Sudoku 数独游戏

2022 年 11 月

目录

1 文件结构

1.1 sudoku.h

```
constexpr int BOARD_SIZE = 9;
constexpr int BLOCK_SIZE = 3;
```

表示数独总棋盘边长、数独九宫格边长。

```
constexpr int SCREEN_WIDTH = <>;
constexpr int SCREEN_HEIGHT = <>;
```

表示屏幕输出区的大小。(未最终确认大小)

```
constexpr int NUMBER_SIZE = 9;
```

表示数独中可填数的上界。

```
constexpr int KEY_TYPE_SIZE = 21;
```

表示按键类型的数量（见下方 Key_type)。

```
enum Key_type {
    K_UP,
    K_DOWN,
    K_LEFT,
    K_RIGHT,
    K_REGAME,
    K_UNDO,
    K_REDO,
    K_ZERO,
    K_ONE,
    K_TWO,
    K_THREE,
    K_FOUR,
    K_FIVE,
    K_SIX,
    K_SEVEN,
    K_EIGHT,
    K_NINE,
    K_COMPLETE,
    K_COPY,
    K_QUIT,
    K_ENTER
};
```

定义不同类型的按键的名字。

```
struct Board {
    int init[BOARD_SIZE][BOARD_SIZE];
    int now[BOARD_SIZE][BOARD_SIZE];
};
```

用于储存数独信息的结构体。init 表示数独的初始情况，now 表示现在情况。游戏开始时 init=now。保留 init 信息方便判断操作的合法性。

```
typedef std::pair<int, int> Key_value;
```

用于保存 getch() 函数的返回值的类型。因为 getch() 函数对于一些按键需要获得两次，故使用一个 pair。

```
struct Setting_info {
    int difficulty;
    int open_hint;
    Key_value key_board[KEY_TYPE_SIZE];
};
```

用于保存当前配置信息的结构体。

difficulty 表示难度，取值在 $1 \sim 3$。

open_hint 表示是否开启提示，1 表示开启，0 表示不开启

key_board 存储的是所有按键类型对应的按键值。

```
typedef std::set<int> Hint;
```

定义 Hint 类型，表示提示信息。

```
struct Operation_node {
    int x, y, fr, to;
};
```

定义 Operation_node 类型，表示操作序列中的一个节点，从而方便 Redo 和 Undo 操作。表示在 (x, y) 将 fr 改成 to。

```
typedef std::pair<int, int> Coord;
```

定义棋盘上的一个点。

```
typedef std::vector<Operation_node> Operation_sequence;
```

定义 Operation_sequence 类型，表示操作序列，方便 Redo 和 Undo。

1.2 aid

1.2.1 draw.cpp

```
void Gotoxy(int x, int y);
```

将控制台光标放置到 (x, y)，从而方便 menu 函数定点输出。

使用时应该注意，定位后的输出不能超过 SCREEN_WIDTH 和 SCREEN_HEIGHT。

```
void Clear_screen(void);
```

将屏幕全清。

```
void Set_color(int color);
```

修改之后的字体颜色。

详情见这篇文章。

```
void Set_window_color(int color);
```

设置整个窗口的字体前景色、背景色。

详情见这篇文章。

1.2.2 copypaste.cpp

```
void Put_to_cp(std::string s);
```

将字符串 s 的值放置到剪切板里。

```
std::string Get_from_cp(void);
```

从剪切板里的内容获取并存放到 s 中。

1.2.3 stringboard.cpp

```
std::string Board_to_string(Board NowBoard);
```

将 NowBoard 里的内容转换成字符串。

该字符串如下：先行后列输出 init[][] 的内容后，再输出 now[][] 的内容。输出时先行后列，有空格与换行。

如果 init 和 now 相同，则输出 81 个字符，仅包含 init[][] 的内容。

```
Board String_to_board(std::string s)
```

将 s 的内容转换成 Board 并返回。

字符串规则同上。

如果不符合规则，则返回的 Board 的 init[0][0] 为 -1。

1.2.4 get_key.cpp

```
Key_value Get_key(void);
```

获得一个按键。如果是单个键值的，则返回 $< value, 0 >$，如果是两个键值的，则返回 $< value1, value2 >$。注意需要把小写字母转化成大写字母。

```
std::string Get_keystring(Key_value key);
```

