基于鼠标手势识别的 WINDOWS 快捷启动插件

汇编语言程序设计大作业

庄晨帆 2011013246

王学成 2011013252

宋艺博 2011013268

2014 年 4 月

目录

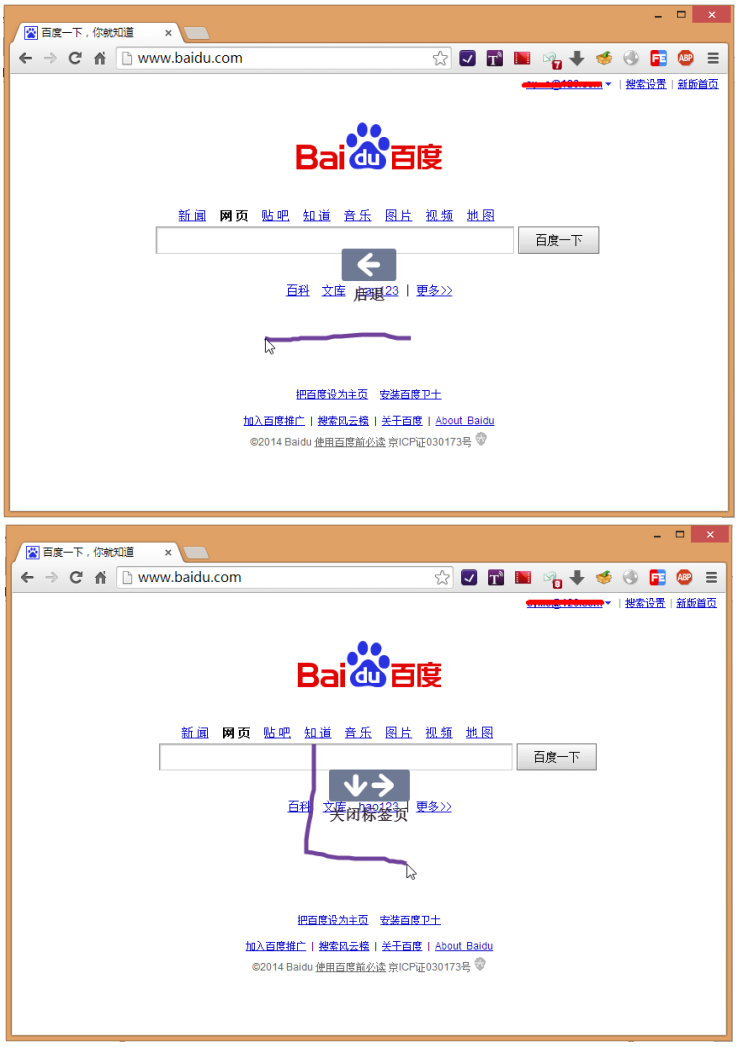
1. 实验选题

本次汇编语言程序设计实验选择的题目为鼠标手势识别。

1.1. 需求来源

我们的目的是设计一款实用性较强的 Windows 鼠标手势识别插件，为此，我们参考了 Chrome 浏览器上的一款手势识别插件 CrxMouse。该浏览器插件能在用户使用 Chrome 浏览器浏览网页时利用用户的鼠标手势快捷地调用浏览器功能，如网页的前进、后退、关闭、刷新，打开新标签页、打开扩展程序设置、重新打开关闭的标签页等功能，为使用 Chrome 浏览器提供了极大的方便。

该插件的使用简图如下：



受到该插件的启发，我们想到将该插件的功能从浏览器平台移至传统的

Windows 平台上，并添加适当的功能，做出自己的创新，从而方便用户在 Windows 平台上进行操作。

1.2. 需求分析

Windows 资源管理器实现的搜索功能不够强大和方便，相较 Mac OS 系统上的 Finder 显得功能较弱。为了能在 Windows 平台上实现根据用户的需要快捷地启动程序、打开文档、浏览网页等需求，我们想到使用鼠标手势这一方便的操作方式。

用户可以在程序窗口中拖动鼠标划出手势，程序会对手势进行识别，将用户的鼠标轨迹转化成特定的方向序列，再打开此方向序列对应的程序、文档等。鼠标手势相较键盘快捷键更易于记忆，且操作更加简洁方便，更适合于非专业的 Windows 用户使用。

为了实现方便用户操作的需求，我们支持手势和所要打开的程序之间的映射关系由用户自己来设定。这是相较于 Chrome 插件的一个进步。用户可以根据自己平时使用 Windows 系统的习惯，为自己的手势定义自己的动作，从而定制自己的功能。为此我们也设计了较为友好的 UI 进行支持。

由于程序需要支持识别多种手势，因此我们需要对用户鼠标手势转化成的方向序列进行识别和匹配，并给用户提供良好的匹配信息，提示用户当前正在划的手势能够执行什么功能，免去用户记忆手势的麻烦。

