《软件设计与开发实践 A》课程报告

任务名称：声音交互游戏《输出全靠吼》游戏开发

班　　级：5　　组号：8

学　　号：180110518

姓　　名：胡智胜

学年学期：2019 年秋季

任课教师：李旭涛

哈尔滨工业大学（深圳）计算机科学与技术学院

前 言

《软件设计与开发实践 A》是基于自选项目的实践训练，学生将综合利用《集合论与图论》、《数据结构》、《算法设计与分析》、《高级语言程序设计 I 及 II》等方面的基本概念、原理、技术和方法，开展实际应用问题设计求解和对应系统软件开发两大方面的实践。

通过本课程的学习、训练和实践，引导学生熟练掌握问题设计求解和软件编程开发的一般过程、相关技术、方法和途径；训练综合运用所学的理论知识和方法独立分析和解决问题，提高问题分析、问题求解和软件开发能力；培养学生能够针对实际问题，选择适当的数据结构、设计有效算法，提高程序设计的能力和编码质量；训练和学会用系统的观点和软件开发一般规范进行软件设计开发，培养软件工作者所应具备的科学工作方法和作风，提高工程素质；并通过采用团队协作、构建项目组的形式，来培养学生的团队合作与交流能力。

本课程要求学生分组进行（每组 1~3 人），通过一定的调研来自行结合实际应用需求来选题，并由任课教师来对学生选题做筛选评定。要求所设计开发的软件具有一定的实用性和系统完整性，要有较友好的图形交互操作界面，并对输入数据有较强的完整性约束，要以用户需求作为出发点来设计软件界面和功能模块。本课程主要教学环节包括：学生自选任务、软件开发、软件验收、任务报告撰写提交和任务资料整理归集等。

教师评语：   报告成绩：

1. 选题背景与应用意义

（描述选题的背景、所针对的具体实际问题及任务所体现的实用性价值和意义所在等）

①选题背景

如今市场上各种人体交互的小游戏越来越受到大众的欢迎,特别是 VR 设备和 AR 技术的推广，使游戏带给玩家的浸入感越来越强烈，电影《头号玩家》就向我们展示了未来游戏的神奇与无限可能，游戏带给玩家的浸入感越强，这个游戏的玩家群体粘度也就越高，游戏也就自然而然地会获得成功。而游戏提升交互感的方式主要则是人体的动作或声音的交互，前不久一款名叫《八分音符酱》的游戏引起了许多游戏玩家的注意，其中也包括我，这款游戏主要是一个不停向前走的小人，玩家传给麦克风的声音越大，小人跳跃的高度越高，以此来闯关的冒险游戏。这款游戏有趣之处就在于玩家发出的声音千奇百怪，玩的时候需要大声吼叫，既能释放生活中的压力，也十分搞笑，起到很好的娱乐作用。于是我心里也萌生了开发一款类似用音量来控制游戏角色的游戏的想法。在和组员的交流中我们探讨了要做什么类型的声音交互游戏，其中便想到了之前在小游戏网站上玩过的一款名叫《混乱大枪战》的多人小游戏，这个游戏的玩法是：控制人物射击，被击中的玩家会后退一定的距离，而最终的胜利则是要把对方射出平台掉落至屏幕外才算成功。于是我们结合这款游戏的玩法和声音交互这个概念的结合，产生了我们的《输出全靠吼》游戏的计划。

《输出全靠吼》的玩法是：依旧是键盘控制人物移动，但是人物的射击改用声音触发，当玩家对着麦克风吼叫时，人物即会向前开枪，吼叫的声音越大，子弹击退敌人的效果越强，同时我们还打算采用连按两下方向键冲刺以此来抵消一部分冲击力的玩法增强这款游戏的刺激和技术性。此外，人物还可以选择自己的技能，技能只能携带一个，但是开始前玩家需要自己为发动这个技能录入声音，例如，为技能 A 录入“冲击波”的语句，这样当玩家说出“冲击波”三个字时人物即会发动技能 A。

②实际问题及应用意义

目前市场上音乐交互的游戏仍然比较稀少，国内许多游戏产商基于公司盈利等原因，往往不敢大力投资开发玩法特别新颖的游戏，因此国内市场上的许多游戏的模式都类似，盈利模式也类似。针对这样的问题，我们开发的声音交互的新游戏就能带给用户一种全新的体验，并且采取的“声音越大攻击力越大”的游戏方式能够有效地带给用户正向反馈，使用户的游戏体验极大地改善。

一款游戏的好坏可以从多方面评价，许多人会认为一款游戏的开发价值在于能不能为游戏公司带来很好的商业收益，但相信更多的年轻人、有梦想的游戏开发者会认为一款游戏只要是能为每天疲于工作的你我带去缓解压力的时光，抑或是在网络的世界里回忆起自己曾经每天和朋友们一起战斗的热血青春，或者是，成为一代人的青春回忆，这样的游戏便是好游戏，才是有意义的。我们设想的《输出全靠吼》这款游戏，是作为和朋友一起聚会时玩的游戏，它也许不适合一个人单独玩很久，但是相信在朋友聚会娱乐时玩一玩这样需要大吼大叫而又紧张刺激的游戏一定会给聚会时光带来更多欢笑，留下更美好的记忆。而这，就是游戏存在的价值和它的应用意义。

