

HRM游戏制作报告

计48-经42 王鹏杰

计48-经42 高雅菁

零、注意事项

视频链接: <https://cloud.tsinghua.edu.cn/f/26a813345ebd45a9a0d5/>

1. 若使用非Mac平台运行,可能会出现字体格式不匹配现象。您可以考虑选用关键字为 `windows` 的 `ui` 文件。
2. 您可能需要对 `cmake` 文件进行适当修改来适配您的编译环境。
3. 运行前,需要将 `mainwindow.cpp` 程序第30, 235, 1597行的图片地址更新为本机机器人gif图片地址, 第761, 1086, 1240行的图片地址更新为本机烟花 `gif` 图片地址。注意: 使用相对地址可能无法正常读取。

一、游戏概述

本游戏基于Human Resource Machine进行改编, 共分为五关。前三关为固定关卡, 后两关为自由创新关卡。 **本游戏增加的主要亮点如下: ** **1. 在自由创新关卡通过添加指针、扩充地毯、创造新指令等算法与手段让用户得以较轻松地实现数组、循环的编程。 ** **2. 自由创新关卡的关卡背景是人见人爱的oj题目哼哼哈兮。 ** **3. 在用户游玩关卡时添加单步模拟、连续模拟的选项, 同时可以控制连续模拟的展示速度, 也可以跳过模拟直接展示结果。 ** **4. 在游戏界面中插入欢迎、结算页面动画, 机器人也有细节的手部动作更为美观。 **

二、设计逻辑

按照玩家真实游玩体验一步一步梳理游戏运行逻辑:

1. 关卡选择:

是否通过; 选择关卡判断; 游戏结束返回关卡选择界面

2. 进入游戏内部:

1. 显示的内容

- Level;关卡描述;可以使用的指令;可以使用的空地数目
- 键盘输入或文件输入; 然后点击 `start` 启动

2. 具体操作实现逻辑

注: 1. 数据实际存储形式为 `std::string`, 此处省略部分 `std::stoi` 及安全判定的说明

2. 由于新加入指令, 又新加入了许多判定, 在此处没有进行说明

总体思路:输入指令, 逐步执行指令, 并在执行过程中返回操作合法性, 如果所有操作都合法, 最后校验结果是否正确。

inbox: 判定 `inbox` 是否为空；如果当前执行指令为`jump`则可以为空，并结束循环。

将 `inbox` 的 `front` 赋值给`hand`；将 `inbox` 的 `front` 擦除

outbox: 判定 `hand` 是否为空；

`outbox` 中 `push_back` `hand` 的内容；`hand` 清零

copyto: 判定 `hand` 是否为空；判定 `para` 是否超范围。

将 `hand` 写入 `carpet`；`hand` 不清空。

copyfrom: 判定 `para` 是否超范围；判定`carpet`是不是空的；把 `carpet` 内容赋给 `hand`

add: 判定 `hand` 是否为空；判定 `para` 是否超范围；判定 `carpet` 是否为空；

执行 `hand+=carpet`

sub: 判定 `hand` 是否为空；判定 `para` 是否超范围；判定 `carpet` 是否为空；

执行 `hand+=carpet`

jump: 先判定那个指令是否存在；是否在`jump`之前；

将实际试行的指令的`index`修改，实现指令跳转；若 `inbox` 为空，则不执行

jumpifzero: 先判定参数是否正确；再判定指令是否存在；判定手里是否有东西；再判定 `hand` 里面是否为0，若为0，执行 `jump`

新加入指令：相关说明请参考自由创新关卡部分

模拟：输入结束之后进行逐步模拟

3. 返回SUCCESS和FAIL

三、代码工程结构(前三关)

工程框架

```
|— CmakeLists.txt
|— Header Files
|   |— game.h
|   |— humanmachine.h
|   |— mainwindow.h
|   |— nunchunk.h
|— Source Files
|   |— game.cpp
|   |— humanmachine.cpp
|   |— main.cpp
|   |— mainwindow.cpp
|   |— nunchunk.cpp
|— mainwindow.ui
|— Game Info Files
|   |— gameinfo.txt
|   |— archive.txt
|— Playgame Files
|   |— level1.txt
```

```
| | | level2.txt
| | | level3.txt
| | | level4.txt
| | | level5.txt
| | | nunchunk.txt (有更详细的解法解读)
| | Report
| | | HRM游戏制作报告.md
| | | HRM游戏制作报告.pdf
| | | md文件依赖的图片
```

1.编译运行说明

参见 `CMake` 文件，您可能需要进行适当修改以确保适配您的平台环境。

2. 程序结构及主要函数类

2.1 game

程序的核心，储存并处理游戏的所有状态和行为。

成员变量包括：初始输入箱（`vector<string> initialInbox`）、输入箱条（`inboxBar`）、输出箱条（`vector<string> outboxBar`）、地毯条（`vector<vector<string>> carpetBar`）、可用操作（`vector<string> availableOps`）、手头的数字（`string hand`）和目标状态（`vector<string> goal`），关卡描述（`string descrip`）。

Game 构造函数：

初始化游戏对象，设置初始输入箱、可用操作、目标状态、游戏描述，以及初始化地毯条的大小。

goalReached 函数：

`bool goalReached()` 目标达成判断函数，检查当前的输出箱条是否与目标状态相匹配。

isLegalOperation 函数：

`bool Game::isLegalOperation(string& command)` 操作合法性判断函数，检查给定的命令是否是可用操作之一。

updateState 函数：

展示当前游戏状态，通过 `mainwindow` 中的回调函数，将 `inboxBar`、`outboxBar`、`hand`、`carpetBar` 追加到 `mainwindow` 中的 `queue` 中

inputProcess 函数：

```
bool Game::inputProcess(string command,int param,int paramW,std::string extraParam,bool&
jumpInputJudge,bool& endRun,int& numSteps)
```

输入处理函数，根据输入的命令和参数执行相应的操作，并返回操作合法性。

- 操作合法性判断：与 `availableOps` 比较，判断操作是否可用。
- 状态更新：根据命令更新手头数字、输入箱条、输出箱条和地毯条的状态。
- 错误处理：如果命令不合法或参数不正确，返回错误。

- 循环结束控制：通过 `jumpInputJudge` 和 `endRun` 判断是否是循环中的空读入，进行循环结束判定。

playgame 函数：

```
bool Game::playgame(istream& inputStream)
```

游戏主函数，从输入流中读取输入，解析指令，并分步交由 `inputProcess` 处理，最后返回游戏胜负。

- 输入读取：从 `istream` 流输入读取一系列命令。
- 命令解析：解析命令和参数，执行相应的操作。
- 执行指令：执行每条命令，直到完成所有命令或游戏结束。
- 循环：针对循环命令类型（如 `jump` 和 `jumpifzero`）进行条件判断和循环控制。
- 游戏结束判断：检查是否达到目标状态或出现错误。

2.2 humanmachine

游戏过程中的小人 `machine` 的绘制与移动。

成员变量包括：初始位置，角度 `int inixPos, iniyPos, iniangle`；位置以及手的角度，`int xPos, yPos, handangle`；移动过程中的目标 `int aimX, aimY, aimAngle`；小人的动作参量 `int status`。

Humanmachine 构造函数

设置 `machine` 的初始位置和角度，绘制 `hand` 文字框（`textBrowser`）

moveMachine 函数

```
void moveMachine(int aimXset, int aimYset, std::string action="empty")
```

