**程序设计基础大作业**-**作业报告**

---by Group#44 软件33 2023012169陈禹默 计34 2023010789张轶程

**概述**

本项目实现了在Windows控制台中模拟机器人完成指令的游戏，也包含了 许多的扩展功能。

**适用运行环境**

本项目的需求运行环境为Windows系统 ，且要求控制台支持ANSI转义序 列，这是由于使用了 <windows.h> 以及转义序列实现的彩色输出和定位导 致的。此外，如果需要重新编译检验，那么对于编译器的标准要求至少为 C++17 ，这是由于使用 <filesystem> 库导致的。

为了方便测试时确认环境正常，这里我们给出推荐的测试配置：

系统——Windows10及以上

编译器——

|  |
| --- |
| gcc version 13.2.0 (MinGW-W64 x86\_64-ucrt-posix-seh, built by Brecht Sanders) |

特别地，项目中已经包含了经 -static 编译后的.exe文件，可以在不编译的 情况下在正常的计算机上正常运行。如需再次编译，我们推荐使用已给的 compile\_all\_at\_once.cpp 或 compile\_all\_at\_once.exe 进行编译运行， 该 程 序 会 将 同 目 录 下 的 launcher.cpp ， game\_loader.cpp ， level\_chooser.cpp ， level\_remover.cpp ， main.cpp ， customize.cpp ， cuslevel\_chooser.cpp ， ID\_remover.cpp 一并编译， 使用的编译命令如下

|  |
| --- |
| g++ $file -std=c++17 -o $file\_base\_name -static |

 **注：启动游戏时请务必从**launcher.exe**进入！**

**设计思路**

项目的设计灵感主要来源于Human Resource Machine这款另类解谜游戏以 及大作业要求说明。**在项目的结构方面** ，我们主要采用了高内聚低耦合的 设计思路，将项目分成前端以及后端。前端主要处理启动器部分，实现关 卡选择，不同存档的创建、记录，以及临时文件的分配更改等操作。而后 端则主要关注于核心游戏部分，力求仅针对某一次的游戏需求服务，将前 后端明确分离。分多个.cpp文件实现分块的功能，同时通过结构体保存多次 重复的元素如数字块，极大地降低了调试难度，增强了代码可读性，使得 项目条理清晰明了。**在项目的内容方面** ，我们主要结合了原游戏与大作业 要求，设计了美观的动画窗口与方便实用的文件输入、键盘输入以及倍速 调整与单步调试等功能，力求功能完备齐全，尽量降低用户的使用难度。

**项目结构**

项目结构的设计思路已给出。下面，我们将对项目中的文件进行说明。

项目的核心功能由 .exe 文件实现：

 launcher.exe ：游戏启动器，作为整个游戏的**唯一合法入口**

 main.exe ：游戏主体，针对一次已确定配置的游戏提供服务，完成 一轮游戏

 game\_loader.exe ：存档读取器，将选择进入的存档读入并启动选关

 level\_chooser.exe ：选关器，实现主游戏的选关

 ID\_remover.exe ：存档删除器，实现对存档的选择式删除

 customize.exe ：自定义游戏入口，导引玩家进行自定义操作进而完 成自定义游戏

 cuslevel\_chooser.exe ：自定义选关，导引玩家选则已创建的关卡

其对应的源代码为：

 同名 .cpp 文件

·header.h: 用于存放多次反复调用的位置控制、数据转换函数以及

较为通用的结构体定义以及常量定义

**小组分工**

(名字的排序不分先后，按照拼音的字典序)

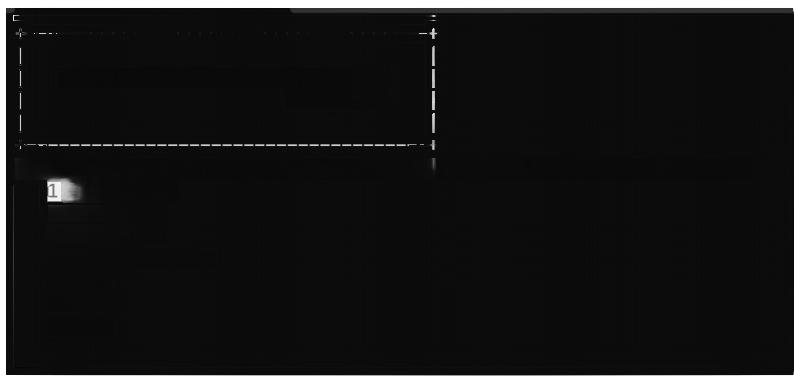
陈禹默：编写以Launcher.cpp为代表的前端代码，进行部分调试与代码重 构，合并二人代码，参与编写 final\_judge.cpp, 测试项目

张轶程：编写 main.cpp 以及header.h 大部分，绘制了outline和机器人并 设定好其动画效果，参与编写 final\_judge.cpp , 测试项目

**整体游戏界面设计**

·启动界面 采用光标控制的方式，通过WS或UP DOWN叠加ENTER

的方式实现模块的选择与确认



**Human Resource Machine Laucher**

Ver.1.0.0

press W(or UP)and S(or DOWN)to choose,and press Enter to confirm.

