Gomoku3d

— 3D 五子棋 —

Nekomaru@pku.edu.cn

And other Gomoku3d contributors

项目简介和实机演示

我们综合使用了较前沿的 Web 前端和后端技术，实现了一款面向浏览器的 **3D 五子棋**游戏。

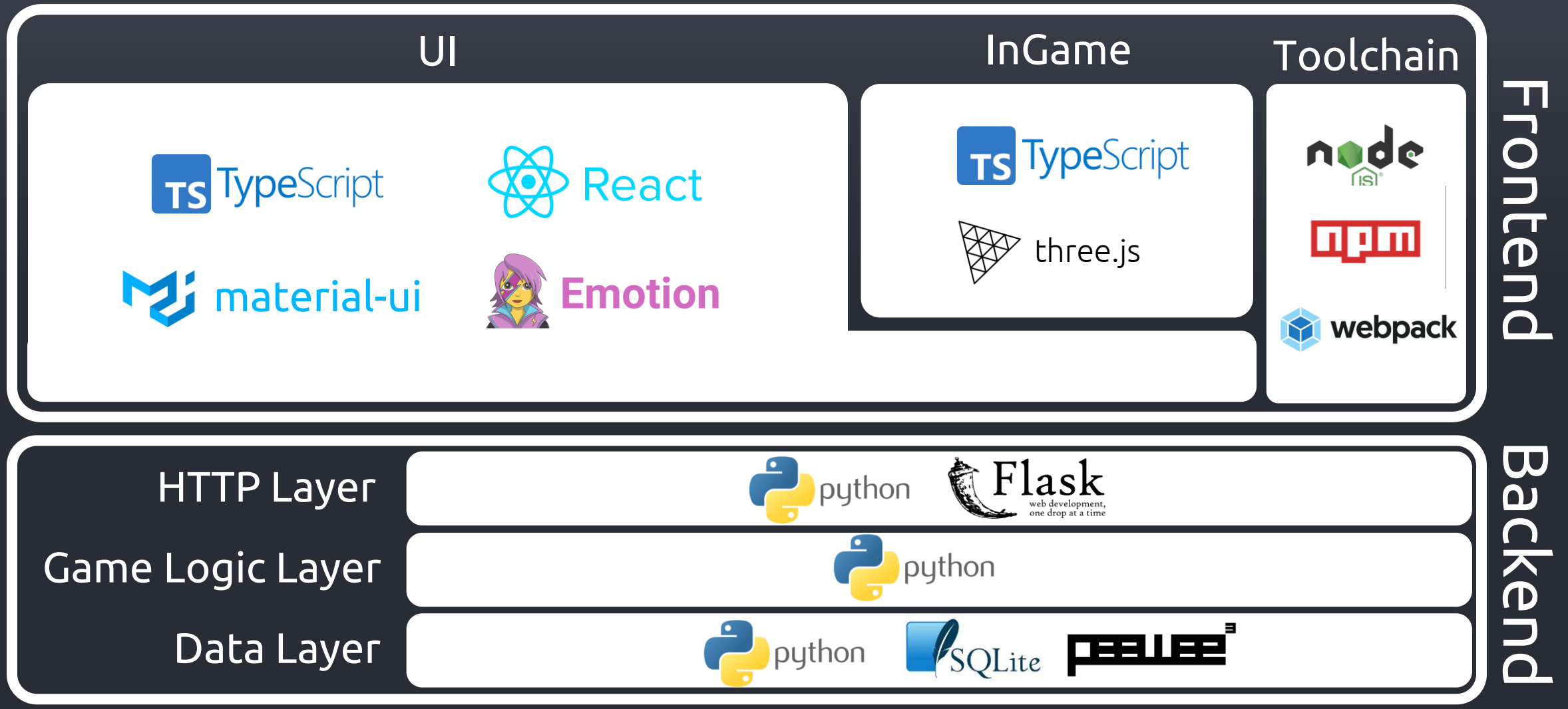
首先我们将现场开启一场双人对局，展示本次项目的主要成果。

此外项目展示期间，校园网内的同学们可以访问：

http://10. ▮▮▮. ▮▮▮. ▮▮▮:5000/

进行匹配对战或人机对战。（请不要攻击此服务器 orz）

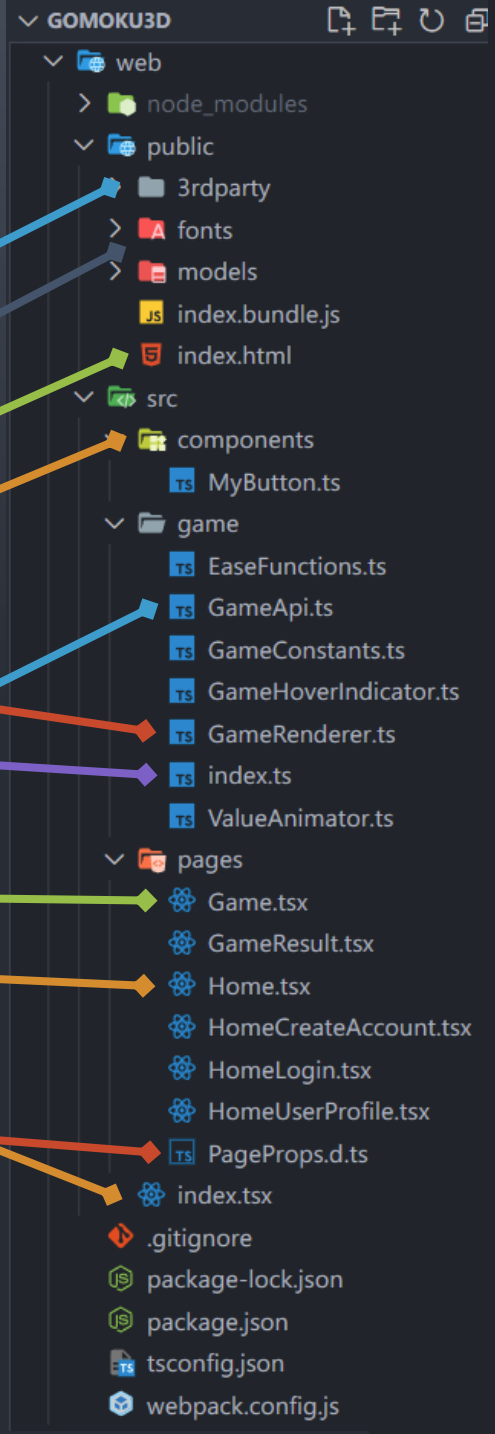
项目整体架构和技术选型



* 连线颜色仅作区分

前端目录结构

