**改自微机课报告2018/11/8**

**一、主要思路**

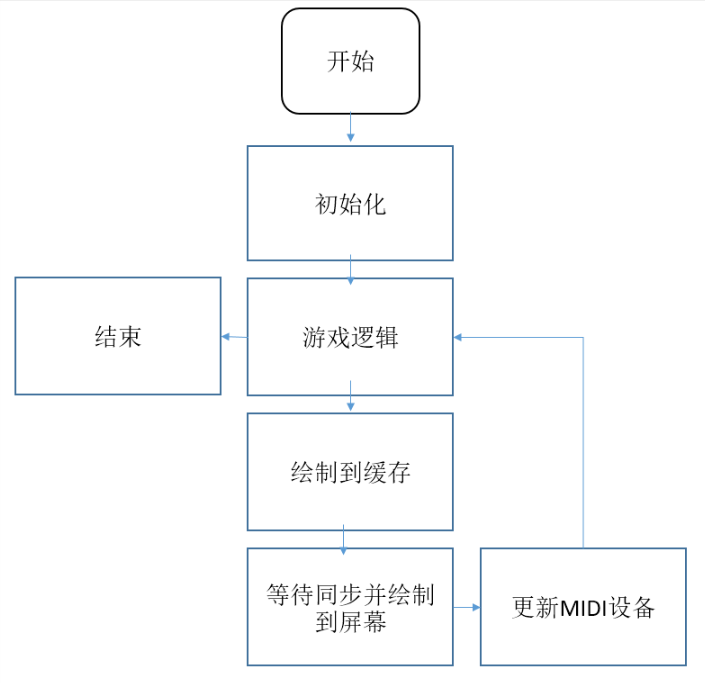
在立项的最初决定是制作一个小游戏。而游戏程序应当具有在绘制每一帧时进行逻辑更新的特点。经过对开发环境的探索，决定采用320\*200、256色的VGA显示模式，音频输出使用mpu401作为midi输出设备。

出于方便协作考虑，尽量将各个功能部分模块化。虽然最后没有足够的人手，但是这样的组织方式在调试上仍然有极大的好处。

细节的思路请参考第三部分，先阅读第五部分程序效果、实际尝试游戏将有助于理解程序。

**二、程序流程图**

下图为主要流程的框图。细节部分的框图见第三部分。



**三、程序解释**

源程序比较大，因此在这里按照功能和组织的方式对源程序做出解释。具体的代码和细节注释参见附件。

**1.概览**

reveu.asm 主文件，包含主要程序循环，维护各个段数据，包含其他所有引入（INCLUDE）文件。

startup.s 在程序最开始处被引入，包括初始化显示模式、调色板、段数据、中断

level1.s 在循环开始前被引入，加载所有位图资源

update.s 在循环中引入，包含游戏的逻辑更新（处理输入、准备绘制数据）

END.s 在循环结束处引入，恢复系统中断、等待退出

btblt2.s 包含一个子程序，用于向缓存绘制位图

loadas.s 包含一个子程序，用于加载资源

mints.s 包含一个中断子程序，用于处理键盘中断

as目录 包含图像资源、调色板资源

LOGU16

**2.图像绘制**

我们选用的显示模式包含256色调色板，320\*200像素的分辨率。显示缓存位于从0a000h开始的连续数据段。通过10h号中断、AL为13H启用。向外部接口3c8h、3c9h分别写入调色板的颜色编号、RGB颜色可以置调色板。

针对这个显示模式，我设计了本次程序的显示系统及工作流。

**(1)位图数据的准备**

BMP格式图像中，存在位深度为8（即256色）、使用调色板的模式，和我们的显示设备十分接近。但是BMP格式的图像元数据中的调色板以BGR顺序存储，而且还有填充字节，读取上有些不便。

Adobe Photoshop可以导出一种以.act结尾的文件用于描述调色板。这种文件的结构十分单纯，按照RGB顺序将三个分量平铺，十分适合本次的使用场景。

另外，作为游戏显示资源，需要背景透明，故将一个颜色保留用于标记不绘制。

因此，整个工作流为：将一个场景中同时可能出现资源平铺，使用PS的自带算法计算出最优的255色调色板。使用该调色板调整资源，并且用最后一个颜色标记透明像素。最后，导出颜色表和bmp图像，处理bmp图像仅保留图像数据（对调色板颜色的索引）。