将 key 值转化用于描述的 std::string。这个 std::string 在文件内静态定义。如果未定义，返回空串。

1.3 generate

1.3.1 generate_lib.cpp

```
bool Generate_lib(void);
```

从“棋盘文件”获得一个相应难度的棋盘，装载到 g_game_board 中。成功返回 1，否则返回 0。

棋盘文件保存在 sudoku_data/boards[123] 中，按照难度获取不同的文件。

文件内保存格式为:81 个字符为一组,有空格和换行。详情请见 Put_to_cp;

1.3.2 generate_saves.cpp

```
bool Generate_saves(void);
```

获取存档位的信息，并装载到 g_game_board 中。成功返回 1，否则返回 0。

存档文件保存在 sudoku_data/saves 中。格式为 81 / 162 个为一组。(即为 Board_to_string) 的格式。

1.3.3 generate_cp.cpp

```
bool Generate_cp(void);
```

从剪切板中获得棋盘，成功返回 1，否则返回 0。这里运用 Get_from_cp 与 String_to_board 函数完成。成功返回 1，失败返回 0。

1.3.4 generate_menu.cpp

```
void Generate_menu(void);
```

提供可视化菜单：需要有"Start from game library"，"Start from last saves"，"Start from copy & paste"，"Go back" 四个选项。使用 K_UP 和 K_DOWN 进行操作，使用 K_ENTER 进行确认，同时可以使用 K_ONE，K_TWO，K_THREE, K_QUIT 等等。

根据选项，分别调用 Generate_lib，Generate_saves，Generate_cp，三个函数。

1.4 play

1.4.1 game_board.cpp

```
Board g_game_board;
```

全局变量，表示目前的棋盘。

```
Operation_sequence g_operation_sequence;
```

全局变量，保存操作的序列。则 Undo 和 Redo 就可以借此完成。

```
int g_operation_sequence_id;
```

全局变量，和上面的 g_operation_sequence 一起使用，用来帮助 Redo。（即 Undo 时不立即弹出）

```
void Init_board(void);
```

初始化 g_game_board、g_operation_sequence、g_operation_sequence_id。

1.4.2 game_aid.cpp

```
Hint Get_hint(int x, int y);
```

得到棋盘上 (x, y) 这点的提示。注意 x, y 的下标范围均为 $0 \sim 8$。返回的 Hint 即 std::set<int> 为该点可填的数的集合。

```
bool Judge_legal(int x, int y);
```

判断 (x, y) 现在所填数是否合法。合法返回 1，不合法返回 0。

```
std::set<Coord> Get_illegal(void);
```

获得棋盘上不合法的棋子。

```
bool Judge_win();
```

判断游戏是否胜利，胜利返回 1，未胜利返回 0。

胜利的条件：棋盘全部填满；符合数独的规则(这个可以调用 Judge_legal 完成)。

1.4.3 operation

1.4.3.1 move.cpp

```
int g_cursor_x;
int g_cursor_y;
```

全局变量，为棋盘上的光标（取值均在 $0 \sim 8$)。

```
void Move_init(void);
```

初始化光标。

```
bool Go_up(void);
bool Go_down(void);
bool Go_left(void);
bool Go_right(void);
bool Move_cursor(int x, int y);
```

移动光标。成功移动返回 1，否则返回 0。即改变 g_cursor_x 和 g_cursor_y 的值。如果到边界，则失败。除了 Move_cursor 外，都会加入 operation_sequence。

1.4.3.2 write_number.cpp

```
bool Write_number(int x, int y, int val, int ty);
```