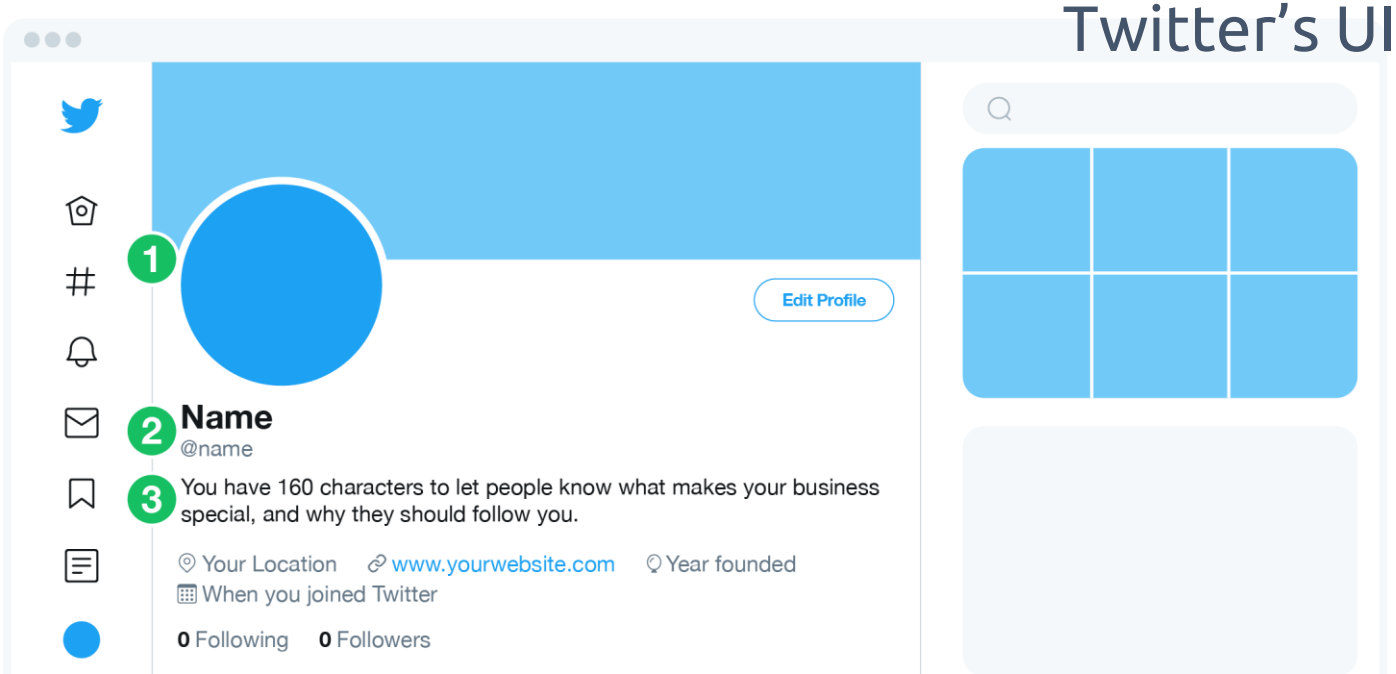
- 不经过 Webpack 打包的 JS 依赖库（react, react-dom, three）
- 字体和 3D 模型资源
- Webpack Bundle 和空荡荡的 HTML 页面
- 自定义 UI 控件库
- 游戏渲染层
- 游戏逻辑层
- 游戏和后端交互的 API 封装层
- 游戏页面
- 主页面及构成主页面的各个组件
- 打包入口，其实是个页面选择器
- 页面选择器的前向声明



前端 UI 简介/平面设计

我们整个前端采用了 Google 提出的 Material Design 设计语言，
主要的参考对象为 Twitter。
而我们全面使用 Ubuntu Font 则纯粹是个人喜好 XD

Twitter's UI



Our UI

前端 UI 简介/ React & Emotion

我们整个前端基于 React 库实现为单页面应用，页面间使用根 Props (type PageProps) 共享状态数据。

我们的所有 React 组件均为函数式组件，通过 React Hooks 实现状态保持和生命周期。

我们采用了 Emotion 库 (@emotion/styled, @emotion/react) 作为 CSS-in-JS 解决方案。

Discussions:

- 按道理我们应该上 React Router 最后没有上纯粹是因为项目太小了ww

前端 UI 简介/ Material UI

我们的前端中大量使用了 Material UI (@mui/material) 库提供的 React 组件作为构成 UI 的基本单元，并在此基础上实现了响应式布局等 UI 特性。

Discussions:

- Ripple animation yyds!!