.Start Game

2.Create ID

3.Delete TD

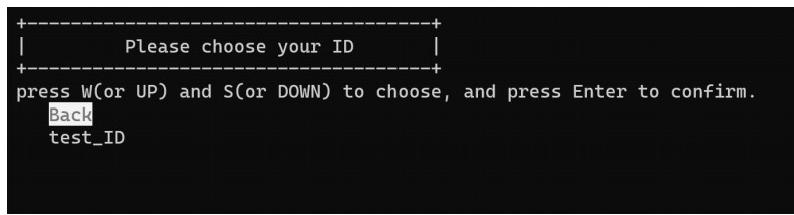
4.Customize Game

5.Help

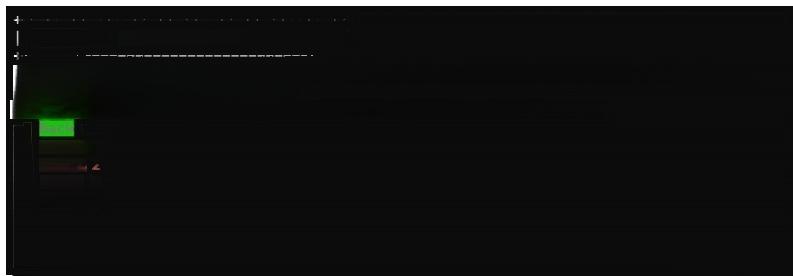
6.About

7.Quit

**·ID 选择界面**



**·关卡选择界面**

Level

Level

press

CHOOSE A LEVEL!

x means that it is completed,while Level

x

x is your current progress.

W(or UP)and S(or DOWN)to choose,and

press

means that it is not

Enter to confirm.

completed yet.

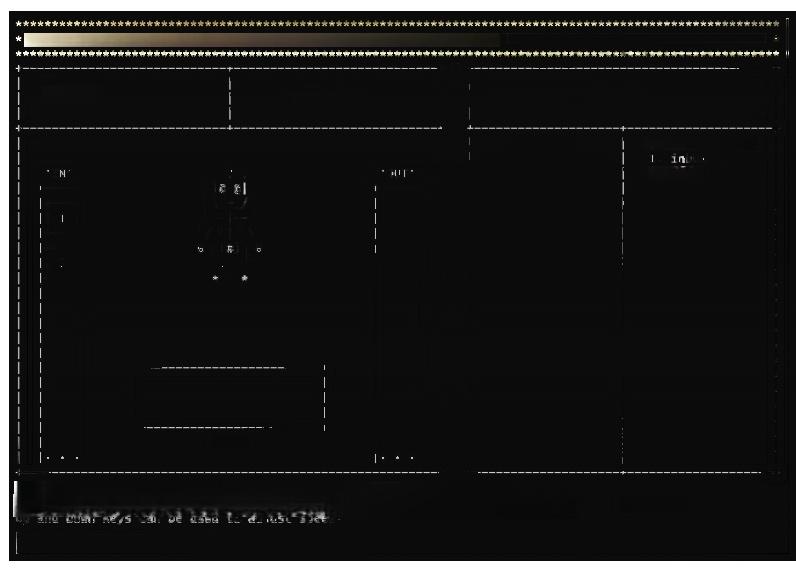
Level 1

Level

Level 3

Level 4

**·游戏界面**



For each one number,directly output it into the output sequence.

(1/3)

Valid Operations 2.outbox

SPACE

Press left and right keys to select a panel.Press Enterto confirm.

Speed:1X

If your code is ready,press Is to start!You can also press Esc to go

Mail Room

Your Code

Level

Command

Panel

back.

以上是主要的界面内容，此外，还有许多其他界面也有类似的设计思路。 可以看出，界面主要注重简洁性与简易性，以及用户操作的便捷性，实现 了控制台游戏中较友好的操作界面。

**功能介绍**

除大作业要求中的基础功能外，我们还实现了如下功能：

·**倍速功能：** 玩家可以在游戏中通过个键自由调节游戏运行时的倍速

·**单步调试功能：** 玩家可以在尝试将倍速调整为“0x”时进入单步调试模

式，使得游戏的运行以单步调试方式进行

·**多代码面板功能：**参照原游戏，我们认为多代码面板功能可以改善游 戏体验，因而允许每一关每存档保存三份代码

 **存档功能**：玩家可以创建多个存档，并保存各自的进度进行游玩

 **自定义关卡功能** ：玩家可以通过自行编写关卡配置文件的方式导入新 关卡游玩

除了以上的游戏功能外，在用户友好性方面，我们额外添加了如下功能：

 **更好的界面** ：我们实现了一套简易的UI以及光标控制功能，方便了用 户的使用

 **全屏窗口** ：滚屏时控制台游戏不可避免的问题，为了部分地解决这个 问题，我们设定了窗口为全屏尽量避免滚屏

 **完善的输入查错** ：我们在遵循大作业要求“对于未运行到的指令不做 异常情况判断” 的前提条件下进行了用户的输入判断，此外，我们还 对文件路径输入进行了判错，并给出了错误原因。

**功能的代码实现逻辑**

 下面先对**光标选择**的实现进行介绍

|  |
| --- |
| // in header.h  /\* 关于颜色以及明亮度的宏定义，可用于ANSI转义序列的构造 \*/  #define BLACK 30  #define RED 31  #define GREEN 32  #define YELLOW 33  #define BLUE 34  #define MAGENTA 35  #define CYAN 36  #define WHITE 37  #define FORE\_BRIGHT (0b01)  #define FORE\_NORMAL (0b00)  #define BACK\_BRIGHT (0b10)  #define BACK\_NORMAL (0b00)  void set\_xy(int x, int y) { // 设置光标位置  SetConsoleCursorPosition(GetStdHandle(STD\_OUTPUT\_HANDLE),  (COORD){(short)x, (short)y});  }  void hide\_cursor() // 隐藏光标 |

