程序结构

```
src
├── main.rs              // 主函数模式控制
├── game.rs              // 游戏相关函数
├── arg.rs               // 命令行参数解析
├── stats.rs             // 游戏状态存储
├── tui_mode.rs          // TUI
└── builtin_words.rs     // 词典
```

程序说明

- 在主函数中判断模式：
  - `TUI`
  - 测试模式
  - 交互模式
- `game.rs`中，结构体`Game`存储本局游戏状态，主要功能如下
  - 根据猜测的单词更新字母表状态，并获取当前猜测单词颜色状态
  - 打印单词和字母表
  - 对答案（`-w`模式下）和猜测判断是否合法，在困难模式下判断是否严格使用提示
  - 获取伪随机单词
  - 判断游戏是否结束
  - （附加功能）筛选符合条件的剩余单词列表
  - （附加功能）给单词评分并返回分最高的5个候选词
  - （附加功能）全局下计算尝试次数
- `args.rs`中，命令行读取游戏初始状态，遇到冲突触发错误
- `stats.rs`中，存储和多局游戏相关的信息
  - 其中结构体`Stats`中的`Vec` `games`存储所有单局游戏的答案和猜测，以便写入`json`文件
- （附加功能）`tui_mode.rs`中主要功能为绘制界面和处理按键事件，可以完成交互模式中的基本所有功能，包括但不限于
  - 进入随机模式并指定day和seed
  - 进入指定答案模式，包括命令行指定答案和每局开局输入答案两种模式
  - 开启困难模式
  - 输出统计数据
  - 指定候选词和可用词列表
  - 读取json文件并将游戏状态存入json
  - 读取config文件中的配置

游戏主要功能说明和展示

参数	可选参数	功能	备注
-w/--word	单词，如crane	指定答案模式	参数为空默认为指定答案模式
-r/--random	/	随机模式	随机模式和指定参数模式只能二选一
-D/--difficult	/	困难模式	必须使用上一步状态给出的提示
-t/--stats	/	统计输出游戏信息	/
-d/--day	局数，如5	指定开始时局数	默认值为1，大小不能超过答案词库的大小；只能在随机模式下使用
-s/--seed	种子，如101	决定答案词库打乱顺序	默认值固定；只能再随机模式下使用
-f/--final-set	候选词库文件名	使用指定词库作为答案词库	默认取内置候选词库
-a/--acceptable-set	可选词库文件名	使用指定词库作为可用词库	默认取内置可用词库
-S/--state	保存有游戏状态的json文件名	保存和加载随机模式的游戏状态	不合法错误异常退出
-c/--config	配置文件文件名	指定启动配置	命令行参数优先级高于配置文件，可能会发生覆盖