2. 实验环境

实验使用 Intel 汇编语言进行开发，辅以 MASM 汇编库。

开发人员在开发过程中使用不同的实验环境：

1、Windows 8

2、Windows 8.1 64 位企业版

3、Mac OS 下的 Windows 8 虚拟机

在前两个平台上实验程序均运行正常，在 Mac 平台虚拟机下字符编码会出

现问题。

开发使用的集成开发环境：

Microsoft Visual Studio 2012

鉴于汇编语言的平台相关性，建议使用 Windows 8 运行本实验程序。

3. 相关知识点

本次实验涉及到的汇编知识如下：

1、Intel 汇编指令：数据传送、算术运算、寻址操作、条件处理等。

2、过程的定义和调用，结构的定义，字符串和数组的处理。

3、MASM 定义的一些伪指令、宏定义等。

4、基于 Win32 API 的图形界面程序设计。

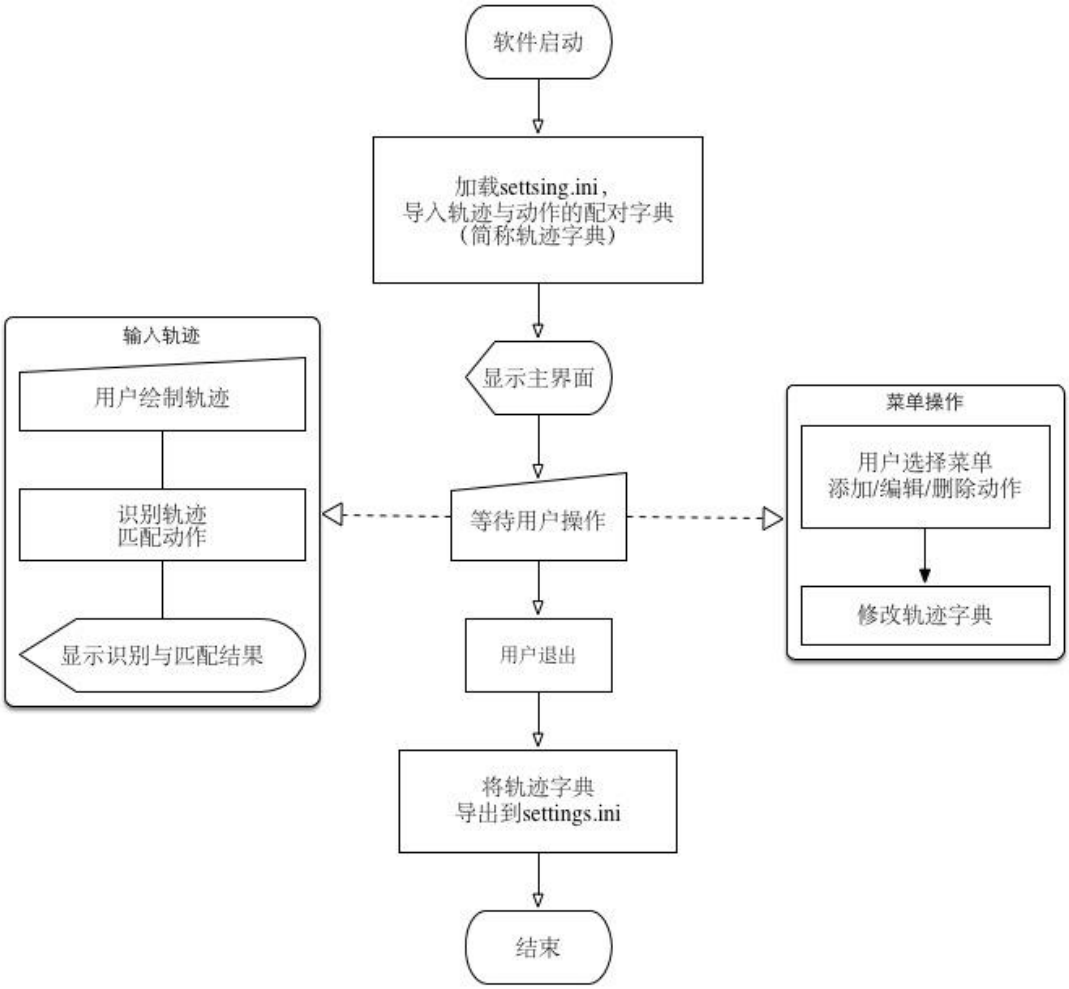
5、通用对话框的使用、文件读写操作。

4. 程序设计

本部分主要从代码实现的角度介绍本实验的内容。

4.1. 逻辑和功能

程序的简要逻辑流程如下图所示：



下面对程序内部的逻辑和呈献给用户的功能做简要说明。

4.1.1. 启动

进程启动时需要进行的操作主要如下：

1、注册窗口类

2、获取命令行参数

3、读取字体、菜单、位图、图标等资源文件

4、根据命令行参数，后台运行或前台启动窗口

5、建立窗口消息循环

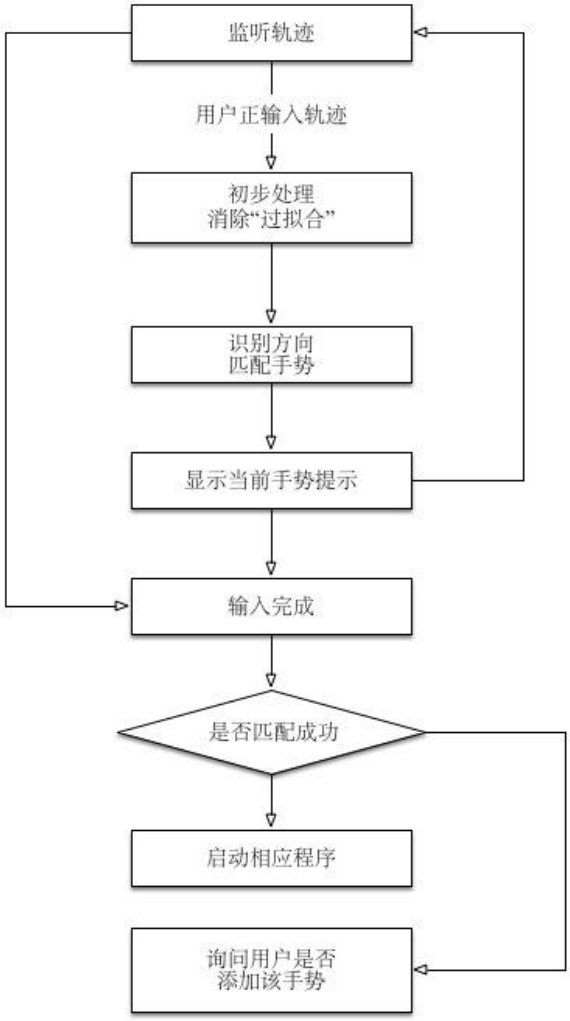