根据传入的目标值修改 `aimX` 和 `aimY`，启动计时器，通过 `void moveMachineStep` 进行逐步移动，移动到目标位置后根据 `action` 中指示的动作演示一些动作动画。

rotateHand 函数

```
void Humanmachine::rotateHand(int angle=30)
```

根据传入值修改 `aimangle`，启动计时器，通过 `void rotateHandStep` 进行逐步转动。

updateHand 函数

如果状态值为0，`handTextBrowser` 跟随手按几何学的角度旋转。

如果状态值为1，`handTextBrowser` 移到头顶。

paintEvent 函数

```
void Humanmachine::paintEvent(QPaintEvent *event)
```

，继承自 `QWidget` 的虚函数。

每当有 `update()` 或 `repaint()` 等函数被调用时，触发 `QPaintEvent`，进行绘制。

先绘制躯体和双腿，然后将躯体旋转，绘制一条手臂，再旋转回，得到倾斜的手臂。

2.3 mainwindow

游戏图形界面的核心，这里只解释跟游戏游玩与内容显示相关的核心逻辑。具体的操作逻辑在第四部分说明，文件处理相关操作在 3.文件操作 中说明。

主要成员变量为小人的指针（`Humanmachine *machine`），`Logbar`，`inboxBar`，`outboxBar`，`carpetBar` 显示的队列 `queue<string>`

槽函数 `showGame`

由选择关卡界面的关卡选择按钮被点击信号触发。

首先初始化小人 `machine`，将它放入 `playgame` 页面的 `layout` 中（如果已经存在一个小人，进行深删除，确保指针指向的所有控件都被删除）。然后在游玩页面（`playgame`）的 `Logbar` 中设置游戏说明，`inboxBar` 中设置初始输入。

槽函数 `startJudgeClicked`

由游戏游玩界面的 `START` 按钮被点击事件触发。

将 `inputPlaygameCommand` 中的内容转化为 `QString`，如果内容为空，弹窗报错。如果是正常的输入，将其转化为输入流 `istringstream`。先初始化与 `Game` 之间的回调函数，然后把输入流交由 `Game::playgame` 处理，获取胜负的返回值。

接下来初始化显示游戏过程的计时器，开始显示内容。

槽函数 `updateProcessingState`

由 `startJudgeClicked` 函数中初始化的计时器结束信号触发。

首先根据 `actionLog` 中的内容操控 `machine` 展示相应动作动画，并 `pop` `actionLog`。

接着分别将日志，输入栏，输出栏，`carpet` 栏的显示队列的内容显示在对应位置，并执行 `pop`。

当所有显示队列为空时，结束显示逻辑，清空所有显示栏，并根据胜负切换至对应页面。

`geneLevelNunchunk` 函数

负责“哼哼哈兮”关卡的随机生成。

随机树种子设置为 `time(0)`，在一定范围内生成 `m` 与 `n`。然后生成一组可行解。

生成可行解的逻辑如下：为了确保至少存在一组解能够合成这样的长度，首先构造这样一组解，只需要让前面生成的棒子长度都在0和剩下的长度之间就可以。存在一组解之后，其他的解可以完全随机生成。

接下通过生成的参数实例化 `Game` 类，并 `push` 到 `games` 矢量中。

2.4 nunchunk

接受 `MainWindow::geneLevelNunchunk` 生成的数据形成的输入流，返回解。

2.5 main

实例化 `MainWindow w` 和 `vector<Game> games`

3.文件操作：

`mainwindow` 中包含了文件读写操作，用于读取关卡信息和存档信息。

`loadLevelInfo` 函数

弹出窗口，让用户打开 `level.txt` 文件，将输入流传递到 `parseLevelInfo` 函数中解析文本文件，将解析出的关卡信息储存在许多 `vector` 中，然后根据这些信息实例化 `Game`，并 `push` 到 `games` 中。

loadArchive 函数

首先检测是否进行了 `levelInfo` 加载，如果没有加载，弹出报错信息。

如果已经加载，弹出窗口，让用户打开 `archive.txt` 文件，通过 `parseArchive` 函数解析文本文件，根据这个修改 `games` 中 `Game` 类的 `passed` 信息。

parseLevelInfo 函数

接受输入流，根据 `|` 分隔符将同一行中的信息分隔，然后找到 `;`，根据 `;` 的位置分隔 `key` 和 `value`，根据 `key` 信息储存 `value`。

parseArchiveInfo 函数

接受输入流，根据 `;` 分隔键值对，储存信息。

存档：游戏结束后（点击退出按钮 `MainWindow::buttonExitClicked()` 被调用），会将关卡完成状态写入存档文件 `archive.txt` 中。