{

HANDLE h\_GAME =GetStdHandle(STD\_OUTPUT\_HANDLE);

CONSOLE\_CURSOR\_INFO cursor\_info;

GetConsoleCursorInfo(h\_GAME, &cursor\_info);

cursor\_info.bVisible=false;

SetConsoleCursorInfo(h\_GAME, &cursor\_info);

}

void show\_cursor() // 显示光标

{

HANDLE h\_GAME =GetStdHandle(STD\_OUTPUT\_HANDLE);

CONSOLE\_CURSOR\_INFO cursor\_info;

GetConsoleCursorInfo(h\_GAME, &cursor\_info);

cursor\_info.bVisible=true;

SetConsoleCursorInfo(h\_GAME, &cursor\_info);

}

bool is\_pressing(int v kval) { // 获取键盘物理信息

return bool(GetAsyncKeyState(vkval));

}

void clear\_buffer() { // 清空缓冲区，减少误触

cin.ignore(numeric\_limits <streamsize>::max(), '\n'); }

// 转换字符串颜色

string convert(string str, int foreground\_color, int

background\_color, int is\_bright) {

string cmd = "\033[" + to\_string(foreground\_color + 60 \* (is\_bright & 0b01)) + "m" +

"\033[" + to\_string(background\_color + 60 \* (is\_bright & 0b10 ? 1 : 0) + 10) + "m" +

str +

"\033[0m";

return cmd;

}

string convert(string str, int color, int is\_bright) {

string cmd = "\033[" + to\_string(color + 60 \* is\_bright)

" "

+ m +

str +

"\033[0m";

return cmd;

|  |
| --- |
| }  struct Query { // “询问”结构体，主要记录一个询问窗口的信息 vector <string> choice; // 总共的选择内容  string title; // 选择窗口的标题信息  int strow, stcol; // 开始显示选项的起始横纵坐标，方便控制位置 }; |

如上代码是实现该操作的基本需求 ，下面 ，我以内容相对简单的 ID\_remover.cpp 的部分进行具体应用的解释

|  |
| --- |
| Query Remove({  {  "Back"  },  "Choose an ID to delete.\n",  2, 3  }); // 在此情形下的Query  void restore(Query &x, int row) { // 复原第row行的高亮状态 set\_xy(x.stcol, row);  int p = row - x.strow;  cout << convert(x.choice[p], WHITE, FORE\_NORMAL); }  void highlight(Query &x, int row) { // 使第row行高亮 set\_xy(x.stcol, row);  int p = row - x.strow;  cout << convert(x.choice[p], (p ? WHITE : BLACK), (p ? RED : WHITE), FORE\_BRIGHT | BACK\_BRIGHT); // 这里使用位运算实现对不同 状态同时要求的拼接  }  void init(Query &x) { // 实现对x的初始化，开始选择  hide\_cursor(); // 隐藏光标，增强美观性  system("cls"); // 清屏，确保没有其他字符干扰  cout << x.title;  cout << "press W(or UP) and S(or DOWN) to choose, and press Enter to confirm." << endl;  // 首先输出窗口标题（提示信息） |

for(int row = x.strow; row <= x.strow +

int(x.choice.size()) - 1; row++) {

restore(x, row); // 先将每一行以复原的形式输出

}

set\_xy(x.stcol, x.strow);

highlight(x, x.strow); // 高亮显示第一行

}

/\* 该函数用于处理第x条被选择后的执行 \*/

bool tackle(int x) {

while(kbhit() && is\_pressing(VK\_RETURN)); // 防止误判，确保 用户已抬起Enter键

clear\_buffer(); // 清空Enter带来的缓冲区堆积

system("cls");

if(x == 0) { // Back

return true;

}

else { // 实现删除操作（实际例子）

cout << "Are you sure you want to delete \"" << Remove.choice[x] << "\"? (Y/N)" << endl;

string str;

while(true) {

getline(cin, str);

if(str.length() != 1 || (str[0] != 'Y' && str[0] !=

'N')) {

system("cls");

cout << "Please enter Y/N:" << endl;

continue;

}

break;

}

if(str[0] == 'Y') {

fs::path pth("./identity/" + Remove.choice[x] +

".id");

fs::remove(pth);

Remove.choice.erase(Remove.choice.begin() + x); }

}

return false;

