汇编小组作业 报告
选题：游戏 仿节奏地牢

周雨豪 (组长) 软 82
叶宸 软 82
徐源远 软 82

1 项目介绍

完全仿照游戏《节奏地牢》实现，大部分素材均从《节奏地牢》游戏素材中获取，本项目仅供汇编课程学习用。

《节奏地牢》游戏本质是一个地牢探索游戏，通过移动、攻击怪物，闯关并到达终点；特色在于所有移动、攻击操作只能在其背景音乐的节拍上进行。参考《节奏地牢》，本项目实现了包含节拍判断的移动、攻击，实现了游戏中的多种怪物和陷阱地形等。

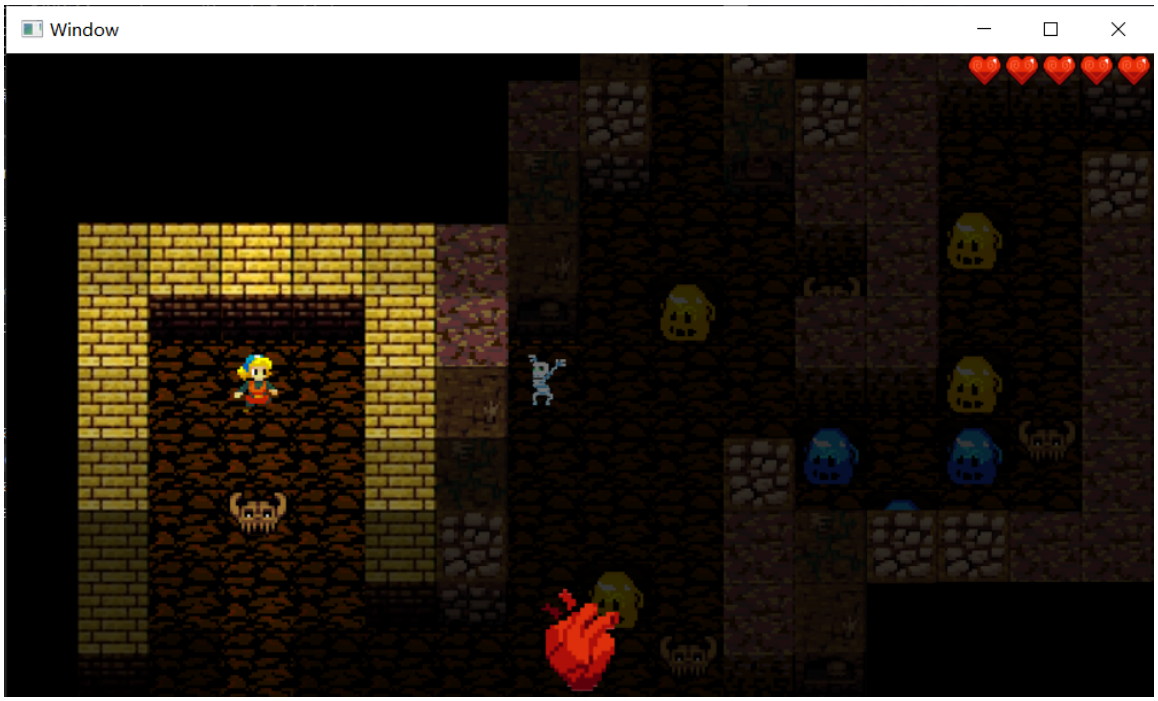
2 开发环境

- Windows 10
- masm32
- Visual Studio 2019, Community

3 实现原理

从游戏有什么讲起，再阐述每个功能点的实现原理。

3.1 绘图



主要使用 Gdi 进行绘制，所有绘图素材均是 BMP

地形 将整个窗口分成 16*9 个 50px*50px 的方格，在网格绘制一个 50*50 的素材即可画出地面；以网格底部线为基准绘制一个 50*82 的素材即可画出立体墙面。绘图时从上到下逐行绘制，因此下面行绘制的墙可以挡住上面行绘制的素材，达到看上去立体的效果。代码中 _PaintObjectAtPos 函数即抽象了在某个网格位置上绘制一个素材的逻辑。

人物、怪物 在绘制某个网格上的地形时，判断该位置上是否有玩家或敌人，有则在该位置额外绘制一个敌人、玩家的贴图，玩家和敌人的贴图是不包含透明信息的 50*50 的 bmp，但却不会挡住该网格的地面贴图，这由 TransparentBlt 实现

心脏、血量红心 直接计算其在窗口中的绝对坐标并绘制。心脏会在每个节拍上跳动，这通过在对应时间点上将小心脏替换成大心脏再替换回来实现

玩家居中 在玩家移动时，游戏将尽量让玩家居于窗口的中部，这通过将当前绘制的窗口抽象成一个在整个地图上滑动的视窗实现，玩家移动将导致这个视窗和玩家同步移动，因此玩家总位于窗口中心，看上去的效果是地面相对整个窗口移动了。

