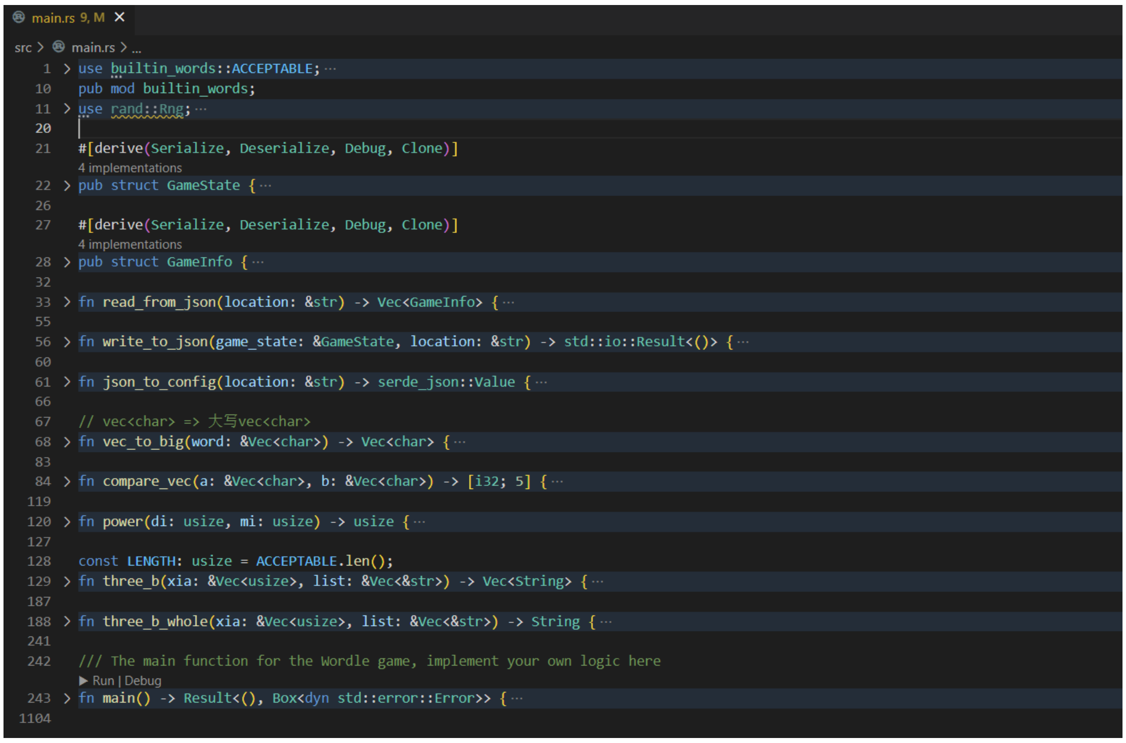
Wordle 大作业报告

计 15 宋驰 2021010797

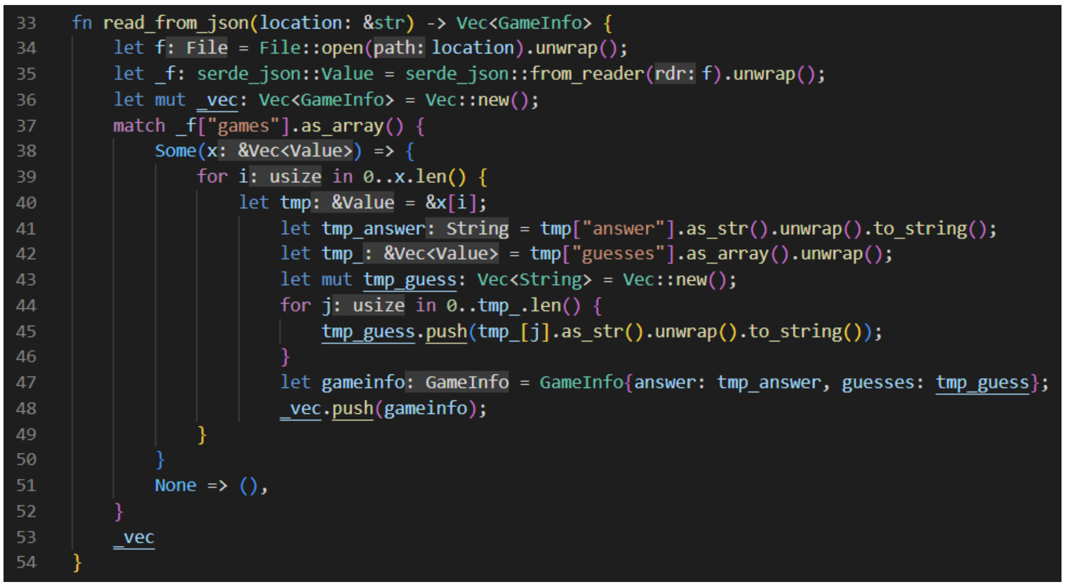
一、　简单的程序结构和说明

整体框架如下图：

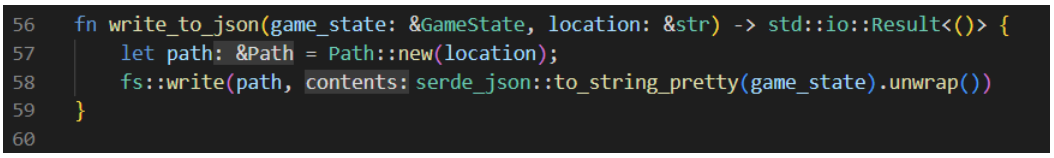


```rust
use builtin_words::ACCEPTABLE;···
pub mod builtin_words;
use rand::Rng;···

#[derive(Serialize, Deserialize, Debug, Clone)]
pub struct GameState {···

#[derive(Serialize, Deserialize, Debug, Clone)]
pub struct GameInfo {···

fn read_from_json(location: &str) -> Vec<GameInfo> {···

fn write_to_json(game_state: &GameState, location: &str) -> std::io::Result<()> {···

fn json_to_config(location: &str) -> serde_json::Value {···

// vec<char> => 大写vec<char>
fn vec_to_big(word: &Vec<char>) -> Vec<char> {···

fn compare_vec(a: &Vec<char>, b: &Vec<char>) -> [i32; 5] {···

fn power(di: usize, mi: usize) -> usize {···

const LENGTH: usize = ACCEPTABLE.len();
fn three_b(xia: &Vec<usize>, list: &Vec<&str>) -> Vec<String> {···

fn three_b_whole(xia: &Vec<usize>, list: &Vec<&str>) -> String {···

/// The main function for the Wordle game, implement your own logic here
fn main() -> Result<(), Box<dyn std::error::Error>> {···
```

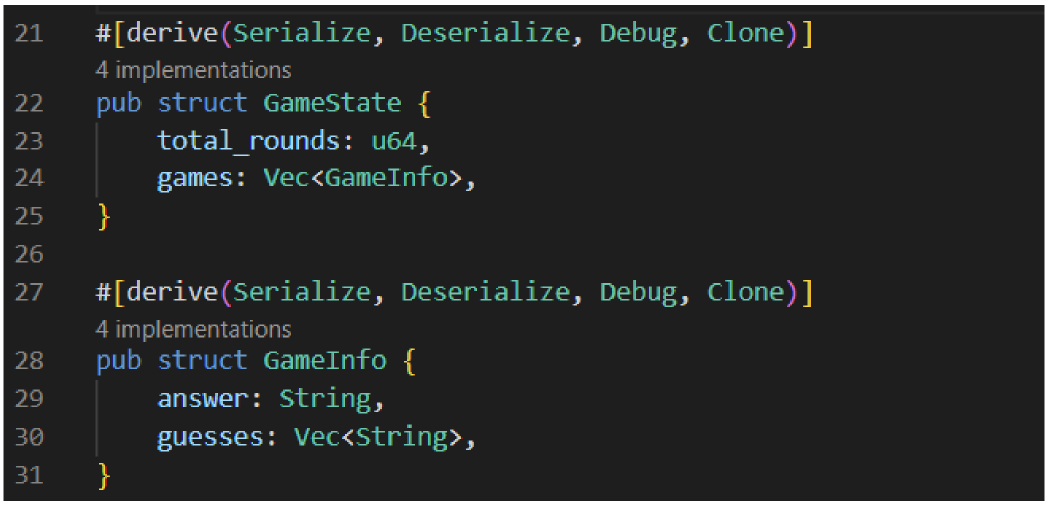
在主函数 main 之外，read_from_json 函数用来读取 json 文件的信息并加载之前的游戏状态，write_to_json 函数用来将每局游戏结束后的游戏状态写入 json 文件，同时定义了两个结构体 GameState 和 GameInfo 用来暂时存储游戏状态。



```rust
fn read_from_json(location: &str) -> Vec<GameInfo> {
    let f: File = File::open(path: location).unwrap();
    let _f: serde_json::Value = serde_json::from_reader(rdr: f).unwrap();
    let mut _vec: Vec<GameInfo> = Vec::new();
    match _f["games"].as_array() {
        Some(x: &Vec<Value>) => {
            for i: usize in 0..x.len() {
                let tmp: &Value = &x[i];
                    let tmp_answer: String = tmp["answer"].as_str().unwrap().to_string();
                    let tmp_: &Vec<Value> = tmp["guesses"].as_array().unwrap();
                    let mut tmp_guess: Vec<String> = Vec::new();
                    for j: usize in 0..tmp_.len() {
                        tmp_guess.push(tmp_[j].as_str().unwrap().to_string());
                    }
                    let gameinfo: GameInfo = GameInfo{answer: tmp_answer, guesses: tmp_guess};
                    _vec.push(gameinfo);
            }
        }
        None => (),
    }
    _vec
}
```

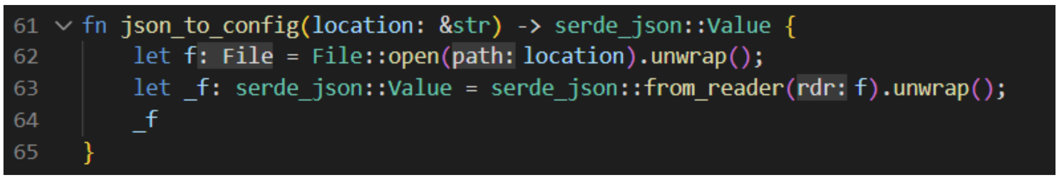


```rust
fn write_to_json(game_state: &GameState, location: &str) -> std::io::Result<()> {
    let path: &Path = Path::new(location);
    fs::write(path, contents: serde_json::to_string_pretty(game_state).unwrap())
}
```