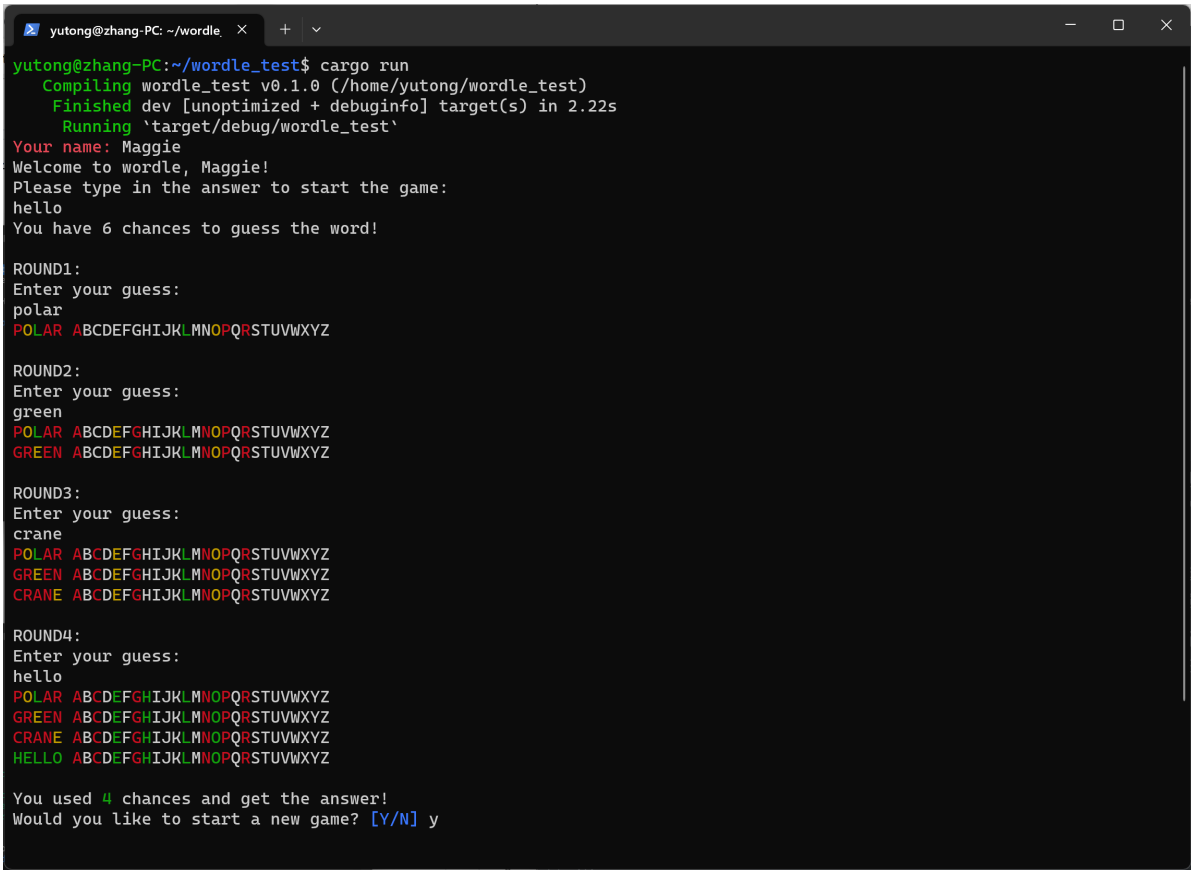
以下为提高功能部分

参数	功能	备注
--tui	开启TUI模式	TUI模式下仍可指定参数进行初始化，TUI界面中也可输入并指定答案
--hint	启用提示模式	将在每局输入猜测后筛选出符合状态的候选词，并输出
在指定提示模式下，将会对用户进行询问，输入y/n，即是否需要提示	根据算法，选出得分最高的5个词作为推荐词	只选出得分最高的至多5个词，只在提示模式下存在
--test	根据算法计算平均尝试次数	/

