Wordle 大作业报告

张子颖 2021010734

一、程序结构和说明

整个项目分为三个部分：基础功能部分（包括提高要求中 Wordle 求解部分的第一、二两个小点，位于根目录下的 src 文件夹中），UI 模式（对应提高要求中 UI 部分的 tui，位于文件夹 ui 中）和 WordleSolver 功能（对应提高要求中 Wordle 求解部分的第三、四两个小点）。

1、基础功能部分

基础功能部分采用面向过程的编程理念，按照游戏进行的顺序展开。

首先是处理命令行参数和 config 配置文件。为了使这两者统一便于读者后续处理，创建 bool 等类型变量以获得最终参数是否存在及对应的 value。

然后是获取答案词库、排序和判断词库是否合法。为了便于后续统一处理，无论用户是选择内置词库还是自定义词库，都会读入将其名为 `answer_words` 和 `acceptable_words` 的向量。

之后是获取游戏状态。这里新建并初始化一个名为 `GameState` 的结构体来从指定 json 文件中获取游戏状态（没有指定只初始化），在后续游戏中持续更新 `GameState` 中的相关值。

最后是通过循环处理游戏逻辑。在每局游戏中，首先获取该局的答案词（从词库中随机产生或等待用户输入），然后进行猜词的循环。在每次循环中，首先获取用户猜词并判断是否有效，再进行与答案词的比对，然后进行数据处理（更新键盘状态、更新 `GameState`、写入 state 文件等）并判断进入新一局或者退出游戏还是继续猜词。

2、UI 部分

UI 模式整体流程与基础功能部分类似，差异在于其处理游戏逻辑部分不是采用循环进行，而是一种递归等方式，即通过 `show_ui` 函数根据当前游戏状态（用 `GameState` 变量存储）来展示界面，当用户输入猜测词并按下“Ok”button 时更新 `GameState`，然后 `pop` 出当前页面并递归调用 show_ui 函数根据新的 `GameState` 展示新页面。

3、WordleSolver 部分

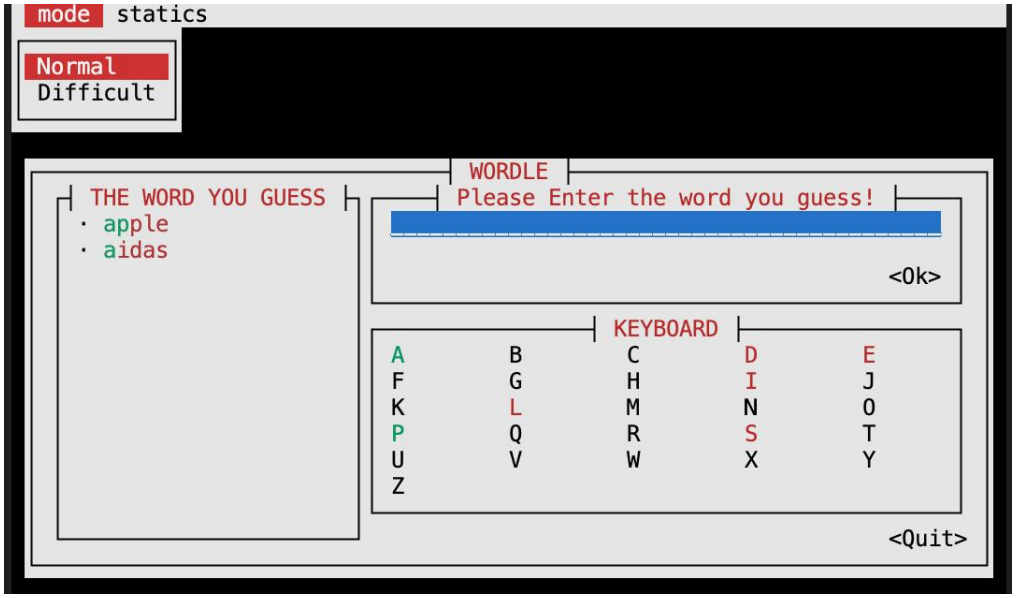
WordleSolver 部分在每次猜词前根据当前游戏状态（主要是此前历次猜测结果）从 acceptable 词库中筛选出可能的答案词，然后计算 acceptable 词库中所有单词的信息熵并按照降序排序从而给出推荐词，当 acceptable 词库中只剩一个单词时则确定为答案。