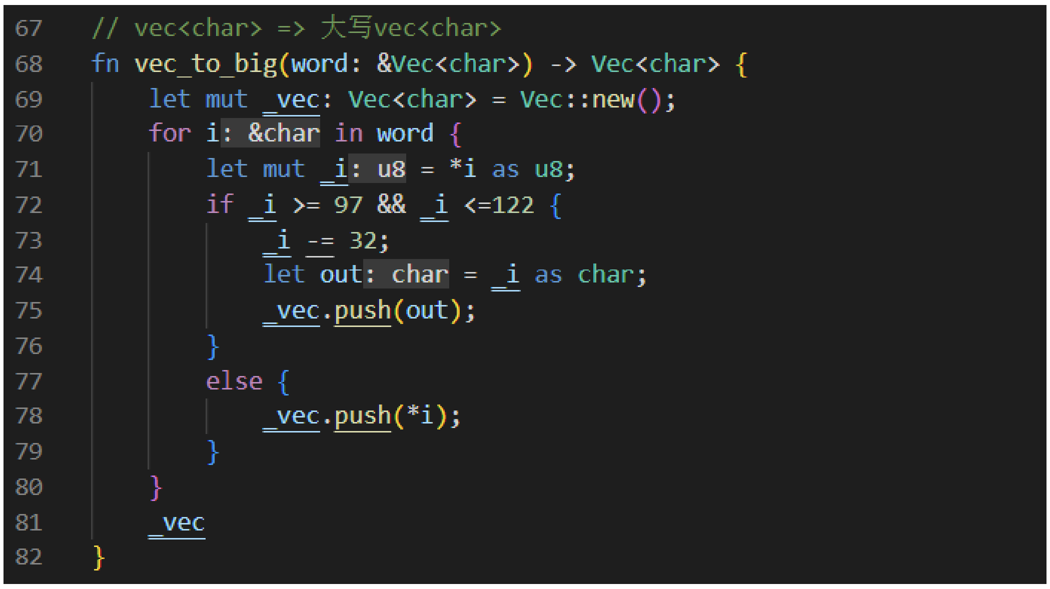
```rust
#[derive(Serialize, Deserialize, Debug, Clone)]
4 implementations
pub struct GameState {
    total_rounds: u64,
    games: Vec<GameInfo>,
}

#[derive(Serialize, Deserialize, Debug, Clone)]
4 implementations
pub struct GameInfo {
    answer: String,
    guesses: Vec<String>,
}
```

json_to_config 函数用于解析读取的 json 格式的 config 文件。



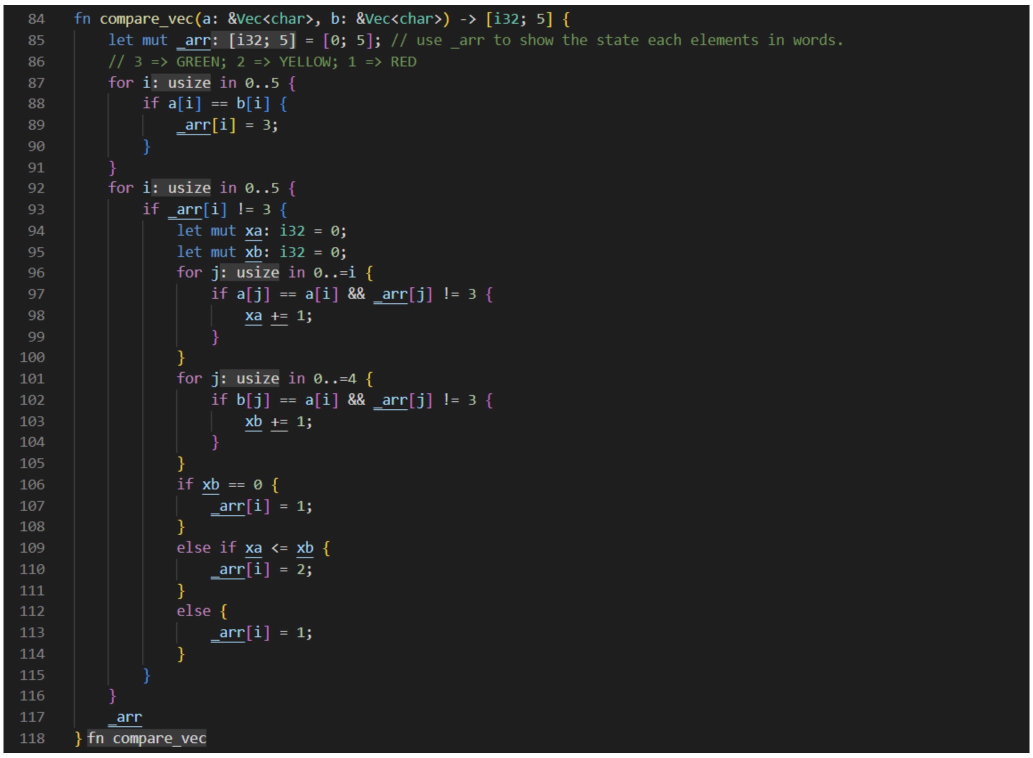
```rust
fn json_to_config(location: &str) -> serde_json::Value {
    let f: File = File::open(path: location).unwrap();
    let _f: serde_json::Value = serde_json::from_reader(rdr: f).unwrap();
    _f
}
```

vec_to_big 函数用来将 vec<char>中的小写字母转换为大写字母。



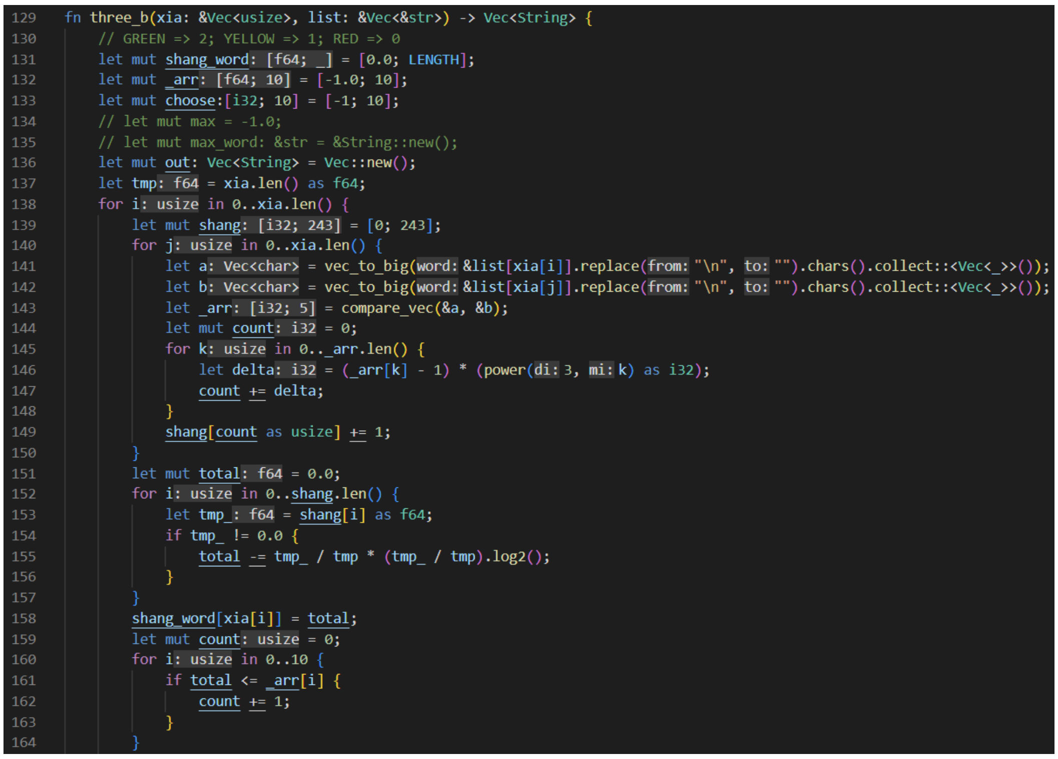
```rust
// vec<char> => 大写vec<char>
fn vec_to_big(word: &Vec<char>) -> Vec<char> {
    let mut _vec: Vec<char> = Vec::new();
    for i: &char in word {
        let mut _i: u8 = *i as u8;
        if _i >= 97 && _i <=122 {
            _i -= 32;
            let out: char = _i as char;
            _vec.push(out);
        }
        else {
            _vec.push(*i);
        }
    }
    _vec
}
```

compare_vec 函数实现了两个 vec<char>之间（猜测和答案）的比较，同时返回一个数组来记录颜色的状态。



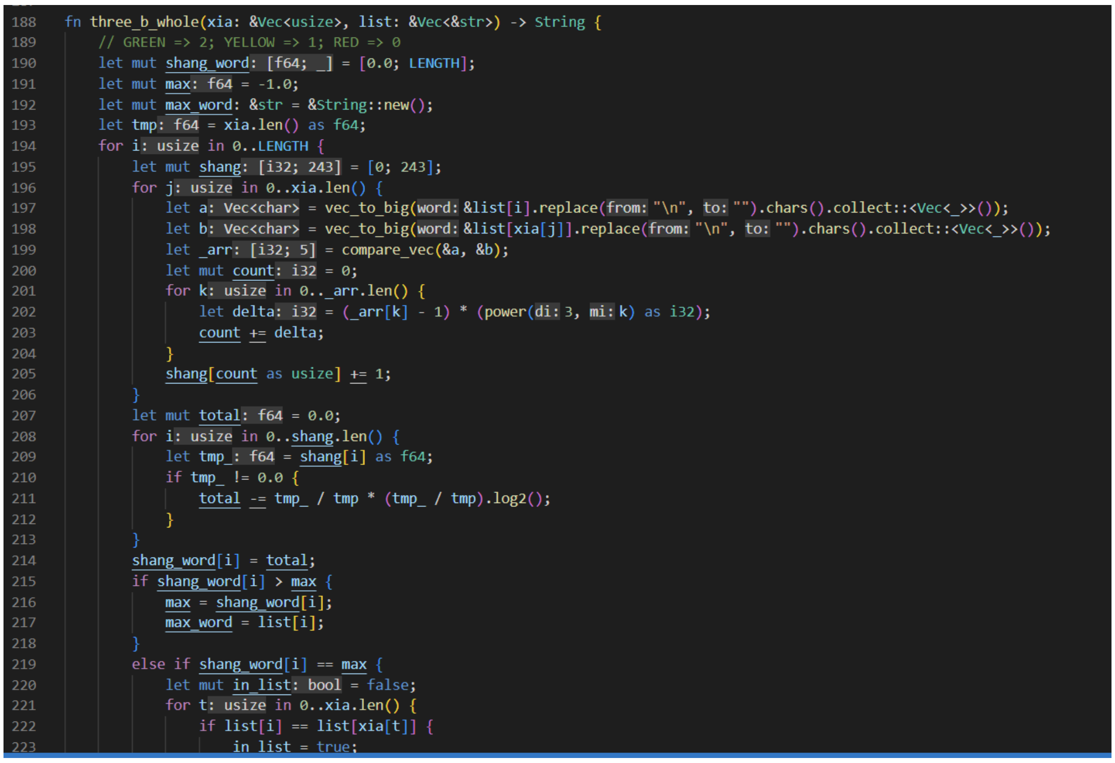
```rust
fn compare_vec(a: &Vec<char>, b: &Vec<char>) -> [i32; 5] {
    let mut _arr: [i32; 5] = [0; 5]; // use _arr to show the state each elements in words.
    // 3 => GREEN; 2 => YELLOW; 1 => RED
    for i: usize in 0..5 {
        if a[i] == b[i] {
            _arr[i] = 3;
        }
    }
    for i: usize in 0..5 {
        if _arr[i] != 3 {
            let mut xa: i32 = 0;
            let mut xb: i32 = 0;
            for j: usize in 0..=i {
                if a[j] == a[i] && _arr[j] != 3 {
                    xa += 1;
                }
            }
            for j: usize in 0..=4 {
                if b[j] == a[i] && _arr[j] != 3 {
                    xb += 1;
                }
            }
            if xb == 0 {
                _arr[i] = 1;
            }
            else if xa <= xb {
                _arr[i] = 2;
            }
            else {
                _arr[i] = 1;
            }
        }
    }
    _arr
} fn compare_vec
```