AD -> MD

1. 将所有的 Blur Effects 区域填充为纯白；
2. 将应用最上面和最下面的区域填充为高饱和度的纯色；
3. 将一些具有可视边界的元素填充为另一种更加鲜艳的高饱和度的纯色；
4. 把所有连在一起的按钮掰开；
5. 给所有的具有可视边界的元素添加 Drop Shadow Effect；
6. 给所有的可点击元素添加 Ripple Effects；
7. 添加大量的过于花哨的 Transitions & Animations；
8. 买一个绿得恰如其分的 Android 小机器人手办；
9. 享受美好生活；

前端游戏引擎简介

我们的游戏引擎基于知名 3D 库 Three.js 开发，底层渲染器是 WebGL。我们使用了 Three.js 库中的场景管理，预置材质，光线投射和 OBJ 模型加载特性。

但因为 Three.js 库是个 3D 渲染库而非完整的游戏引擎，我们重新开发了简单的插值动画系统（以 ValueAnimator.ts 为代表），并以此为基础实现了游戏内的交互反馈。

Discussions:

- Easing functions 来源于 https://github.com/AndrewRayCode/easing-utils 。

前端游戏逻辑与交互简介

因为写逻辑很无聊，我们的游戏逻辑大多交给后端，前端则在非玩家落子的轮次轮询等待，而在玩家落子的轮次提供交互。

我们也实现了断线重连等功能。

为了处理 3D 五子棋的交互，我们在场景中维护了三套棋盘：一套用于实际落子，一套用于鼠标悬浮的判定，一套用于可交互位置的指示。

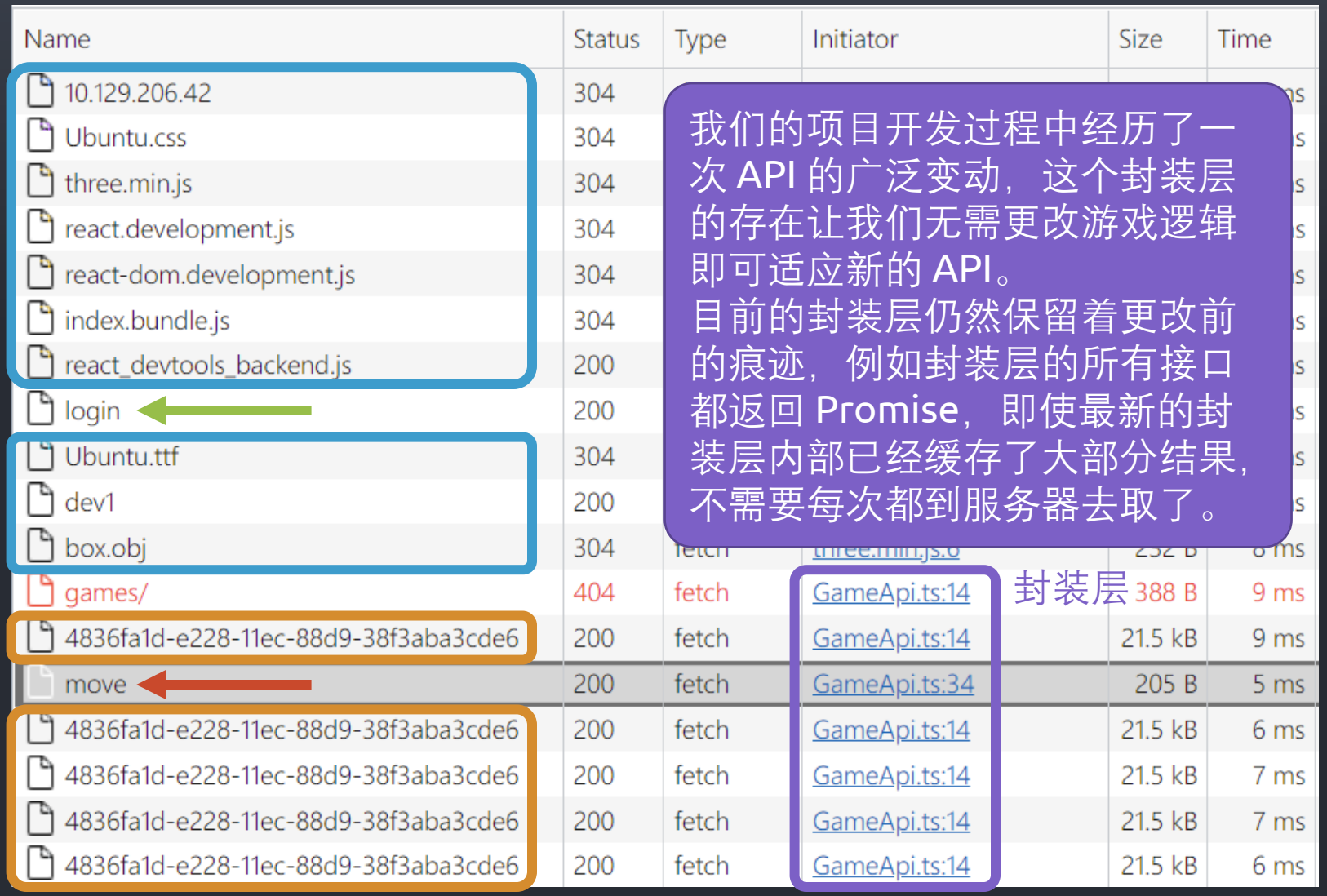
值得一提的是，这三套棋盘中所有格子上都放着棋子，按需显示隐藏而非按需添加删除。这是因为对于现代图形硬件而言，更新已经存在的场景物体的状态比增加新的场景物体开销要低。

