

# Wordle 大作业报告

BY 张扬 致理书院 2023012455

## 1 程序结构说明

程序主要分为 `config`, `interactor`, `plate`, `statistic`, `word_gen`, `web` 几个模块。

### 1.1 config

代码文件：`src/config.rs`。

主要实现 `Args` 和 `Config` 类。`Args` 类代表用户指定的所有参数，使用 `clap` 库从命令行解析，使用 `serde_json` 库从配置文件解析。随后将 `Args` 转换为程序的配置 `Config`，主要过程为读取、解析后选词列表，与解析目标词选取方式（分为 `Ask`, `Select`, `Random` 三种）。

### 1.2 interactor

代码文件：`src/interactor.rs`, `src/interactor/cmd.rs`, `src/interactor/tty.rs`。

`interactor.rs` 指定了交互器的 `trait`。`cmd.rs` 和 `tty.rs` 分别为测试模式、交互模式的实现。

### 1.3 plate

代码文件：`src/plate.rs`。

`plate` 实现了游戏过程的主体。通过指定 `goal`（猜测目标）和 `difficult`（是否开启困难模式）构造一个 `Plate` 对象。`Plate` 对象实现了 `guess` 方法，用于处理用户的猜测，该方法通过返回 `Result<(), >` 来处理困难模式下的非法猜测等情形。`Plate` 对象实现了 `is_win()`, `history()` 等接口，用于外部读取游戏信息。

### 1.4 statistic

代码文件：`src/statistic.rs`。

`statistic` 实现了游戏的统计信息。`Statistic` 对象通过 `load_from_file`, `load_from_json`, `store_to_file`, `store_to_json` 等方法，实现了数据持久化。在每局游戏结束后，通过 `add_plate` 方法收集该轮游戏的统计信息。`Statistic` 对象通过 `HashMap` 统计单词使用频率，通过 `TreeSet` 实时维护使用频率最高的单词列表，提供了 `top5_words` 接口返回使用频率最高的 5 个单词的迭代器。

### 1.5 word\_gen

代码文件：`src/word_gen.rs`。

`word_gen` 模块读取 `Config` 中的相关信息，提供了待猜测词的（多态的）迭代器接口。对于 `Ask`, `Select`, `Random` 三种模式，分别实现了其迭代器，特别的，对于 `Ask` 和 `Random` 模式，通过读取用户输入的 `Y/N` 来判断是否开启下一局。

### 1.6 web

代码文件：`src/bin/web.rs`。

使用 web\_sys 模块和 yew 框架实现了网页中的 Wordle 游戏。web\_sys 提供了 DOM/Js 对象到 rust 对象的接口，yew 框架使用类 React 的 props/state 模式，配合 VDOM 动态渲染页面。游戏主体由 App 组件构成，其 state 包括游戏配置 Config。初始时 Config 为 None，此时渲染 SetConfig 子组件。SetConfig 组件通过 Html 的 Form 表单收集游戏配置信息，在提交时通过 yew 框架提供的 Callback 对象将信息传递给父组件 App。App 在收到配置信息后开始一局游戏，渲染 GameBoard，Keyboard，Statistic 三个子组件，分别对应游戏中的猜测区域，键盘区域和统计信息区域。GameBoard 组件显示历史猜测和当前的输入，对于当前的输入，通过对不可见的 input 节点添加 keydown 事件的监听器，实现用户的动态输入与显示。Keyboard 组件使用了类似键盘的字母布局，显示所有字母的状态。Statistic 组件显示统计信息，并提供 clear statistic 选项用于清空历史统计信息。

## 2 游戏的主要功能

### 2.1 测试模式

测试模式按照要求以纯文本方式输出所有信息，能通过所有测试用例。

### 2.2 交互模式

交互模式提供了对用户更为友好的颜色输出：

```
---
WANNA
ABUSE
CARGO
TESTY
VAGUE
---
ABCDEFGHIJKLMN OPQRSTUVWXYZ
---
CORRECT 5
Y■
```

### 2.3 网页模式

Difficult mode: ☐

Game mode: Random ▾

Select Word:

Random seed: 2

Acceptable list:

aahed

aalii

aargh

aarti

abaca

abaci

Final list:

aback

abase

abate

abbey

abbot

abhor

Start Wordle!



### 3 提高要求的实现方式：

见 1.6 节 web.rs 的实现。

除此之外，`index.html` 和 `index.css` 实现了网页的整体框架和样式配置。

使用 `yew` 官方推荐的 `trunk` 作为调试运行和打包的工具。

### 4 作业感想

感觉 `rust` 写前端这事就是为了用 `wasm` 硬造出来的东西。写代码的体验来说 `rust` 完全比不上 `typescript`，运行效率来说，先不管 `Js` 对象到 `rust` 对象转换造成的性能损失，`rust` 里面（几乎是唯一解的）大片的 `Rc RefCell` 未必比纯 `Reference type` 和 `GC` 的 `js` 加上 `v8` 的极致优化跑得快。何况前端这玩意的对象依赖复杂起来，不太像是纯靠引用计数就能很好解决的。

然后是 `rust` 的异步，`async` 在 `rust` 里似乎被搞得很复杂。本来 `async` 方式和 `callback` 方式只是差一个 `cps` 变换，但似乎是因为 `rust` 在类型系统和生命周期上的一些缺陷使得 `cps` 变换不能以 `trivial` 的方式进行。

一个例子是，`rust` 对于返回值简单地将其生命周期绑定到 `caller` 的函数体上。这对于值类型是合理的，但是对于借用类型来说是完全错误的。借用类型的生命周期与其被定义的地方没有任何关系。这导致如下等价的写法中：

```
fn foo_return<'a>() -> &'a String {
    return &"foo".to_string() /* temporary value */;
}

fn foo_ref<'a>(res: &mut &'a String) {
    *res = &"foo".to_string() /* temporary value */;
}

fn foo_k<K: FnOnce(&String) -> !>(k: K) -> ! {
    k(&"foo".to_string() /* temporary value */)
}
```

只有 `foo_k` 能通过 `borrow checker`。而 `foo_k` 事实上是一个作弊的实现，它把临时值绑定到了 `foo_k` 函数提上，对于 `cps` 而言这就是 `'static`。如果 `rust` 的优化器在生命周期擦除之前（而不是依赖 `llvm`）的话，这个 `foo_k` 都没法做尾递归优化。

此外，`rust` 在生命周期的指定上似乎过度依赖自动推导，缺乏足够强大的手动标注方式。最简单的例子是我们不能写 `let _: &'a _ = ...` 或者类似 `decl_lifetime<a>` 的东西，而只能以来模板参数推导。个人认为 `rust` 起码要把 `lifetime` 做到 `type` 同等地位才能合理的解决许多问题。