6、读取用户配置文件

7、等待用户操作

启动后程序界面如下：

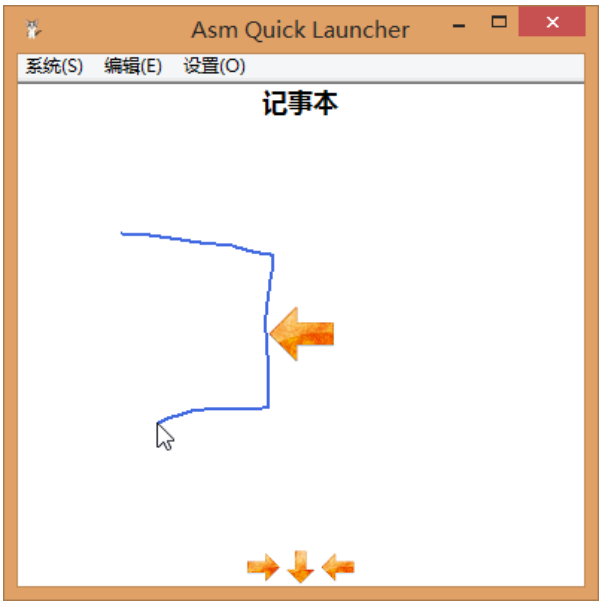
4.1.2. 手势

用户可以在主窗口中拖动鼠标左键进行手势绘制。在窗口消息循环中捕捉 WM_LBUTTONDOWN 消息，并开始识别记录鼠标坐标，在 WM_MOUSEMOVE 消息中将鼠标坐标点列记入数组 trackPoint。此处需要注意，如果把所有 WM_MOUSEMOVE 获取到的点都加入到 trackPoint 中，会导致方向判断过于敏感，降低用户体验。因此，我们队对写入 trackPoint 的点有一个最小步长限制，存储在 RECOGNIZE_DISTANCE 常量中，即 trackPoint 中相邻的两个点曼哈顿距离必须大于等于 RECOGNIZE_DISTANCE。程序的简要流程如下图所示：



下面对流程进行具体说明：

每当用户鼠标轨迹方向改变时，将新的方向记录到方向序列中并进行序列匹配，如果匹配到已有的手势，就显示提示信息，效果如图：



如上图所示，屏幕中央会显示用户鼠标轨迹当前的朝向，窗口底部会显示出用户鼠标轨迹的方向序列，窗口上方会显示当前轨迹匹配的动作，若匹配不上则不会显示文字。

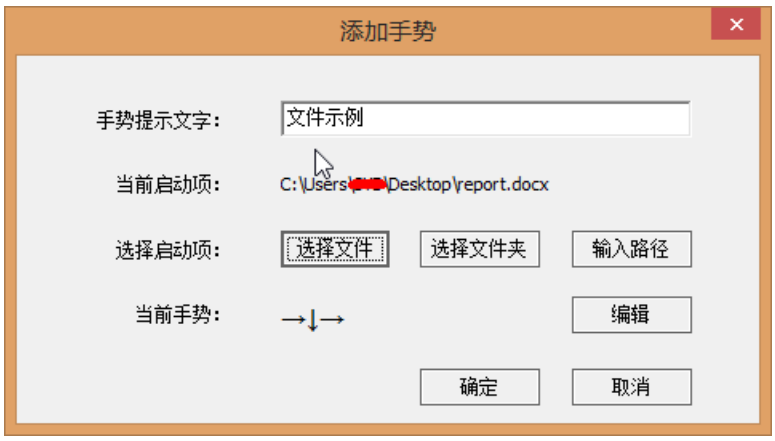
若用户的手势匹配到动作，则当 WM_LBUTTONUP 被触发时，会调用 Win32 的 ShellExecute 函数打开用户定义过的程序或文件、URL。

