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 Python 后端和 Web 前端技术，实现了一款面向浏览器的 **3D 五子棋**游戏。

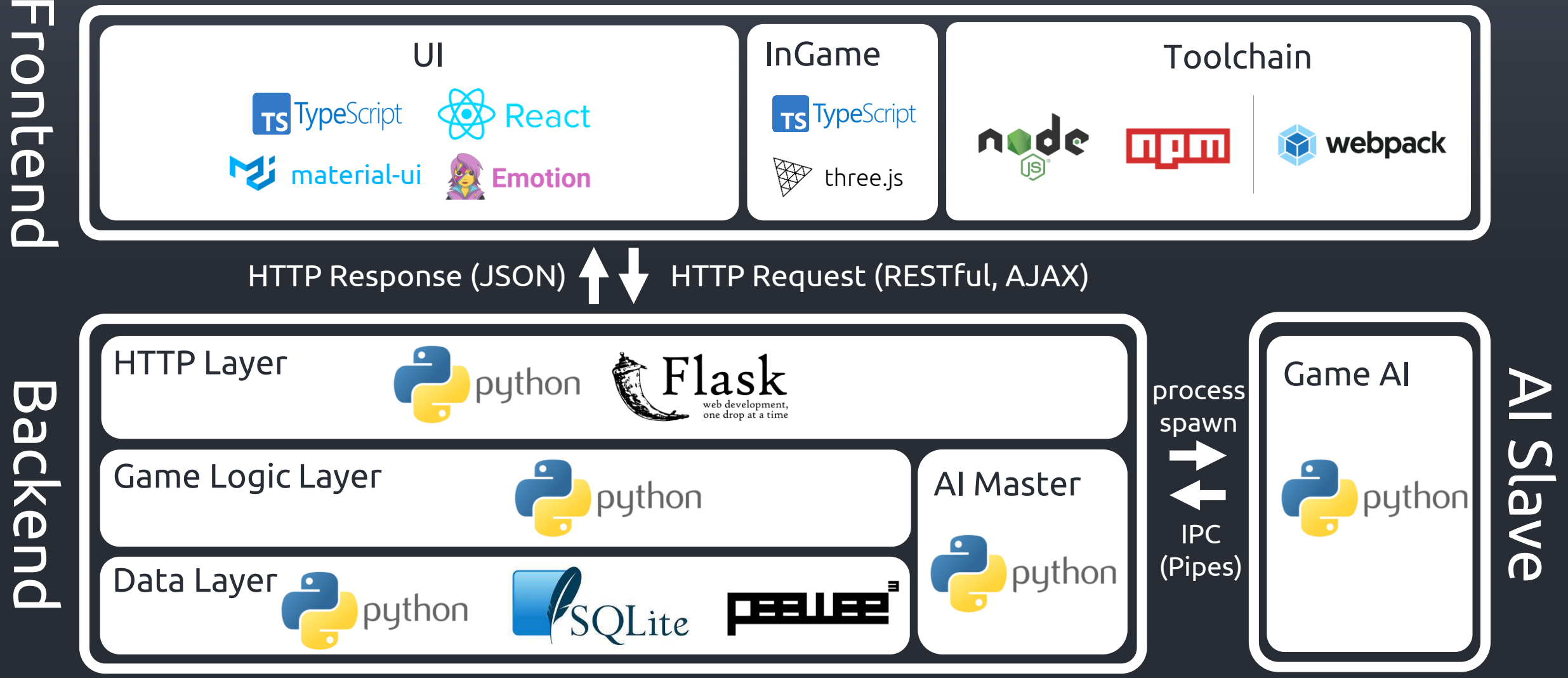
首先我们将现场开启一场人机对战，展示本次项目的主要成果。

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）

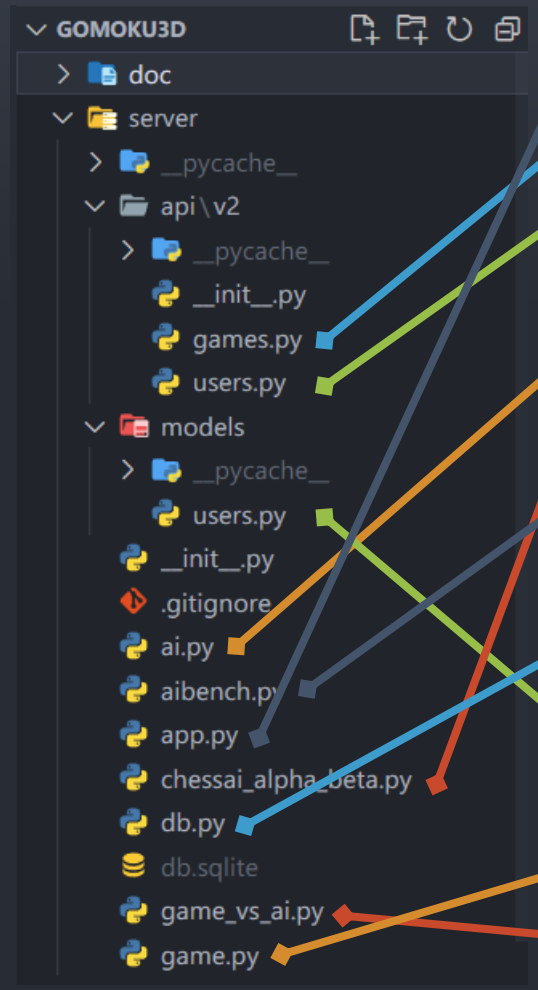
项目整体架构和技术选型



项目目录结构

* 连线颜色仅作区分

Backend & AI



Flask App

游戏相关 RESTful API

用户注册登录相关 RESTful API

初版人机对战 AI

威力加强版人机对战 AI

人机对战 AI 的单元测试

peewee 数据库初始化

用户表的数据模型定义

游戏逻辑 (Game 类)

人机对战逻辑 (GameVsAI 类，继承自 Game 类)

Frontend

JS 依赖库（react, react-dom, three）

字体和 3D 模型资源

打包好的 HTML-JS

自定义 UI 控件库

游戏渲染层

游戏逻辑层

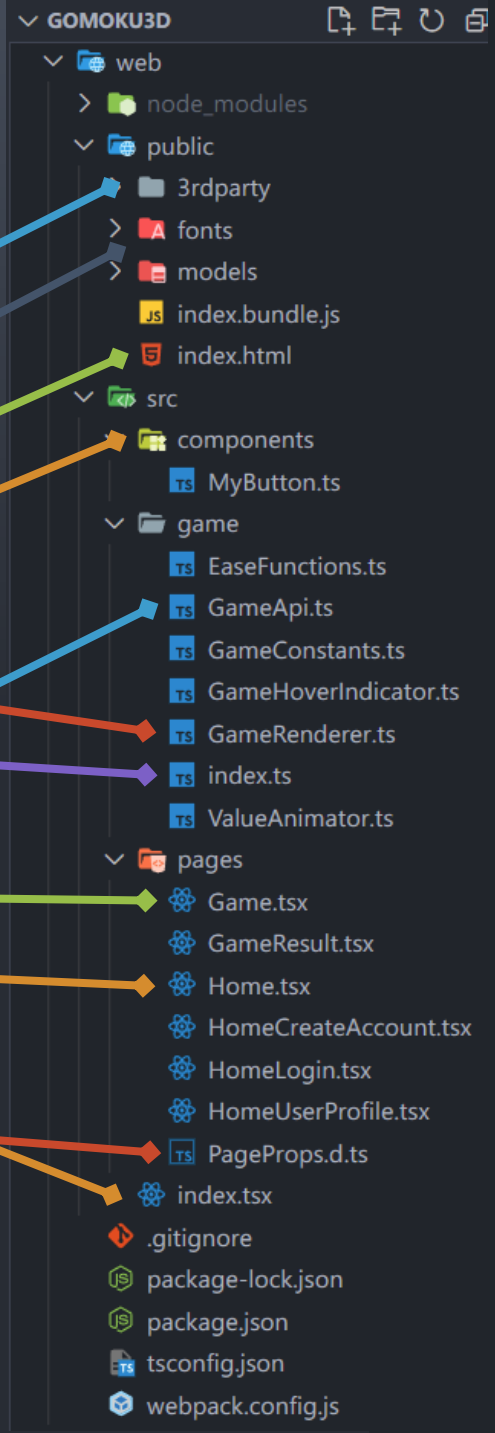
游戏 API 封装层

游戏页面

主页面及各个组件

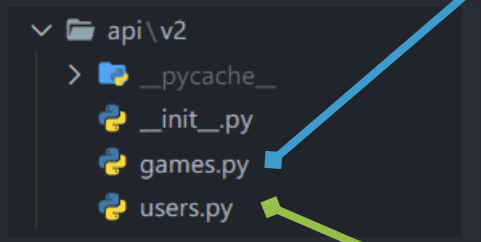
页面选择器

页面选择器的前向声明



后端简介 / Flask-based HTTP Server

我们的 HTTP 后端基于 Python (Flask) 实现，并利用 Flask 2+ 引入的蓝图特性进行模块化。



api.v2.games 模块
负责游戏逻辑

```
@app.route("join", methods=["GET"])
@app.route("join-vs-ai", methods=["GET"])
@app.route("games/<string:game_id>", methods=["GET"])
@app.route("games/<string:game_id>/move", methods=["POST"])
@app.route("games/<string:game_id>/giveup", methods=["POST"])
```

api.v2.users 模块
负责用户注册登录

```
@app.route("login", methods=["GET"])
@app.route("login", methods=["POST"])
@app.route("login", methods=["DELETE"])
@app.route("users/<string:username>", methods=["HEAD"])
@app.route("users/<string:username>", methods=["GET"])
@app.route("users", methods=["POST"])
```