四、游戏界面设计

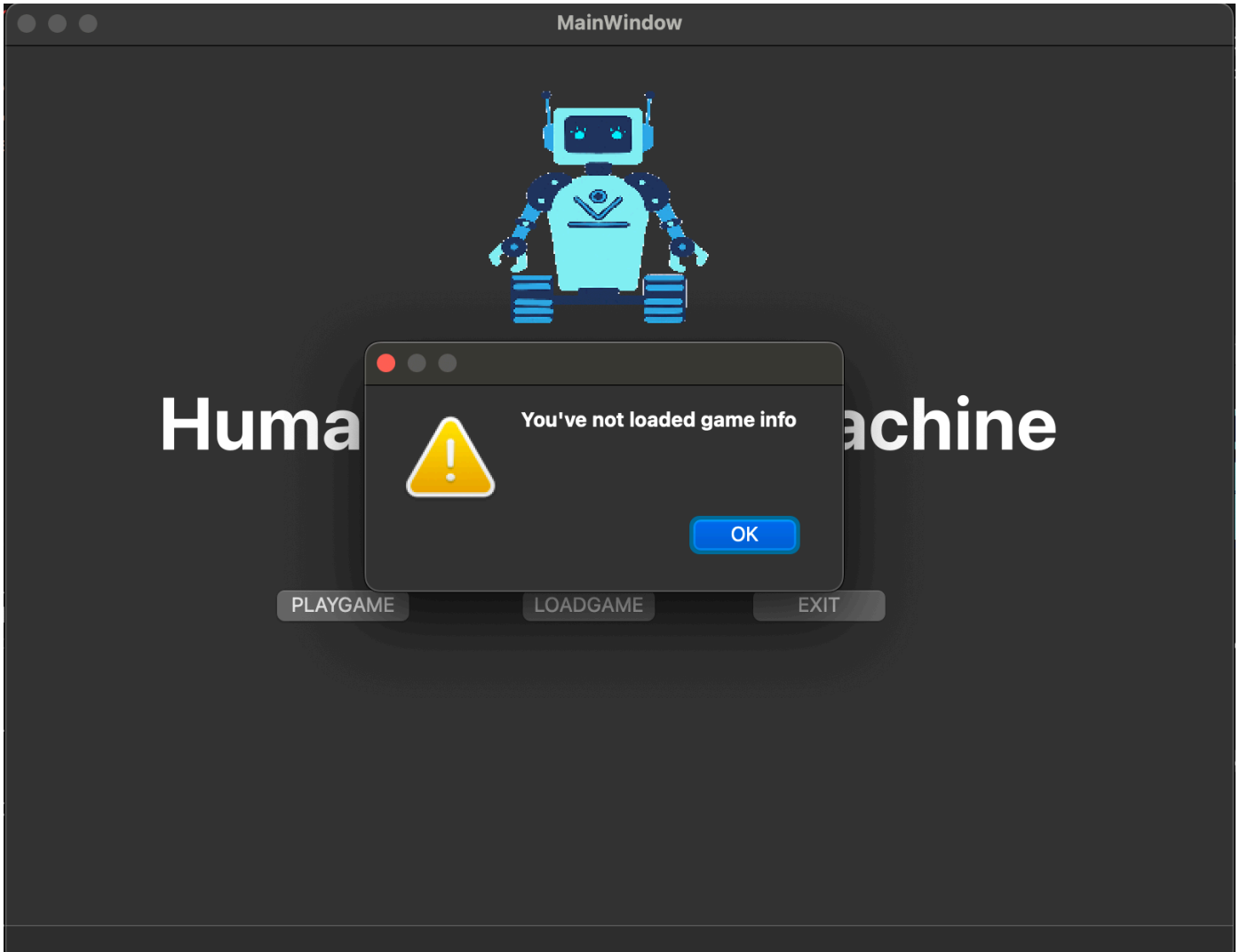
游戏界面采用Qt Creator实现。

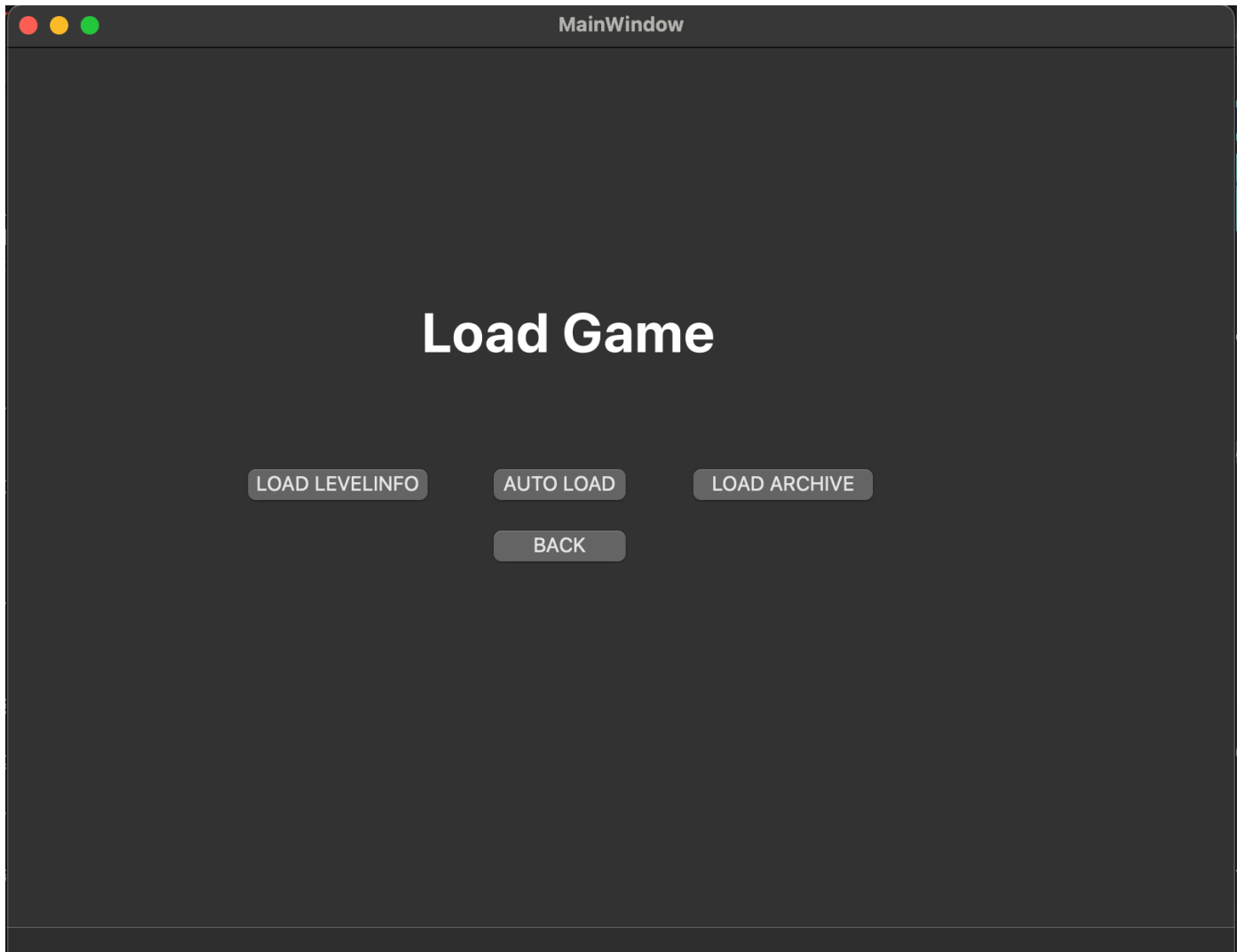
1. 游戏首页



首页中间是游戏名称，其下分为三个按钮：`PLAYGAME`——选择关卡，`LOADGAME`——初始化游戏，`EXIT`——退出游戏。

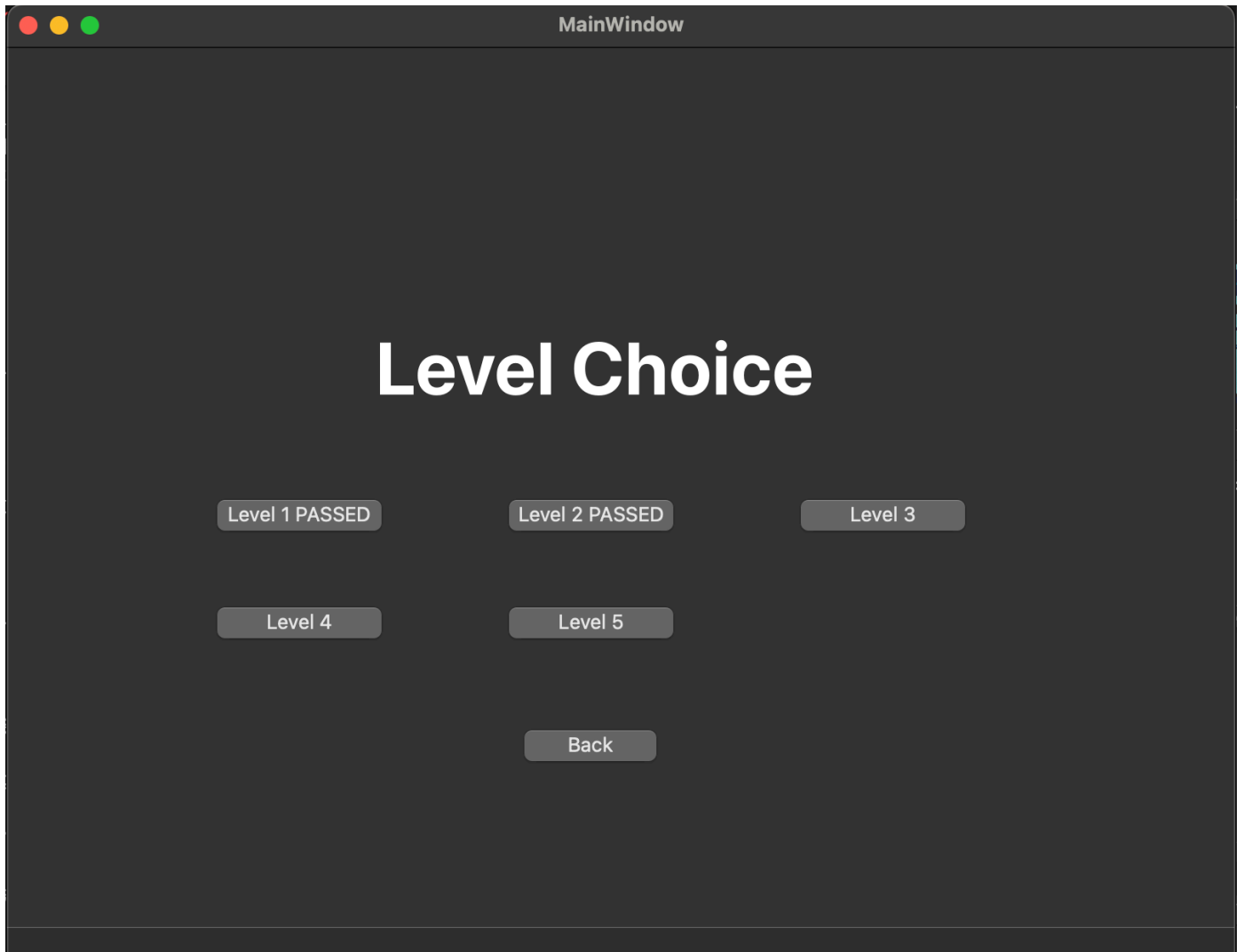
用户首先需要点击 `LOADGAME`，分别加载 `levelinfo`（关卡信息）和 `archive`（游戏日志），也可选择 `autoload` 一键加载（注意：`autoload` 识别与游戏程序同文件夹下名称为 `levelinfo.txt` 和 `archive.txt` 的文件，不能自由选择文件）。随后点击 `BACK` 返回首页，然后点击 `PLAYGAME` 选择关卡。倘若没有初始化游戏就点击 `PLAYGAME` 则会出现 `You've not loaded game info` 的提示框。