three_b 函数用来实现提高功能 2 的信息熵算法，下图只是函数的一部分，该函数通过返回一个 vec<String>得到每次猜测后的最优的十个猜测词。



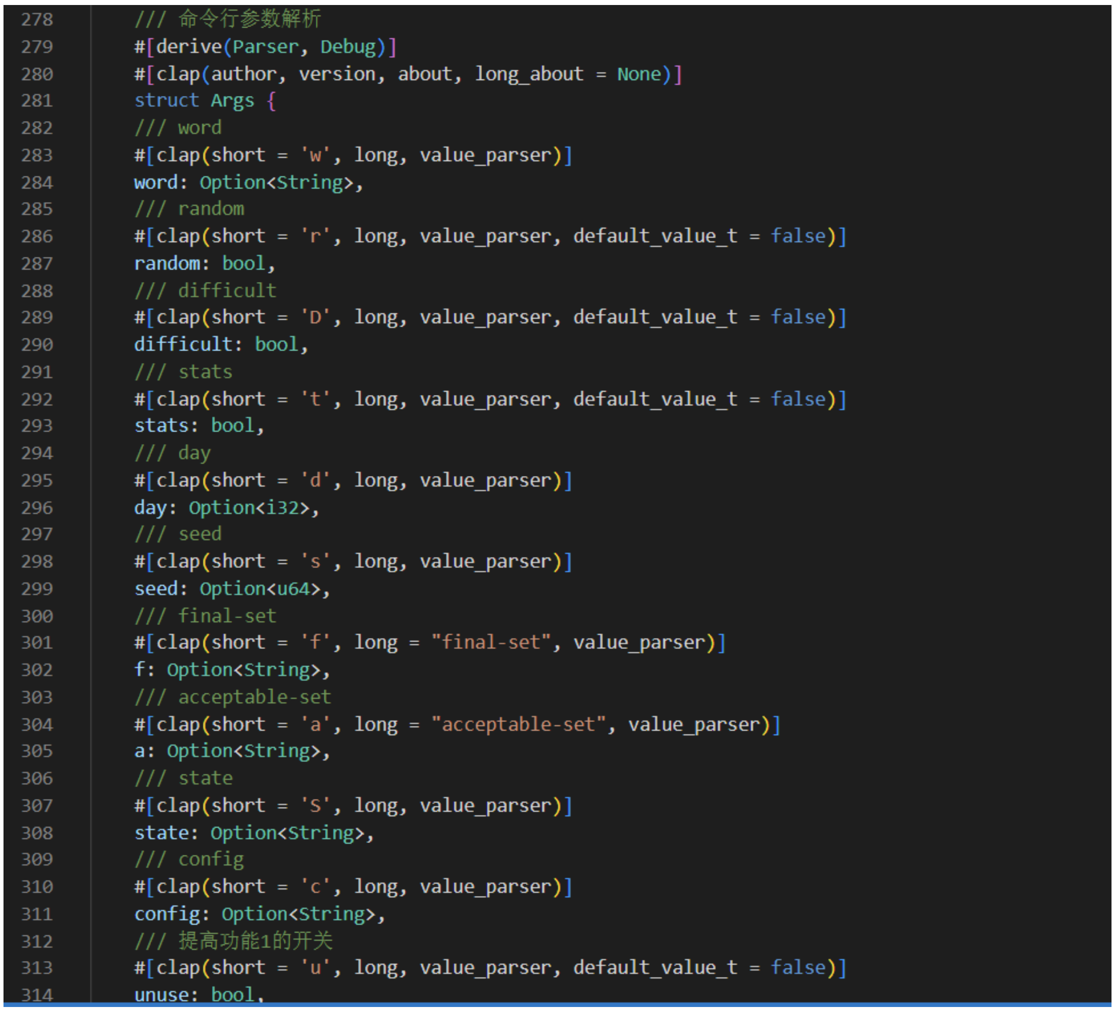
```rust
fn three_b(xia: &Vec<usize>, list: &Vec<&str>) -> Vec<String> {
    // GREEN => 2; YELLOW => 1; RED => 0
    let mut shang_word: [f64; _] = [0.0; LENGTH];
    let mut _arr: [f64; 10] = [-1.0; 10];
    let mut choose:[i32; 10] = [-1; 10];
    // let mut max = -1.0;
    // let mut max_word: &str = &String::new();
    let mut out: Vec<String> = Vec::new();
    let tmp: f64 = xia.len() as f64;
    for i: usize in 0..xia.len() {
        let mut shang: [i32; 243] = [0; 243];
        for j: usize in 0..xia.len() {
            let a: Vec<char> = vec_to_big(word: &list[xia[i]].replace(from: "\n", to: "").chars().collect::<Vec<_>>());
            let b: Vec<char> = vec_to_big(word: &list[xia[j]].replace(from: "\n", to: "").chars().collect::<Vec<_>>());
            let _arr: [i32; 5] = compare_vec(&a, &b);
            let mut count: i32 = 0;
            for k: usize in 0.._arr.len() {
                let delta: i32 = (_arr[k] - 1) * (power(di: 3, mi: k) as i32);
                count += delta;
            }
            shang[count as usize] += 1;
        }
        let mut total: f64 = 0.0;
        for i: usize in 0..shang.len() {
            let tmp_: f64 = shang[i] as f64;
            if tmp_ != 0.0 {
                total -= tmp_ / tmp * (tmp_ / tmp).log2();
            }
        }
        shang_word[xia[i]] = total;
        let mut count: usize = 0;
        for i: usize in 0..10 {
            if total <= _arr[i] {
                count += 1;
            }
        }
```

three_b_whole 函数通过在 three_b 函数的基础上进行修改，实现了给出每次猜测后的全局最优解。



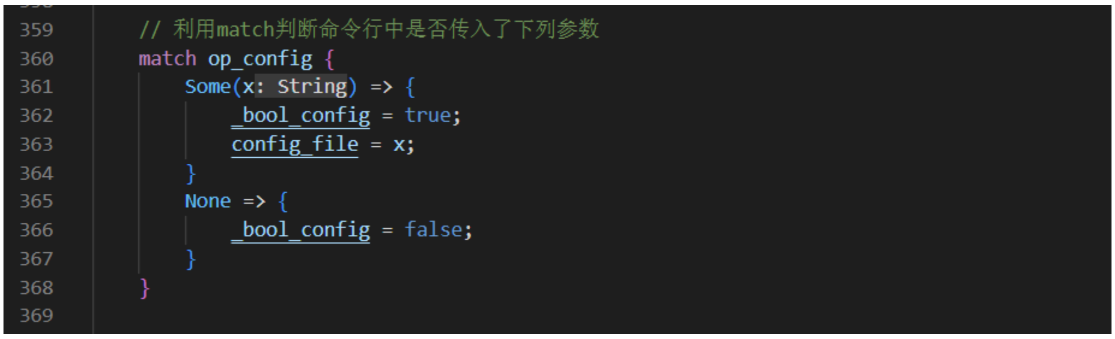
```rust
fn three_b_whole(xia: &Vec<usize>, list: &Vec<&str>) -> String {
    // GREEN => 2; YELLOW => 1; RED => 0
    let mut shang_word: [f64; _] = [0.0; LENGTH];
    let mut max: f64 = -1.0;
    let mut max_word: &str = &String::new();
    let tmp: f64 = xia.len() as f64;
    for i: usize in 0..LENGTH {
        let mut shang: [i32; 243] = [0; 243];
        for j: usize in 0..xia.len() {
            let a: Vec<char> = vec_to_big(word: &list[i].replace(from: "\n", to: "").chars().collect::<Vec<_>>());
            let b: Vec<char> = vec_to_big(word: &list[xia[j]].replace(from: "\n", to: "").chars().collect::<Vec<_>>());
            let _arr: [i32; 5] = compare_vec(&a, &b);
            let mut count: i32 = 0;
            for k: usize in 0.._arr.len() {
                let delta: i32 = (_arr[k] - 1) * (power(di: 3, mi: k) as i32);
                count += delta;
            }
            shang[count as usize] += 1;
        }
        let mut total: f64 = 0.0;
        for i: usize in 0..shang.len() {
            let tmp_: f64 = shang[i] as f64;
            if tmp_ != 0.0 {
                total -= tmp_ / tmp * (tmp_ / tmp).log2();
            }
        }
        shang_word[i] = total;
        if shang_word[i] > max {
            max = shang_word[i];
            max_word = list[i];
        }
        else if shang_word[i] == max {
            let mut in_list: bool = false;
            for t: usize in 0..xia.len() {
                if list[i] == list[xia[t]] {
                    in_list = true;
```