于是，只需要在main()中不断检测用户的键盘输入，即可通过调用上述函 数实现选择。

 **选择关卡**

关卡选择主要是通过如上的光标输入结合文件的写入实现的，这样也自然 避免了非法输入。 level\_chooser.exe 会将用户的选择（包括存档、关卡 号）通过 level.dat 和 code.cd 的文件形式存放在 ./temp 下，进而供主程 序使用。

|  |
| --- |
| create\_file("temp/level.dat");  copy\_file("level\_info/level" + to\_string(x) + ".dat", "temp/level.dat");  // 将已有的关卡配置文件复制一份作为level.dat  of stream fout("./temp/code.cd", ios::out);  for(int i = 0; i < 3; i++) {  Code &tmp = cd[x][i];  fout << tmp.lines << endl;  for(const auto &j : tmp.cmds) {  fout << j.label;  if(j.label == -2) { // wrong input  fout << ' ' << j.content << endl;  }  else if(j.label >= 2) { // with opnum  fout << ' ' << j.num << endl;  }  else { // without opnum  fout << endl;  }  }  fout << endl;  }  fout.close();  // 输入初始（可能已有过的）代码  system("main.exe"); // 启动游戏  // 在main.exe执行完毕后，恰好可以继续下列代码的执行，即实现了游戏结 束后返回选关界面 |

上述代码展示了这种文件的迁移的实现。

 **游戏界面**

游戏界面主要由outline等的绘制以及文件读入，定点输出信息实现。

|  |
| --- |
| // outline 与 robot 的图形详见header.h，考虑到报告的精简性这里予以 省略  string posit(string str,int x,int y){ // 设置坐标与输出字符串二合 一，方遍大量更改位置时使用  string out = "\033[" + to\_string(x + 1) + ";" + to\_string(y + 1) + "H" + str + "\n";  return out;  } |

|  |
| --- |
| full\_screen(); // 全屏，防止滚屏  hide\_cursor(); // 隐藏光标  Sleep(100); // 等待全屏完成，防止没有完成全屏就输出仍然导致滚屏  block\_on\_hand.num\_hide();  system("cls");  getdata(); // read level.dat  getfileop(cd); // read code.cd  system("cls");  basic\_output(); // 界面输出  for(int i = 0; i < spacenum; i++){  space[i].set(spapos.x, int(spapos.y - (spacenum - 1) \*2.5 + i \* 5));  space\_loc.push\_back((pos){spapos.x - 7, int(spapos.y - (spacenum - 1) \*2.5 + i \* 5) - 2});  }  for(int i = 0; i < innum; i++){  block a;  a.write(input\_line[i]);  in.push\_back(a);  }  //此时in是block容器，tmp.out是输出序列的答案  show\_space();  show\_file\_op(-1, youpos, cd[cmd\_panel].cmds, notelines); |

|  |
| --- |
| init\_set();  show\_inbox();  show\_outbox();  robot.change\_status(Norm);  robot.set(init\_pos.x + notelines, init\_pos.y);  robot.show(); |

通过 basic\_output() 函数 ，我们可以实现输出基本游戏框架 ，再加上 show\_space() show\_inbox() show\_outbox() robot\_show() show\_file\_op() 实现对于其余信息的显示，进而完成游戏界面。

如果用户在正式进入之前选择提供了代码文件 ，那么代码会保存 于 ./temp/code.cd 中被解析读出。无论用户如何选择，我们都允许对代码 做进一步的修改。

|  |
| --- |
| if(kbhit() && is\_pressing(VK\_RETURN) && !is\_pressing(VK\_LEFT)  && !is\_pressing(VK\_RIGHT)) {  Sleep(10);  if(!kbhit() || !is\_pressing(VK\_RETURN)) continue; clear\_buffer();  set\_xy(0, newoppos.x + notelines);  cout << "  " << endl <<  "  " << endl <<  "  " << endl; start();  set\_xy(0, newoppos.x + notelines);  cout << instruction;  show\_spd(speedx, notelines);  while(kbhit());  } |

我们通过如上代码判定 用户是否想编辑代码 ，如确认Enter ，则调 用 start() 函数开始编辑命令的读入，进而使得用户可以自由修改代码。

机器人执行动画部分主要由 simulate() 函数完成，下面是对 simulate() 部分的解释：

//simulte() 返回值类型为int，结合宏定义表示该行代码的运行结果是如何 if(compute\_times >= 200) return ENDLESS\_LOOP; // 死循环判定

int &label = cd[cmd\_panel].cmds[pointer].label;

int &num = cd[cmd\_panel].cmds[pointer].num;

if(label >= 0 && limit[label] == false) { // 并非当前关可用代码

return pointer;

}

if(label == 0)

...

// 对于label代表的代码一一实现相对应操作，这里不再赘述

而机器人、数据块的移动等操作是通过 block 类以及 man 类实现的，下面 以 block 为例进行简要说明

|  |
| --- |
| class block {  public:  block(){ // 构造函数  number = 0;  block\_pos.x = 3;  block\_pos.y = 3;  appear = true;  }  block(int num){ // 构造函数  number = num;  block\_pos.x = 3;  block\_pos.y = 3;  appear = true;  }  void num\_appear(){ // 使数字设定在显示状态  appear = true;  }  void num\_hide(){ // 使数字设定在隐藏状态  appear = false;  }  bool available(){ // 询问可访问性，即是否可用  return appear;  }  void set(int x,int y){ // 设定数据块坐标  block\_pos.x = x; |

|  |
| --- |
| block\_pos.y = y;  }  void up(){...} // 上移动画  void down(){...} // 下移动画  void write(int num){...} // 更改数字值  void show(){...} // 显示  void hide(){...} // 隐藏  void erase(){...} // 全部擦除，一般用在手上拿的物块  void move(int to){...} // to存放了一个方向数据，根据to的值实现 单步的移动  int read(){...} // 返回数据块上的值  private:  int number;  pos block\_pos;  bool appear; //是否可见  }; |

