作，往年的代码无法大量复用，甚至基本上要重写，往年花费大量时间编写好的评判端，客户端，在比赛过后即封存，导致了资源的一些浪费。所以编写一个比赛对抗的平台，将比赛中的对抗过程抽象起来，提供一套对抗程序进行在线对抗的接口，并提供一套可视化的界面，以更加调动学生的积极性，显得很有必要。

二、相关成果

截止目前，已有一些类似的对抗平台，比如：

1. 北京大学信息科学技术学院人工智能实验室开发的 Botzone 平台 (https://www.botzone.org/)，跟我们想做的很相似，提供了一个创建新奇规则的比赛(管理员)，提供上传代码，循环对抗，挑战，以及回放等功能，开放给外界使用。



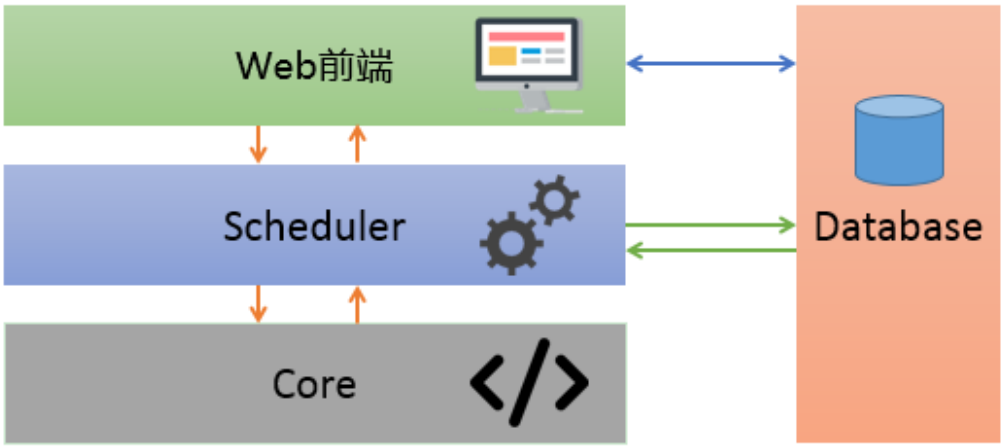
2. 电子科技大学 Online Judge 平台(http://uestc-acm.github.io/CDOJ)
ACM 相关，主要是 ACM 赛题的 judge 功能，但是开源，可以参考系统设计部分。

3. 中国人民大学信息学院计算机系设计的 OVS 系统

等，可以看出，除了 ACM 竞赛平台外，许多平台都是在人工智能课程上延伸出来的，作为教学之用，对抗程序也基本都是人工智能 AI。

三、平台基本架构

平台初步的一个设计如下图：



平台大致分为三层结构，包括 web 前端层，调度层（或称服务层 Service），以及执行程序的核心层。下面对每一层基本功能，使用的技术和架构，数据库以及各层功能之间的互动做一个介绍。

3.1 Web 前端层

功能包括向用户展示所有功能，包括比赛界面，回放界面，排名界面，以及提供用户的登录和注册，以及比赛的添加，修改，删除等操作等。
这部分会频繁地与数据库进行交互，用户信息，比赛信息，程序代码等等都保存在数据库中。
利用 js 提供动态回放，这点很重要。

技术方面，这部分可采用 bootstrap，或者 jQuery 等前端技术做前端展示，显示内容以 javascript 控制为主，使用 Spring + Hibernates + Struts 的经典框架，做一个经典的 MVC 模型。