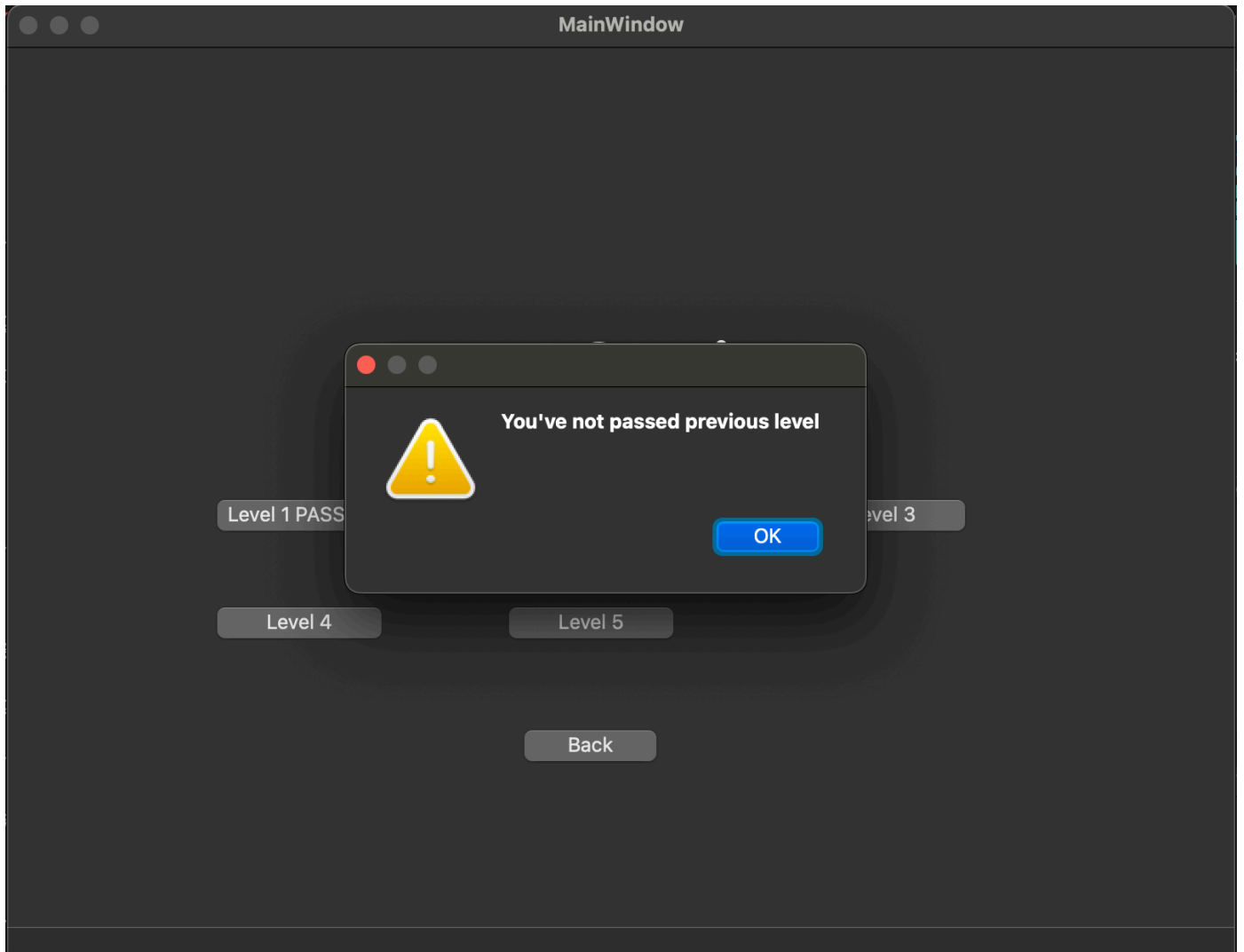


2.关卡选择

进入PLAYGAME关卡选择界面后，可看见LEVEL1-5五个选项，如果上传的游戏日志，相应的关卡后会显示PASSED。

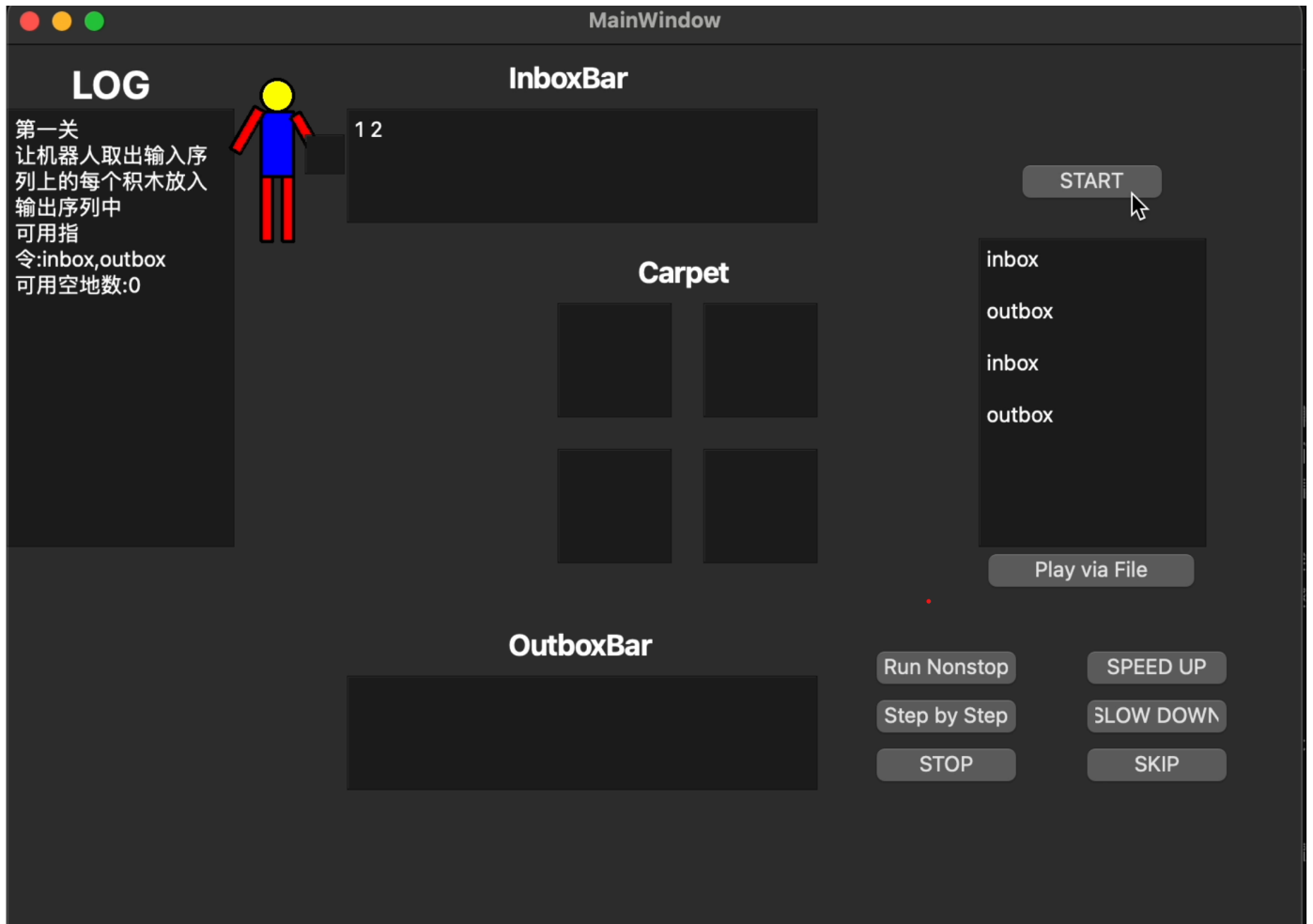


用户可选择想要游玩的关卡，倘若尚未通过前置关卡则会出现“You've not passed previous level”的提示框。



3.游戏内部

选择关卡后正式进入游戏内部界面。界面上包括多个模块：



LOG：在关卡开始之前呈现关卡信息描述，在用户完成输入点击运行后逐步呈现当前执行的指令。

InboxBar：输入传送带，随着每一步指令的读取实时更新。

Carpet：空地，随着每一步指令的读取实时更新

OutboxBar：输出传送带，随着每一步指令的读取实时更新。