将 (x, y) 处的数更改为 val，成功返回 1，不成功返回 0。

如果 (x, y) 处为 init 中就存在的值（即不可填写的值），则失败。如果 ty 为 1，表示需要加入队列。

1.4.3.3 regame.cpp

```
bool Regame(void);
```

发出重玩请求，需要确认，确实重玩返回 1，否则返回 0。即将 now 赋值为 init，再调用 Init_board，再 Print_board。

1.4.3.4 undo_redo.cpp

```
bool Undo(void);
bool Redo(void);
```

撤销、恢复操作。成功返回 1，失败返回 0。

Undo 只需让 g_operation_sequence_id 自减。若为 0，则失败。

Redo 只需让 g_operation_sequence_id 自增。若为 size-1，则失败。

1.4.3.5 solve.cpp

```
Board Solve_game(Board NowBoard);
```

求解 NowBoard，返回解后 Board。这里计划采用 Dancing Link X 算法求解。如果无解，返回的 Board 的 now[0][0] 为-1。

1.4.3.6 complete_game.cpp

```
bool Complete_game(void);
```

一键完成游戏。成功返回 1，失败返回 0。无解则失败。

1.4.3.7 copy_to_paste.cpp

```
bool Copy_to_paste(void);
```

将当前游戏复制到剪切板，成功返回 1，失败返回 0。一般不会失败。

1.4.3.8 save_game.cpp

```
bool Save_board(void);
```

将当前游戏保存到存档位中。成功返回 1，失败返回 0。存档不能正确打开则失败。

存档文件保存在 sudoku_data/saves 中。格式为 81 / 162 个为一组。(即为 Board_to_string) 的格式。

1.4.4 print

1.4.4.1 print_board.cpp

```
void Print_board(void);
```

初始化输出整个棋盘框架。(重刷)

包括边界、初始数字、光标。

```
void Delete_cursor(int x, int y);
```

删除 (x, y) 处的光标。(用空白覆盖)

```
void Add_cursor(int x, int y);
```

在 (x, y) 处增添光标。

```
void Modify_number(int x, int y, int val);
```

将 (x, y) 处的数字改为 val (覆盖)。

1.4.4.2 print_hint.cpp

```
void Print_hint(int x, int y);
```

在提示区打印 (x, y) 的提示。

1.4.4.3 print_message.cpp

```
void Print_message(std::string s, int color);
```

在通知区打印通知。

1.4.5 run_game.cpp

```
void Run_game(void);
```

游戏的主体运行逻辑。

首先调用 Init_board，然后 Print_board。接下来不断调用 Get_key 等，然后根据操作分别调用 operation 中不同的函数。每次操作后，及时打印到屏幕。

当对棋盘做出改变时，要及时写入文件（调用 Save_Board 函数）

1.5 setting

1.5.1 setting_menu.cpp

```
Setting_info g_setting_info;
```

保存配置信息的全局变量。

```
void Setting_menu(void);
```

显示设置菜单。需要有"Difficulty", "Hint", "Keys", "Go back" 等几个选项。在每次调整后，自动写入文件（调用 Load_setting）

1.5.2 difficulty.cpp

```
void Modify_difficulty(int val);
```

将难度调整为 val。

1.5.3 open_hint.cpp

```
void Modify_open_hint(int val);
```

将提示开启与否调整为 val。

1.5.4 key_board_menu.cpp

```
void Key_board_menu(void);
```

快捷键菜单。需要提供 KEY_TYPE_SIZE 种快捷键的更改操作。注意大小写字母都转化为大写字母。

```
bool Check_key(Key_type kt, Key_value key);
```

判断 kt 键位的值是否为 key。

1.5.5 key_board.cpp

```
void Modify_key_board(Key_type kt, Key_value key);
```

将 kt 键位的值改为 key。

1.5.6 settingfile.cpp

```
void Load_setting(void);
```

从 sudoku_data/setting 文件中读取，并储存到 setting_info 中。

```
void Save_setting(void);
```

将 setting_info 的内容发送到 sudoku_data/setting 中。

1.6 start.cpp

```
void Start_menu(void);
```

开始菜单。提供 State Game（调用 Generate_menu），Settings（调用 Setting_menu），Exit 等功能。

1.7 main.cpp

```
int main(void);
```

主函数。是整个程序的入口。

首先先调用 Load_setting() 读取配置，然后调用 Start_menu()，进入开始页面。