游戏模式展示

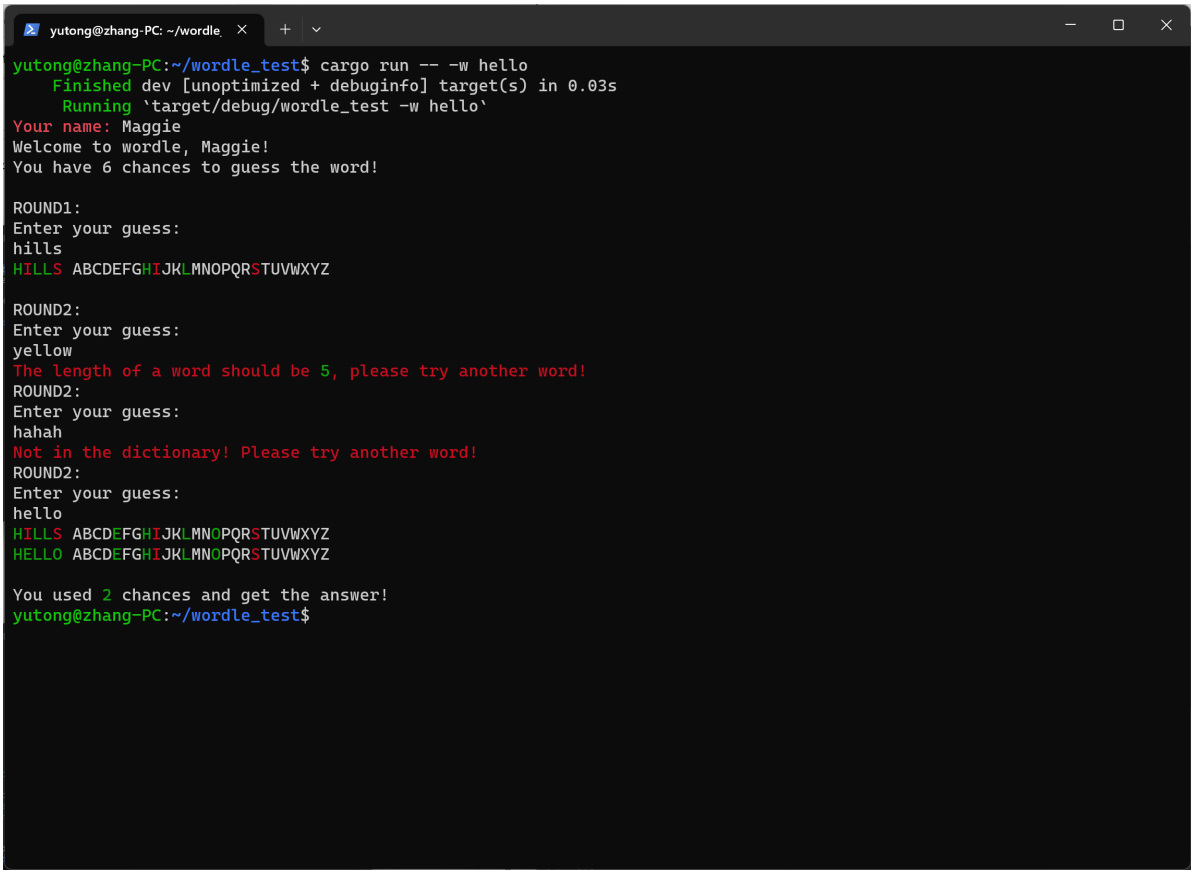
基础功能

`cargo run` 进入指定答案模式，将在每局开头对用户进行答案询问，并在游戏结束询问是否开启下一局游戏



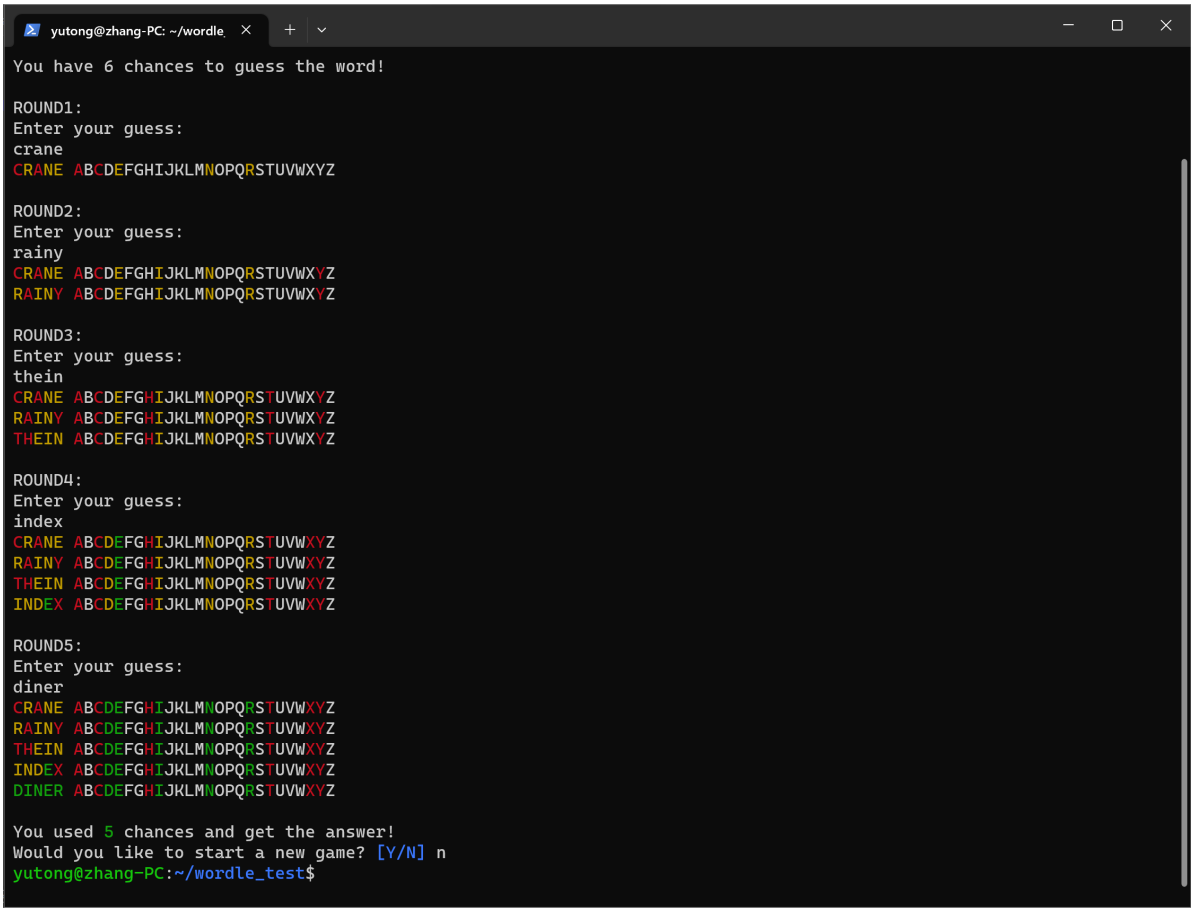
`cargo run -- -w hello` 进入指定答案模式，且指定答案为hello