机器人程序：用户在此处输入完整指令后点击START运行测试。

Play via File: 用户在此处可以选择文件输入形式。

START：点击后界面进入单步模拟

Step by Step：用户可以随时点击模拟下一步

Run Nonstop：点击该按钮可进入自动连续模拟状态运行测试。

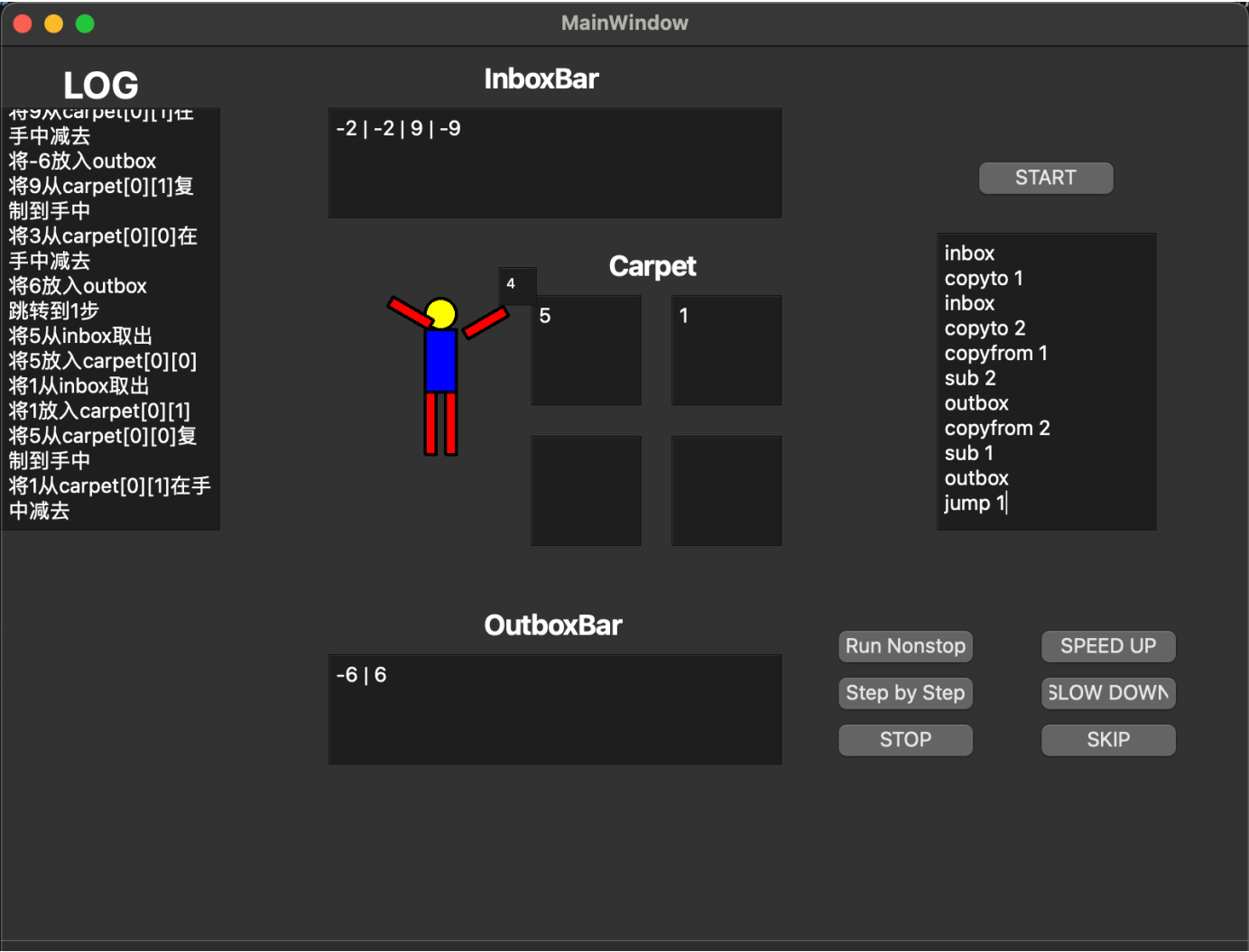
Stop：用户可随时暂停自动连续模拟进行查看

Speed Up：运行速度*10

Slow Down：运行速度/10

Skip：跳过逐步模拟，直接显示最终结果

机器人是一个可移动，手部可旋转的小人，右手上有储存数字的方框。随着指令的读取，机器人可实时连续移动至inboxbar、outboxbar、carpet旁边，对右手方框中储存的数字进行操作。