2. 需求分析

（根据任务选题的要求，充分地分析和理解问题，明确用户要求做什么？完整性约束条件是什么？运行环境要求、图形操作界面要求等）

① 功能性需求

需要设计出一款能够利用声音进行交互的横板动作游戏，游戏中的角色还是采用键盘控制移动，角色的攻击则是由麦克风监听用户的声音，当音量较大时则发动攻击，且攻击的效果随声音变大而变得更明显；每个角色拥有一个技能，技能需要用户提前录入一段语音，每当用户对着麦克风说出之前为技能录入的语音时，游戏中的角色就会发动这个技能。角色发动攻击会产生一道“冲击波”，当角色受到“冲击波”攻击后会产生明显的后退效果，若角色被攻击到地图外则角色的生命值应该减一，并且重生。每个重生的角色应具有几秒的“保护时间”，处在“保护时间”的角色受到攻击后的后退效果较弱。

同时为了使游戏更加的好玩，还应设置闪避按键，每当用户按下 shift 键角色就可以向前闪避，期间不会受到“冲击波”的攻击。

游戏中总共设计三个技能。第一个技能是“隐身”，选择该技能的角色在发动技能后，在自己的画面中看到的是自己的角色被一块“灰色”的半透明图片遮盖，代表当前处在隐身状态，而在其他玩家的界面中这个角色将会“消失”，每当这个角色攻击或被受到攻击时，角色应短暂地“现身”被其他玩家看见。第二个技能是“造墙”，选择该技能的角色在发动技能后将在自己角色的前方造出一面墙，这面墙将会自动向前缓慢移动，并且可以阻挡飞行的“冲击波”和游戏角色。第三个技能是“飞行”，选择该技能的角色在发动技能后，在角色的背后会显示出一双“翅膀”，此时用户按下飞行键后角色可以缓慢向上飞行。

游戏中总共设计十张地图，每张地图的主题各不相同，包括“草原”、“太空”、“沙漠”、“火山”等，每张地图的样式应根据对应主题来设计，地图中的供玩家站立的“地图块”分布各不相同。

这款游戏应具有单人模式和多人模式。单人模式里和电脑 AI 进行游戏，AI 角色应当根据用户控制的角色位置来进行相应的移动，并在恰当的位置对玩家发动攻击，AI 角色的动作应自然流畅，尽可能地和玩家控制的角色行为一致。多人模式则应该是和其他玩家通过局域网联机进行游戏，局域网联机时应当保证游戏画面的一致性，包括角色的动画、位置，子弹的生成、消失、位置。局域网联机时，玩家可以查看附近局域网中的对应“游戏房间”，玩家应该可以反复进出同一局游戏，并且局域网对战的房主退出游戏后房间应当还存在，并且房主可以反复进出自己开创的游戏。

② 非功能性需求

运行这款游戏需要在 windows 系统，要求该系统能够运行 exe 文件即可。不需要用户的电脑已安装 python 环境和相关的第三方包。

运行该游戏时，文档里需要包含游戏运行所需的相关资源(图片、音乐等)。

用户交互界面的设计应当简洁明了、美观。局域网游戏的房间列表应当清楚明白地显示出房间房主选择的地图和当前房间里的人数。

③ 设计约束

游戏的游戏逻辑用 python 编写，利用 pygame 库来进行游戏开发。

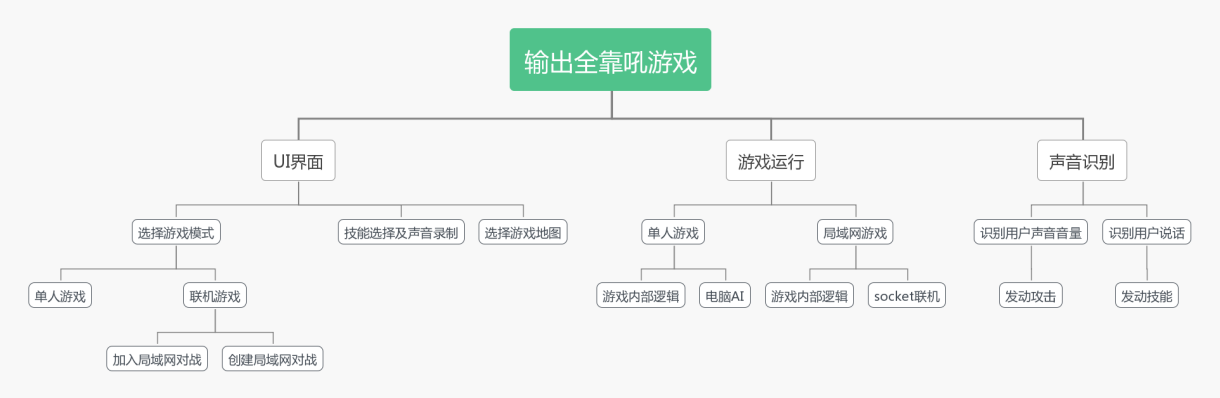
游戏的 UI 界面用 html 编写，与 python 代码的交互部分利用 eel 库进行开发。

游戏的语音识别部分用 python 编写，利用 pyaudio 与电脑的麦克风模块交互，利用 sci 库开发语音识别的功能。游戏中的语音识别不需要太精确只需要把用户的发动攻击的声音和发动技能的语音分清即可。识别玩家发动技能采用 DTW 算法来识别语音。

3. 系统主要功能设计

（根据需求分析来详细设计软件系统的主要功能模块，要求画出功能模块图，含子功能模块图，并给出详细文字描述）

系统结构图：



游戏分为 UI 界面、游戏运行、声音识别三个部分。

UI 界面负责游戏模式的选择、技能声音的录制、游戏地图的选择、游戏的启动、加入局域网游戏、创建局域网游戏。

游戏运行部分则主要是游戏运行时的各部分的逻辑组织，分为单人游戏部分和局域网游戏部分。单人游戏部分分为游戏内部逻辑和 AI。局域网游戏部分分为游戏内部逻辑和游戏联网部分。