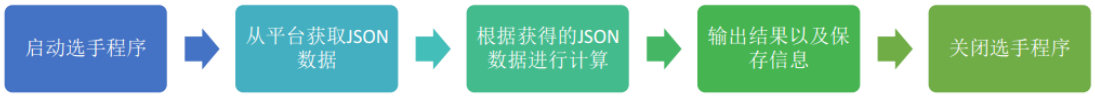
3.2 调度层

调度层(Scheduler)主要负责调度执行未完成的对战，这个阶段维护一个对战信息数据库，存储对战日志和对战结果。

以前我们是做了一个服务器，再编写了一个客户端，将客户端的通信模块（socket 通信）为同学们写好，然后通过同步阻塞 IO 来一步一步发送消息，客户端接收消息，做出决策，发送消息，再接收消息...如此循环往复，直到完成比赛。但是这种方式有一个缺点就是网络传输环境很复杂，在登入服务器和消息传递的过程中可能会遇到莫名奇妙的错误，所以客户端程序经常输出一些错误信息，但是又找不到原因，虽然说这不影响比赛过程，但是可能会影响比赛公平性和同学们的体验。

在对抗平台中，由于情况更加变幻莫测，我们应该尽量规避以 socket 形式来进行

通信。借鉴已有的平台的经验，我们可以采用循环开启和关闭程序的方法来避免两个程序同时开启和服务进程通信的情况，直接将消息的传递转变为调用程序时将消息作为输入/参数传入。过程如下：

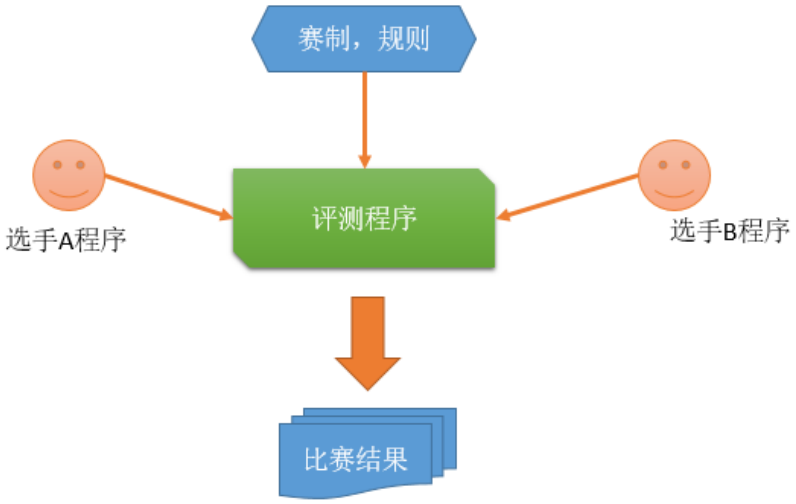


循环执行如上过程，直到平台判定游戏结束，或者一方出现错误，游戏停止，产生游戏结果。

比如对于需要进行的一场对战 A vs. B，开启一个进/线程来进行对战，首先向 A, B 发送初始化信息，向先手一方同时发送 step 请求，表示此时由该方做出决策/动作（这里的“发送”可以是开启 A 程序，然后将以 json 格式表示的信息作为输入），然后关闭 A 程序，将 A 的输出做一定处理，并判定 A 的 step 是否正确，比赛是否结束等等，然后产生给 B 的输入，开启 B 程序，将输入传入，循环往复，直到结束。

调度层只管调度程序执行，程序的真正执行为下面的核心层。

调度层（评测程序）大致功能如下：



可参考架构：使用原内核，一个主线程，多个子线程，主线程管理全局对战队列，子线程用来执行单场对战。

3.3 核心层

核心层(core)负责真正地执行用户程序，接收 scheduler 的输入，然后运行产生 step，返回给 scheduler。

这部分需要用到 linux 系统中许多底层的功能，需要对 linux 较熟悉的人员来完

成。

可参考架构：使用原内核，对战过程中，编译和运行过程分为两个进程，编译时，子进程编译，父进程监控编译的时间。运行时，子进程运行，父进程监控运行时间以及占用的内存等信息，判断超时和超内存，一旦出现，停止程序，避免对系统的损害。