需要注意的是，VGA的RGB分量是6位的，而现代的RGB分量普遍是8位，因此配置调色板时需要减少两位精度。

相关代码：

startup.s:25~46；加载颜色表

**(2)绘制图像**

图像资源在level1.s中调用loadas子程序被全部加载。加载的资源信息被维护在DATA数据段。每条信息7个WORD，按照模式（是否显示、显示模式）、实际数据的段基址、偏移地址、位置的X坐标、Y坐标、原始图像的长、宽存储。这些数据首先在游戏逻辑中被更新，随后在绘制阶段被绘制到缓存。

相关代码：

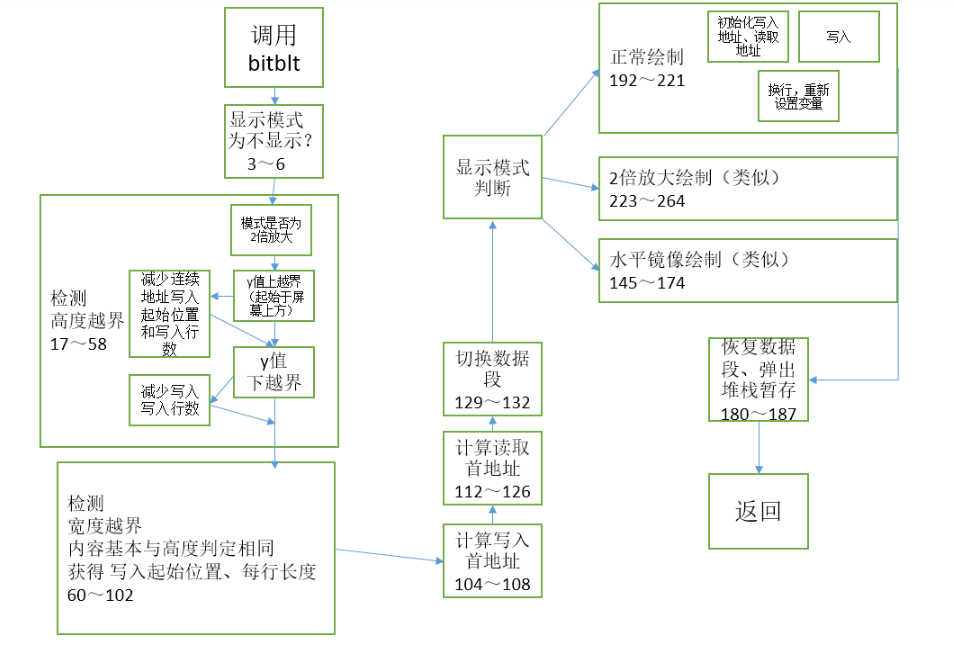
level1.s;调用loadas读取数据。

revue.asm:27~38; 位图资源的信息。

loadas.s;加载资源的子程序。调用约定：DX:路径偏移地址 CX:数据大小 BX:目标缓存的偏移地址 AX:缓存段基址

绘制工作主要调用子程序bitblt完成，包含在btblt2.s中。程序内部向es数据段特定位置写入颜色索引。在这里没有使用系统调用，因为系统调用在每次设置像素时都有安全检查，而我们需要大量写入，因此安全检查统一执行后，向连续地址写入图像数据。在第一次编写此程序的时候，仅实现了正常情况下的传输，而第二版实现了二倍放大和水平镜像，以及绘制时的裁切。程序中使用的是第二版。该子程序接收一个资源信息的地址，根据这个地址读取图像资源的各类信息（上一部分所述）。

下图为配合框图和程序行号的解释。



在主循环中，对每一条图像资源信息调用bitblt，由子程序处理显示与否、显示模式，将绘制结果写入ES数据段。

在绘制一帧的时候，如果直接向显存缓冲绘制，有可能造成未完成的图像出现在屏幕上，类似于“撕裂”现象。这样的情况会使人物在屏幕上闪烁。为了解决这个问题，我们需要建立一个缓存。尽管VGA模式的显存具有与内存的直接映射关系，这个映射的关系并不能改变，因此不能简单点实现现代渲染常用的图形交换链。因此我们的程序的流程上先将所有绘制操作在自建的缓存上完成，随后等待时钟信号，将自建缓存复制到VGA缓存。在开发初期，ES数据段直接指向VGA显示缓存，在发现这个问题之后，ES指向一个自建的缓存，之后将该缓存复制到VGA的显示缓存。