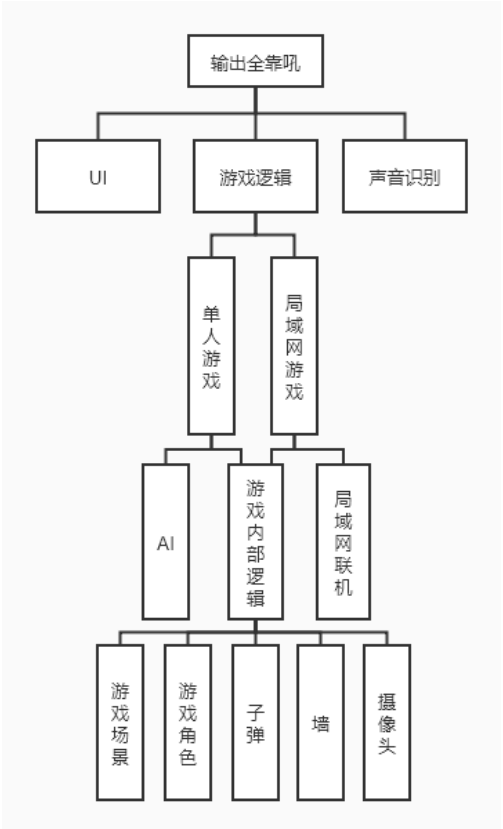
游戏内部逻辑包含游戏场景类、游戏角色类、子弹类、摄像头类、墙类。

AI 部分则负责单人游戏模式时电脑 AI 角色的移动和攻击的控制逻辑。

游戏联网部分则负责局域网游戏时各种数据传输的逻辑。

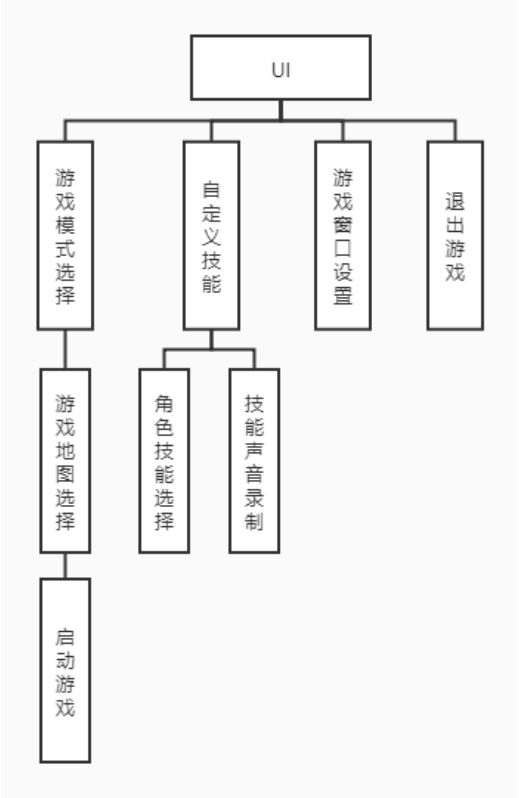
声音识别部分负责游戏时随时监听电脑麦克风信号，当用户说话时则开始识别，每当用户声音大于一个阈值时用户控制的角色发动攻击，当用户说出技能语音时用户控制的角色发动技能。

功能模块图：



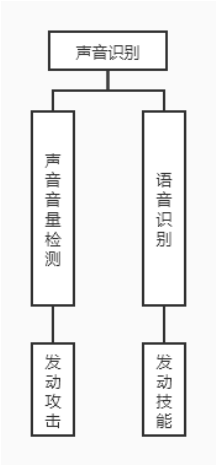
子功能模块：

① UI 界面



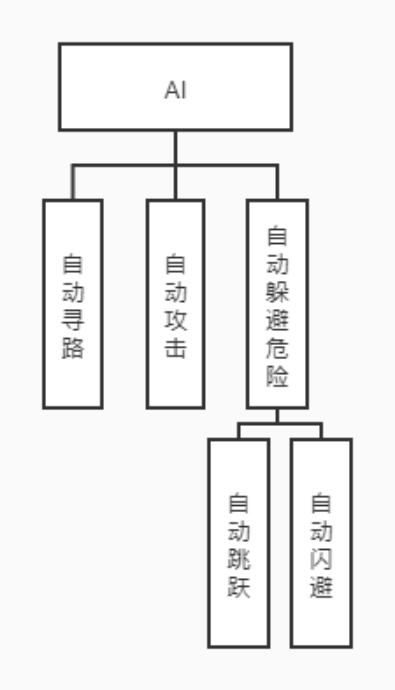
UI 界面首先会展示一个开场动画，动画结束后将会显示游戏菜单，菜单供玩家选择单人游戏还是局域网联机、游戏窗口是全屏还是窗口模式、角色技能选择及录制语音、退出游戏。其中角色技能设置这部分会让玩家先选择一个技能，再点击相应按钮开始录制两次技能发动语音，录制完毕即可在游戏里通过刚说出的声音来发动所选技能。

② 声音识别



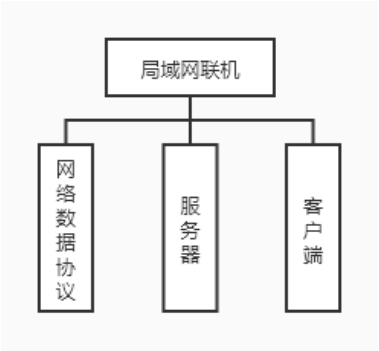
声音识别部分首先是使用 pyaudio 获取系统麦克风的数据，同时运行两个线程，一个线程负责检测声音音量大小，若音量大于设定的阈值则让角色发动攻击，发射出来的冲击波的攻击力与音量大小成正比；另一个线程负责语音识别，每当声音大于设定的阈值，则开始录制一段与玩家之前录制的声音相同时长的语音，用 DTW 算法进行声音识别，若识别的结果认为声音匹配则让角色发动技能。