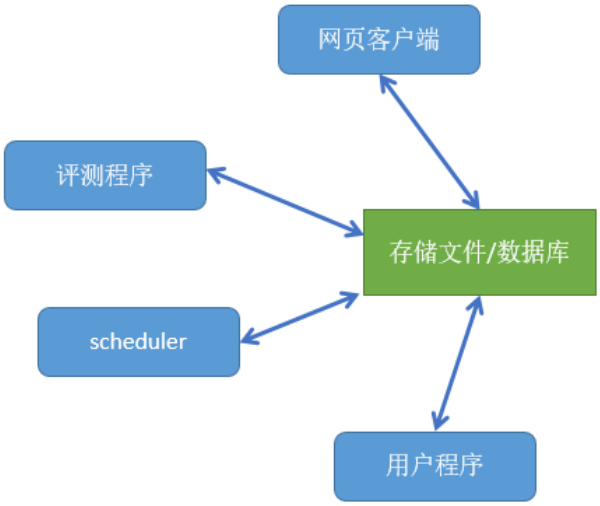
3.4 存储层 (数据库或文件)

数据库或者文件负责存储诸如用户密码信息，比赛信息，对局信息（结果和日志）等。

可采用 MySQL 引擎或者 SQLite 轻量级引擎等，比赛记录等信息也可直接采用文件存储。用户名密码最好使用数据库。

如使用数据库，需要对数据库表进行一系列详细可靠的设计。

存储层是一个很重要的模块，许多模块都与它有所交互。



四、调试

同学们在阅读比赛规则后，开始编写程序，在上传之前需要对他们的程序进行调试，发现各种错误并解决，以确保程序正确性，并且可以测试其程序的效果。所以需要提供一个调试功能。

调试功能初步设想是后端做一个调试调度程序，也可 scheduler 做一个调试部分，系统默认几个标准 AI 供同学们测试，由于我们消除了通信部分，所以只能是同

学们上传他们可能包含错误的程序，然后同学点击调试功能，选择对战测试的 AI，然后系统进行评测对战，返回调试信息，同学们根据调试信息来改进程序。
(这样可能的弊端在于，需要频繁地上传程序，频繁调试，给系统增加了很多负载，并且同学们的程序错误可能会影响系统。)

除了上传调试，还可以进行本地调试，即手动自己与自己写的 AI 对战，也能够发现许多错误。

五、对战回放

对战回放直接读取数据库（或者存成文件也可以？）中的比赛日志，然后通过 js 等渲染，通过一个良好的可视化界面展示出来，对于每个比赛都应该有一个通用的比赛界面模板，然后回放或者是实时对抗的时候，直接开启一个新页面进行信息读取和展示。

六、消息格式

由于 JSON 格式的标准性和可读性，可以选用作为我们的输入输出格式。

比如 Botzone 的 JSON 消息格式如下：

输入 JSON 数据格式：

Bot Input

```
{
    "requests" : [
        "Judge request in Turn 1", // 第 1 回合从平台获取的信息
        "Judge request in Turn 1", // 第 2 回合从平台获取的信息
        ...
    ],
    "responses" : [
        "Bot response in Turn 1",  // 第 1 回合输出的信息信息
        "Bot response in Turn 2",  // 第 2 回合输出的信息信息
        ...
    ],
    "data" : "saved data",   // 保存的信息
    "time_limit" : "",       // 时间限制
```

Bot Output

```
{
    "response" : "response msg",     // 选手此回合的输出信息
    "data" : "saved data"            // 选手此回合的保存信息
}
```

七、错误信息

1. TimeLimitExceed：超时
2. MemoryLimitExceed：超内存
3. OutputLimitExceed：输出数据量过大
4. NoJSON：输出不是 JSON 格式
5. RuntimeError：运行时错误

八、约战

可以支持创建房间，单独约战的模式，大致流程如下：

1. 在首页创建房间
2. 给各个位置分配用户程序(Robot)或等待人类玩家加入
3. 开始游戏并实时观看（作为人类玩家，可以在实时对局界面参与游戏）
4. 对局结束后可以进行回放