main 函数中，命令行参数的解析引入了 clap 包来进行。



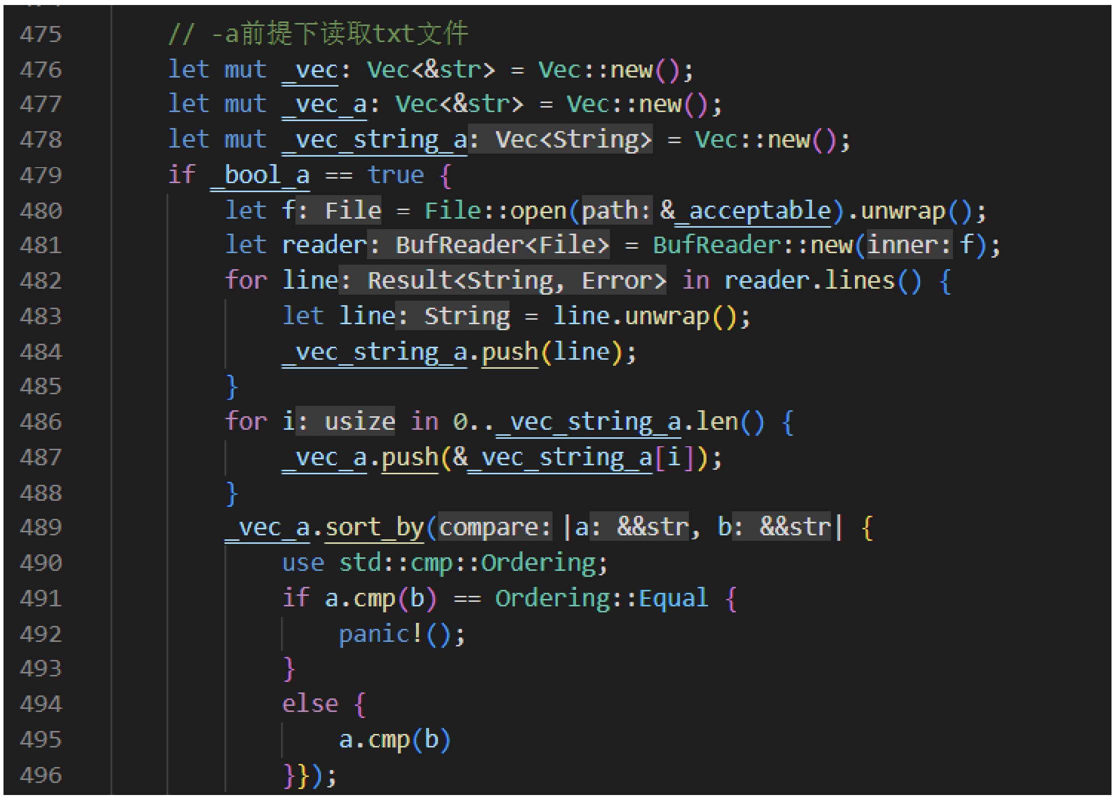
```rust
/// 命令行参数解析
#[derive(Parser, Debug)]
#[clap(author, version, about, long_about = None)]
struct Args {
/// word
#[clap(short = 'w', long, value_parser)]
word: Option<String>,
/// random
#[clap(short = 'r', long, value_parser, default_value_t = false)]
random: bool,
/// difficult
#[clap(short = 'D', long, value_parser, default_value_t = false)]
difficult: bool,
/// stats
#[clap(short = 't', long, value_parser, default_value_t = false)]
stats: bool,
/// day
#[clap(short = 'd', long, value_parser)]
day: Option<i32>,
/// seed
#[clap(short = 's', long, value_parser)]
seed: Option<u64>,
/// final-set
#[clap(short = 'f', long = "final-set", value_parser)]
f: Option<String>,
/// acceptable-set
#[clap(short = 'a', long = "acceptable-set", value_parser)]
a: Option<String>,
/// state
#[clap(short = 'S', long, value_parser)]
state: Option<String>,
/// config
#[clap(short = 'c', long, value_parser)]
config: Option<String>,
/// 提高功能1的开关
#[clap(short = 'u', long, value_parser, default_value_t = false)]
unuse: bool,
```

解析参数后，除了简单的 bool 值以外，利用 Option，采用 match 的方法判断是否有参数传入，并将传入的数值/String 记录下来。



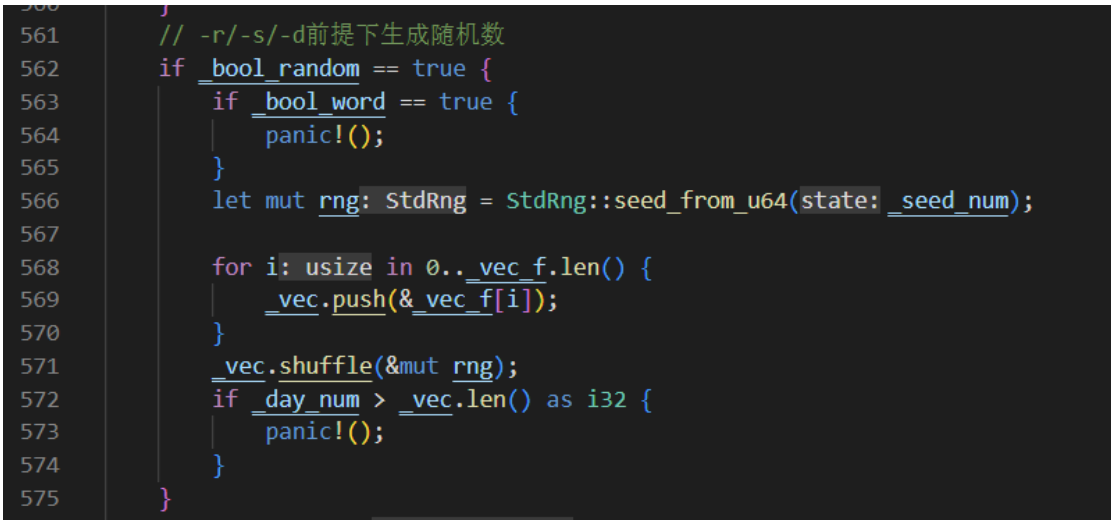
```rust
// 利用match判断命令行中是否传入了下列参数
match op_config {
    Some(x: String) => {
        _bool_config = true;
        config_file = x;
    }
    None => {
        _bool_config = false;
    }
}

```

在-a/-f 参数指定候选词库和可用词库的前提下，进行 txt 文件的读取并检查是否符合要求，以及按照字典序排序。



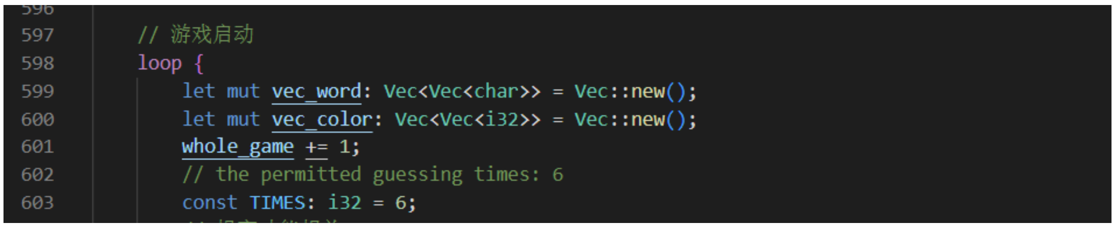
```
    // -a前提下读取txt文件
    let mut _vec: Vec<&str> = Vec::new();
    let mut _vec_a: Vec<&str> = Vec::new();
    let mut _vec_string_a: Vec<String> = Vec::new();
    if _bool_a == true {
        let f: File = File::open(path: &_acceptable).unwrap();
        let reader: BufReader<File> = BufReader::new(inner: f);
        for line: Result<String, Error> in reader.lines() {
            let line: String = line.unwrap();
            _vec_string_a.push(line);
        }
        for i: usize in 0.._vec_string_a.len() {
            _vec_a.push(&_vec_string_a[i]);
        }
        _vec_a.sort_by(compare: |a: &&str, b: &&str| {
            use std::cmp::Ordering;
            if a.cmp(b) == Ordering::Equal {
                panic!();
            }
            else {
                a.cmp(b)
            }});
```

随机模式中按照参数（随机种子）打乱词库。



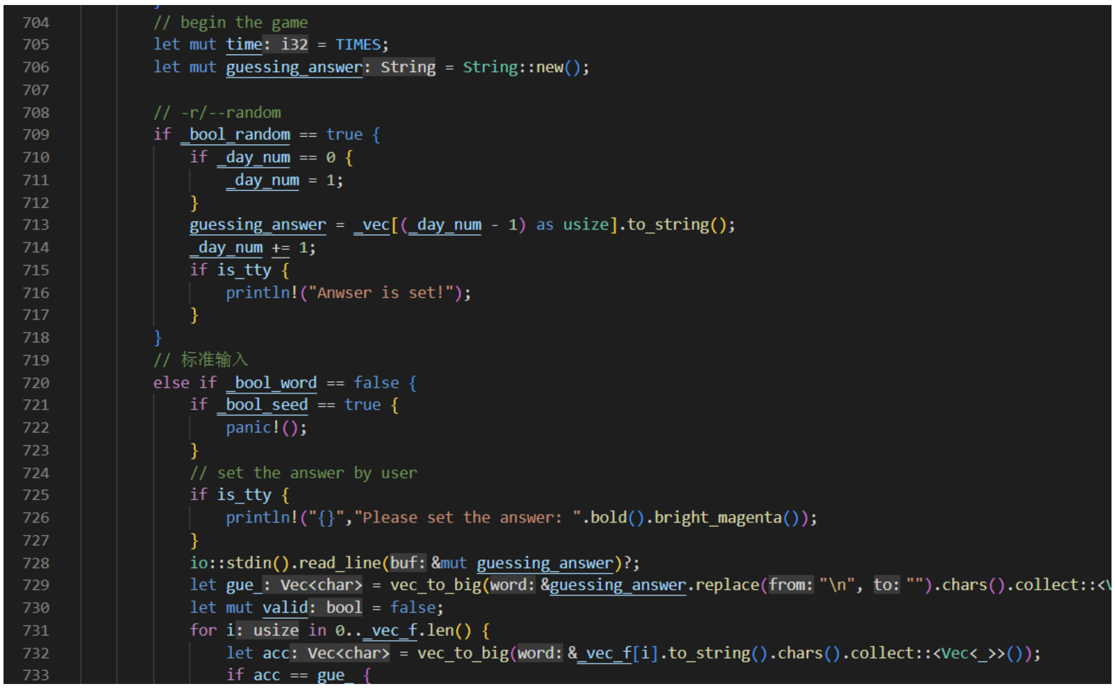
```
    // -r/-s/-d前提下生成随机数
    if _bool_random == true {
        if _bool_word == true {
            panic!();
        }
        let mut rng: StdRng = StdRng::seed_from_u64(state: _seed_num);

        for i: usize in 0.._vec_f.len() {
            _vec.push(&_vec_f[i]);
        }
        _vec.shuffle(&mut rng);
        if _day_num > _vec.len() as i32 {
            panic!();
        }
    }
```