③ 游戏 AI



单人游戏模式在地图开始加载时，首先会将每个平台分割成若干个点，例如把一段较长的平台分为左、中、右三个点，用这些点进行 Floyd 算法计算出两点之间的最短路径并存储起来。而 AI 每次会根据自己到玩家的距离随机选择一个竖直高度与玩家角色等高、水平位置在玩家和自己之间随机的一个位置的点进行最短路径移动（更好的模拟人的控制）；每次当玩家和自己的竖直高度相差小于一个阈值时 AI 便会对玩家发动攻击；每次当 AI 角色检测到自己人物的垂直下面没有平台可站立时便会进行“躲避危险”操作，具体做法是随机的跳跃和朝向最近的平台进行闪避。

④ 局域网联机



网络联机部分采用 socket 架构，网络数据协议 protocol 负责如何打包和解析数据，例如：规定前四位数据代表数据包的长度，前两位数据代表当前的数据类型等等；服务器负责接收数据和发送相应数据给各个客户端；客户端负责接收服务器发来的数据包和发送数据包给服务器。特别当玩家中途加入一个对战时，服务器要将房间已有的玩家发送给新加入的玩家，并把新玩家发送给服务器之前的其余玩家。

4. 核心算法设计与分析

（根据软件功能设计，概述所用到的主要数据结构；用伪代码或程序流程图的形式来详细描述核心算法的功能及过程，并定性分析其时间、空间复杂度）

数据结构：

UI 界面使用 HTML+python 的跨平台开发，主要借助 python 的 eel 库来实现，因此 UI 界面使用到了 HTML 中常见的 DOM 树结构；

在 UI 界面中显示地图列表采用的是队列结构，总共有十张地图，每次只显示 5 张，玩家点击空白区域即可循环展示全十张地图；

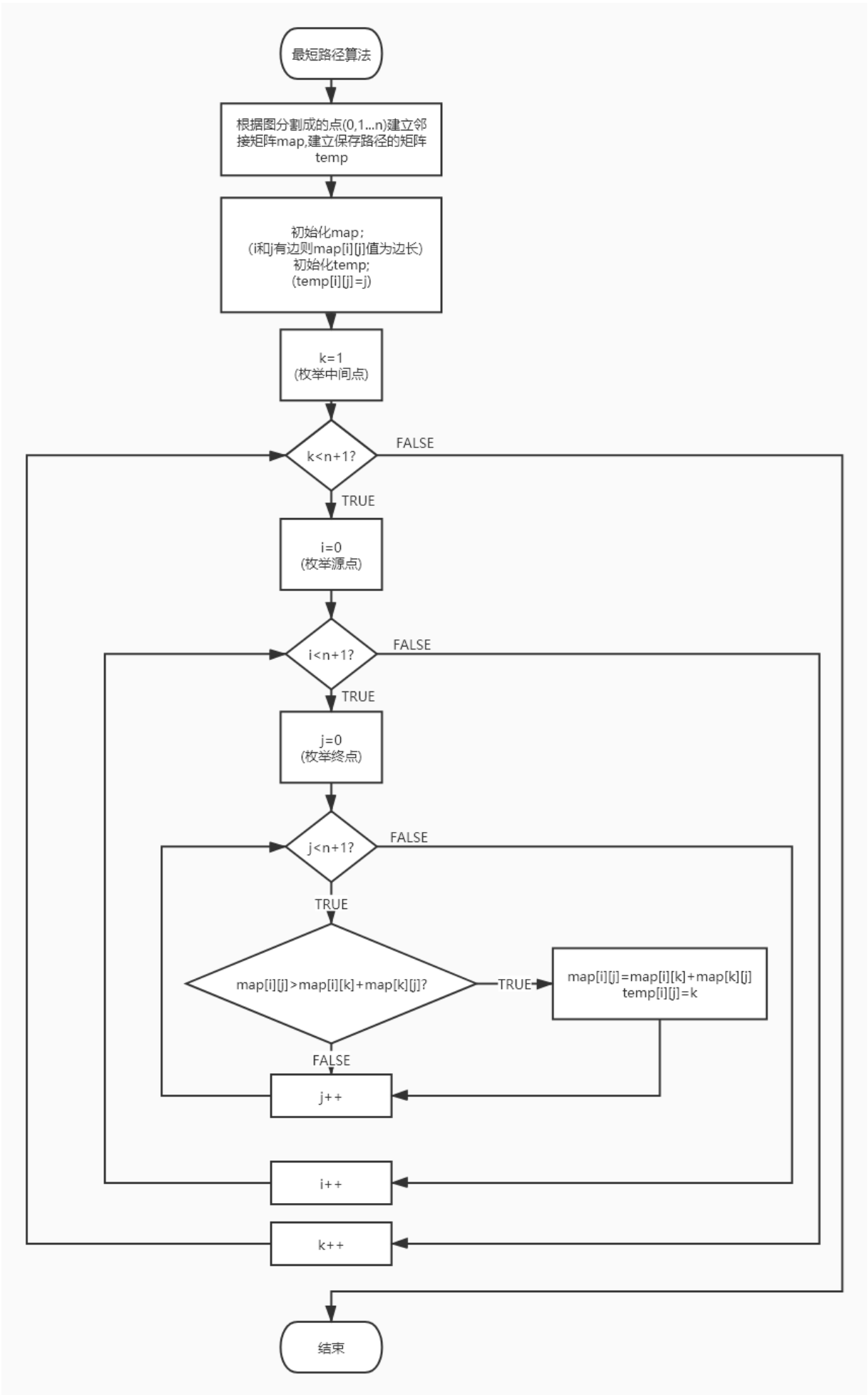
在游戏中 AI 角色的自动寻路算法用到了图的最短路径算法，我们把图分割成了若干点组成，因此用到了图的邻接矩阵的数据结构；而为了存储点到点的路径，又采用了栈的存储结构。