输入猜测单词后会对单词长度和是否在可用词列表中进行检查，不合法将输入报错，并提示重新输入

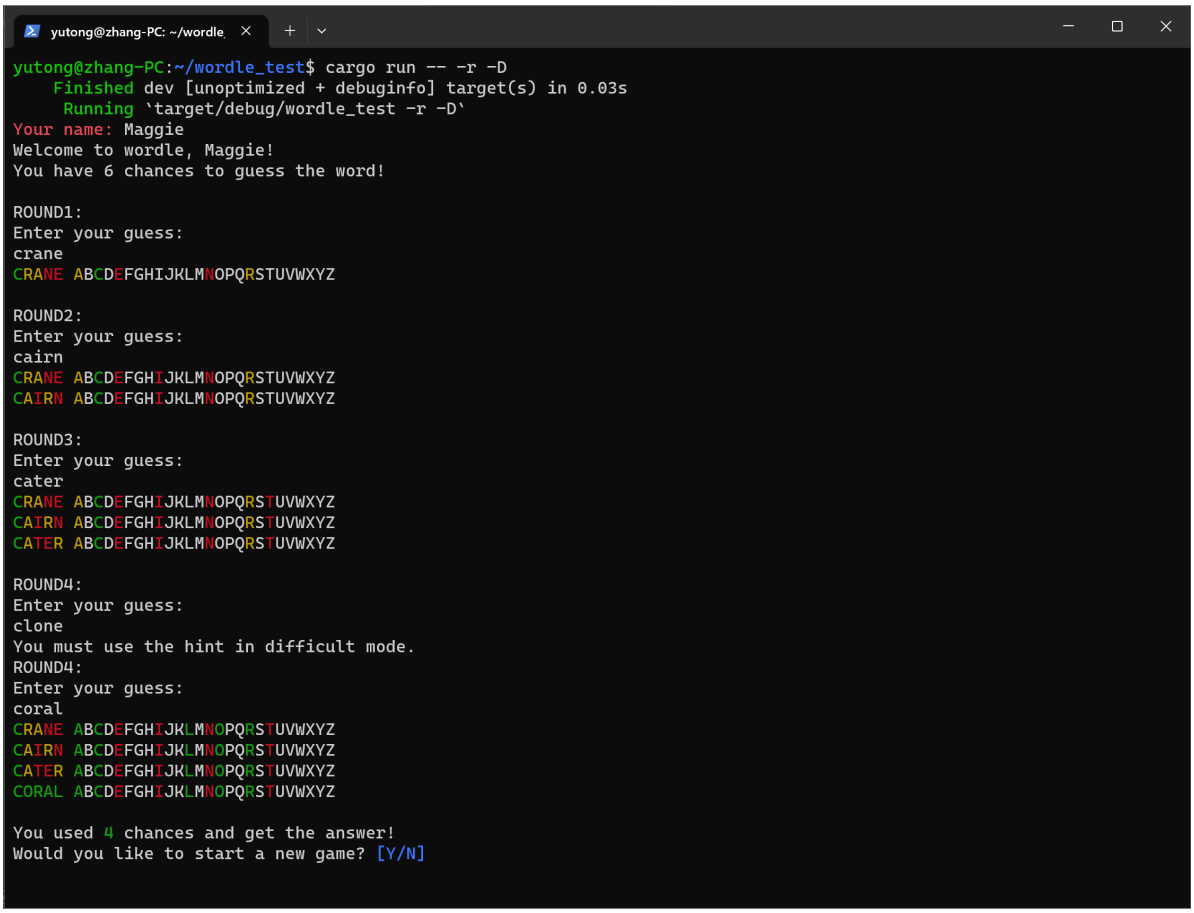


`cargo run -- -r -s 100 -d 10` 进入随机模式

输入n退出游戏，y继续游戏



`cargo run -- -r -D` 开启困难模式，每一步必须利用上一步提示



冲突实例

比如在指定答案模式下不能使用种子和天数参数

```
yutong@zhang-PC:~/wordle_test$ cargo run -- -w -s -d
    Finished dev [unoptimized + debuginfo] target(s) in 0.06s
     Running `target/debug/wordle_test -w -s -d`
thread 'main' panicked at 'Please use -d/--day or -s/--seed options in random mode!', src/args.rs:187:13
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
```

随机模式和指定答案模式不能同时启用

```
yutong@zhang-PC:~/wordle_test$ cargo run -- -r -w
    Finished dev [unoptimized + debuginfo] target(s) in 0.03s
     Running `target/debug/wordle_test -r -w`
thread 'main' panicked at 'The -w/--word option is not allowed in random mode!', src/args.rs:183:13
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
```

天数指定过大或小于等于0

```
yutong@zhang-PC:~/wordle_test$ cargo run -- -r -d 10000000
    Finished dev [unoptimized + debuginfo] target(s) in 0.03s
     Running `target/debug/wordle_test -r -d 10000000`
thread 'main' panicked at 'Invalid value for -d/--day option!', src/args.rs:176:25
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
```