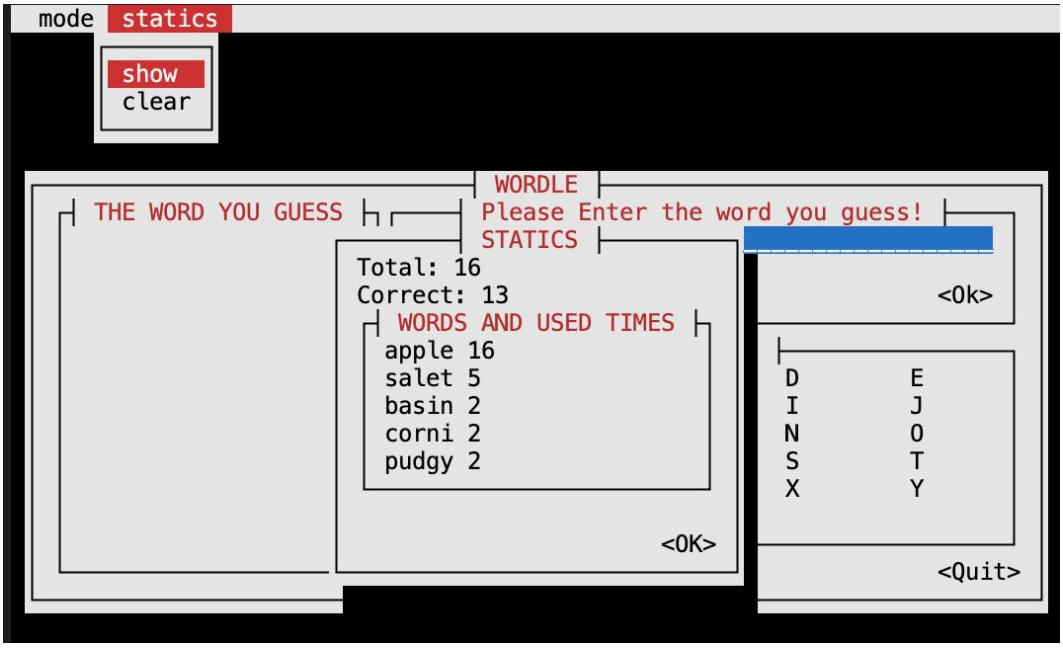
二、游戏主要功能

基础功能部分完成了大作业文档中提出的所有要求，新增了提示功能（加入命令行参数-e 触发），即在每次猜测后可以根据历次猜测结果给出可能是答案的词和推荐词。如图：

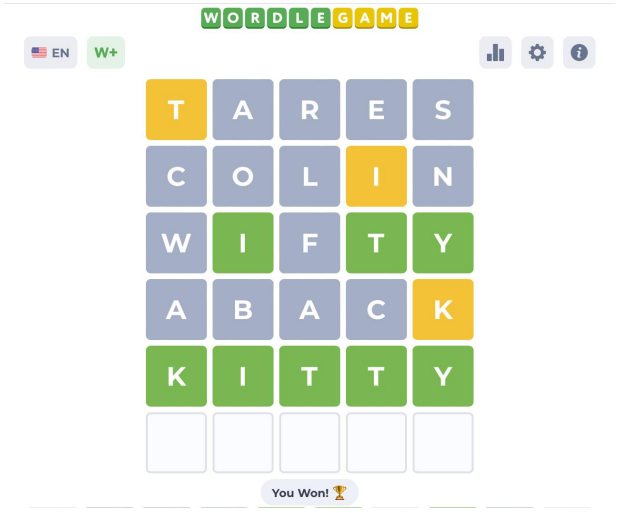
当命令行参数为`-w white -e`（指定答案为 white，指定提示功能）时，每次输入猜测都会给出可能的答案词和信息熵最高的五个推荐词。

UI 模式在功能上基本还原了原版 wordle 游戏，但由于使用了 tui，界面美观性较差。用户可以选择困难模式和普通模式，可以查看统计和清空统计数据（统计数据包括此前所有数据而不是此次启动之后的）。如图：





WordleSolver 部分实现了交互式的 Wordle 求解，用户输入自己在 Wordle 游戏中猜测的单词和游戏的反馈结果，会给出五个推荐词。当答案确定时会直接给出答案词。如图：




```
What do you want to do? S(olver) B(est) A(verage)
S
Start? [Y/N]
Y
Please input the word and its state.
tares YRRRR
colin  5.2850766952505825
count  5.210015788493705
cunit  5.181704275666195
nicol  5.156030544569513
litho  5.121019306489769
Please input the word and its state.
colin RRRYR
wifty  3.782122224145307
withy  3.582610300148535
mifty  3.559900001923084
witty  3.5546515037721105
ditty  3.5266927073956853
Please input the word and its state.
wifty RGRGG
aback  1.584962500721156
abaka  1.584962500721156
aband  1.584962500721156
abask  1.584962500721156
abbed  1.584962500721156
Please input the word and its state.
aback RRRRY
Answer : kitty
Start? [Y/N]
```