AI 角色每次自动寻路都会利用到玩家角色的坐标，而攻击方式是发射子弹，因此重点关注玩家角色的纵坐标，根据纵坐标 AI 角色会得到一系列在这个纵坐标上的点然后 AI 角色移动到其中的某个点对玩家角色发动攻击即可，因此我们用到了链表的结构，每个节点存储了一个纵坐标值，而这个节点又相当于是另一个链表的表头，其中的节点则代表是同一个 Y 值上的点。因此 AI 角色便可根据某个纵坐标值查询出在这个纵坐标附近的一组点。

游戏在运行时，为了方便进行统一的绘制和碰撞检测等操作，我们将同一类物体存储到了同一个线性表中，具体实现是采用了 pygame 中的 group 结构。

核心算法：

① 最短路径：


最短路径算法
根据图分割成的点(0,1...n)建立邻接矩阵map,建立保存路径的矩阵temp
初始化map；
(i和j有边则map[i][j]值为边长)
初始化temp;
(temp[i][j]=j)
k=1
(枚举中间点)
k<n+1?
FALSE
TRUE
i=0
(枚举源点)
i<n+1?
FALSE
TRUE
j=0
(枚举终点)
j<n+1?
FALSE
TRUE
map[i][j]>map[i][k]+map[k][j]?
TRUE
map[i][j]=map[i][k]+map[k][j]
temp[i][j]=k
FALSE
j++
i++
k++
结束

采用 Floyd 算法计算出两点之间的最短路径，并且在更新最短路径时采用一个二维数组来记录这个中间点。

把图用邻接矩阵 G 表示出来，如果从 Vi 到 Vj 有路可达，则 G[i][j]=d，d 表示该路的长度；否则 G[i][j]=无穷大。定义一个矩阵 D 用来记录所插入点的信息，D[i][j]表示从 Vi 到 Vj 需要经过的点，初始化 D[i][j]=j。把各个顶点插入图中，比较插点后的距离与原来的距离，G[i][j] = min(G[i][j], G[i][k]+G[k][j])，如果 G[i][j]的值变小，则 D[i][j]=k。在 G 中包含有两点之间最短道路的信息，而在 D 中则包含了最短通路径的信息。

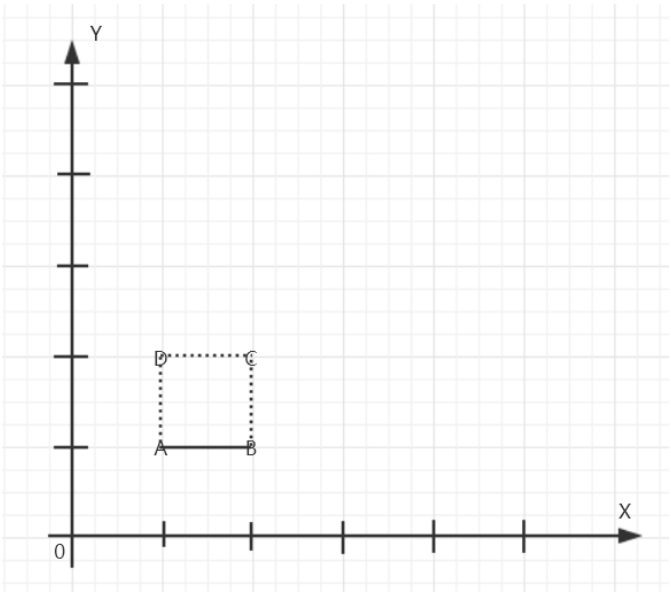
例如：map[1][3]初始值为 1 与 3 之间的边长（若无边则为一个大值），另设一个二维数组 temp,temp[1][3]初始值为 1,若 1->4->3 路径更短,则会在 Floyd 算法在 k=4 时,会更新 map[1][3]的值，并且将 temp[1][3]的值置为 4，代表 1->4->3 为最短路径；若是有更多的中间点，例如 1->5->4->3 为 1 到 3 的最短路径，则最终结果会 temp[1][3]的值是 5，temp[5][3]的值是 4，因此系统在找 1 到 3 的最短路径时，首先把 1 压入栈，再找到 5，把 5 压入栈，查 temp[5][3]的值 4，再把 4 压入栈，差 temp[4][3]的值是 4，不用再查询 temp 数组，因此压入终点 3，随后依次弹出栈，3、4、5、1，将这个顺序反过来即是 1 到 3 的最短路径。

此算法的时间复杂度为 O(n^3)；空间复杂度为 O(n^2)。

② 语音识别：

声音识别分成了音量检测和语音识别两部分，其中音量检测只需要获取麦克风数据即可，这里主要写语音识别算法。

语音识别算法采用的是机器学习算法中的 DTW 算法，识别过程主要是：将用户输入的语音的每一帧转换成 39 个属性值的一维向量，并把一帧看作坐标轴上的一个点，因此由用户当前输入的语音和之前录制的样本语音，我们可以画一个坐标轴，X 轴上每个点都是用户当前输入的语音的一帧，Y 轴上每个点则是用户之前录制的语音的一帧。