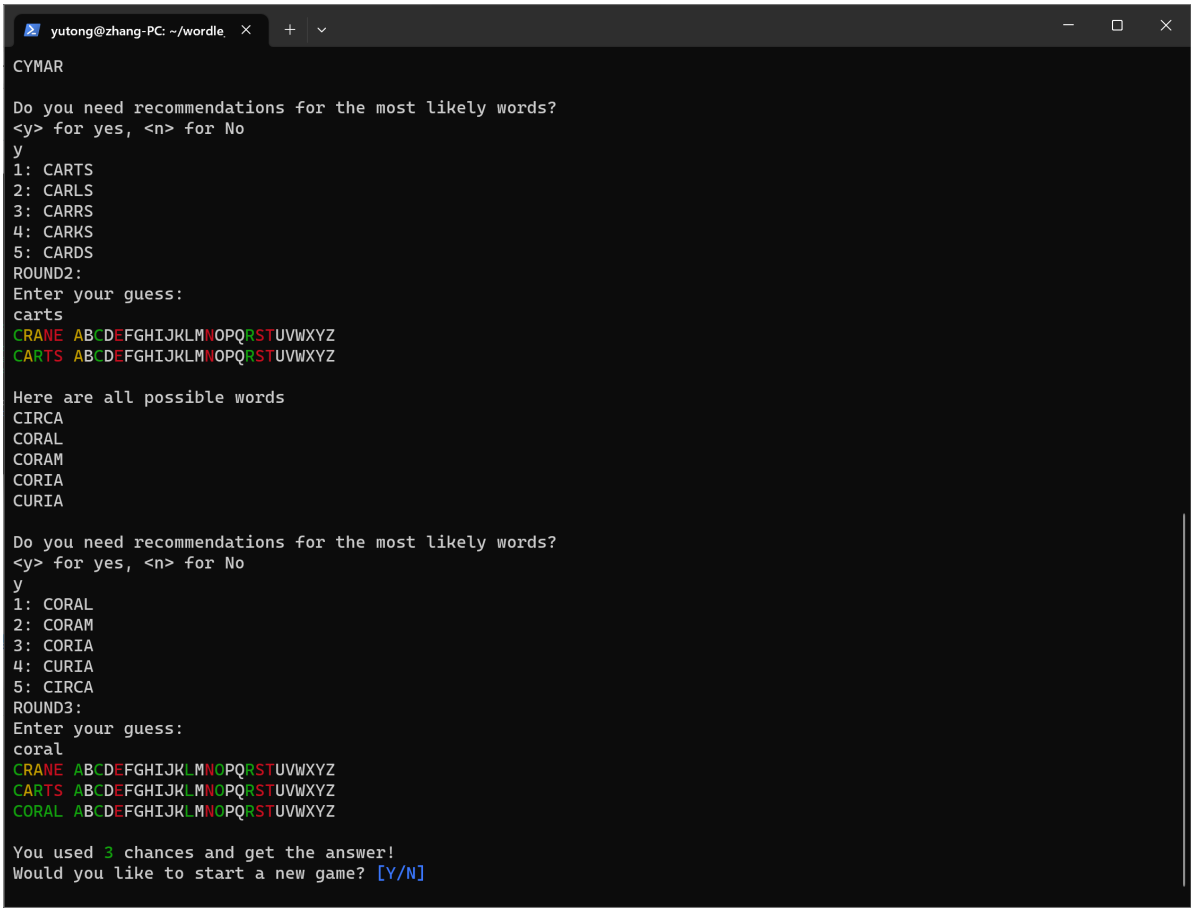
```
yutong@zhang-PC:~/wordle_test$ cargo run -- --r -d -1
    Finished dev [unoptimized + debuginfo] target(s) in 0.24s
     Running `target/debug/wordle_test --r -d -1`
thread 'main' panicked at 'Invalid value for -d/--day option!', src/args
.rs:174:21
note: run with `RUST_BACKTRACE=1` environment variable to display a back
trace
```

等……

提高功能——求解

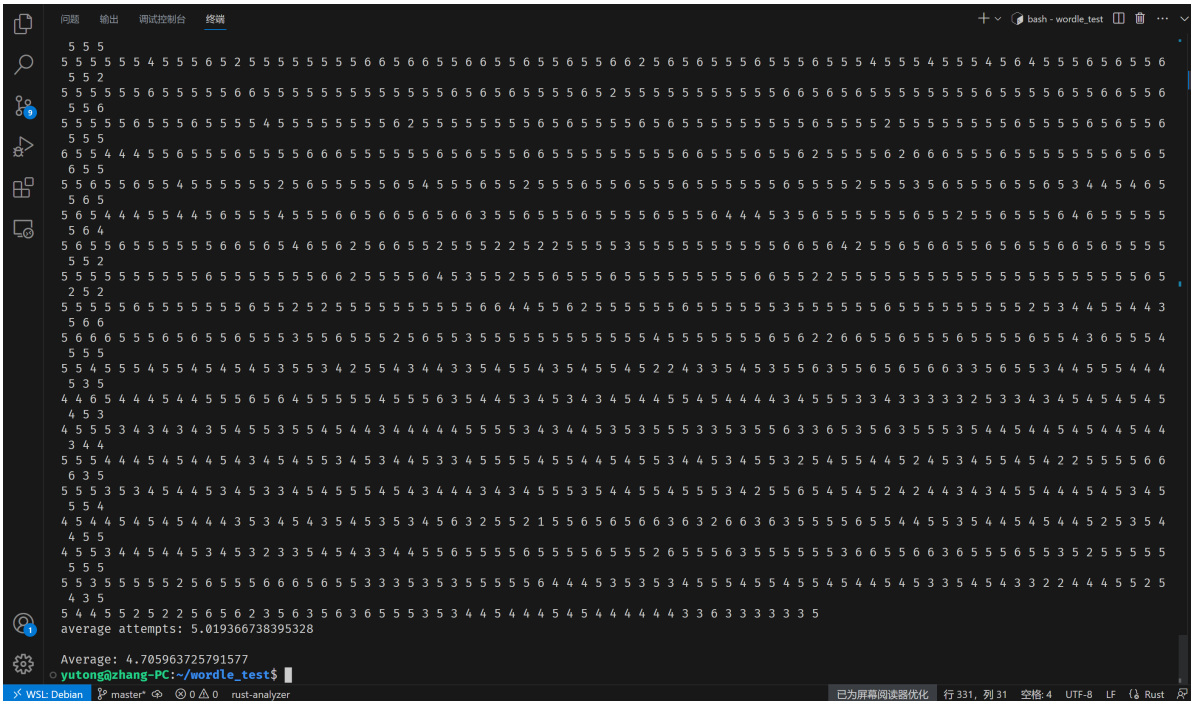
`cargo run --  -r --hint`

每局猜测后将输出筛选后的候选词，并询问是否需要提示



`cargo run -- --test`

遍历候选词列表和可用词列表，计算平均尝试次数，由于计算量过大，在使用并行计算后也未显著加快计算效率，因此使用随机选择的规模为官方候选词库一半大小的词库和包含答案词库前提下随机选择的规模为官方可用词库一半大小的可用词进行计算，进行多次随机选择，结果如下：