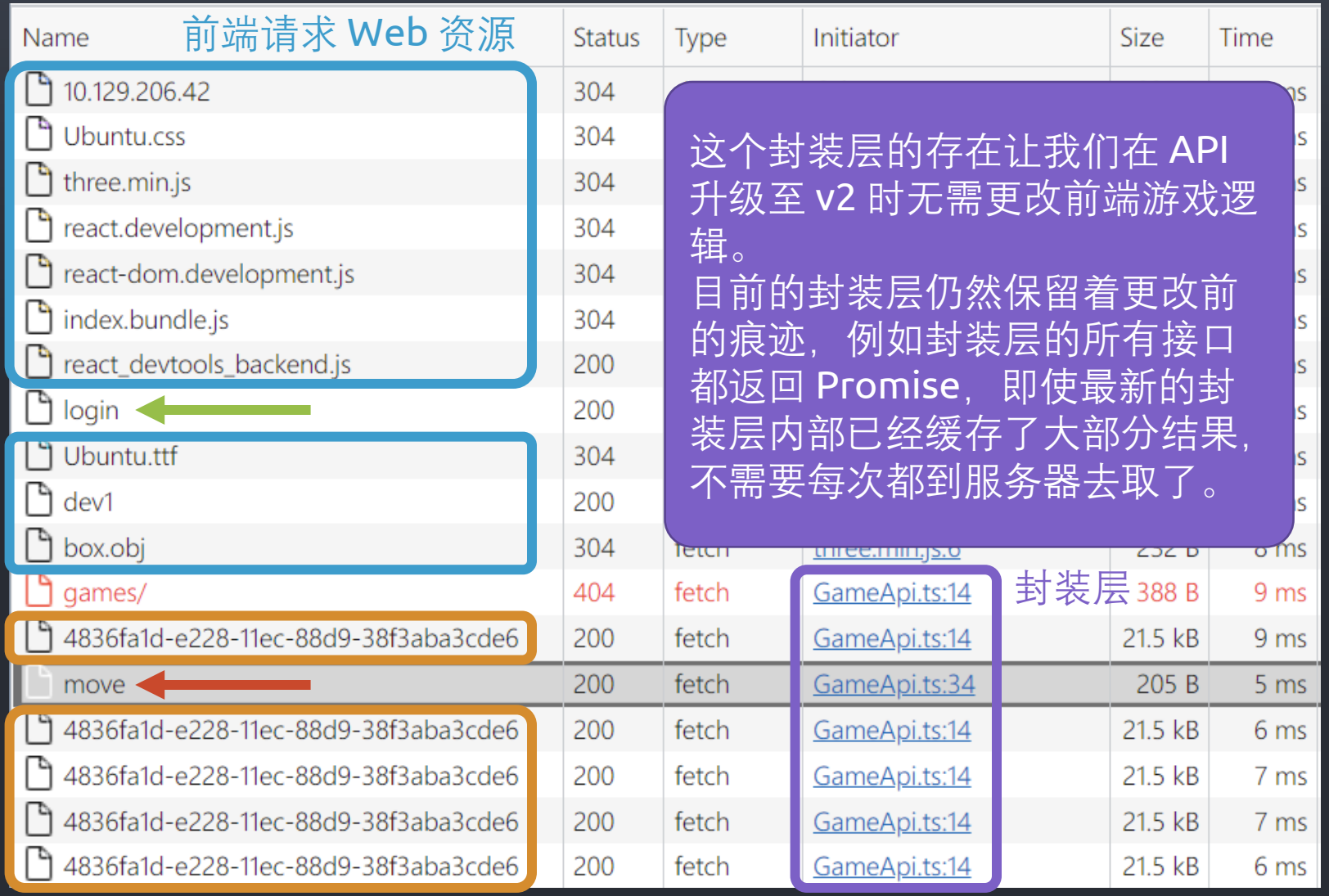
关于消失的 api.v1: 这是我们项目原型阶段（当它还是我的作业 4 的时候）用的一套 API，因为当时实现的略显很草率，后来我决心推倒重做，于是便有了 api.v2。

后端简介 / HTTP Server 请求与响应示例

我们的游戏逻辑由 Python 后端驱动前端实现，前端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。

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



这个封装层的存在让我们在 API 升级至 v2 时无需更改前端游戏逻辑。
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 / ORM-driven Database