假设 A 是[1,1]的点，B 是[2,1]的点...则首先计算 A 对应两帧语音的向量的欧式距离，接着再算 B、C、D 选择其中欧式距离最小的点将其与 A 连接，图中假设 B 的计算距离最小。这样经过多次直线折叠之后，我们再将所有的点的距离之后最为最后判断的距离值，若这个相加的距离值越小，则代表两段语音越类似。我们把用户当前说的一段语音与之前预先录制的语音进行比对，若距离值之和小于我们设定的阈值，即可判断两个声音是相似的，应该让角色发动技能。

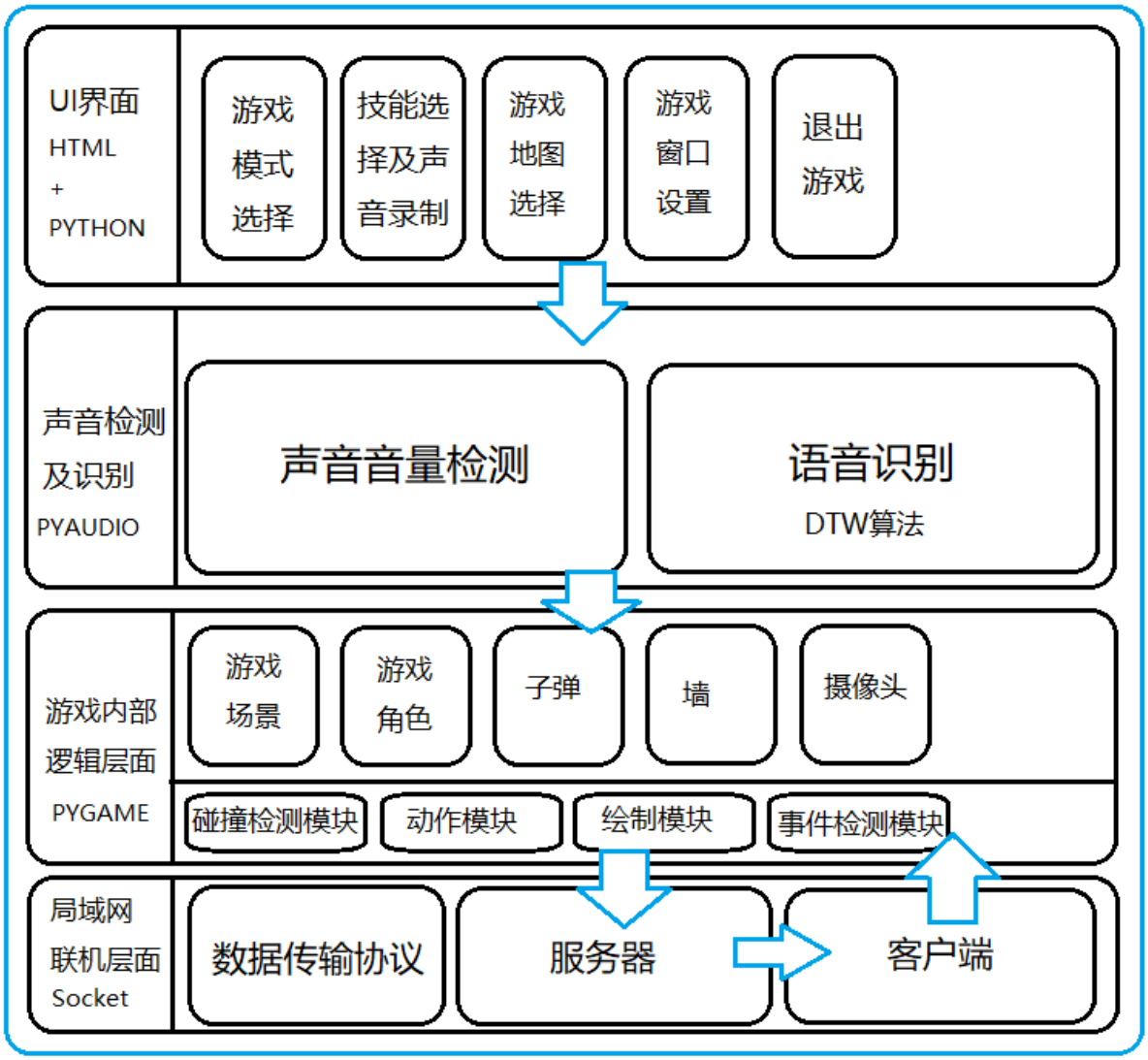
5. 系统核心模块实现

（给出编程语言、开发环境及支撑软件、软件系统架构；给出主要数据结构的定义代码以及核心算法对应的关键函数定义代码，包括输入参数的含义、函数输出结果等；核心函数的实现代码段及注解等；主要功能界面截图及对应文字概述等）

编程语言：python

开发环境：Microsoft Visual Studio

软件系统架构：



数据结构定义：

① HTML 网页 DOM 树：（示例）

```
<div id="page1_main" class="optionsWrapper">
    <div class="options">
        <a onclick="javascirpt: void (0); mapChoose(0)"><span>单人游戏</span></a>
        <a onclick="javascirpt:void(0);runMultiGame()"><span>多人游戏</span></a>
        <a onclick="javascirpt: void (0);setting()"><span>游戏设置</span></a>
        <a onclick="javascirpt: void (0);selfdefine()"><span>自定义技能</span></a>
        <a onclick="javascirpt: void (0);team()"><span>制作团队</span></a>
        <a onclick="javascirpt:void(0);exit()"><span>退出游戏</span></a>
    </div>
</div>
```

② 展示地图的循环队列：(JS 编写）

```
var map_page_queue = ['map_choose_page1', 'map_choose_page2'];
```

实现代码：

```
page.ondblclick = function () {
    var first_page = map_page_queue.shift();
    map_page_queue.push(first_page);
    console.log(map_page_queue);
    showMapPage();
}
```