若当前手势没有匹配到动作，则会弹出对话框询问用户是否要添加当前的手势为新手势：

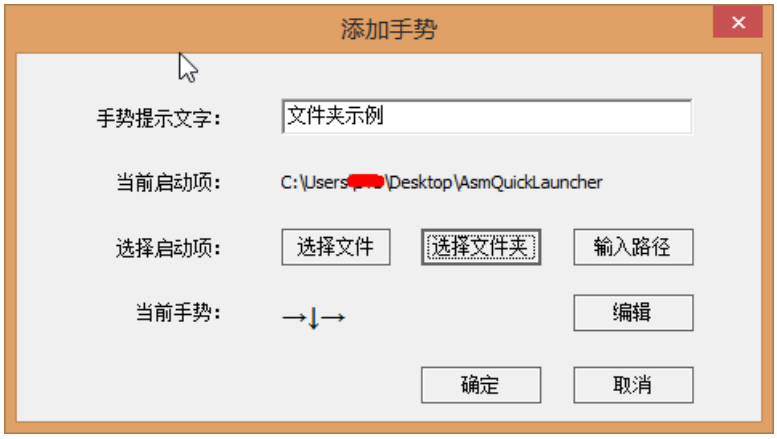
选择“是”则弹出添加手势对话框：

用户需要输入手势提示文字，即匹配到手势时显示在屏幕上方的文字；需要编辑启动项：

点击“选择文件”，会弹出选择文件对话框，用户可以从中选择一个文件，点击“确定”后会得到该文件的完整路径：



点击“选择文件夹”会弹出选择目录对话框，用户可以选择文件夹，这样可以实现打开特定文件夹：



点击“输入路径”，弹出输入路径对话框，用户可以自行输入要打开或执行的内容：

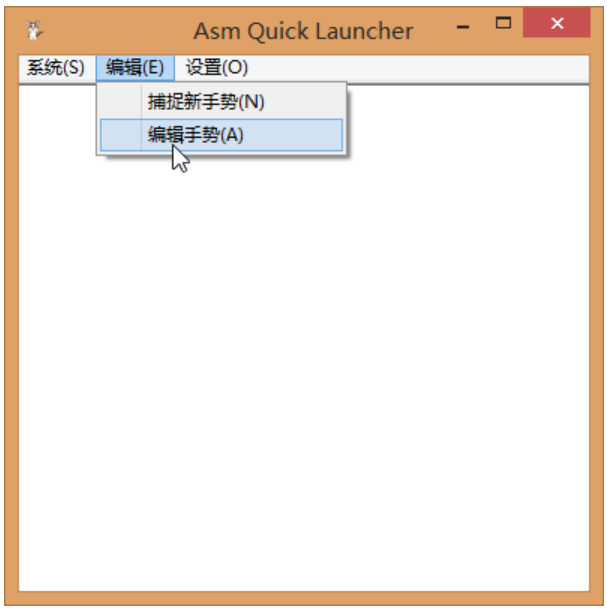
这也是本程序最为灵活的一个功能，根据 ShellExecute 提供的对于路径的解析能力，支持文件、路径、URL、系统 PATH 环境变量中定义的可执行程序、系统虚拟路径、CLSID 等多种启动方式。另外，能在 Windows“运行”功能中打开的程序或文件，也能够在此输入，定义为手势。如下是除路径、URL 之外的一些例子：

explorer	Windows 资源管理器
notepad	记事本
regedit	注册表编辑器
control	控制面板
::{645FF040-5081-101B-9F08-00AA002F954E}	回收站