进入 loop 循环，一局游戏启动。



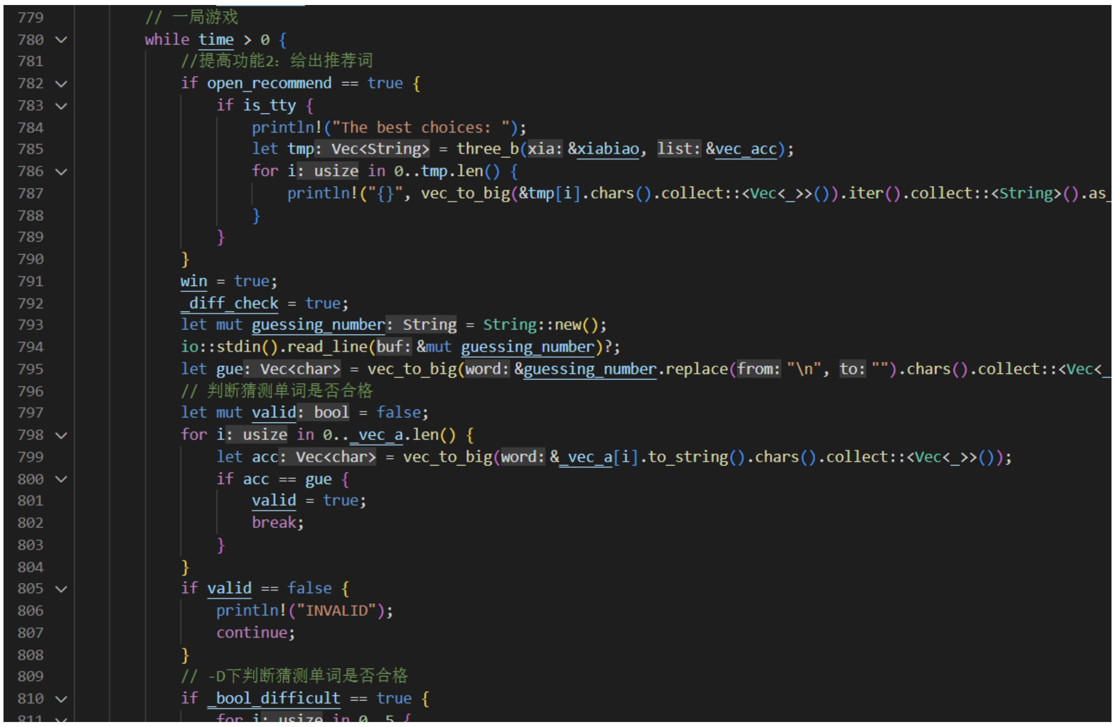
```
    // 游戏启动
    loop {
        let mut vec_word: Vec<Vec<char>> = Vec::new();
        let mut vec_color: Vec<Vec<i32>> = Vec::new();
        whole_game += 1;
        // the permitted guessing times: 6
        const TIMES: i32 = 6;
```

在开始猜测之前，判断是随机模式输入、标准输入还是指定答案模式输入。



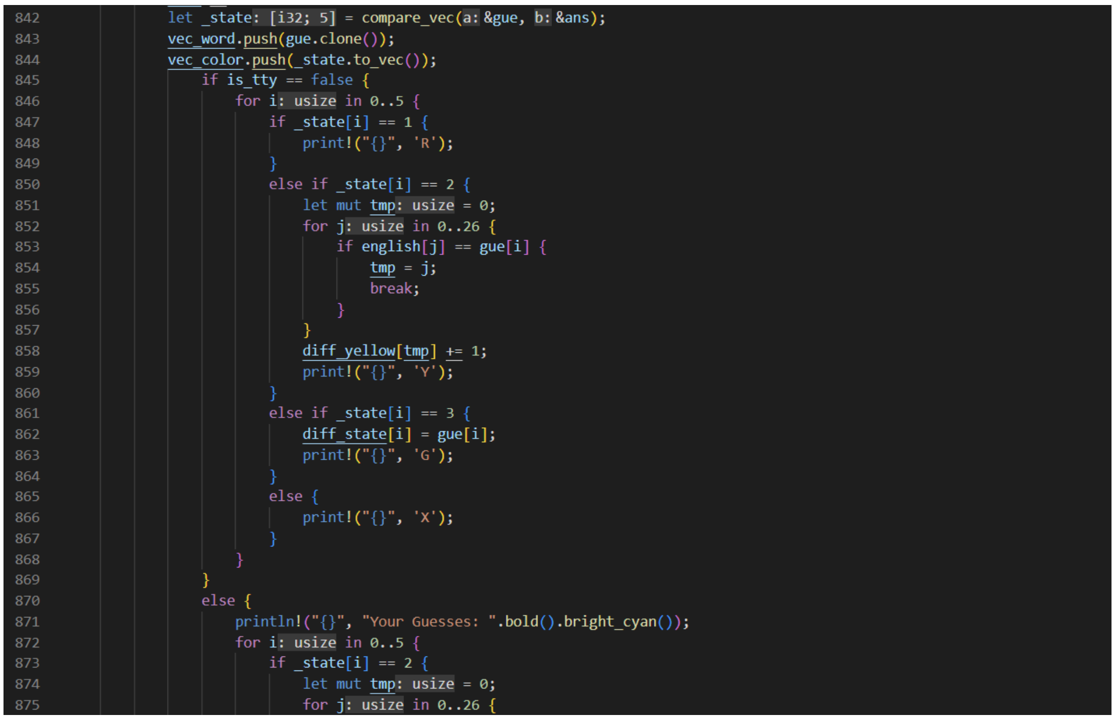
```
        // begin the game
        let mut time: i32 = TIMES;
        let mut guessing_answer: String = String::new();

        // -r/--random
        if _bool_random == true {
            if _day_num == 0 {
                _day_num = 1;
            }
            guessing_answer = _vec[(_day_num - 1) as usize].to_string();
            _day_num += 1;
            if is_tty {
                println!("Anwser is set!");
            }
        }
        // 标准输入
        else if _bool_word == false {
            if _bool_seed == true {
                panic!();
            }
            // set the answer by user
            if is_tty {
                println!("{}","Please set the answer: ".bold().bright_magenta());
            }
            io::stdin().read_line(buf: &mut guessing_answer)?;
            let gue_: Vec<char> = vec_to_big(word: &guessing_answer.replace(from: "\n", to: "").chars().collect::<V
            let mut valid: bool = false;
            for i: usize in 0.._vec_f.len() {
                let acc: Vec<char> = vec_to_big(word: &_vec_f[i].to_string().chars().collect::<Vec<_>>());
                if acc == gue_ {
```

从 while 循环，进入到一次猜测之中。



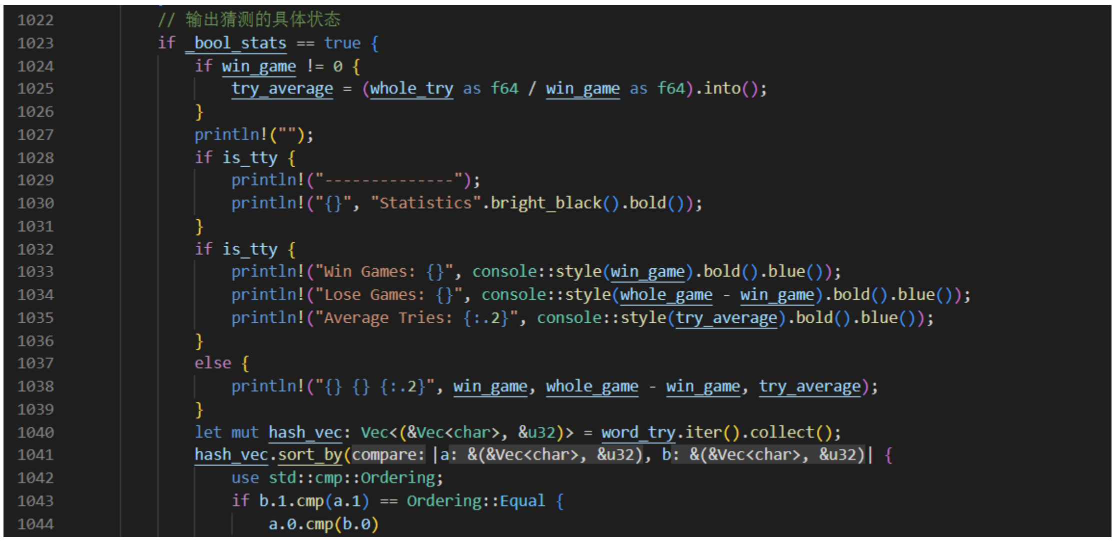
```rust
// 一局游戏
while time > 0 {
    //提高功能2：给出推荐词
    if open_recommend == true {
        if is_tty {
            println!("The best choices: ");
            let tmp: Vec<String> = three_b(xia: &xiabiao, list: &vec_acc);
            for i: usize in 0..tmp.len() {
                println!("{}", vec_to_big(&tmp[i].chars().collect::<Vec<_>>()).iter().collect::<String>().as_
            }
        }
    }
    win = true;
    _diff_check = true;
    let mut guessing_number: String = String::new();
    io::stdin().read_line(buf: &mut guessing_number)?;
    let gue: Vec<char> = vec_to_big(word: &guessing_number.replace(from: "\n", to: "").chars().collect::<Vec<_
    // 判断猜测单词是否合格
    let mut valid: bool = false;
    for i: usize in 0.._vec_a.len() {
        let acc: Vec<char> = vec_to_big(word: &_vec_a[i].to_string().chars().collect::<Vec<_>>());
        if acc == gue {
            valid = true;
            break;
        }
    }
    if valid == false {
        println!("INVALID");
        continue;
    }
    // -D下判断猜测单词是否合格
    if _bool_difficult == true {
        for i: usize in 0..5 {
```

与用户交互输出猜测结果时根据 compare_vec 的结果更新 26 个字母的状态并对于猜测进行颜色输出。



```rust
let _state: [i32; 5] = compare_vec(a: &gue, b: &ans);
vec_word.push(gue.clone());
vec_color.push(_state.to_vec());
    if is_tty == false {
        for i: usize in 0..5 {
            if _state[i] == 1 {
                print!("{}", 'R');
            }
            else if _state[i] == 2 {
                let mut tmp: usize = 0;
                for j: usize in 0..26 {
                    if english[j] == gue[i] {
                        tmp = j;
                        break;
                    }
                }
                diff_yellow[tmp] += 1;
                print!("{}", 'Y');
            }
            else if _state[i] == 3 {
                diff_state[i] = gue[i];
                print!("{}", 'G');
            }
            else {
                print!("{}", 'X');
            }
        }
    }
    else {
        println!("{}", "Your Guesses: ".bold().bright_cyan());
        for i: usize in 0..5 {
            if _state[i] == 2 {
                let mut tmp: usize = 0;
                for j: usize in 0..26 {
```