③ 地图分割成点的邻接图：

```
matrix = []
```

实现代码：

```
for i in range(0, length):
    matrix.append([])
    for j in range(0, length):
        matrix[i].append([])

for i in range(0, length):
    for j in range(0, length):
        if i!= j:
            distance = int(math.sqrt(int(math.pow(points[i][0] - points[j][0], 2) + math.pow(points[i][1] - points[j][1], 2))))
            if i%2 == 0 and j%2 == 1:
                if i + 1 == j:
                    matrix[i][j] = distance
                else:
                    if distance < self.max_leap:
                        matrix[i][j] = distance
                    else:
                        matrix[i][j] = MAX
            elif i%2 == 1 and j%2 == 0:
                if j + 1 == i:
                    matrix[i][j] = distance
                else:
                    if distance < self.max_leap:
                        matrix[i][j] = distance
                    else:
                        matrix[i][j] = MAX
            else:
                if distance < self.max_leap:
                    matrix[i][j] = distance
                else:
                    matrix[i][j] = MAX
        else:
            matrix[i][j] = 0
```

当两点距离小于设定的阈值 self.max_leap 则将 matrix[i][j]设为对应的距离值，否则设为大值 MAX。

④ 存储最短路径的栈：

```
for v in range(v_num):
    for w in range(v_num):
        sequnce = []
        stack = []
        j = w
        if v <= w:
            while True:
                stack.append(distance[v][j])
                if distance[v][j] != v:
                    j = distance[v][j]
                else: break
            while len(stack):
                sequnce.append(stack.pop())
            sequnce.append(w)
        path[v][w] = sequnce
```

利用栈将 Floyd 算法记录的最短路径经过的每个点转化成列表。

⑤ 存储游戏对象的线性表：

```
88          self.bullets=pygame.sprite.Group()
89          self.walls=pygame.sprite.Group()
```

核心算法实现：

① 最短路径：

主要代码：

```
476     def floyd(self,matrix):
477         v_num = len(matrix)
478         copy = []
479         distance = []
480         path = []
481         for i in range(v_num):
482             copy.append([])
483             distance.append([])
484             path.append([])
485             for j in range(v_num):
486                 copy[i].append([])
487                 distance[i].append(-1)
488                 path[i].append([])
489 
490         for i in range(v_num):
491             for j in range(v_num):
492                 copy[i][j] = matrix[i][j]
493 
494         for k in range(v_num):
495             for i in range(v_num):
496                 for j in range(v_num):
497 
498                     if(distance[i][j] == -1):
499                         distance[i][j] = i
500                     if copy[i][j] > copy[i][k] + copy[k][j]:
501                         copy[i][j] = copy[i][k] + copy[k][j]
502                         distance[i][j] = k
503 
504         #shortpath: v->w
505         for v in range(v_num):
506             for w in range(v_num):
507                 sequnce = []
508                 stack = []
509                 j = w
510                 if v <= w:
511                     while True:
512                         stack.append(distance[v][j])
513                         if distance[v][j] != v:
514                             j = distance[v][j]
515                         else: break
516                     while len(stack):
517                         sequnce.append(stack.pop())
518                     sequnce.append(w)
519                 path[v][w] = sequnce
520 
521         for v in range(v_num):
522             for w in range(v_num):
523                 if v > w:
524                     #print(path[w][v])
525                     print(path[w][v])
526                     path[v][w] = path[w][v].copy()
527                     path[v][w].reverse()
528 
529         return path, copy
```

函数输入参数：

Matrix 是二维数组，是地图经过点分割后形成的邻接矩阵，存储的是两点的距离值。
函数返回值：
Path 是三维数组，path[i][j]存储的是由 i 到 j 最短路径所经过的各点。
Copy 是最短路径矩阵，copy[i][j]存储的是 i 到 j 的最短距离。

② DTW 语音识别算法：
主要代码：

```
    #计算最短路径的
    #tf: template_feature
    #nf: now_feature
def DTW(tf,nf):
    len_t = len(tf)                          #预存语音梅谱帧数
    len_n = len(nf)                         #实时语音梅谱帧数

    vector_len = len(tf[0])           #特征向量长度
    #print(len_t,'  ',vector_len)
    print("before:",len(tf),',',len(nf),',',vector_len)
    if len_t >len_n:
        for i in range( len_t-len_n):
            nf = np.append(nf,[nf[len_n-1]],axis=0)
        len_n=len_t
    elif len_t <len_n:
        for i in range( len_n-len_t):
            tf = np.append(tf,[tf[len_t-1]],axis=0)
        len_t=len_n
    D = CalculateDistance(0,0,tf,nf,vector_len)       #起点距离
    #d = []
    #d.append(D)

    i = 0
    j = 0
    #t =[1]
    #n = [1]
    #k = 0       #计数器，用来查看DTW的执行过程
    max_size = 65535
    min_size = 0
```

```
61
62          #print(len_t,',',len_n)
63          while (i<=len_n-1 and j<=len_t-1):
64              #if (j*2>=i and (2*(i-len_n+2)<=j-len_t+1) and i+1<len_n-1) :
65              if (j*2>=i and (2*(i-len_n+2)<=j-len_t+1) and i+1<len_n-1) :
66                  D1 = CalculateDistance(i+1, j, tf, nf, vector_len)
67              else:
68                  D1 = max_size
69              #if (j<=2*i and (0.5*(i-len_n+1)>=(j-len_t+2)) and j+1 <len_n-1) :
70              if (j<=2*i and (i-len_n>=2*j-2*len_t+1) and j+1 <len_n-1) :
71                  D2 = CalculateDistance(i, j+1, tf, nf, vector_len)
72              else:
73                  D2 = max_size
74              if i+1<=len_n-1 and j+1<=len_t-1:
75                  #print(j+1)
76                  D3 = CalculateDistance(i+1, j+1, tf, nf, vector_len)
77              min_size = min(D1, D2, D3)
78              if min_size ==D1:
79                  i = i+1
80              elif min_size == D2:
81                  j = j+1
82              elif min_size == D3:
83                  i = i+1
84                  j = j+1
85              D = D +min_size
86              #d.append(min_size)
87             # k = k+1
88              #print(k,'    ','( ',i,' ',j,' )' )
89          #x = [i for i in range(len(d))]
90          #plt.plot(x, d, label="distance")
91          #plt.legend()
92          #plt.show()
93
94          print(D)
95          return D
```