阴影 离玩家越远，地图越黑暗，且被墙体遮挡的地区是完全黑暗不可见的，这些都是地牢类游戏渲染气氛的常用手段，本项目也近似实现了这些效果。方法是为每个绘制的位置通过其离

玩家的距离和被墙体遮挡的程度计算一个 alpha 值，然后通过 AlphaBlend 将一个纯黑的 Mask 按照这个 alpha 值渲染到绘制位置上，alpha 值越高，黑暗程度越重。

避免闪烁　Gdi 绘图容易出现闪烁问题，而本项目通过双缓冲完全解决了闪烁。绘图时先创建一个和窗体一样大的缓冲区，再对每个要绘制的贴图创建一个小缓冲区并通过 SelectObject 将贴图放到小缓冲区中，将贴图从小缓冲区贴到大缓冲区中对应的位置，最后将大缓冲区一次性贴到窗体中（而非每次都把一个小缓冲区贴到窗体中）。

3.2 音乐与节拍

音乐通过 PlaySound 播放。

事先用其他工具计算出节拍间的时间间隔，然后用 SetTimer 设置计时器，每到节拍的时间点上就触发一次 WM_Timer 事件，在这个事件里，将会完成玩家移动、怪物移动、攻击等等游戏相关的状态更新 (在 updateStatus 中实现)，上述状态更新完成后，触发 InvalidRect 以重绘窗口。不过由于 Windows 的 Timer 并非精确计时，加上 WM_Timer 的优先级较低，因此有时候会出现丢拍，目前没有很好的解决办法。

3.3 游戏逻辑

用户的输入会改变玩家的 nextStep 变量，在每个节拍上触发的 updateStatus 函数中，按以下逻辑执行

1. 通过 player.nextStep 判断玩家下一步将要去的的位置 (nextPos)
2. 通过 checkCollision 函数判断 nextPos 是否可达（有没有墙体、敌人等阻挡），如果可达，前往 3.，否则前往 4.
3. (可达) 更新玩家的位置为 nextPos，清空 nextStep。前往 5.
4. (不可达) 不更新玩家位置，清空 nextStep。判断阻挡玩家的物体类型，如果是可以挖掘的墙壁（土墙），则挖掘之；如果是怪物，则攻击之。前往 5.
5. 遍历敌人列表，对每个还活着的敌人，通过 decideNextStep 函数，根据敌人的**移动类型 ***计算其 nextStep
6. 通过该敌人的 nextStep 判断其下一步将要去的位置 (nextPos)
7. 通过 checkCollision 函数判断 nextPos 是否可达（有没有墙体、敌人、玩家等阻挡），如果可达，前往 6.，否则前往 7.
8. (可达) 更新敌人的位置为 nextPos，清空 nextStep。前往 10.
9. (不可达) 不更新敌人位置，清空 nextStep。判断阻挡敌人的物体类型，如果是玩家，则攻击之。前往 10.
10. 判断玩家血量和位置，如果位置为陷阱，则将其血量减为 0；如果血量为 0，则触发游戏结束函数；如果位置为楼梯，则触发游戏胜利函数
11. 如果游戏没有结束或胜利，则在下一个节拍上重新从 1. 开始执行

*** 移动类型**　按怪物区分，有以下几种

- **蝙蝠**　2 拍动一次，上下左右随机移动
- **蓝色史莱姆**　2 拍动一次，上下来回跳动
- **金色史莱姆**　1 拍动一次，顺时针转圈跳动
- **骷髅**　2 拍动一次，追逐玩家，采用贪心追逐算法

3.4 信息储存

玩家和敌人用结构体储存，地图用一个一维数组 (大小为 MAP_WIDTH*MAP_HEIGHT) 储存，每个位置为一个数字，储存了地形信息

3.5 资源管理

程序打开时加载所有资源；程序关闭时删除所有资源。具体逻辑在 _InitResources 与 _DestroyResources 中实现

4 难点和创新点

4.1 难点

1. 一开始整体上没有一个入门 win32 编程的教材，且网上资料较少，以至于不知道从何入手。后通过《Windows 环境下 32 位汇编语言程序设计》(罗云彬著) 一书 + 微软的 win32 api 文档入门
2. 一开始遇到了严重的绘图闪烁问题，后通过双缓冲解决
3. 节奏需要时间上的严格精确，但 Timer 无法做到十分精确，所以本游戏最关键的节奏其实实现得不完美（丢拍，错节奏），且至今没有解决，最后修改成了每个节拍怪物都会移动，而玩家也可以在非节拍点上移动

4.2 创新点

游戏中阴影的绘制、怪物血量、人物血量、被攻击时的闪屏、跳动的心脏等都可以作为创新点，让这个游戏的完成度显得更高。

5 小组分工

- **周雨豪**　组织协调工作；调查 win32 音频播放方法；实现节奏相关部分和用户输入处理
- **叶宸**　实现主要的游戏逻辑 (如 updateStatus 等)；地图设计；敌人设计；文档编写
- **徐源远**　调查 win32 的绘图方法；实现主要的绘图函数 (如 _PaintObject 等)