对于游戏的猜测状态的输出。

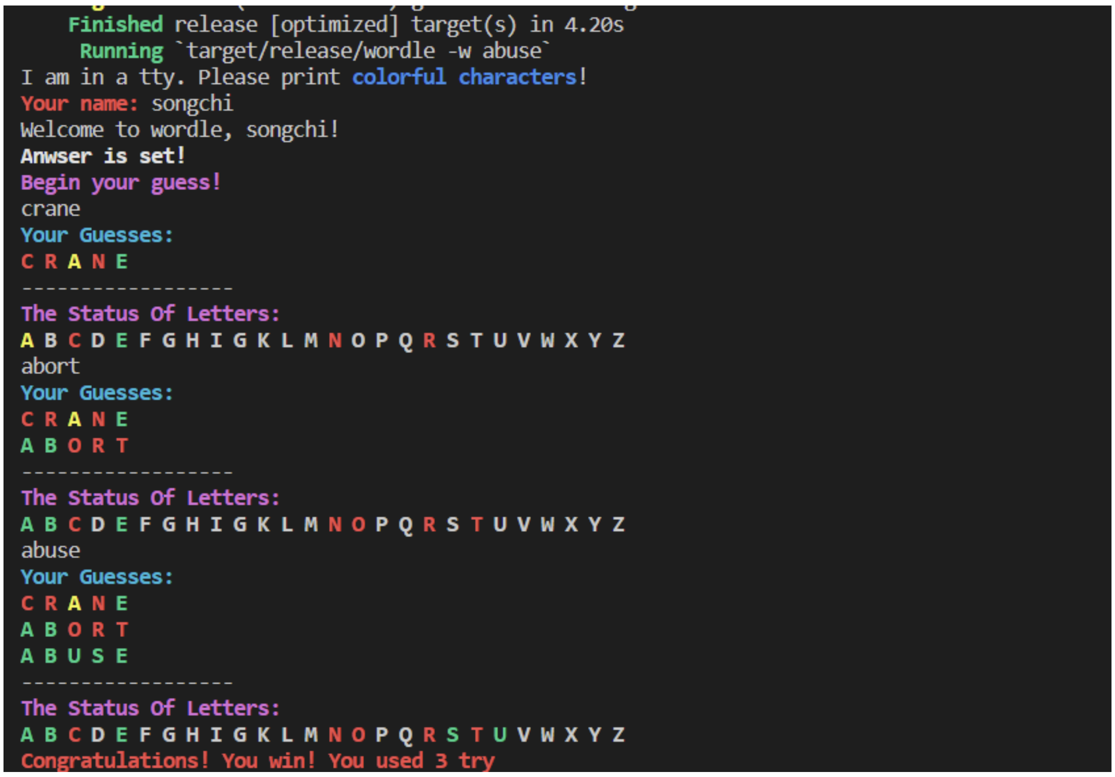


```
        // 输出猜测的具体状态
        if _bool_stats == true {
            if win_game != 0 {
                try_average = (whole_try as f64 / win_game as f64).into();
            }
            println!("");
            if is_tty {
                println!("--------------");
                println!("{}", "Statistics".bright_black().bold());
            }
            if is_tty {
                println!("Win Games: {}", console::style(win_game).bold().blue());
                println!("Lose Games: {}", console::style(whole_game - win_game).bold().blue());
                println!("Average Tries: {:.2}", console::style(try_average).bold().blue());
            }
            else {
                println!("{} {} {:.2}", win_game, whole_game - win_game, try_average);
            }
            let mut hash_vec: Vec<(&Vec<char>, &u32)> = word_try.iter().collect();
            hash_vec.sort_by(compare: |a: &(&Vec<char>, &u32), b: &(&Vec<char>, &u32)| {
                use std::cmp::Ordering;
                if b.1.cmp(a.1) == Ordering::Equal {
                    a.0.cmp(b.0)
```

二、　游戏主要功能说明和截图

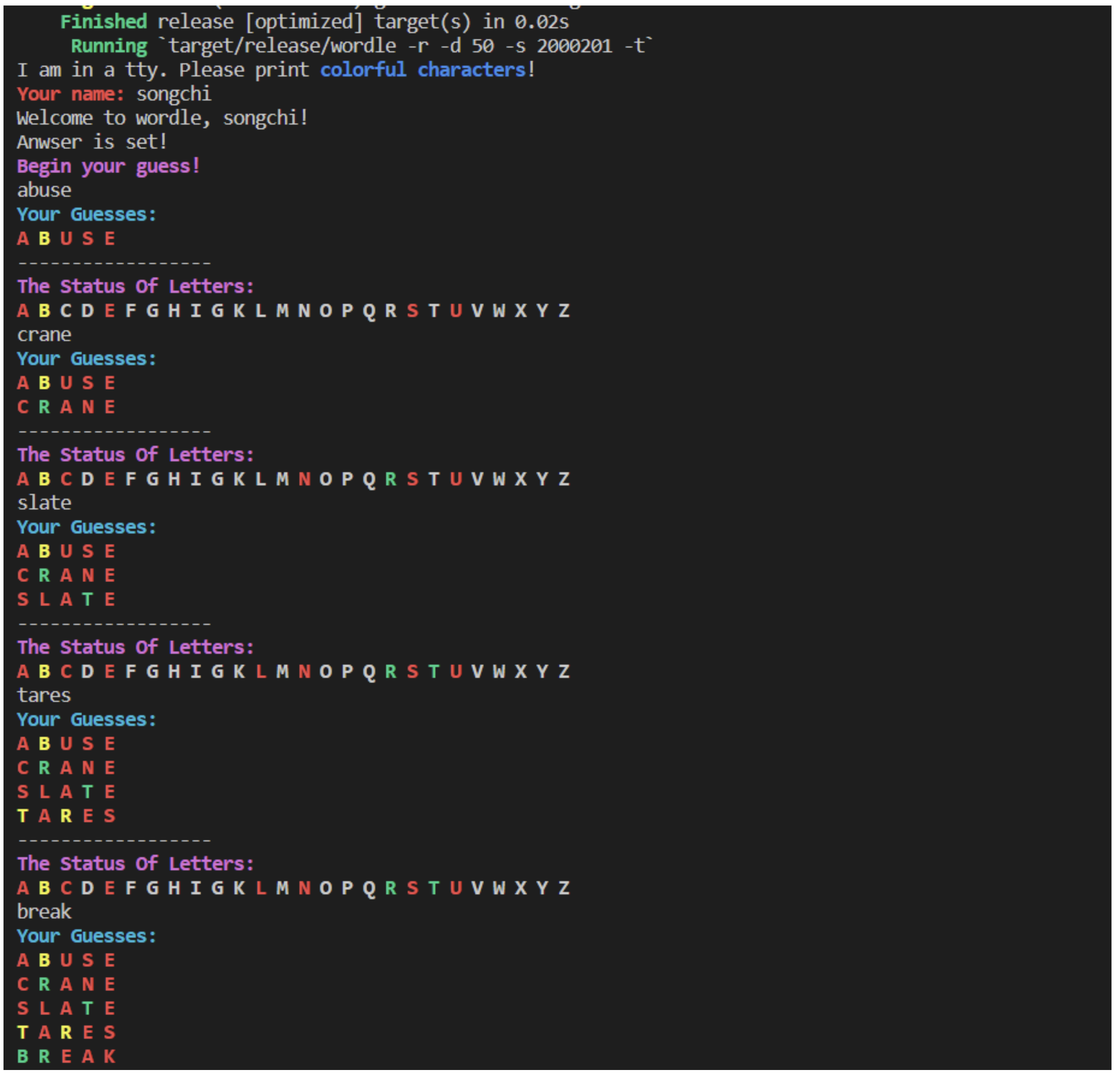
在我的大作业 wordle 中，实现了基础功能的测试模式和交互模式，以及提高功能，下面我将针对我的交互模式的实现举例。

例：在-w 指定答案的模式下进行猜测，每次反馈所有猜测的结果以及 26 个字母的状态，猜测成功后也会有相应的输出。

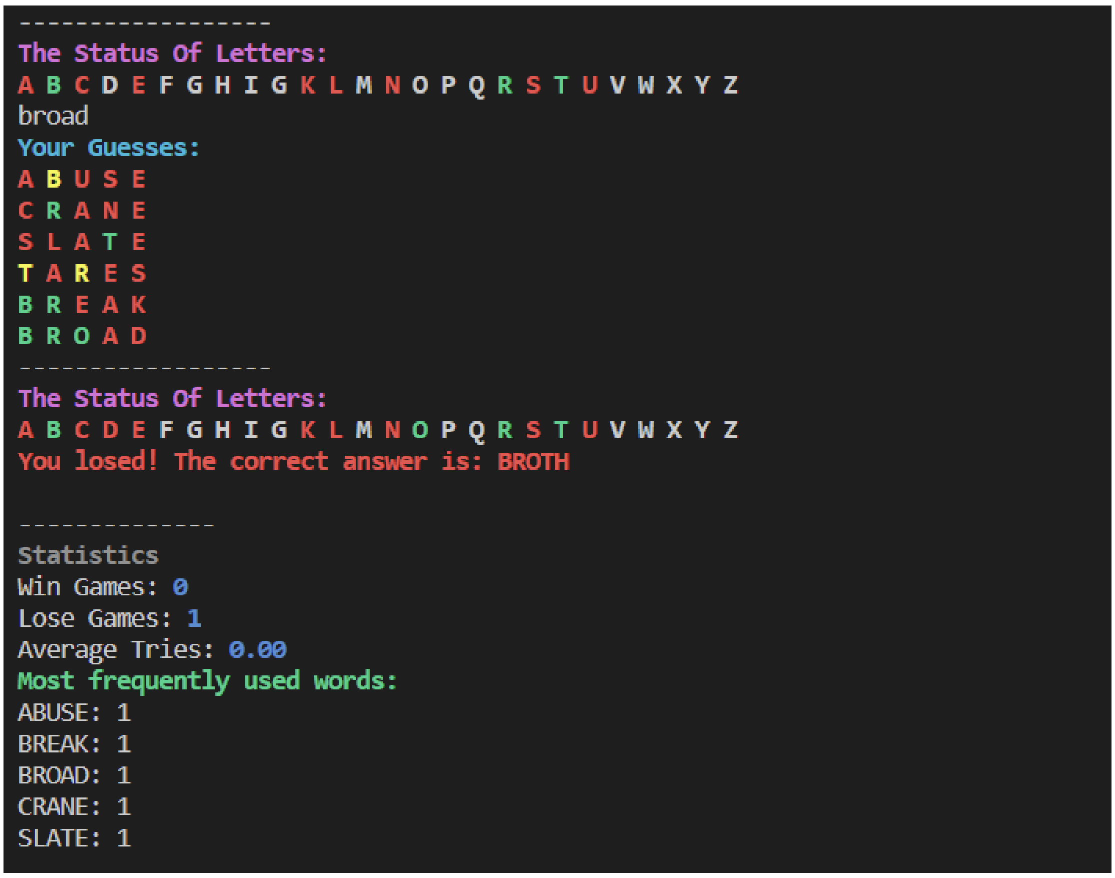


```
    Finished release [optimized] target(s) in 4.20s
     Running `target/release/wordle -w abuse`
I am in a tty. Please print colorful characters!
Your name: songchi
Welcome to wordle, songchi!
Anwser is set!
Begin your guess!
crane
Your Guesses:
C R A N E
------------------
The Status Of Letters:
A B C D E F G H I G K L M N O P Q R S T U V W X Y Z
abort
Your Guesses:
C R A N E
A B O R T
------------------
The Status Of Letters:
A B C D E F G H I G K L M N O P Q R S T U V W X Y Z
abuse
Your Guesses:
C R A N E
A B O R T
A B U S E
------------------
The Status Of Letters:
A B C D E F G H I G K L M N O P Q R S T U V W X Y Z
Congratulations! You win! You used 3 try
```