相关代码：

btblt2.s；子程序

revue.asm:301～312；针对每一条图像信息调用bitblt

revue.asm:358～368；将缓存绘制到屏幕

**3.输入和时间**

在游戏程序中，为了保持帧率的稳定，程序的流程控制不应该依赖于键盘输入，而是自动更新并且监听键盘的输入。针对这样的需求，我采用了键盘中断和监控DOS时间的方法。

1. **键盘中断**

虽然我们需要“监听”键盘状态，但是我们每一帧的处理都有时间，如果仅仅读取处理时的状态，过于快速的操作将被忽略。要想处理在每次处理与上次处理前的所有输入，就要记录每个输入，将输入缓存下来，在处理时一并处理。记录输入就需要使用键盘中断。

键盘中断为IRQ9，在键盘被按下和抬起的时候被触发。原本由DOS系统处理，我们暂时将键盘中断替换为我们自己的处理程序。我们的处理程序记录每个键盘输入到缓存，在游戏逻辑中处理，之后清空缓存。

尽管替换了中断，这个键盘中断仍然不是一个理想的游戏设备。键盘中断的调用在按下后调用一次，若不放开则隔的一段时间继续调用，并且连续发送键盘按下的消息。而抬起中断的调用如果抬起过快将会被忽略。

相关代码：

startup.s:6~15；替换键盘中断，将DOS系统的中断程序地址暂存

mint.s；键盘中断处理程序。我们直接忽略按键抬起。

end.s:3~14；恢复DOS系统中断

1. **时间控制**

最开始，我们希望能够尽可能快地更新图像，随后获取实时时间计算帧间时间以使角色以恒定速率在屏幕上移动。但是，在DOS环境下实时时钟并不是标配，而DOS的实时时间依靠一个低精度的时钟中断。这个中断每55ms调用一次，因此快速的更新会造成诸多不稳定因素。我决定以这个时钟中断的速率更新屏幕。时钟由DOS维护，我们这一次选择不去修改中断，而是监控DOS维护的时间。DOS维护的时间在0040h:006ch处，每次变化说明DOS的中断被调用。在主循环中的图像缓存就绪后，等待这个值变化，之后进行后续的屏幕更新操作。另外，音乐的更新也依靠这个机制进行。