可以看出，通过定义 block 类，程序很好地实现了对物块操作的封装，并 且使得物块功能、位置的修改变得更加得心应手。

 **机器人的指令集**

这一部分由 level.dat 实现，关卡数据文件会告知 main.exe 给出怎样的指 令集，并通过 simulate() 函数一步步执行，这里不再赘述

 **异常情况处理**

异常情况包括不在指令表中的未定义指令，不属于当前关卡固定的可用指 令集，不符合指令表规定的指令使用（如操作数非整数、指令特定的错误 情况、指令后面的操作数数量与要求不符） 。指令非法判定由 simulate() 函数完成。而对于未执行到的指令不予判断这一功能的实现则是由输入时 给予标记实现的，这里对输入的分析函数进行简述：

|  |
| --- |
| Console\_Cmd analyze\_console\_input(string &str) { // 返回包装好的 Console\_Cmd类，表示一条解析过后的指令  // 实现了对于add, ins, del三种命令编辑指令的解析  const Console\_Cmd FAIL = (Console\_Cmd){-1, -1, -1, -1, ""};  // 编辑指令有误，无效指令  for(int i = 0; i < (int)str.size(); i++) if(str[i] == '$') return FAIL; // 如果输入包含非法字符（已被文件输入时的空格顶替符占 用） ，则不可接受 |

stringstream sstream(str); // 利用sstream实现对整行字符串的拆 解

vector <string> v;

while(sstream) {

v.push\_back("");

sstream >> v[v.size() - 1];

}

if(v.size() <= 2) return FAIL;

string y = "";

for(int i = 2; i < (int)v.size(); i++) {

y += v[i];

if(i != (int)v.size() - 1) y +=

' '

;

}

string x = v[1] + ' ' + y;

if(v[0] != "add" && v[0] != "ins" && v[0] != "del") return FAIL; // 操作指令错误，无法操作

if(v[0] == "add" && (v.size() != 4 && v.size() != 3))

return (Console\_Cmd){0, -2, -2, -1, x}; // 这里的意思是确认指令不 可能正确，因而标记指令为非法并仅保留其字符串形式，使得其能够显示但是

一经执行必然报错

if(v[0] == "ins" && !is\_opnum(v[1])) return FAIL; if(v[0] == "del" && v.size() != 3) return FAIL; if(v[0] == "del") {

if(!is\_opnum(v[1])) return FAIL;

return (Console\_Cmd){2, atoi(v[1].c\_str()), -1, -1, x}; }

else {

if(v[0] == "add") {

int op = is\_cmd name(v[1]);

if(op == -1 || (op >= 2 && !is\_opnum(v[2])) || (op < 2 && v[2].size() != 0)) return (Console\_Cmd){0, -2, -2, -1, x};

return (Console\_Cmd){0, op, (v[2].size() == 0 ? -1 : atoi(v[2].c\_str())), -1, x};

}

if(v[0] == "ins") {

//if(!is\_opnum(v[1])) return FAIL;

int line = atoi(v[1].c\_str());

int op = is\_cmd name(v[2]);

|  |
| --- |
| if(op == -1 || (op >= 2 && !is\_opnum(v[3])) || (op < 2 && v[3].size() != 0)) return (Console\_Cmd){1, line, -2, -1, y};  return (Console\_Cmd){1, line, op, (v[3].size() == 0 ? -1 : atoi(v[3].c\_str())), y};  }  }  return FAIL;  } |

利用上述函数 ，结合用户输入的命令 ，我们可以实现对于非法指令的保 存、记录，并使其在且尽在被执行时产生错误。

 **正确性检测**

这一功能由两部分组成 ：正确性检测与统计信息的输出。下面分别对两个 功能的实现做解释

bool test\_output(){ // 输出是否和标答一致

if(outnum != (int)out.size())

return false; // 输出长度与标答不一致，显然错误 for(int i = 0; i < (int)out.size(); i++){

if(out[i].read() != expected\_out[i])

return false; // 有一个不正确即为不正确

}

return true; // 全部正确

}

可见 test\_output() 函数实现了输出与答案的对比，并判断正误。

|  |  |
| --- | --- |
| int compute\_times;  int simulate(){...}  // simulate()的过程中统计计算次数  void terminate\_output(string x, bool pass) { // 结束游戏输出信息 | |
| set\_xy(term\_pos.y, term\_pos.x + notelines);  cout << "  set\_xy(term\_pos.y, term\_pos.x + notelines);  cout << x << endl; // 关卡结束状态，Success，Fail， instruction X三种 | ";  Error on |

|  |
| --- |
| if(pass) { // 如果通关，则比较代码行数与计算次数  int lines = cd[cmd\_panel].cmds.size();  if(line\_limit != -1) {  cout << "Lines of Codes: " << lines <<  "\t\tRequirement: "  << convert(to\_string(line\_limit), lines <= line\_limit ? GREEN : RED, FORE\_BRIGHT) << endl;  //根据是否达到最高标准选择合适的颜色显示  }  if(compute\_limit != -1) {  cout << "Times of Computing: " << compute\_times << "\t\tRequirement: "  << convert(to\_string(compute\_limit), compute\_times <= compute\_limit ? GREEN : RED, FORE\_BRIGHT) << endl;  }  }  update\_code\_file(pass); // 根据结果返回信息进而更新文件 system("pause"); // 等待用户确认以结束  exit(0); // 彻底结束程序  } |