例：在-r 随机模式下，通过-s 和-d 确定答案进行猜测，同时通过-t 进行游戏局数以及最常用的猜测单词的输出。

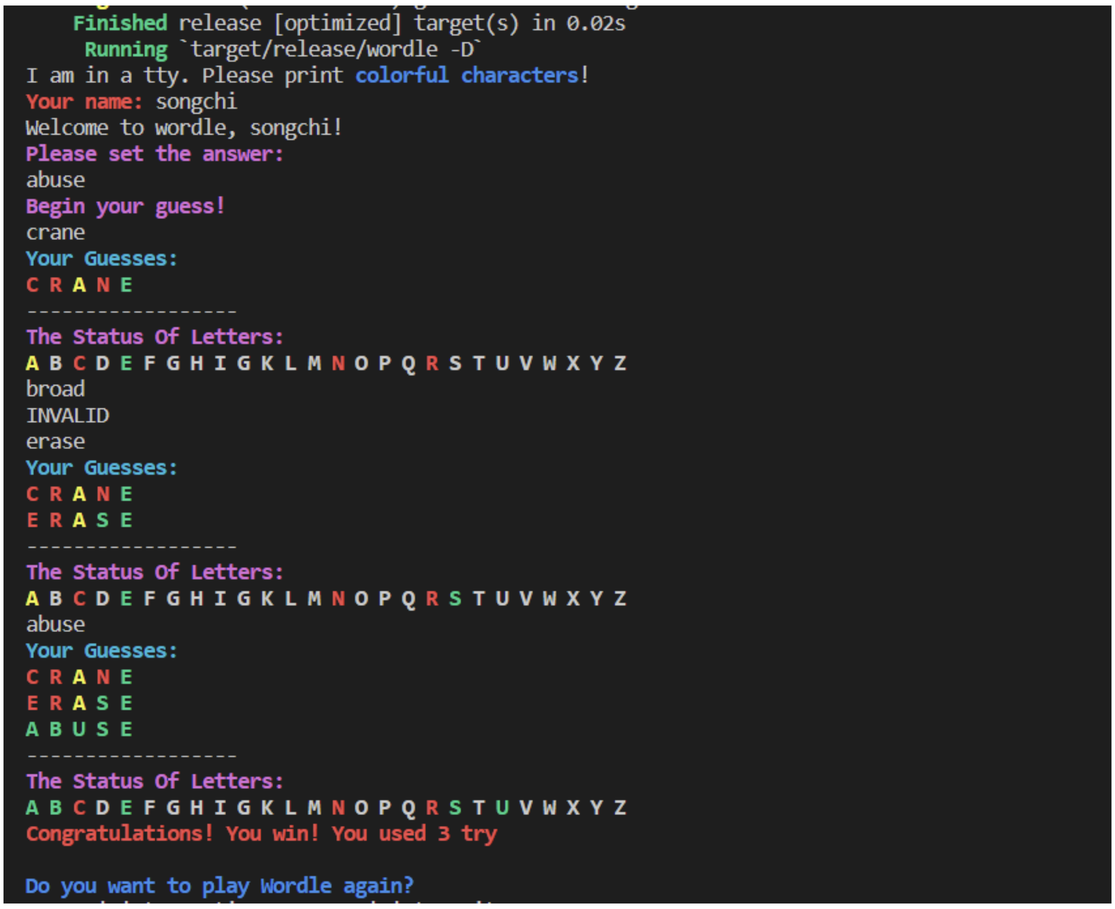


```
    Finished release [optimized] target(s) in 0.02s
     Running `target/release/wordle -r -d 50 -s 2000201 -t`
I am in a tty. Please print colorful characters!
Your name: songchi
Welcome to wordle, songchi!
Anwser is set!
Begin your guess!
abuse
Your Guesses:
A B U S E
------------------
The Status Of Letters:
A B C D E F G H I G K L M N O P Q R S T U V W X Y Z
crane
Your Guesses:
A B U S E
C R A N E
------------------
The Status Of Letters:
A B C D E F G H I G K L M N O P Q R S T U V W X Y Z
slate
Your Guesses:
A B U S E
C R A N E
S L A T E
------------------
The Status Of Letters:
A B C D E F G H I G K L M N O P Q R S T U V W X Y Z
tares
Your Guesses:
A B U S E
C R A N E
S L A T E
T A R E S
------------------
The Status Of Letters:
A B C D E F G H I G K L M N O P Q R S T U V W X Y Z
break
Your Guesses:
A B U S E
C R A N E
S L A T E
T A R E S
B R E A K
```



```
------------------
The Status Of Letters:
A B C D E F G H I G K L M N O P Q R S T U V W X Y Z
broad
Your Guesses:
A B U S E
C R A N E
S L A T E
T A R E S
B R E A K
B R O A D
------------------
The Status Of Letters:
A B C D E F G H I G K L M N O P Q R S T U V W X Y Z
You losed! The correct answer is: BROTH

--------------
Statistics
Win Games: 0
Lose Games: 1
Average Tries: 0.00
Most frequently used words:
ABUSE: 1
BREAK: 1
BROAD: 1
CRANE: 1
SLATE: 1
```

例：difficult 模式下对输入单词进行严格的判断，并及时反馈 INVALID。

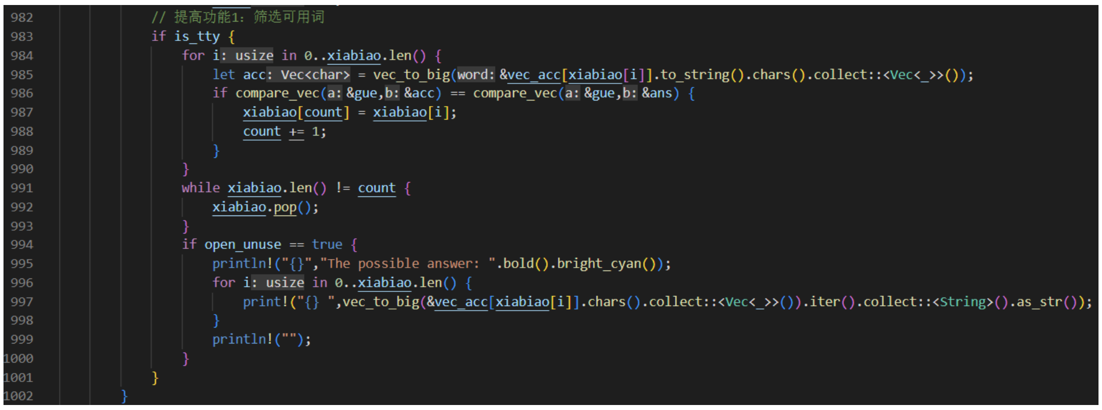


```
    Finished release [optimized] target(s) in 0.02s
     Running `target/release/wordle -D`
I am in a tty. Please print colorful characters!
Your name: songchi
Welcome to wordle, songchi!
Please set the answer:
abuse
Begin your guess!
crane
Your Guesses:
C R A N E
------------------
The Status Of Letters:
A B C D E F G H I G K L M N O P Q R S T U V W X Y Z
broad
INVALID
erase
Your Guesses:
C R A N E
E R A S E
------------------
The Status Of Letters:
A B C D E F G H I G K L M N O P Q R S T U V W X Y Z
abuse
Your Guesses:
C R A N E
E R A S E
A B U S E
------------------
The Status Of Letters:
A B C D E F G H I G K L M N O P Q R S T U V W X Y Z
Congratulations! You win! You used 3 try

Do you want to play Wordle again?
```

三、 提高要求的实现方式

提高要求 1：筛选可用词

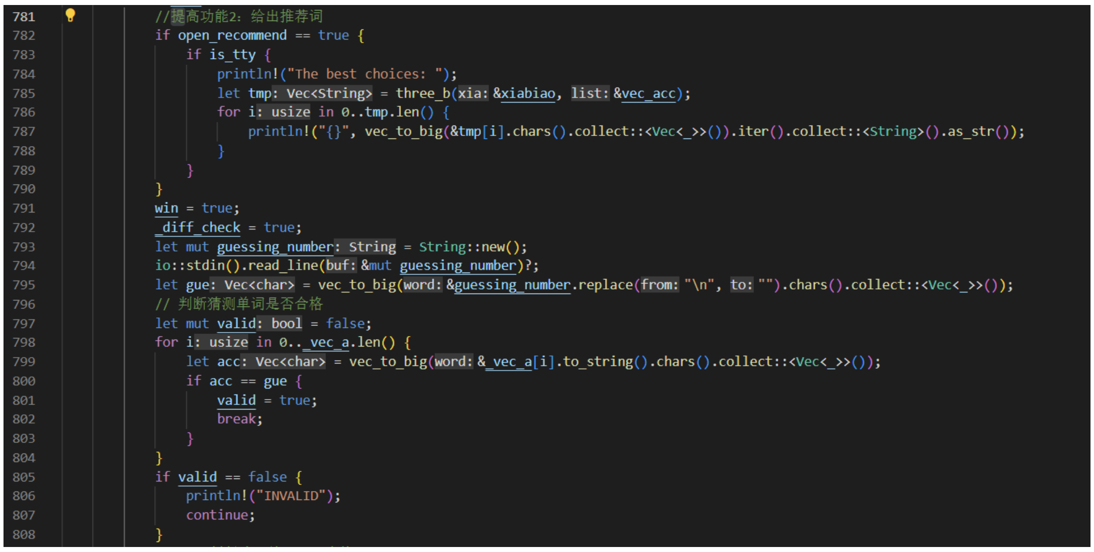
利用 compare_vec 函数判断候选词库的单词是否可能成为正确答案，并及时更新并输出。



```rust
            // 提高功能1: 筛选可用词
            if is_tty {
                for i: usize in 0..xiabiao.len() {
                    let acc: Vec<char> = vec_to_big(word: &vec_acc[xiabiao[i]].to_string().chars().collect::<Vec<_>>());
                    if compare_vec(a: &gue,b: &acc) == compare_vec(a: &gue,b: &ans) {
                        xiabiao[count] = xiabiao[i];
                        count += 1;
                    }
                }
                while xiabiao.len() != count {
                    xiabiao.pop();
                }
                if open_unuse == true {
                    println!("{}","The possible answer: ".bold().bright_cyan());
                    for i: usize in 0..xiabiao.len() {
                        print!("{} ",vec_to_big(&vec_acc[xiabiao[i]].chars().collect::<Vec<_>>()).iter().collect::<String>().as_str());
                    }
                    println!("");
                }
            }
        }
```