相关代码：

revue.asm:320~327;等待时钟变化。在这个循环后面是音乐更新和屏幕更新。这两项操作需要相对准确的时间，因此应该紧跟着时钟执行。

1. **清空输入缓存**

输入缓存在等待时间前,如果处理器频率足够高，等待的时间将占用大部分帧间间隔。另外，在这里有检测是否退出的操作。检测到ESC将跳出主循环进行结尾操作。

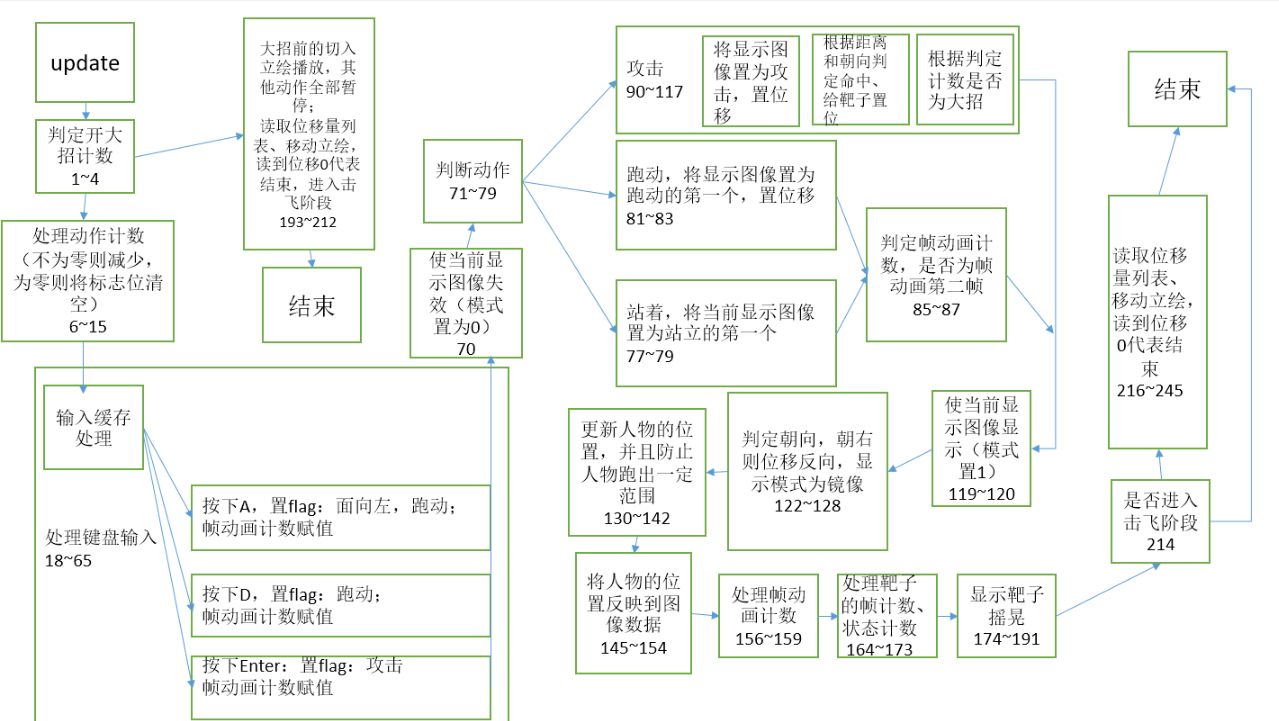
相关代码：

Revue.asm:315~318；上述操作。

**4.游戏逻辑**

游戏逻辑部分实现了整个游戏的资源管理。这部分处理键盘输入，决定哪个图像显示，维护人物的位置信息、帧动画信息。

下图为带行号的框图注解。



上图中的过程包含在update.s中，在每次绘制一帧时进行。更新完毕后，图像绘制部分的程序会根据更新后的图像数据绘制。

**5.背景音乐**

1. 主要设计思路，工作原理

思路：首要原则，不能占用过多资源，尽可能的简单。

工作原理：利用外部MIDI设备播放音乐，cpu只用于与外部接口进行通信。将音乐简化为时序乐谱，定时向外部接口传送相应的音乐数据。

2. 前期准备

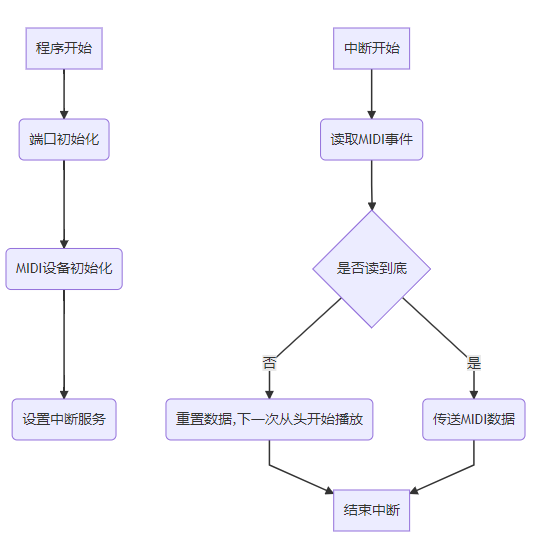
Dosbox 配有虚拟midi设备 Roland MPU-401。有两种工作模式：Intelligent 和Uart modes。