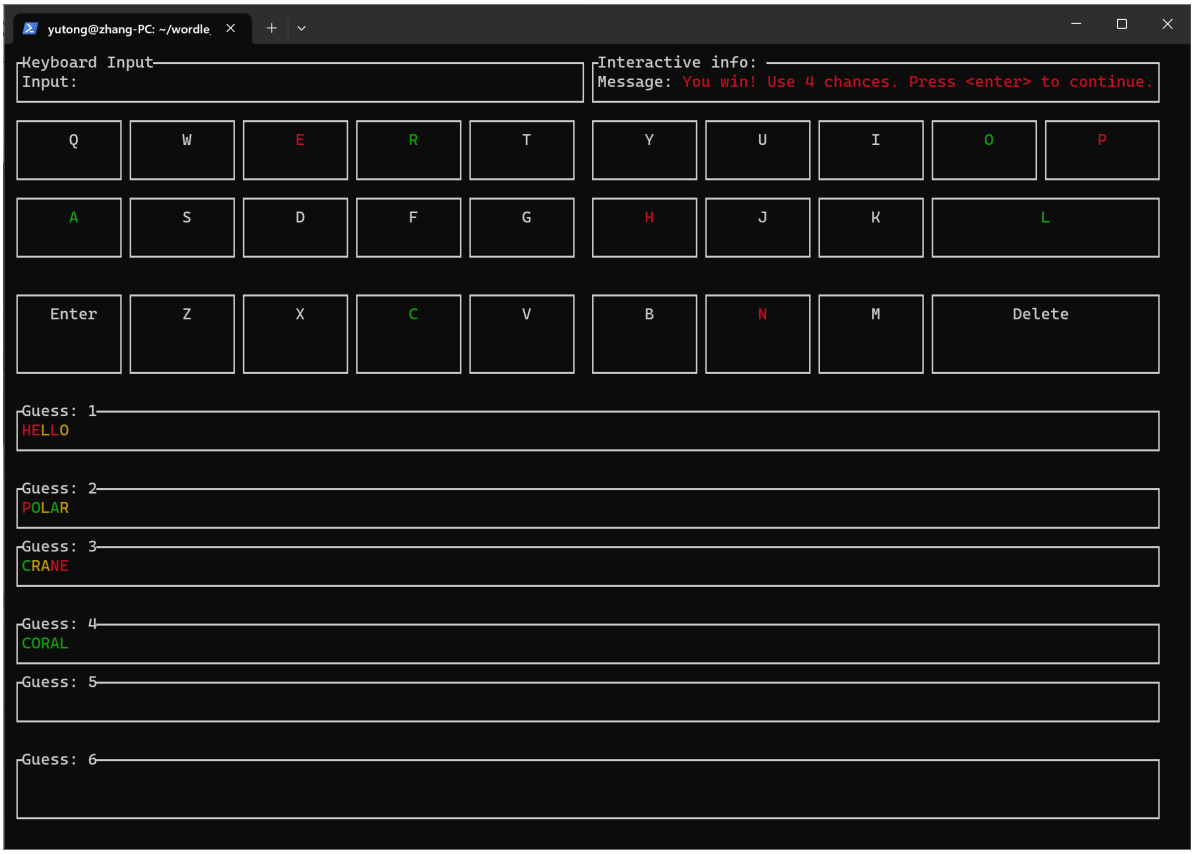
可见此算法还是有一定准确性的，只是计算复杂度过大，有待优化

提高功能——TUI

TUI模式下通过键盘输入进行交互，交互模式的信息在TUI模式下可以展示

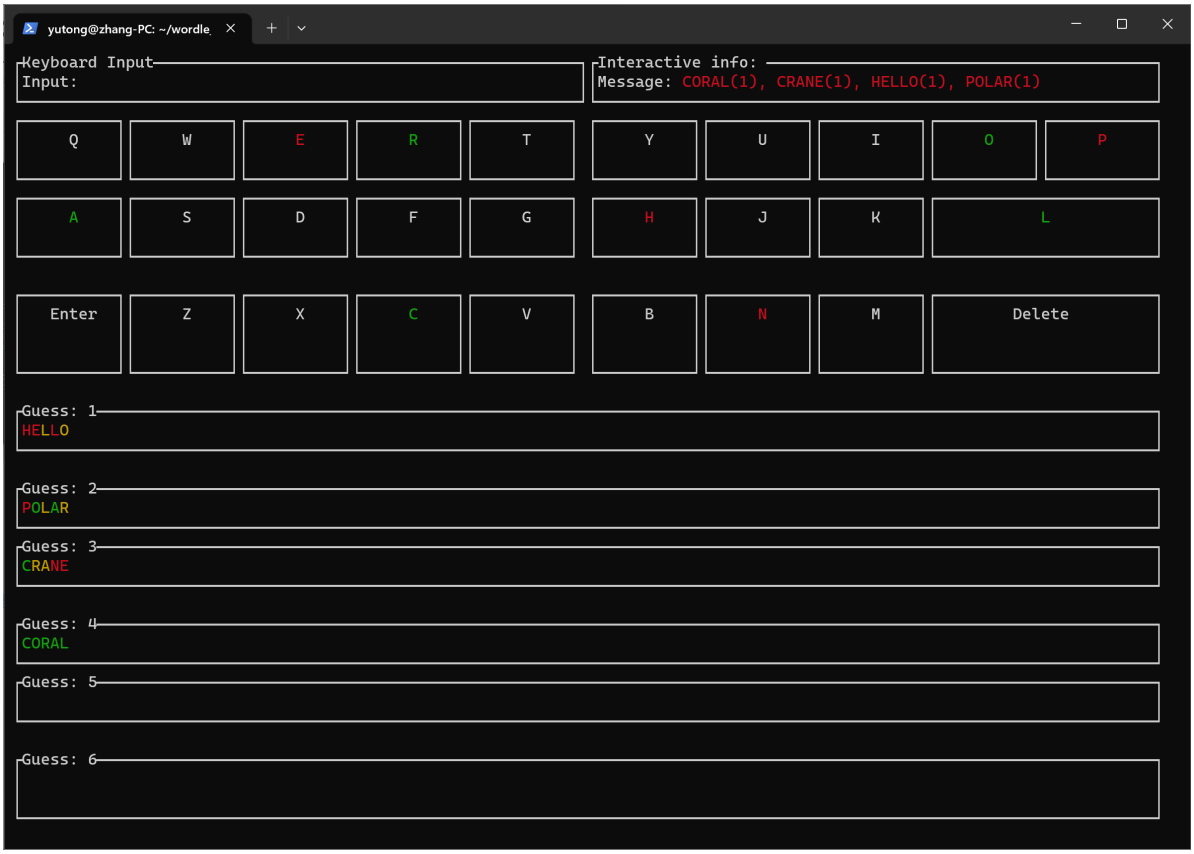
天数、种子、候选词列表、可用词列表、游戏状态保存路径、游戏配置文件路径等参数均可正常选用

`cargo run -- -r -d 1 -s 101 --tui` 进入随机模式：

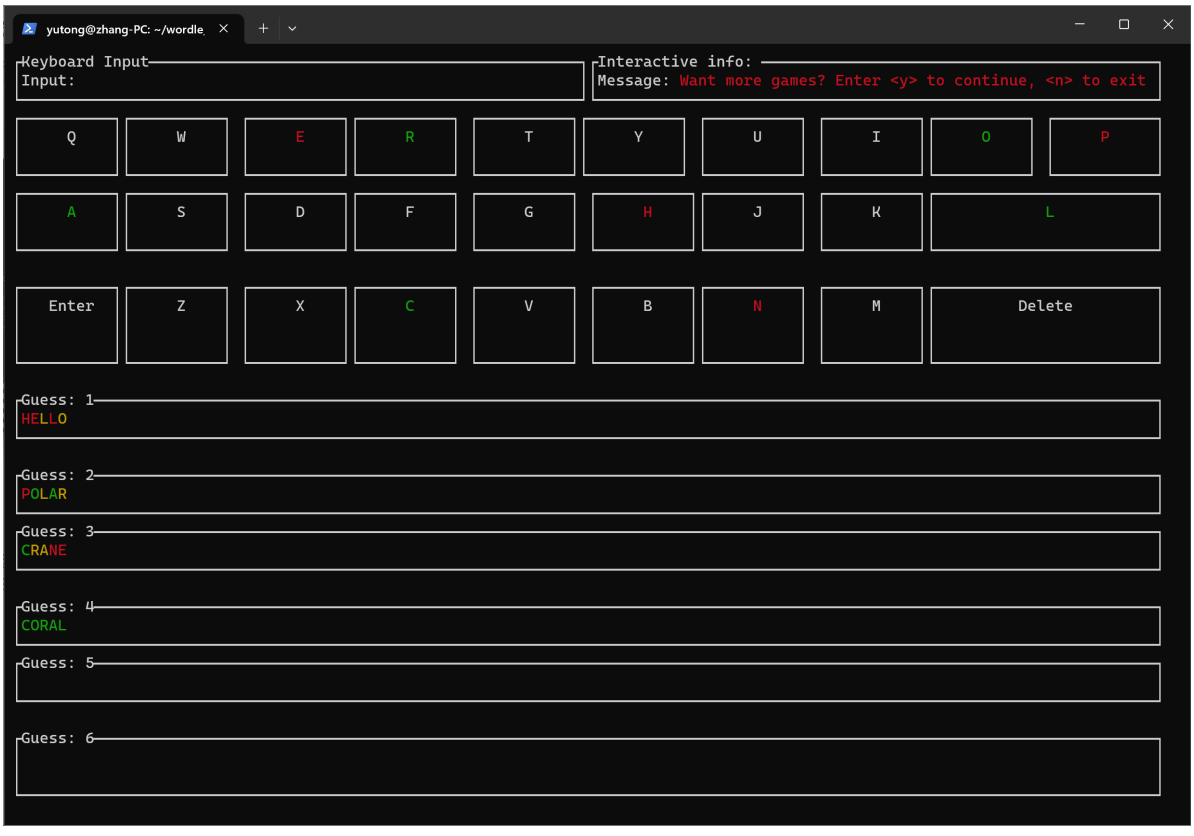

yutong@zhang-PC: ~/wordle
Keyboard Input
Input:
Interactive info:
Message: You win! Use 4 chances. Press <enter> to continue.
Q W E R T Y U I O P
A S D F G H J K L
Enter Z X C V B N M Delete
Guess: 1
HELLO
Guess: 2
POLAR
Guess: 3
CRANE
Guess: 4
CORAL
Guess: 5
Guess: 6