在“添加手势”对话框点击“确定”即可完成添加。添加完毕的手势就可以立即使用。

4.1.3. 编辑

通过创建模态对话框，实现便捷的手势编辑功能。点击菜单：



弹出“编辑手势”对话框：

通过下拉列表框可以选择要编辑的手势，修改手势提示文字或选择启动项均与添加手势类似。

点击右下方的“编辑”按钮，可以弹出“编辑方向序列”对话框，直接对手势进行编辑：

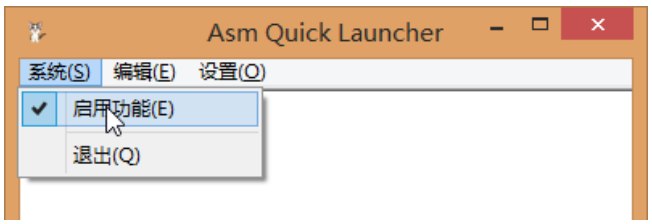
使用四个方向按钮在当前序列后追加方向，或使用“清空”按钮清空方向序列重新安排。

点击“确定”完成编辑。

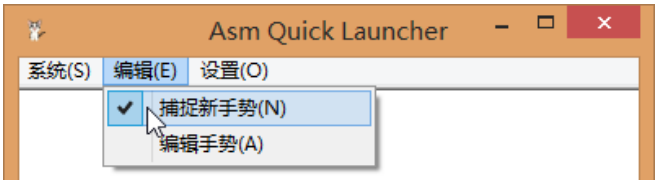
使用“删除当前手势”按钮可以将当前的手势删除。

4.1.4. 菜单

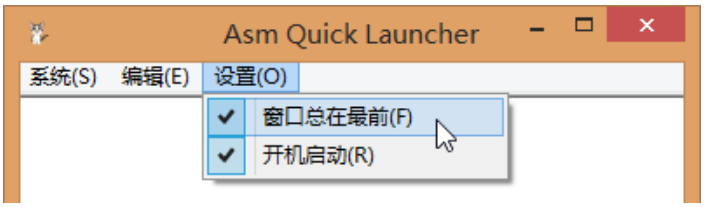
在“系统”菜单中可以启用或禁用本插件的功能。处于禁用状态时停止捕捉鼠标轨迹。



“退出”菜单项用于退出程序。



“编辑”菜单中的“捕捉新手势”选项用于快速添加手势。当此项被选中时，用户在主窗口中绘制鼠标手势时若没有匹配到对应的手势则直接弹出“添加手势”对话框。



“设置”菜单中提供一些常用设置。

“窗口总在最前”选中时可以保持窗体始终在桌面最顶层，方便用户在进行工作时绘制手势启动其他程序。

“开机启动”可以设置本程序开机启动。由于需要写注册表，因此需要管理员权限，如果没有以管理员权限启动本程序，会弹出提示框提示用户以管理员身份启动。

4.1.5. 托盘

如许多长期启动驻留后台的系统插件一样，本程序支持后台运行。

程序运行时点击最小化按钮，程序窗口会隐藏起来并在任务栏显示托盘图标：

单击托盘图标窗口会恢复显示，也可右键点击托盘图标选择菜单：

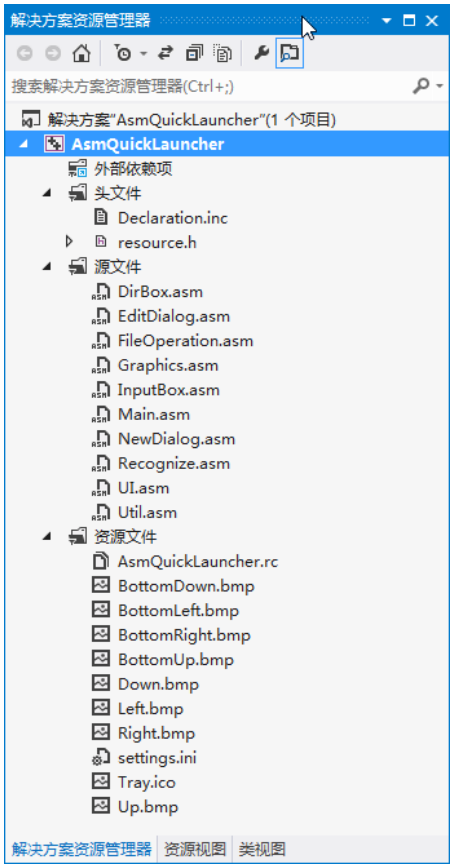
4.1.6. 热键

系统驻留在托盘区时，为了方便用户呼出主窗口，我们为程序设置了全局快捷键。预置了 Ctrl+Q 作为程序窗口切换的快捷键，可以将主窗口从托盘状态呼出，也可以在主窗口显示时隐藏主窗口。

热键的设置对于用户使用本插件是较大的帮助。

4.2. 代码结构

本程序使用多文件的方式组织汇编代码，文件组织结构如图：



头文件：

Declaration.h：包含了项目中需要 include 的头文件和库文件，定义函数原型，定义结构体，定义全局变量，定义资源符号。

resource.h：定义资源相关的一些配置，由 Visual Studio 自动生成。

源文件：

Main.asm：程序启动逻辑和主窗口的创建，以及主窗口过程、消息循环等。

NewDailog.asm、EditDialog.asm、InputBox.asm、DirBox.asm：定义“添加手势”、“编辑手势”、“输入路径”、“编辑方向序列”四个对话框的对话框过程。相应对话框中的消息。

Recognize.asm：用户鼠标轨迹识别、方向序列提取、方向序列匹配等与手势识别相关的核心代码。

Graphics.asm：主窗口上的鼠标轨迹绘制、提示文字显示、位图绘制相关代码，实现了双缓冲机制避免屏幕闪烁。

FileOperation.asm：文件读写操作，用于在程序启动和结束时读写配置文件。

UI.asm：处理菜单事件响应。

Util.asm：一些工具性质的函数。

资源文件：

AsmQuickLauncher.rc：主要的资源文件，定义了菜单、对话框、位图、图标等资源的具体参数。

*.bmp：程序中使用的位图，主要是主屏幕上用来显示方向的箭头图形。

Tray.ico：程序图标文件，显示在托盘区、标题栏中，同时也是.exe 文件的图标。

settings.ini：保存用户手势配置的文件，每次程序启动时读取，程序结束时写入。

4.3. 程序结构

代码架构上，借鉴了 MVC 的思想，将整个程序的功能分几个层次进行考虑：

1、前台主界面（View）：监听用户操作，例如输入轨迹、请求编辑手势等。

2、中间传递模块：负责对前台数据进行初步处理，并传递给后台进行处理，再将后台返回的数据进行包装，传送给前台用于显示。

3、后台数据处理：对得到的数据进行识别操作，与“字典”中的数据进行匹配；对字典进行增、删、查、改操作。

使用这种架构方式，有如下优点：

1、让代码结构变得清晰、有逻辑。

2、便于多人合作，每位组员负责一个模块，各有专攻，不需要对整个工程的代码都了解，只需要与组员商议好接口即可。使用 git 工具来管理代码，冲突少。

3、便于测试。各位模块负责人确保自己模块的正确性。在后期测试时，只要关注各个模块的输出是否正常。如果有异常发生，交给相应的组员，debug 的效率大大提高。

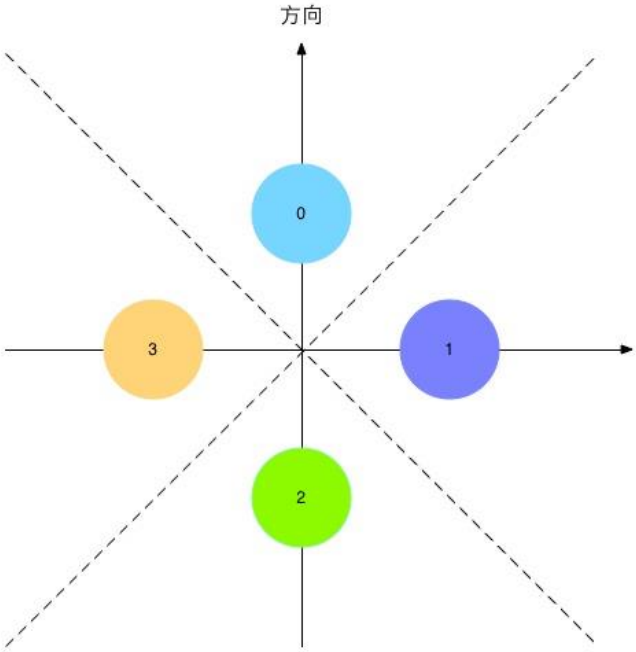
4、便于迭代开发。每次开发重点“升级”一个模块，保持接口统一，其他模块基本不需要变动，省去了大量的修复工作。并且借助其他模块来对当前的修改部分进行测试，省时省力。