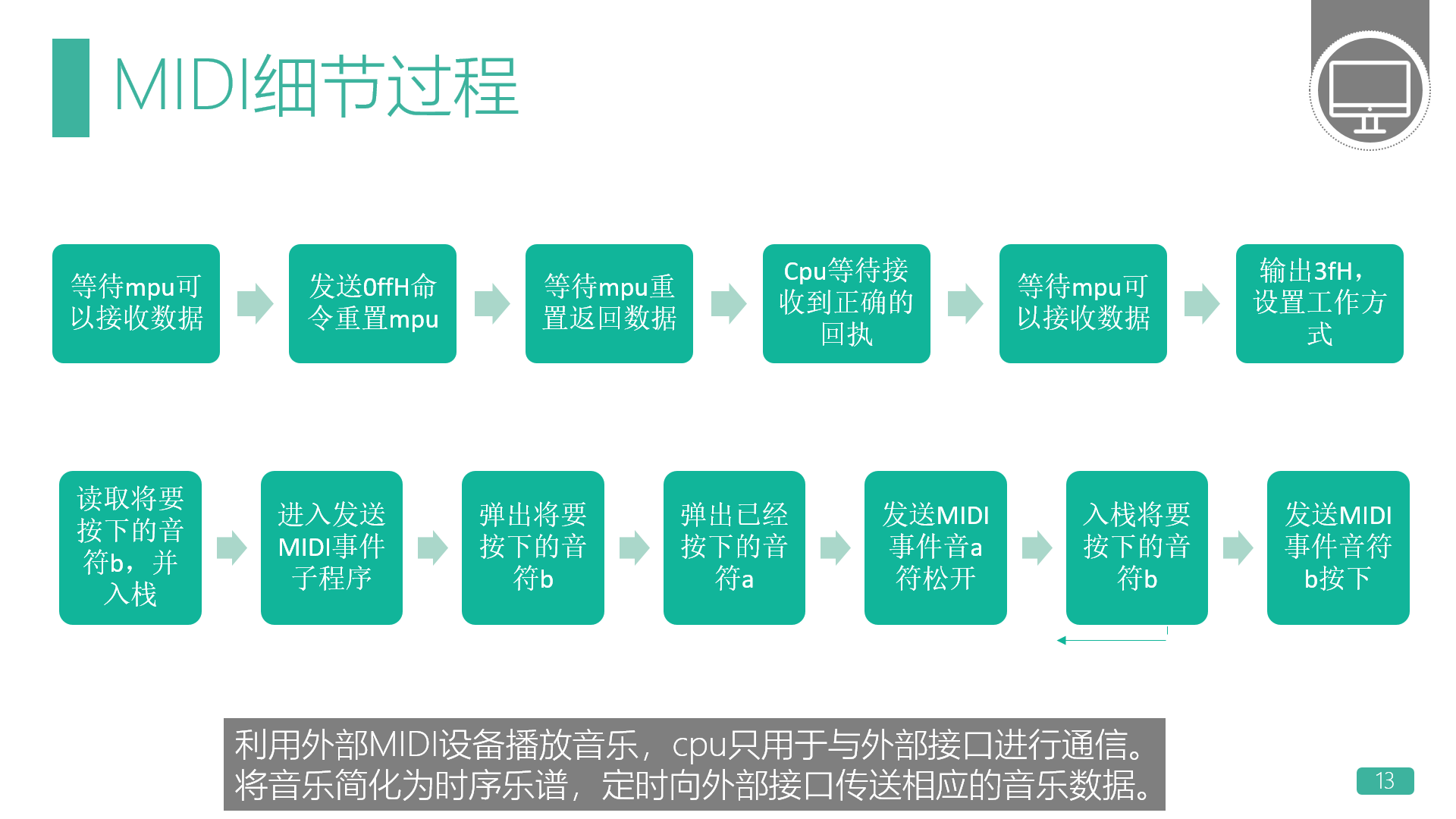
在Intelligent模式下，MPU使用其板载电路为应用程序提供大量服务。例如，MPU具有一个板载硬件定时器，用于对传入的MIDI事件加时间戳，并可用于确定何时输出MIDI事件以进行有序播放。MPU-401还有一个内部节拍器，可以设置为以相对于此计时器的速率自动发出蜂鸣声。Intelligent模式做了很多，因此需要应用程序提供相当精细的中断处理程序，因为MPU需要为了许多不同的目的而中断计算机。这显然是我们不期望遇到的。

UART模式会丢弃上述所有功能，并将MPU转换为计算机从属设备的简单设备。只要应用程序将MIDI字节写入MPU，MPU就会输出每个MIDI字节，并且只要应用程序选择从MPU读取，app就会读取传入的MIDI字节。该应用程序可以更好地控制何时选择输出和输入字节。当然，音序器应用程序必须管理自己的定时器以进行排序。

MPU-401是一个8位卡。它有2个8位端口：DATA和 STATUS \ COMMAND。MPU可以设置为各种基本（I / O）地址。如果它在Dosbox中默认基址为330H，则DATA端口为330，STATUS \ COMMAND端口为331。

UART模式下：DATA端口是输入和输出MIDI字节的位置。这个端口是双向的。将命令字节写入COMMAND端口后，MPU会确认此命令。MPU确认后将FE（十六进制）字节发送到其DATA端口。因此，在向COMMAND端口写入一个字节后，需要继续从DATA端口读取字节，直到遇到FE字节。在进行下一个指令输出。