4.完成界面

当所有操作执行完成或者出现错误指令游戏终止后，游戏界面上会呈现Success和Fail的最终结果。

Success：成功后显示实际执行步数、指令数、output，还会有庆祝的烟花特效。

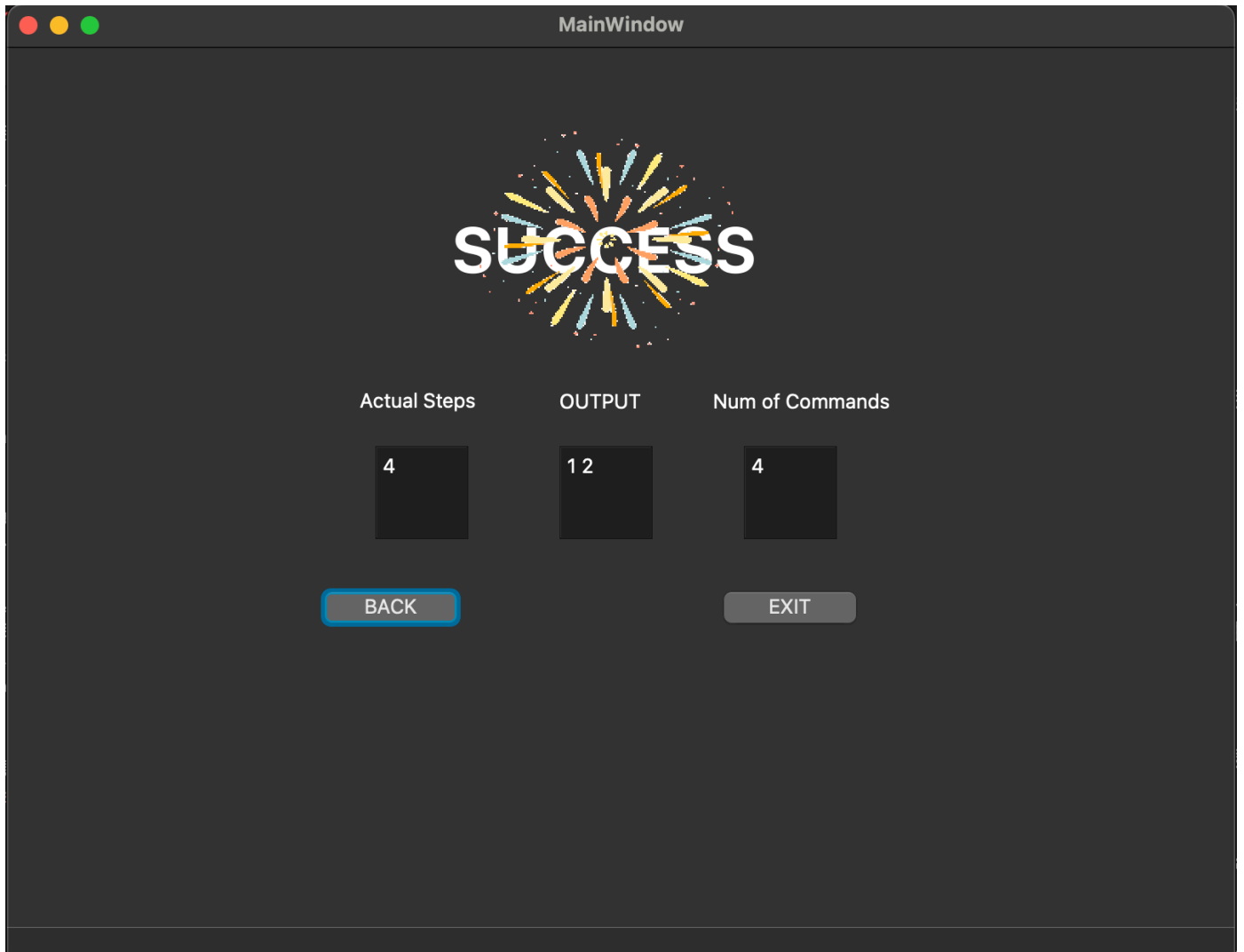
Fail：

- 错误指令：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）。在执行一条指令时，如果遇到了上述异常情况，Fail界面上会显示Error on instructionX
- 错误答案：Fail界面上会显示Wrong Answer (*3) **
- 用户可以选择restart尝试重玩游戏