提高要求 2：给出推荐词

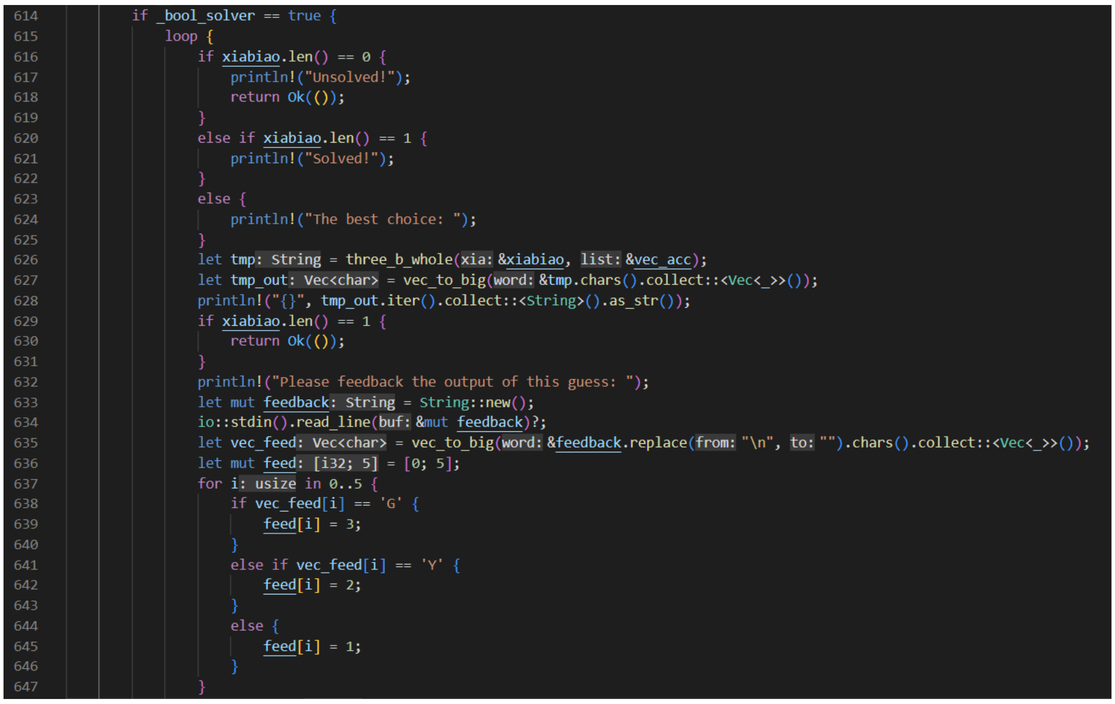
利用此前写好的 three_b 函数给出推荐词，同时在提高要求 1 的基础上及时更新候选词库中的单词（筛掉不可能成为答案的单词，减少计算信息熵的时间）。



```rust
            //提高功能2: 给出推荐词
            if open_recommend == true {
                if is_tty {
                    println!("The best choices: ");
                    let tmp: Vec<String> = three_b(xia: &xiabiao, list: &vec_acc);
                    for i: usize in 0..tmp.len() {
                        println!("{}", vec_to_big(&tmp[i].chars().collect::<Vec<_>>()).iter().collect::<String>().as_str());
                    }
                }
            }
            win = true;
            _diff_check = true;
            let mut guessing_number: String = String::new();
            io::stdin().read_line(buf: &mut guessing_number)?;
            let gue: Vec<char> = vec_to_big(word: &guessing_number.replace(from: "\n", to: "").chars().collect::<Vec<_>>());
            // 判断猜测单词是否合格
            let mut valid: bool = false;
            for i: usize in 0.._vec_a.len() {
                let acc: Vec<char> = vec_to_big(word: &_vec_a[i].to_string().chars().collect::<Vec<_>>());
                if acc == gue {
                    valid = true;
                    break;
                }
            }
            if valid == false {
                println!("INVALID");
                continue;
            }
```

提高要求 3：wordle-solver

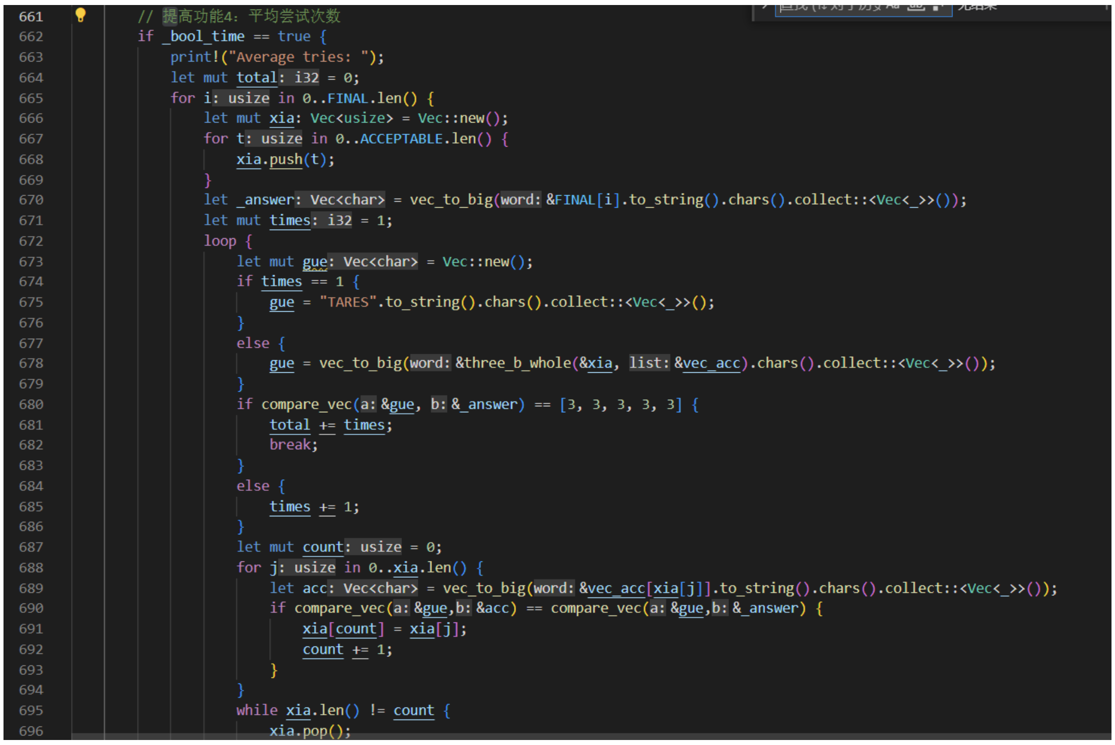
利用 three_b_whole 函数对于每一次猜测后，根据用户反馈的颜色信息，在全局单词中给出推荐词。



```rust
        if _bool_solver == true {
            loop {
                if xiabiao.len() == 0 {
                    println!("Unsolved!");
                    return Ok(());
                }
                else if xiabiao.len() == 1 {
                    println!("Solved!");
                }
                else {
                    println!("The best choice: ");
                }
                let tmp: String = three_b_whole(xia: &xiabiao, list: &vec_acc);
                let tmp_out: Vec<char> = vec_to_big(word: &tmp.chars().collect::<Vec<_>>());
                println!("{}", tmp_out.iter().collect::<String>().as_str());
                if xiabiao.len() == 1 {
                    return Ok(());
                }
                println!("Please feedback the output of this guess: ");
                let mut feedback: String = String::new();
                io::stdin().read_line(buf: &mut feedback)?;
                let vec_feed: Vec<char> = vec_to_big(word: &feedback.replace(from: "\n", to: "").chars().collect::<Vec<_>>());
                let mut feed: [i32; 5] = [0; 5];
                for i: usize in 0..5 {
                    if vec_feed[i] == 'G' {
                        feed[i] = 3;
                    }
                    else if vec_feed[i] == 'Y' {
                        feed[i] = 2;
                    }
                    else {
                        feed[i] = 1;
                    }
                }
```

提高要求 4：测试平均猜测次数

遍历 FINAL 中的所有单词，根据 three_b_whole 函数以及算法给出推荐词来模拟猜测的过程操作，最终得到猜测次数并进行累加计算平均猜测次数。



```rust
        // 提高功能4：平均尝试次数
        if _bool_time == true {
            print!("Average tries: ");
            let mut total: i32 = 0;
            for i: usize in 0..FINAL.len() {
                let mut xia: Vec<usize> = Vec::new();
                for t: usize in 0..ACCEPTABLE.len() {
                    xia.push(t);
                }
                let _answer: Vec<char> = vec_to_big(word: &FINAL[i].to_string().chars().collect::<Vec<_>>());
                let mut times: i32 = 1;
                loop {
                    let mut gue: Vec<char> = Vec::new();
                    if times == 1 {
                        gue = "TARES".to_string().chars().collect::<Vec<_>>();
                    }
                    else {
                        gue = vec_to_big(word: &three_b_whole(&xia, list: &vec_acc).chars().collect::<Vec<_>>());
                    }
                    if compare_vec(a: &gue, b: &_answer) == [3, 3, 3, 3, 3] {
                        total += times;
                        break;
                    }
                    else {
                        times += 1;
                    }
                    let mut count: usize = 0;
                    for j: usize in 0..xia.len() {
                        let acc: Vec<char> = vec_to_big(word: &vec_acc[xia[j]].to_string().chars().collect::<Vec<_>>());
                        if compare_vec(a: &gue,b: &acc) == compare_vec(a: &gue,b: &_answer) {
                            xia[count] = xia[j];
                            count += 1;
                        }
                    }
                    while xia.len() != count {
                        xia.pop();
```

四、　完成此作业感想

在短短一周的时间完成这样一个大作业，对于我来说是一个挑战，当时开始的时候也没曾想自己能在这样短暂的时间里完成这样一份对自己来说还算满意的作业。对于一门新接触的语言，自己在这一周中没日没夜地查找资料，不停地调试代码，并与同学交流如何写才是正确的，在这样不断获取知识并交流的过程中我逐渐地熟悉了这门语言，也锻炼了我的代码能力，过程是痛苦的，但也给予我不小收获，尤其是自己对于上网搜索并阅读文档的能力有了不小提升。最后，写了这样的代码实现这样一个 wordle 游戏，还是让自己比较有成就感的，也理解了 wordle 求解背后的一些知识。