**四、问题与解决**

**图形、游戏逻辑**

调试过程中难免出现错误，而图形程序调试在DOS下比较困难。解决方法是，使用输出数字的子程序写了个宏，遇到可疑的地方监视变量，打印在屏幕上。

程序比较复杂的时候，合并在一个文件里比较容易出错，因此把各个功能分出去，逻辑更清楚一点。

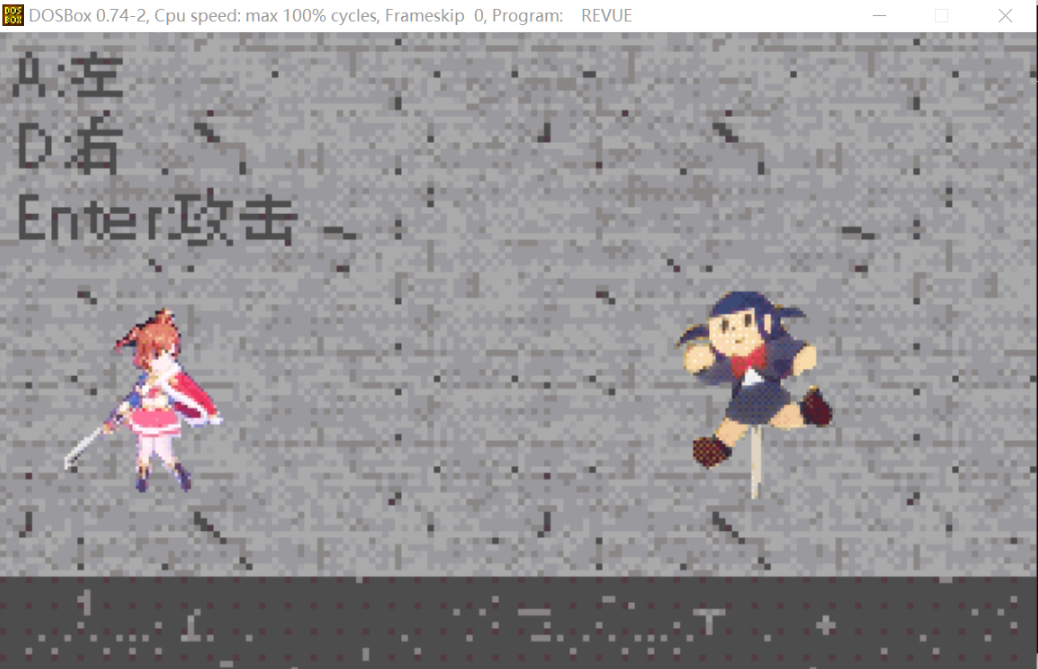
**音乐部分**

(1) 计划利用8253 可编程定时器产生定时脉冲，再连8259a进行中断，无法确定dosbox的具体连接，查阅资料，自己尝试无法实现。更换中断，利用1CH中断，重写其终端服务。

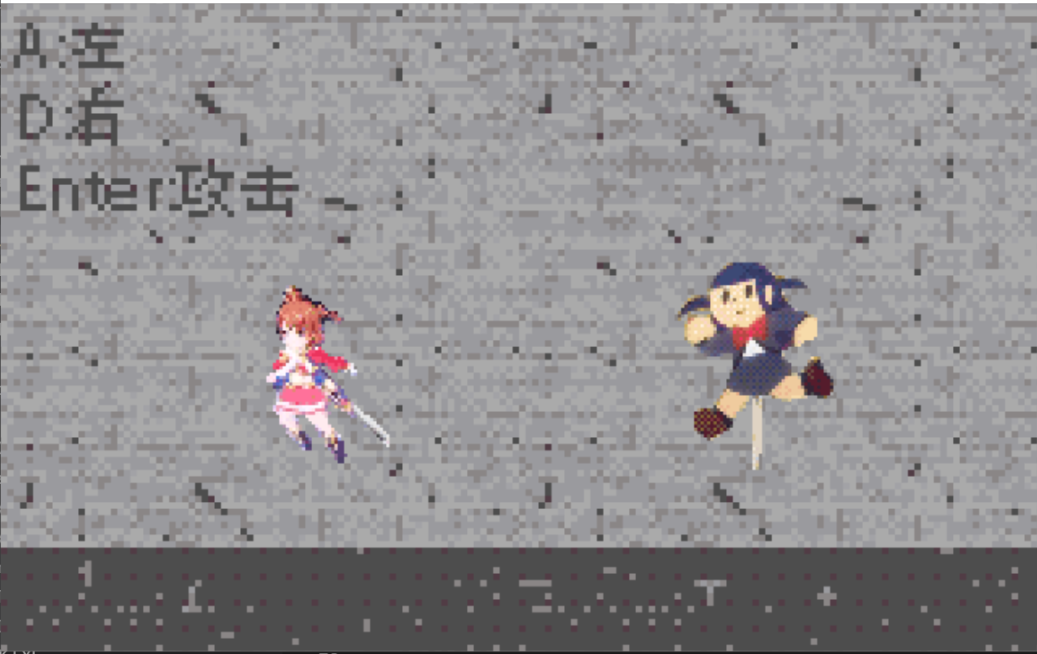
(2) 程序对接时，发现主程序的屏幕刷新同样利用了这个1CH，但是利用方式不一样，利用不断读取1CH中断对应的计数器进行定时操作。所以对接时为了不破坏源程序直接将中断服务部分加入屏幕更新部分，去除了设置中断部分。