程序在开始时刻确认了代码的行数 ，并在执行的过程中记录了运算的次 数，进而在正确性检验过后输出统计信息。

 OJ**自动化测试**

考 虑 到 simulate() 函 数 实 在 难 以 分 离 ， 因 此 我 们 特 地 另 写 final\_judge.cpp 以用于OJ测试，其中也由多组随机生成数据的功能方便 我们自行检验程序的正确性。

以上为基础功能的实现，下面我们对部分扩展功能的实现做解释

 **倍速功能**

倍速功能通过按键←→ 调节，可选倍速为1x到64x ，并且附加单步调试功 能。具体实现主要为数学计算。

|  |
| --- |
| void show\_spd(int spd, int shiftx) { // 显示倍速  set\_xy(spd\_pos.y, spd\_pos.x + shiftx);  cout << " "; |

|  |
| --- |
| set\_xy(spd\_pos.y, spd\_pos.x + shiftx);  if(spd != 0)  cout << convert("Speed: " + to\_string(spd) + 'X', BLUE, FORE\_BRIGHT);  else // 0x被调节为单步调试  cout << convert("Speed: Single Step Mode", YELLOW, FORE\_BRIGHT);  }  int simulate(){  ...  Sleep(speedx == 0 ? 1 : SHORT\_WAIT / speedx); // 特判是否为 单步调试，不是，则进行数学计算，算出间隔时间  ...  } |

特别地，在 speedx 为0时，程序会在主函数执行完一次 simulate() 后执行 system("pause") 等待用户确认后再进入下一步命令

 **自定义关卡**

该功能 的实现主要来 自于 customize.exe ， cuslevel\_chooser.exe ， level\_remover.exe 。关于用户配置文件的格式要求，详见大作业文件夹 下的 customize\_tutorial.txt 。下面主要对核心的解析用户给出的配置文 件部分解释：

|  |
| --- |
| create\_file("./customize\_level/" + name + ".dat");  if stream fin(str); // 从用户给出的路径出发（由于C++相关函数的限制， 无法支持中文路径！建议使用 ../等方式从当前目录导引）  of stream fout("./customize\_level/" + name + ".dat"); // 将转化 后的配置文件输出至custom\_name.dat中  fout << -1 << endl; // customize number  getline(fin, name);  fout << name << endl; // level name  string note;  getline(fin, note);  fout << note << endl;  int innum; fin >> innum; fout << innum << ' '; // 输入序列 |

for(int i = 1; i <= innum; i++) {

int x;

fin >> x;

fout << x <<

' '

;

}

fout << endl;

int outnum; fin >> outnum; fout << outnum << ' '; // 输出序列 for(int i = 1; i <= outnum; i++) {

int x;

fin >> x;

fout << x <<

' '

;

}

fout << endl;

int spacenum; fin >> spacenum; fout << spacenum << ' '; // 空地 个数

for(int i = 1; i <= spacenum; i++) {

string x;

fin >> x;

if(x == "blank") fout << 114514

'

'

<<

;

else fout << atoi(x.c\_str()) <<

' '

;

}

fout << endl;

int cmd num; fin >> cmd num; fout << cmd num << ' '; // 指令集基数 for(int i = 1; i <= cmd num; i++) {

string cmd;

fin >> cmd;

fout << is\_cmd name(cmd) <<

' '

;

}

fout << endl;

int line\_limit, compute\_limit; // 代码行数限制以及计算次数限制 fin >> line\_limit >> compute\_limit;

fout << line\_limit << ' ' << compute\_limit << endl;

fin.close(); fout.close();



|  |
| --- |
| cout <<'\"'<<convert(name,WHITE,BLUE,BACK\_BRIGHT |  FORE\_BRIGHT)<<'\"'<<"has been successfully created as a new level!"<<endl; |

**·帮助选项以及帮助文档**

考虑到游戏的用户友好性，我们在launcher 中设置了Help选项，该选项会 打开HELP.pdf ,辅助用户了解有关该游戏的玩法机制等。此外，我们也提 供上述pdf的 .md版本，方便有条件的玩家阅读。并且，对于自定义关卡， 我们也提供了 customize\_tutorial.txt 用于帮助用户理解自创关卡方法。

以下是HELP.pdf的部分截图

**Human Resource Machine帮助文档**

*o°*







**Tomorrow Corporation**

**声明：原游戏为Tomorrow Corporation开发的另类解密游戏，大作业名称借用了其名字**

**简** **介**

这是一款杂糅编程元素的解密游戏，你将操控主人公(在大作业中是机器人)进行一次次的工作，不断 通关升级。在每一关中，你将有不同的语句、空地可以使用。同时，每一关还有代码行数与运算次数的 挑战要求，尝试达到最优吧!