关卡选择界面上的是否通过状态也会立即更新。

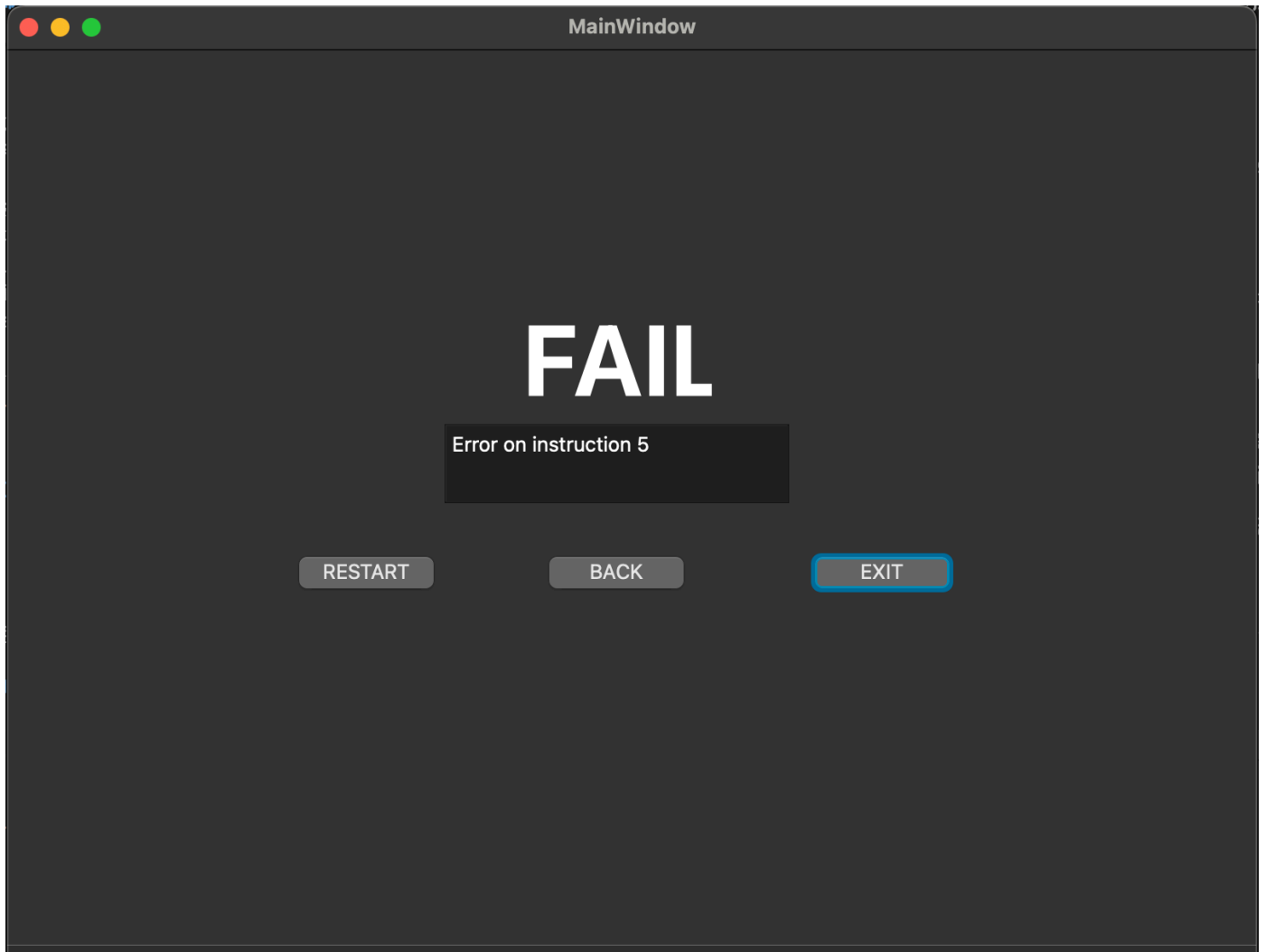
五、游戏测试（前三关）

1. Success

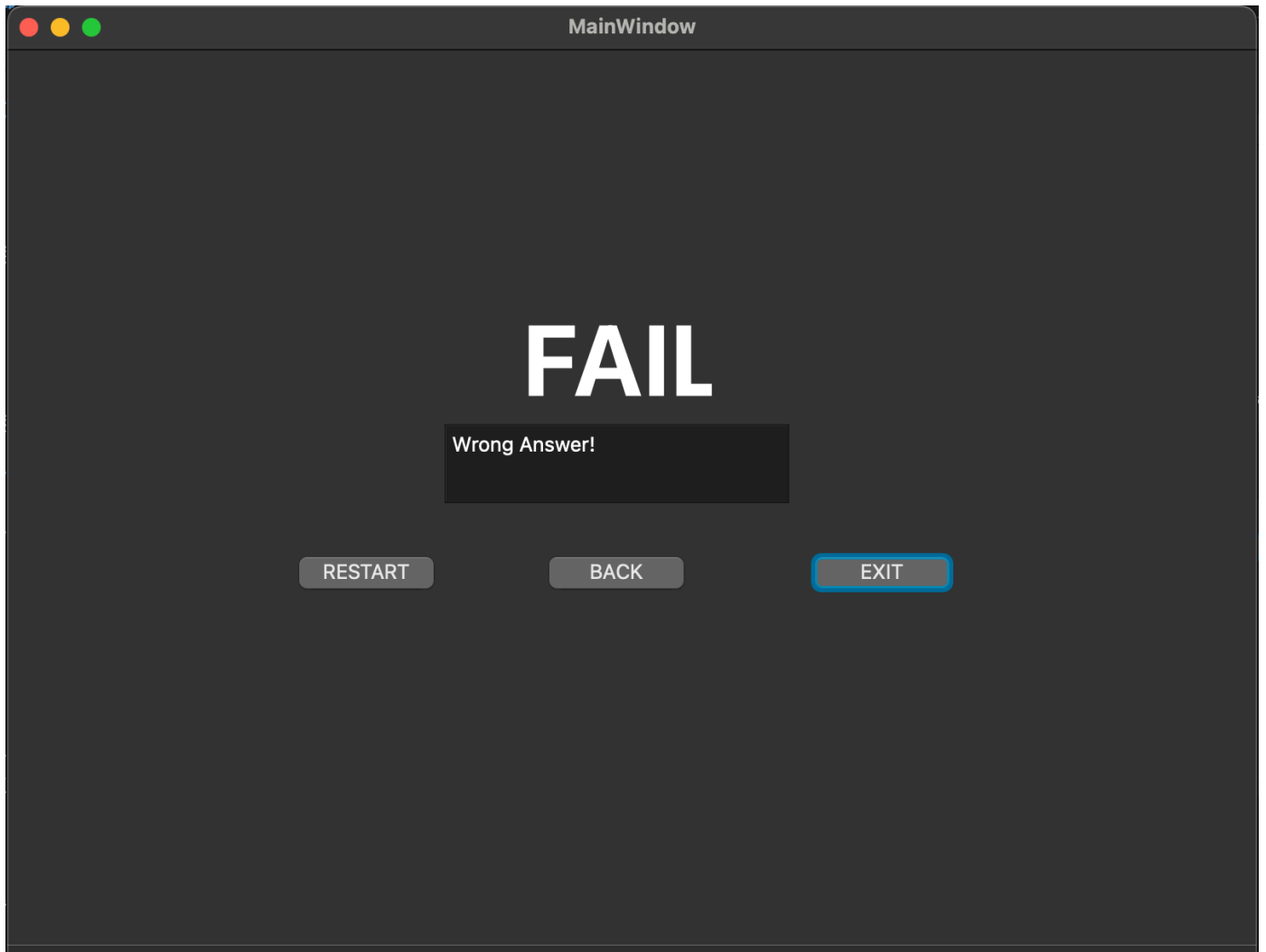


2. Fail

2.1 Error



2.2 Wrong Answer



六、自由创新关卡

1.关卡背景

第四第五关灵感来源于第十二周oj题目：哼哼哈兮。第四关提供固定的样例输入，希望用户运用肉眼观察法找到最大武力值。第五关则由特殊到一般，样例输入是随机生成的，希望用户能够写出程序实现哼哼哈兮的完整解法。在Human Resource Machine能够支持的操作框架下，这类似于一个原始汇编程序的写作，需要用户进行手动的内存管理和调用。