**五、程序效果**

游戏过程中按照设置的节拍进行读谱，可以听到洋娃娃和小熊跳舞这首歌。



（人物是在动的）



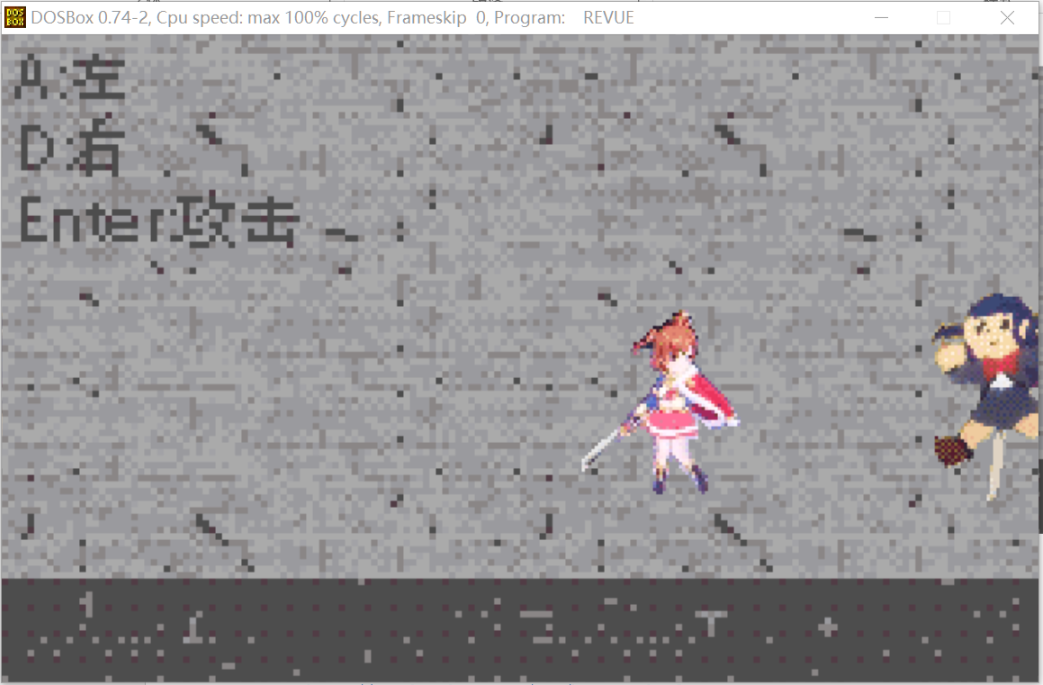
（可以左右移动）



（可以揍这个靶子）



（多揍几次会出现从左到右飞过的立绘，大招）



（靶子被打飞了，结束）

**参考资料：**

VGA相关：

<https://en.wikipedia.org/wiki/Mode_13h>

<http://www.brackeen.com/vga/basics.html>

<http://www.brackeen.com/vga/bitmaps.html>

<http://www.wagemakers.be/english/doc/vga>

键盘中断：

<http://inglorion.net/documents/tutorials/x86ostut/keyboard/>

定时：

<https://stackoverflow.com/a/41714028>

为什么不能实现图形交换链:

<https://alt.lang.asm.narkive.com/kxJUnxOo/does-vga-or-svga-have-a-screen-pointer-you-can-change-dos>

MPU音频：

<http://www.piclist.com/techref/io/serial/midi/mpu.html>

ACT格式调色板：

<https://www.adobe.com/devnet-apps/photoshop/fileformatashtml/#50577411_pgfId-1070626>