**主游戏部分**

**ID意义下的游戏**

游戏设计了ID系统，以保存不同的存档，你的全部代码以及主游戏通关记录都将被保存其中。你可以随

**自由创新关卡**

这里分别以大作业要求的Level 4与自定义关卡可支持的自创关卡Multiply 为例进行介绍

level4.dat

|  |
| --- |
| 4  8x Intensifier  For each number X in the input sequence, output 8X.  10 1 -2 3 6 9 10 -11 -12 5 8  10 8 -16 24 48 72 80 -88 -96 40 64  1 114514  6 0 1 5 4 2 6  9 90 |

该内置自创关要求玩家对于每一个输入序列上的数X输出它的8倍。具体通 关方法采用的是自身相加代替\*2操作，进而通过复制结果降低计算次数， 完美通关思路类似快速幂，适合Level 4的位置。具体测试过程详见视频。

multiply.txt （用户端配置的文件）

Multiply

For each two numbers X and Y in sequence, output X \* Y.

6 1 2 3 4 6 5

3 2 12 30

5 blank blank blank 0 1

8 inbox outbox add sub copyto copyfrom jump jumpifzero

-1 -1

本关在已给空地0和1的基础之上要求玩家实现两数相乘。具体做法为将X累 加Y次输出，其中巧妙借助了已给的1与jumpifzero配合。

**游戏测试**

**云盘链接如下：**

|  |
| --- |
| <https://cloud.tsinghua.edu.cn/d/9a4aa91f6db140e2a04e/> |

鉴于报告的形式难以呈现许多动态过程的测试结果，故在这里只针对大多 数静态过程做测试，并给出测试结果

 **代码的文件输入**

以Level 4为例，测试./ans/lvl4.ans中的代码是否能正常导入

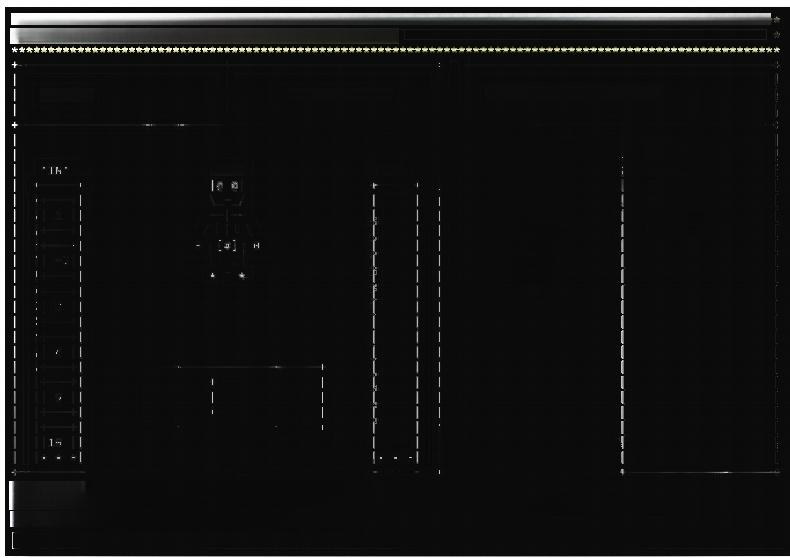
lvl4.ans

|  |
| --- |
| inbox  copyto 0  add 0  copyto 0  add 0  copyto 0  add 0  outbox  jump 1 |

导入结果

4.copyto

g.jump 1

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*For each number X in the input sequence,output 8X.

Level

4

8x Intensifier

Command Panel(1/3)

???

^OUT^

Your Code

..o θ

3.addθ

4. copyto

cto θ θ

7.add e

8.outbox

Valid Operations

1.inbox

2.outbox

3.add

5.copyfron

6.jump

SPACE

Press left and right keys to select a panel.Press Enter to If your code is ready,press 'S'to start!You can also press Up and Down Keys can be used to adjust speed.

confirm.

Esc to go back

Speed:1X

**·自创关卡配置文件的导入**

以 样 例 自 定 义 关 Multiply 为 例 ， 测 试

C:\test\_dir1\random\_dir2\unimportantdir3\config.txt 下配置文件的

导入

config.txt

|  |
| --- |
| Multiply  For each two numbers X and Y in sequence,output X\*Y.  6123465  321230  5 blank blank blank 01  8 inbox outbox add sub copyto copyfrom jump jumpifzero  -1-1 |

**导入后读取结果**



\*For each two numbers X and Y in sequence,output X\*Y.

(1/3)

Valid Operations

copyto

6.copyfrom

8.jumpif

SPAO CE

PIfr dni rreigahdtys 'e'ttt!pYal.ssaltes osnrtmo. go back.

Up and Down Keys can be used to adjust speed.