也可以依照同样的原理计算最佳开局词和完成一局 Wordle 游戏所需的平均局数。

```
/*
tares  6.1940525443754435
lares  6.149918742453124
rales  6.114343099454227
rates  6.096242642514605
teras  6.076619177276175
*/
```

```
开局词固定为本算法计算出的初始信息熵最大的词 tares
计算出的平均猜测次数为4.149892008639309
```

三、提高要求的实现方式

1、UI 部分

UI 部分基于 Rust 语言的 Cursive 库实现, Cursive 库自带 `EditView`(输入框), `ListView`（列表）等控件，可以在 `LinearLayout`（线性布局）下选择 `Vertical`（垂直）和 `Horizontal`（水平）。绘制界面时，首先对整个界面使用 `LinearLayout::horizontal()`将其分割为左侧的使用过的单词列表 (`ListView`) 和右侧区域，右侧区域再使用 `LinearLayout::vertical()` 分割为输入框 (`EditView`)、键盘区和底部退出按钮。带颜色的文字使用 `Cursive` 库中的 `StyledString` 完成。

游戏开始前首先使用 `set_user_data()` 函数向 `Cursive` 变量传入 `GameState`, 之后使用 `show_ui()` 函数绘制界面并依据 `GameState` 为猜词列表和键盘区的字母着色。当用户点击 Ok 按钮时，则处理游戏逻辑并更新 `GameState`，然后剔除当前页面并递归调用 `show_ui()`，根据最新的 `GameState` 绘制新的页面和着色。

2、WordleSolver 部分

WordleSolver 的核心部分是计算整个 acceptable 词库中各单词的信息熵并按照降序排序，这一部分由 `cie()` 函数完成:

对 acceptable 词库中的每一个单词$word_1$, 遍历所有可能为答案的单词$word_2$。假定$word_2$为答案，$word_1$为猜测词，得到由五个颜色组成的状态（state)，遍历所有$word_2$即可得出对于$word_1$各类状态出现的概率，由公式

$$S(word_1) = -\sum_s p(state)\log_2 p(state)$$

可以计算出$word_1$的信息熵。计算出所有$word_1$的信息熵后，按照信息熵降序进行排列，即可得到信息上较大的几个推荐词。

令$word_2$为 acceptable 词库中所有的词, 即可给出该算法下的最优开局词 tares。

计算 Wordle 平均次数时，令开局词为“tares”，以后历次猜测均为信息熵最大的单词，当可能为答案的单词只剩一个时，则将其作为答案词。计算出平均猜测次数后，我与其他同学进行了比较，发现我的平均猜测次数较大，可能有以下原因:

就像 3Brown1Blue 的视频里说的，我在计算信息熵时，只关注了接下来的一

步，如果多关注几步可能会获得更优的推荐词；

虽然我们不确定答案词库中具体有哪些单词，但一般来说应该是常用词。如果能根据各单词在日常使用中的词频，按照公式$f(x) = (1 + e^{-x})^{-1}$进行加权，可能能得到更优的推荐词；

当若干个单词同时具有最大的信息熵时（比如可能为答案的单词剩两个，则有包括这两个单词在内的多个单词信息熵为 1），我没有将可能为答案的单词设为优先。如果将其设为优先，有概率在更少的回合数内获得正确答案。

遗憾的是，碍于时间不足，没有来得及根据以上原因对算法进行改进。

四、完成作业感想

1、学会阅读官方文档和 Github 上的示例代码

Rust 作为一门年轻的语言，在国内的普及程度与之前接触到比较主流的 C/C++、Python、Java 等相差较大，对于一些问题和库的使用很难找到中文的教程或解答。这迫使我去阅读以前很不愿意接触的官方英文文档和示例代码。但通过阅读这些内容，我发现这些示例往往编写的非常清晰，能让我快速上手第三方库的使用，比如 Github 上几个有关 Cursive 库控件和 StyledString 的示例。

2、代码风格仍有待改善

这次大作业我采用了分文件编写和函数式编程的方式，但依然出现了重复代码等问题，某些地方的逻辑也不太清晰。整体代码风格仍然体现出“幼稚”的特点，尤其是在基础功能部分。这也使得我很难将提高功能部分与原有的基础功能部分合在一起。在 UI 部分，我尝试对代码风格进行了一些改变，使其逻辑上更清晰一些，似乎有些成果？但整体上，代码风格仍有待改善。