RESTful API 示例

单页面应用的共用资源

前端查询登录 Session 状态
当前用户仍有正在进行的对局
故路由至游戏页面继续游戏

拉取/轮询当前棋局，以 UUID 标识
无想的一着



我们的项目开发过程中经历了一次 API 的广泛变动，这个封装层的存在让我们无需更改游戏逻辑即可适应新的 API。
目前的封装层仍然保留着更改前的痕迹，例如封装层的所有接口都返回 Promise，即使最新的封装层内部已经缓存了大部分结果，不需要每次都到服务器去取了。

后端简介

我们的后端 HTTP Server 基于 Python (Flask) 实现。后端数据库使用了 SQLite，同时引入 ORM 库 peewee 来简化数据库访问。

我们的人机对战 AI 与 HTTP Server 以 Master-Slave 架构运行在不同的进程内，通过管道交互，不会阻塞 HTTP 响应。

Discussions:

- 我们的后端没有用 Node.js 纯粹是因为 Node.js 已经玩腻了（笑）
- 我们的人机对战所使用的 AI 不是我们小组成员开发的，不过既然这门课不是算法课（逃）

讨论与展望 / 不打算做的事情

- 游戏大厅（房间，聊天，好友，blahblah）
- 更多用户资料（您已沉迷棋事 xxx 天）
- 既然都五子棋了，围棋 or 象棋 or WHATEVER

无非是 UI 和后端逻辑的堆砌。我们现在做的足以证明我们能做到。

讨论与展望 / 值得做的事情-前端 UI

- 后端报错的处理
  - 目前我们的前端是假设后端完全不出错（用户错误还是能处理的），且 HTTP 完全可信的。而生产标准下的前端是要能够处理比方说 500 或者服务器给出了完全没有设想的响应时的情况的。
- 最基本的安全性
  - 虽然我们不上 HTTPS 是因为买不起域名（不），但是明文传输用户密码永远是罪过 orz

讨论与展望 / 值得做的事情-前端 InGame

- 游戏逻辑与后端的通过 WebSocket 而非 RESTful + 轮询来交互；
- 更好的游戏内交互；
  - 3D 五子棋如何让落子变得自然而不易出错真是个天坑（趴）
- 游戏内增加 HUD UI；
  - 最典型地，显示当前参与对局的玩家名（以及让正在落子的玩家 Label 一跳一跳的 XD）
  - 其实我们现在已经有了调试用的 “局中认输” API，加个按钮就很自然地变成公开功能了；

Thanks

欢迎来玩 ⇒ http://10. ▮▮▮. ▮▮▮. ▮▮▮:5000/

项目已在 GitHub 上以 GPL v3 or Later 协议开源