可逐次展示统计信息：


yutong@zhang-PC: ~/wordle
Keyboard Input
Input:
Interactive info:
Message: Wins: 1; Fails: 0, press <enter> to continue...
Q W E R T Y U I O P
A S D F G H J K L
Enter Z X C V B N M Delete
Guess: 1
HELLO
Guess: 2
POLAR
Guess: 3
CRANE
Guess: 4
CORAL
Guess: 5
Guess: 6



并询问是否需要开启下一轮游戏：



`cargo run -- --tui -D`

开启困难模式，在下一步操作不符合上一步状态提示时与交互模式相同，会显示警告：


```
cargo run -- --w --tui
```

进入指定答案模式，输入自定义答案开始游戏：

输入后自定义答案及时消失，开始游戏猜测：

并在末尾询问是否开启下一轮游戏，与命令行交互模式的游戏逻辑一致。

游戏结束可正常退出程序：

```
yutong@zhang-PC:~/wordle_test$ cargo run -- -w --tui
    Finished dev [unoptimized + debuginfo] target(s) in 0.03s
     Running `target/debug/wordle_test -w --tui`
yutong@zhang-PC:~/wordle_test$
```

提高要求的实现方式

求解器

在求解器部分，对候选词筛选的思路如下：

- 一轮筛选判断位置：判断单词中绿色字母位置不变，单词中包含黄色字母且不在原位置，单词中红色字母不在原位置，并给红色字母标记状态
- 二轮筛选判断数量：遍历字母的5个位置统计各个字母出现的次数，如果某字母被标记了状态，说明被筛选出的单词的该字母的数量只能严格等于已有状态中该字母的黄色+绿色字母的数量；如果未被标记状态，说明被筛选出的单词的该字母的数量需要大于等于已有状态中该字母黄色+绿色字母的数量，例如最后猜测单词为abcxx，给出状态为gyrgr，说明被筛选出的单词必须满足a(?)(!c)x(!x)的形式，且被筛选出的单词中，b至少出现一次，c不能出现，x只能出现1次

对单词评分并给出推荐词的思路如下

- 受二分法启发，分别计算筛选出来的所有候选词0、1、2、3、4号位置的A-Z字母数量总数，给总数进行归一化得到字母权重，得到0-1之间的值
- 对每个单词将5个值相加得到分数，分数在0-5之间
- 使用rayon库中的`par_iter`和`par_sort`，对分数进行排序并选出分最高的5个

计算平均尝试次数的思路如下

- 遍历答案词库和候选词库，后续每局选用推荐词中的第一个
- 当推荐词的第一个和答案相等时返回尝试次数
- 该算法大致稳定在4-5局得到正确答案

TUI

开始选用了cursive包，尝试了一天半后，却发现cursive库的按钮事件非常复杂，且颜色设置遇到了困难，故换用TUI包重新进行尝试，在刚入手的时候进度非常缓慢，遇到了很多未知的bug，比如出了缓冲区文本框会消失等等奇怪的问题。并且网上没有成型的教程，只能参照官方文档和开源代码缓慢学习。

在完成TUI模式时，尽量复用了已有代码，通过参数传入到TUI逻辑中，虽然成品比较简易，但还是很好用的，功能也比较完善！

完成此作业的感想

总体来说，完成本次作业的时间还是非常紧张的，但在这一周左右的时间中我也学到了很多东西。在最开始写代码的时候，由于对Rust语言的不熟悉，简单的判断单词状态都卡了我很久。逐渐对Rust语言熟悉起来，大作业的进度也推进了很多。我使用了Rust的一些库，比如`atty`、`rand`、`serde_json`、`console`、`colored`、`tui`、`crossterm`、`rayon`等等，但在使用这些包的时候我也遇到了很多问题，我不断参阅文档、参考GitHub上的开源代码和访问StackOverflow等社区，我逐渐自己解决了这些问题。在完成作业的过程中，我不少出现编译器报错很多的情况，也因此我认识到了Rust语言的安全性，Rust编译器的强大使得很多bug在编译期就可以被检查出来，比运行再遇到数百行的报错要效率高很多。

我第一次使用了Rust的测试框架进行单元测试和集成测试，我深感这会对开发检查提供很大的便利，而且Cargo作为包管理工具也让开发方便了很多。

在完成提高功能时，我也遇到了很多问题，比如开始设计的算法复杂度过高或者错误、TUI不断摸索成型……

Rust语言对于我来说是一种全新的语言，也是一种完全陌生的体验。通过完成这次作业，我加深了对Rust语言性能和安全性的认识，此外，通过完成这次作业，我不断的修改我已经写过的接口、存入新东西等等，不断的对已有的代码进行修正和改进，才能完成后面的功能和提高功能。期待在接下来的Rust语言学习中学习更多的新东西，也希望我可以继续提高我的代码能力！