4.4. 技术实现

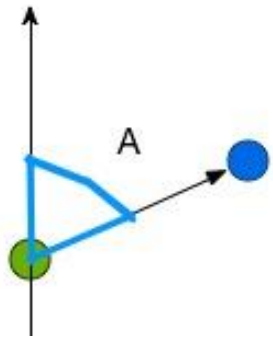
下面对程序中使用的核心技术进行说明。

4.4.1. 手势识别

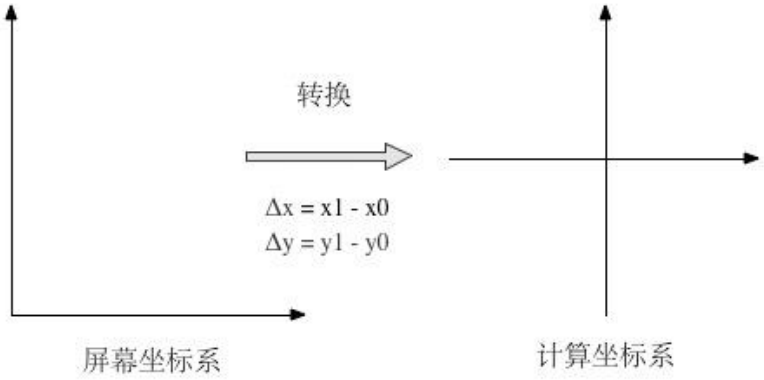
本程序将鼠标轨迹识别为上下左右四个方向，通过识别鼠标轨迹方向的变化来定义手势。方向定义如下：



具体的计算方法为：计算从上一个绘制点P_0到当前绘制点P_1的移动方向，利用向量$\overrightarrow{P_0P_1}$与y轴的夹角，得到当前的移动方向，如图：



需要注意的是，屏幕坐标系与我们常见的坐标系（称为计算坐标系）略有不同，需要进行一个简单的转换，如图：



具体代码见 Recognize.asm。

4.4.2. 序列匹配

考虑到在实际情况中，用户不会记住大量的手势，因此手势字典（actionMap）预设可容纳 1024 个手势。在这个数量级上，逐一枚举手势进行匹配就能达到令人满意的效果。

为了更好的用户体验，我们采取了前缀记录与精确匹配的方法：

首先，为手势字典中的每一个 ACTION 维护一个标志变量，用 prefixMatchArray 数组记录第 k 个手势是否能匹配上，例如：

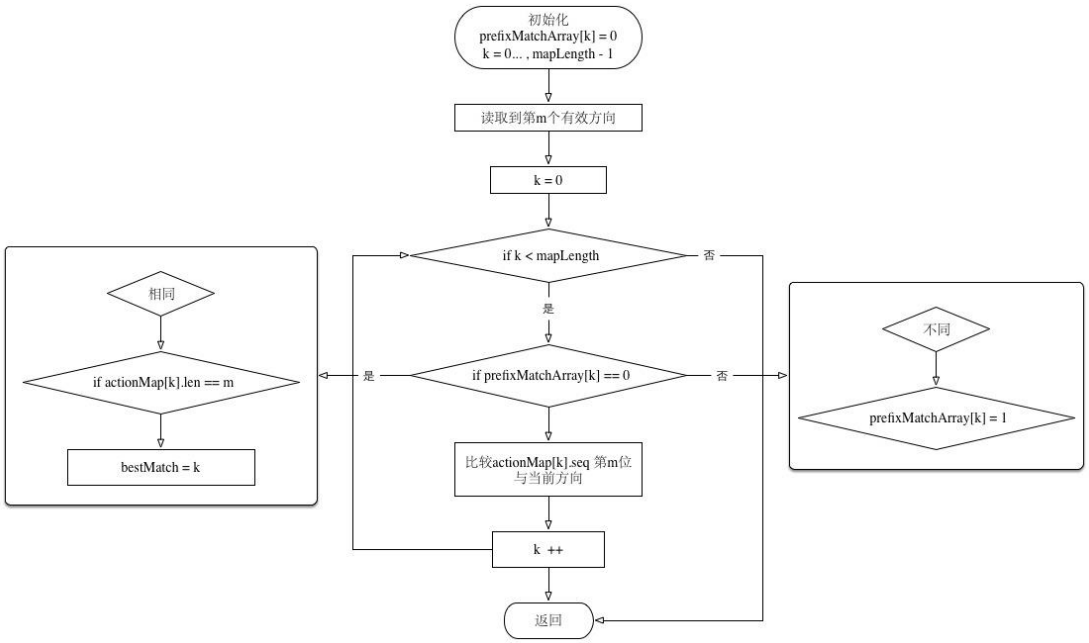
prefixMatchArray[k] = 0 actionMap[k]的前缀能匹配

prefixMatchArray[k] = 1 actionMap[k]的前缀不能匹配

每当后台监听到一个有效点（定义参见上文“手势”部分），不妨设为是当前输入手势的第 m 个有效点，就与当前 prefixMatchArray[k] = 0 的 actionMap[k] 的第 m 个方向进行匹配，若失败了，则 prefixMatchArray[k] = 1。