2.关卡内容介绍

LOG

第四关

有N种类型的短棒，第i种短棒长度为 a_i ，武力值为 b_i ，数量无限

长度为A的棒和长度为B的棒组成一个长度为A+B的棒，此时长度为A+B的棒的武力值为他们的和

如果A与B的长度相等，则其武力值还要再增加233

已知需要一根长M的棒，求最大的武力值

注:这一关可以使用hand+，可用空地数为4

InboxBar

3 10 1 2 3 1000 3000 6000

Carpet

OutboxBar

START

copyto 1

inbox

copyto 2

inbox

copyto 3

add 1

copyto 1

copyfrom 3

add 3

hand+ 233

add 1

outbox

Run Nonstop

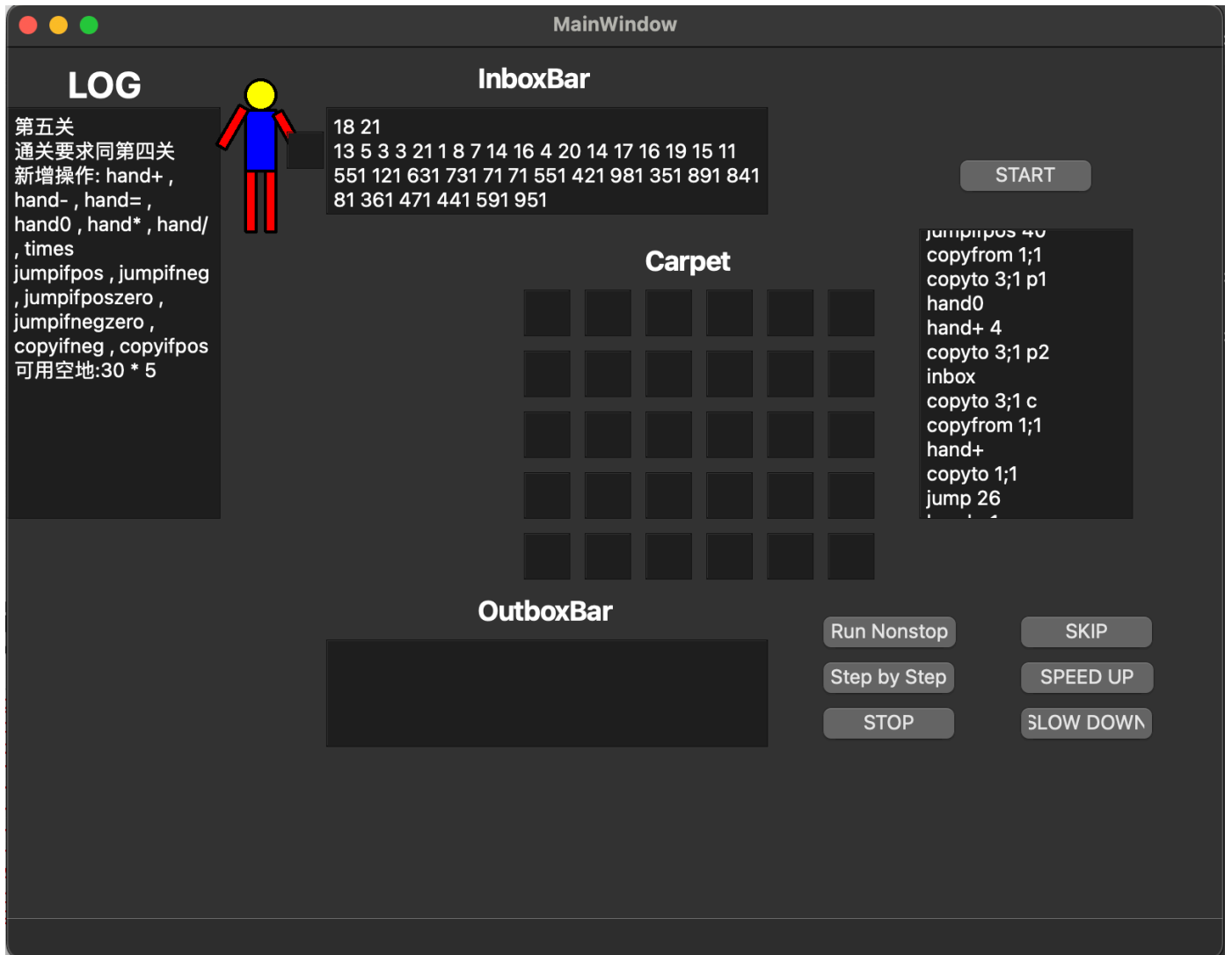
Step by Step

STOP

SPEED UP

SLOW DOWN

SKIP



2.1 输入 (inbox)

第一行输入两个数N, M。

第二行为N个整数a_i, 表示每种短棒的长度。

第三行为N个整数b_i, 表示每种短棒的武力值。

输入数据保证有解。

(5 < N, M, a_i ≤ 20)

(a_i ≤ M)

(0 < b_i ≤ 1000)

2.2 输出 (outbox)

输出最大武力值。

2.3 可用指令集&空地数量

哼哼哈兮的程序涉及数组、循环等操作，故需要引入新操作，同时，我们在最大程度上保证了对原始指令输入的兼容性。

首先，我们需要扩充地毯至无限二维数组，以使用户储存信息，否则在一维的数组上进行内存管理过于地狱。

其次，由于会涉及未知的存储和调用，我们需要添加指针来实现这些功能，这样我们可以实现数组的功能。

最后，由于会涉及到很多乘法，指标加一等运算，为了方便用户游玩，让用户可以专注于核心逻辑实现而非这些细枝末节，我们提供了很多简单的操作函数。但是，原则上，乘除法等操作是可以通过函数调用来实现的，我们现在的代码已经可以实现单次函数调用，对于多次函数调用，我们可能还需要增加 `jump` 指令的 `c` 参数模式，来实现根据储存的返回地址实现跳转。

下面具体说明新加入指令的输入模式。

对于任何一个指令，比如说 `copyto`，后面都可以带有参数。在新的指令中，指令的标准范式为 列;行 额外参数。其中列之所以是第一个参数是为了保证兼容性。

对于坐标参数，如果用户只输入一个数字，那么默认理解为第一个数字是列，游戏会默认填充行为1.这样，在游玩前面几关的过程中，用户不会访问除了第一行之外的 `carpet`，因此可以直接输入单个数字作为坐标参数，保证了兼容性。

额外参数有 `np,c,p1,p2`，其中 `np` 为默认额外参数，用户不需要自行输入，程序会自行补全。

`c` 为 `carpet` 输入模式，表明这个实际执行的参数应该是前面的坐标给出的 `carpet` 中的内容，程序会访问这个 `carpet`，然后解析其中的坐标参数（注意：我们不支持多重指针，因此第二次访问的 `carpet` 的内容不能是 `c` 参数模式）。支持的指令有：`copyfrom` `copyto` `add` `sub` `times`。

`p1` 和 `p2` 为参数写入模式，只支持 `copyto` 指令。`p1` 表示将手中的值写入到 `carpet` 的第一个坐标参数（列），`p2` 表示将手中的值写入到 `carpet` 的第二个坐标参数。具体操作如下：case1，写入的时候为空，直接写入；case2，写入的时候不为空，但是没有 `;`，则根据坐标的表达形式追加，并添加分号；case3，写入的时候不为空，且有分号，根据需求进行对应位置的修改。注意：目前我们并没有实现 `c` 输入模式的指针写入（虽然这应该也很简单）。

新加入的指令还有 `hand*`，`hand/`。只需要加入一个数字参数就可以，但是注意，均不支持负数参数。以及 `hand+` 和 `hand-`。它们具有默认参数 `1`，也可以指定参数，可以为任意整数。

2.4 成功范例

第四关最大武力值答案为19233，一种成功的解决方案如下：

```
inbox
inbox
inbox
inbox
inbox
inbox
inbox
copyto 1
inbox
copyto 2
inbox
copyto 3
add 1
copyto 1
copyfrom 3
add 3
hand+ 233
add 1
outbox
```

第五关见 `nunchunk.txt` 文件，有详细的说明。

3.新代码结构介绍

- 一方面，由于针对“哼哼哈兮”关卡重绘了有多`carpet`的页面，因此许多页面和函数都重写了有`nunchunk`后缀的版本。
- 另一方面，针对新操作指令加入了许多判定、报错、鲁棒性与兼容性逻辑。

七、小组分工

- 王鹏杰：游戏主程序主撰写；图形界面主搭建；报告工程结构撰写；展示视频录制。
- 高雅菁：游戏主程序debug；图形界面机器人动态、页面美化加工；报告设计逻辑、图形界面、创新关卡内容撰写；展示视频录制、剪辑加工。