Level Customized

Your Code

Multiply

Speed:X

Command

Panel

**·存档新建时输入存档名**

例如已经创建了一个ID, 名 称 为test\_ID, 测试重名与相似名的输入

输入1

|  |
| --- |
| test\_ID |

输出1

|  |
| --- |
| You have already created this identity,try again: |

输入2

|  |
| --- |
| test ID |

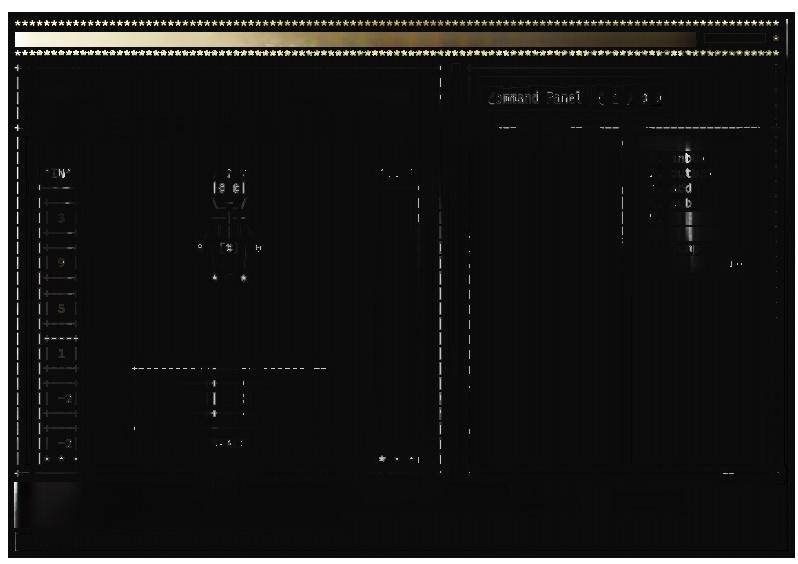
输出2

|  |
| --- |
| "test ID"has been successfully created as your new identity!  Press any key to go back! |

可见ID输入部分具有鲁棒性

**·进入游戏后使用代码编辑命令**

**以第二关为例，测试多面板下输入各种指令，下图为初始状态**

\*For each two numbers X and Y in the input sequence,output two numbers,Cx-Y)and (Y-x).

Level 2

Hall of Addition

Your Code

Valid Operations

copyto

6. copyfrom 8. jumpifze

Press left and right keys to select a panel.Press Enter to confirm.

If your code is ready,press 'S'to start!You can also press Esc to go back. Speed:1

Up and Down Keyscan be used to adjust speed.

输入序列(每一行代表一个输入，空行代表直接回车)

|  |
| --- |
| new inbox  new inbox  ins 1 copyto 0  new Copyto -1  del 4  new Copyto 1  new copyfrom 0  ins -1 outbox  n e w in box  del -1  ins 10 jumpifzero 0.5  new add sub  new jumpifzero 28.632 |

直接反馈(每一行对应输入的每一行)

|  |  |
| --- | --- |
| Successfully Successfully Successfully | Executed!  Executed!  Executed! |

Successfully Executed!

Successfully Executed!

Successfully Executed!

Successfully Executed!

Invalid Command!

Invalid Command!

Invalid Command!

Invalid Command!

Invalid Command!

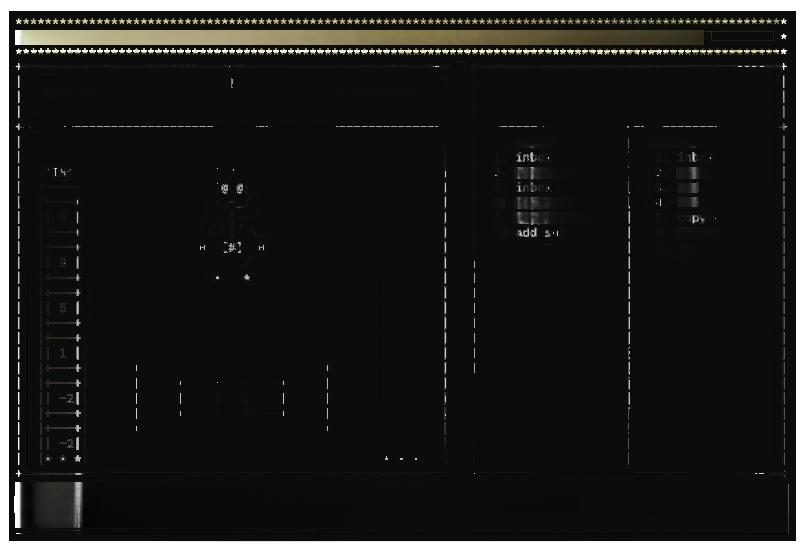
Successfully Executed!

Invalid Command!

是对代码本身做了语法判定)

**(原因是代码过长，认为不合法，而不**

**最终效果**



\*For each two numbers X and Y in the input sequence,output two numbers,CX-Y)and CY-X).

Hall of Addition

Valid Operations

outbox add

sub

6. copyfrom 7. jump

8.jumpifzero

SPACE

Press left and right keys to select a panel.Press Enter to confirm.

If your code is ready,press IS¹to start!You can also press Esc to go back. Speed:1X

Your Code

copyto 0

4. copyto 1

copyfrom e

Up and Down Keys can be used to adjust speed.

Panel(1/3)

Command

^OUTA

Level

6.

?

2

这说明，代码编辑功能是基本完善的

**·关卡选择、运行动画等其他操作**

由于游戏的许多功能是基于选择实现而非打字输入实现，因此关卡选择、 运行动画等其他过程将在视频中测试，报告中已经测试了所有种类的键盘 输入(基本相同的输入格式按同一类算)

**#END**