因此，无论鼠标轨迹有多长，总的匹配次数不会超过 actionMap 中所有 ACTION.len 之和，是一个程序可以接受的常数。

用一个全局变量 bestMatch 记录是否有匹配手势，若无，则置为-1，由负责执行命令的函数来监听 bestMatch，触发相应的操作。



4.4.3. 动作存储

为了将手势映射到打开文件、程序的动作，我们设计了数据结构对动作进行存储。

Declaration.inc 中定义了 ACTION 结构体：

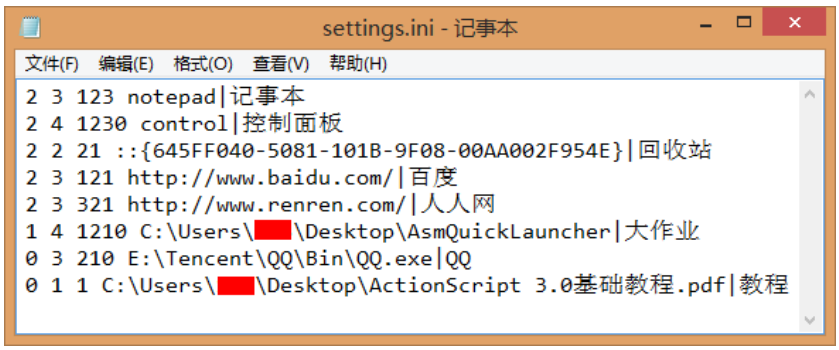
```
ACTION STRUCT
    len dd 0            ; length of seq
    seq dd 32 DUP(0)    ; direction seq
    path db 1024 DUP(0) ; path of responsive action/command
    tip db 1024 DUP(0)  ; tip of responsive action/command
    pathType dd 0       ; action/command type
ACTION ENDS
```

手势的方向序列和要执行的动作以 ACTION 结构体的方式对应起来，在程序执行期间，内存中维护一个 ACTION 数组，用以实现新建、修改、删除手势的功能。

外存方面，程序启动和关闭时，通过文件读写将此数据结构维护到外存中。文件 settings.ini 用于保存用户的手势字典，在程序启动时载入到内存中，其中的每一条记录（一行）对应一个 ACTION 结构体，格式为:

#type #len #seq[1]#seq[2]...#seq[len] path|tip

一个 settings.ini 文件的示例如下：



4.4.4. 屏幕绘制

屏幕绘制运行在 WM_PAINT 消息的响应过程中，具体实现过程在 Graphic.asm 文件中。在屏幕绘制之前，需要获取到当前显示区的宽度和高度。屏幕绘制主要分为四个部分：

第一个部分是鼠标手势当前方向提示，通过之前获取到鼠标手势的当前方向，即 lastDirection 变量，将相应的方向图片显示在显示区的正中间。这里需要用到每次触发 WM_PAINT 消息时重新获取到的显示区的宽度和长度。

第二部分是鼠标手势全部方向提示，通过存储在 trackSeq 数组中的参数，我们可以获取到之前用户已经输入的所有方向手势，并且将这些手势图标按照从左到右、从下到上的先后顺序显示在屏幕的最下方。这里对手势长度进行了限制，如果现实的图标已经超出了显示区的高度，则停止获取鼠标手势。

第三部分是鼠标手势文字提示，提示的内容是当前手势对应的手势提示文，居中显示在屏幕最上方。

第四部分是鼠标手势路径画线，也就是将用户鼠标在操作区滑动的整个路径显示出来。实现的方法就是将 WM_MOUSEMOVE 获取到的所有点存放在 drawPoint 中，通过将 drawPoint 中所有相邻点连线的方式画出整条路径。

为了避免直接在显示区绘制时产生的闪屏问题，我们在绘制的过程中采取了双缓冲的机制。简单来说，就是在内存中建立一个与显示区等大小的位图区域，先把需要绘制的内容绘制到这一块内存中，再将这一块内存通过 BitBlt 函数全部赋值给显示区。

4.4.5. 后台运行

当用户将窗口最小化之后，为了减少任务栏显示负担、提高用户体验，我们采取自动最小化到托盘的方式让程序在后台运行。实现过程中的关键函数是 ShowWindow 和 Shell_NotifyIcon，通过 ShowWindow 将窗口隐藏，再通过 Shell_NotifyIcon 函数显示托盘图标。当用户左键单击托盘图标或者是右键单击托盘图标在弹出菜单中选择显示时，先通过 Shell_NotifyIcon 将托盘图标隐藏，再通过 ShowWindow 函数显示窗口。同样的操作也可以通过快捷键 Ctrl+Q 完成。

4.4.6. 开机启动

设置开机启动的原理就是在管理开机启动项的注册表里添加注册项，这里面有几个关键的函数，先是通过 RegOpenKeyEx 函数打开注册表，再通过 RegSetValueEx 函数添加注册项，最后调用 RegCloseKey 关闭注册表。相反的，取消开机启动就是在管理开机启动项的注册表里删除注册项，先通过 RegOpenKeyEx 函数打开注册表，之后调用 RegDeleteValue 删除注册项，最后再通过 RegCloseKey 函数关闭注册表。

需要注意的是，注册表的操作一般需要管理员权限，所以用户在使用改功能时，需要以管理员身份运行。

4.4.7. UI 设计

为了给用户提供提供良好的体验，方便其定制符合自己需要的手势，我们使用了 Win32 API 为我们提供的菜单和对话框进行 UI 设计。

同时为了与系统紧密结合，较多地使用了 Windows 通用对话框，如浏览文件、浏览目录对话框，即通过系统 Shell 的 GetOpenFileName、SHBrowseForFolder 接口进行实现。

同时，Win32 API 还提供了丰富的组件，如按钮、下拉列表、文本框等常用的控件，与菜单、对话框一同构成了本程序的 UI。

5. 实验亮点

简要说明本次大作业中做得比较好的地方。

5.1. 用户友好性

一些鼠标手势插件的手势是开发者预定义好的，因为需要迎合特定系统的功能。我们为了本插件的实用性考虑，支持用户自己定制手势功能，利用 Windows 较好的 Shell API 满足用户的需求。