我们的后端数据库为 SQLite，同时引入 ORM 库 peewee 来简化数据库访问。

ORM 全称 Object-Relational Mapping (对象-关系映射)，其主要作用是把程序中的类和数据库中的表对应起来，将对象上的操作转化为数据库中记录上的操作。ORM 是下一代的数据库访问技术，免除了手动拼接 SQL 语句的繁琐和可能的安全风险。

下面的示例是最小化的使用 ORM 库 peewee 定义 “用户表” 的代码：

```
from peewee import *

class DbUser(DbModel):
    username   = CharField    (null=False, max_length=31, unique=True, index=True)
    password   = CharField    (null=False, max_length=31)
    nickname   = CharField    (null=False, max_length=31)
    created_at = DateTimeField(null=False, default=datetime.now)
```

* 摘自 server/models/users.py

后端简介 / Game AI

我们的人机对战 AI 由组员 @rongyiming 同学编写，其核心是基于深度优先搜索的 minimax 算法和 alpha-beta 剪枝。它的胜率相当不错。

为了避免人机对战 AI 的高强度计算阻塞 HTTP 响应，我们使用了 Python 标准库中的 multiprocessing 模块，让人机对战 AI 与 HTTP Server 以 Master-Slave 架构运行在不同的进程内。

每场正在进行的人机对战都会派发出新的 AI 工作进程，与主进程之间经由管道通信。AI 工作进程轮询等待主进程向管道内送入最新的对局信息，并将计算结果经由管道返回主进程。

前端简介 / Game Engine

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 。

前端简介 / Game Interaction

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。

我们也实现了断线重连等功能。

前端简介 / React-based UI

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。

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

我们的前端中广泛使用 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。

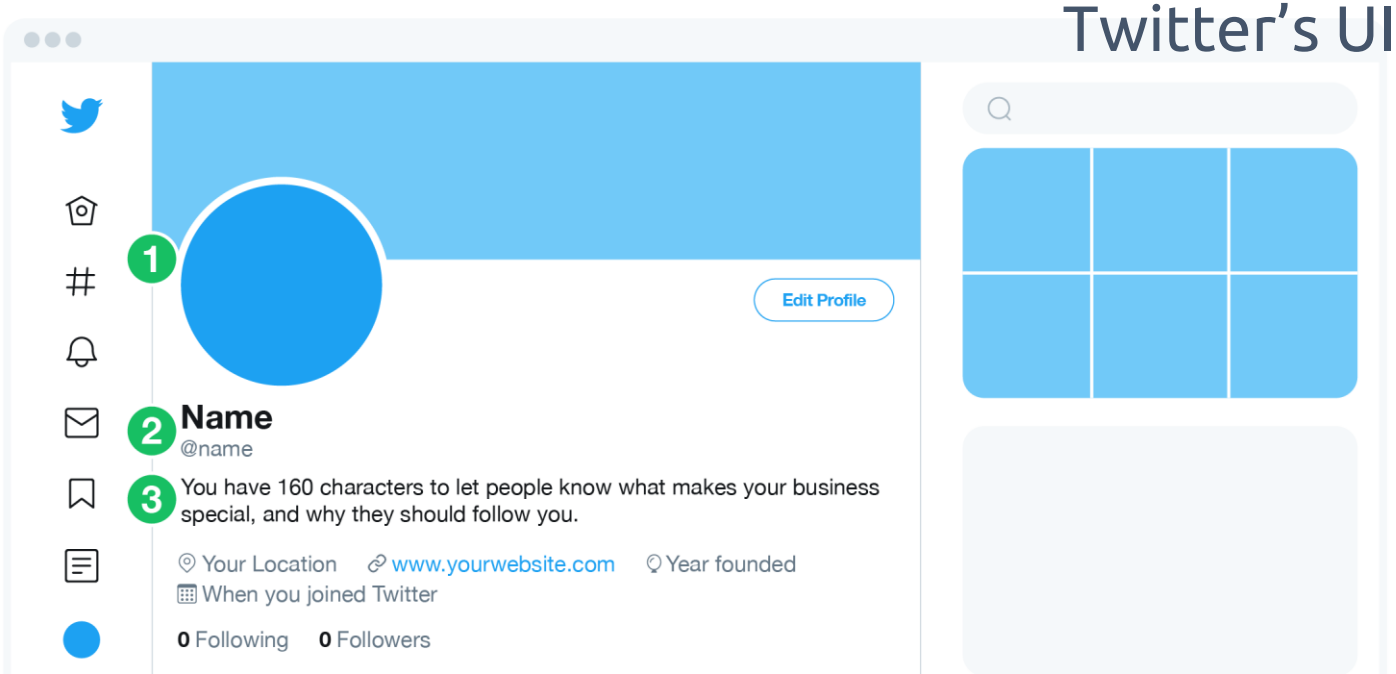
Discussions:

- Ripple animation yyds!!
- 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

讨论与展望 / 堆工作量可以解决的事

- 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

- 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

- 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