围绕着这个需求，我们设计了添加、修改、删除手势的界面，实现了对用户手势的维护。

给用户自主定制软件功能的机会，对于提高软件的实用性价值很高。

5.2. 系统健壮性

编辑手势界面输入框较多，因此需要对输入的合法性进行繁杂的判断，如果用户不输入内容，容易产生一些令系统不稳定的问题。考虑到这个问题，我们对用户输入的合法性做出了一定的检查，不允许空内容的输入框。

再如删除手势功能，如果删除最后一条手势，我们就将编辑对话框的功能改为添加手势的功能，变换按钮上的文字。用户既可以选择不添加返回主界面，也可以选择继续进行手势添加。

诸如此类。这些是我们对系统中的一些边界条件的考虑。

5.3. 功能细节

为了是我们的程序更符合一款软件的标准，我们参照许多程序设计了诸如最小化到托盘区、开机自动后台启动、全局快捷键等功能，使得我们的程序不仅仅是一次大作业，而是能够在课程结束后继续投入使用。

已经有其他非开发人员试用过本系统，反映出的回馈说明，对于一项大作业而言，这个系统还是有着较高的实用价值的。

5.4. 程序模块化

开发过程中，对程序进行功能上的分析，并据此对代码进行了模块划分。文件操作、图形操作、界面操作等模块分开编写代码，便于分工协作。

由于汇编语言的特性，在.data 段中定义的数据都是全局可以访问的，不过我们也采取了一些措施来规避过多全局变量的问题。比如对于会造成耦合度较高的变量尽量分离出来，每个文件定义的变量尽量只由自己修改，仅有其他模块做初始化操作。这样的一些尝试较好地提高了代码的重用性。

5.5. 团队管理

由于是团队开发，我们需要合理的代码管理措施。

本次实验使用 git 作为代码管理工具，开发人员在自己的分支上进行开发，调试正确的代码合并到 master，有效地保证了代码的安全。

同时由于较为合理的模块划分，我们的开发工作进行得有条不紊。合理的分工使得我们不依赖于集中开发，每个组员可以相对独立地完成自己的工作，因此开发效率较高。

6. 开发过程

本部分介绍整个项目的迭代历程。

6.1. 首次开发

1、确定选题，讨论方案的可行性与功能的实用性，找到软件的定位，明确各位组员分工

2、搭建整体框架，创建一个最简单的 demo，商定模块之间的接口，统一数据存储方式

6.2. 第二次开发

1、采用简单粗暴的方式对轨迹进行识别，算法虽然粗糙，但是准确率不低

2、完成轨迹的绘制

3、实现启动其他程序的功能

6.3. 第三次开发

1、对识别算法进行改进

2、采用“双缓冲”，解决主界面更新时“闪烁”的问题

3、实现打开虚拟文件（例如我的电脑、回收站）等等的功能

4、完成菜单栏、弹框等界面

6.4. 第四次开发

1、实现导入、导出程序配置

2、添加实时匹配并显示识别方向的功能

3、实现弹框响应，支持用户新加手势与编辑动作路径

6.5. 第五次开发

1、完善手势的添加与编辑功能

2、实现程序后台托盘运行

3、实现程序开机启动后台运行

7. 系统扩展

本系统的功能存在很大的扩展空间，参考一些优秀的 Windows 系统插件，可以添加的功能还有很多。由于汇编课程课时有限，大作业开发时间较短，很多可行性和实用性都很好的功能在本次大作业中都无法实现，在此列出一些系统的

问题和可能的功能扩展，以便以后的开发和维护参考。

1、虚拟路径支持：目前的选择路径通用对话框禁用了系统虚拟路径的选择功能（如“库”、“网络”、“控制面板”等），因为无法获得系统虚拟路径的字符串形式或者 CLSID，无法进行存储。

2、托盘菜单：当前系统的托盘菜单还只有“显示”和“退出”两个功能，还有很大完善空间。

3、配置文件导入导出：对文件接口稍作修改即可实现配置文件的导入和导出，使用户能将自己的配置文件存储到其他的路径。

4、热键修改：目前的系统预置的全局热键为 Ctrl+Q，应该支持用户修改全局热键。

5、UI 设计：现在的 UI 还较为单调，线条颜色、手势绘板的颜色或图片等都没有修改，整体显出比较传统的 Windows 程序风格，不够活泼明快，可以参考 MASM 库中的一些示例对 UI 进行一些修改。

8. 实验总结

我想很多人都曾经疑惑，为什么要学习汇编语言？

在今日，似乎没有哪家公司会雇佣员工用汇编语言来开发一个大型的软件，因为我们有众多的高级语言可以选择。但是我们不能否认，没有哪个高级语言能比汇编语言“跑得快”。在一些要求速度的关键环节，汇编语言就成为我们提高程序性能的利器。另外，要了解编译相关的知识，更是不能不懂汇编。

更重要的是，汇编能帮助我们更好的理解高级语言。甚至有时在调试 C/C++ 语言时，我们不得不去查看编译后的汇编指令，来找到问题的症结。

纸上得来终觉浅，当我们真正动手、从 0 写起时，才体会到写汇编程序的苦与乐。在这个过程中，遇到了很多问题。尤其是汇编语言的资源又少又旧，大部分时候甚至无从查起。当最后，我们越过了一道道难关，做出成果时，喜悦与骄傲在心头并发。不得不说，又是一次难忘的体验。

在这个过程中，我们收获了搜索能力与自学能力，培养了面对问题的细致与耐心，提高了团队之间的分工合作的能力。也许将来我们不再有机会写汇编语言，

但是这些更深层次的能力，聚沙成塔，无时无刻不在影响着我们的人生。

9. 其他说明

源码随文档一同提交。

另附项目 github 地址：

https://github.com/syb1001/AsmQuickLauncher

感谢老师和助教！