该函数接收代表两段音频的特征值数据 tf 与 nf，计算他们的距离，距离越小代表这两段音频越相似。

函数输入参数：

Tf 是用户预先录制的技能语音提取出来的特征值。

Nf 是当前用于说话的语音提取出来的特征值。

函数的返回值：

D 是两段语音的距离，距离越小代表两段语音越相似。

③ 声音音量检测算法：

主要代码：

```
106     #实时监测麦克风
107     #volume为声音阈值
108     def MonitorMicrophone(volume=800):
109         p = pyaudio.PyAudio()
110         stream = p.open(
111                 format = FORMAT,
112                 channels = CHANNELS,
113                 rate = RATE,
114                 input = True,
115                 frames_per_buffer = CHUNK,
116             )
117 
118         print ('录音开始')
119         for i in range(0, 100):
120             data = stream.read(CHUNK)
121             audio_data = np.frombuffer(data, dtype=np.short)
122             temp = np.max(audio_data)
123 
124             if temp > volume :
125                 print ("检测到信号")
126                 print ('当前阈值：',temp )
127                 return temp
128         return False
```

该函数读取麦克风当前的数据，返回一个当前音量最大值。

函数输入参数：

Volume 是返回信号的阈值，只有当当前音量值大于 volume 时才会返回当前的音量值，否则返回 False。

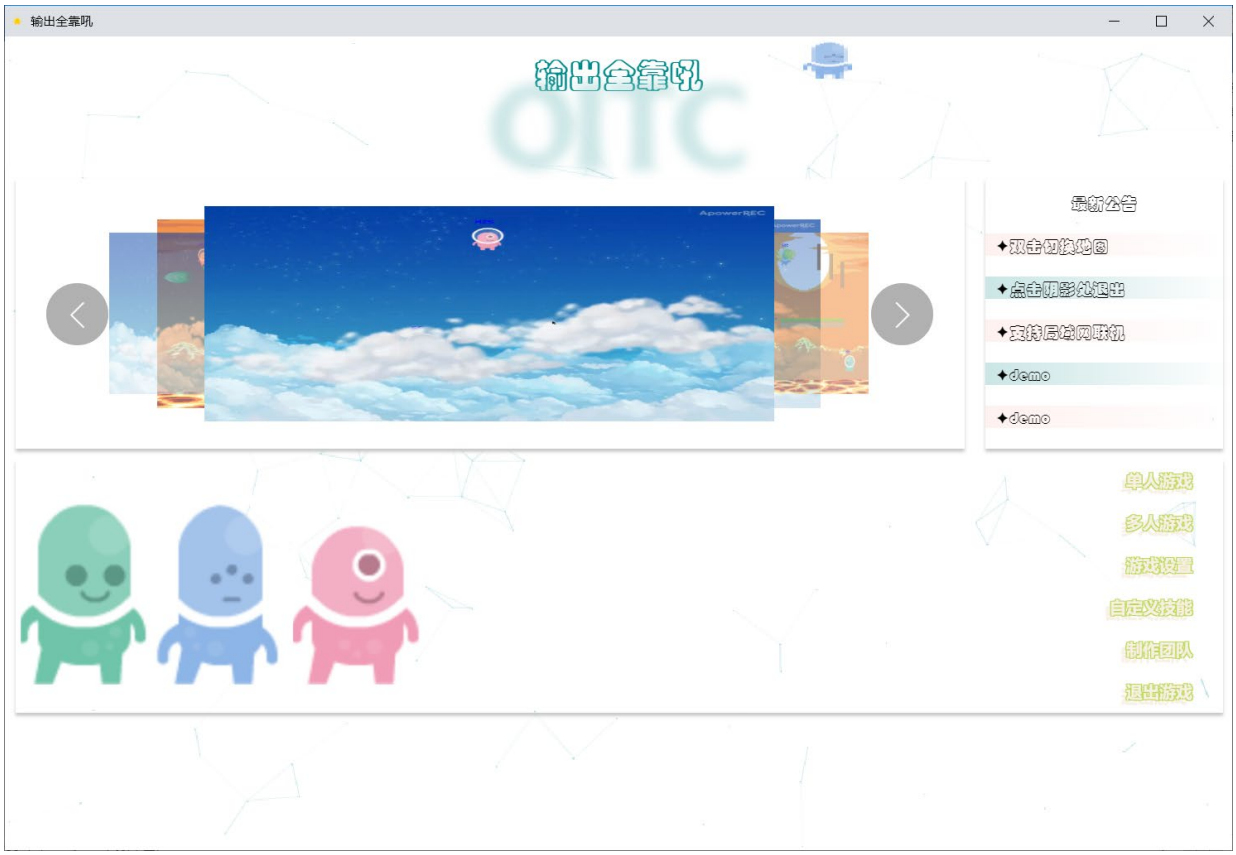
函数返回参数：

Temp 是从麦克风数据流 stream 读取的 CHUNK(代码里设置为 1024)个数据块中最大值，代表当前读取到的最大音量。

主要功能界面截图：

① 开场动画

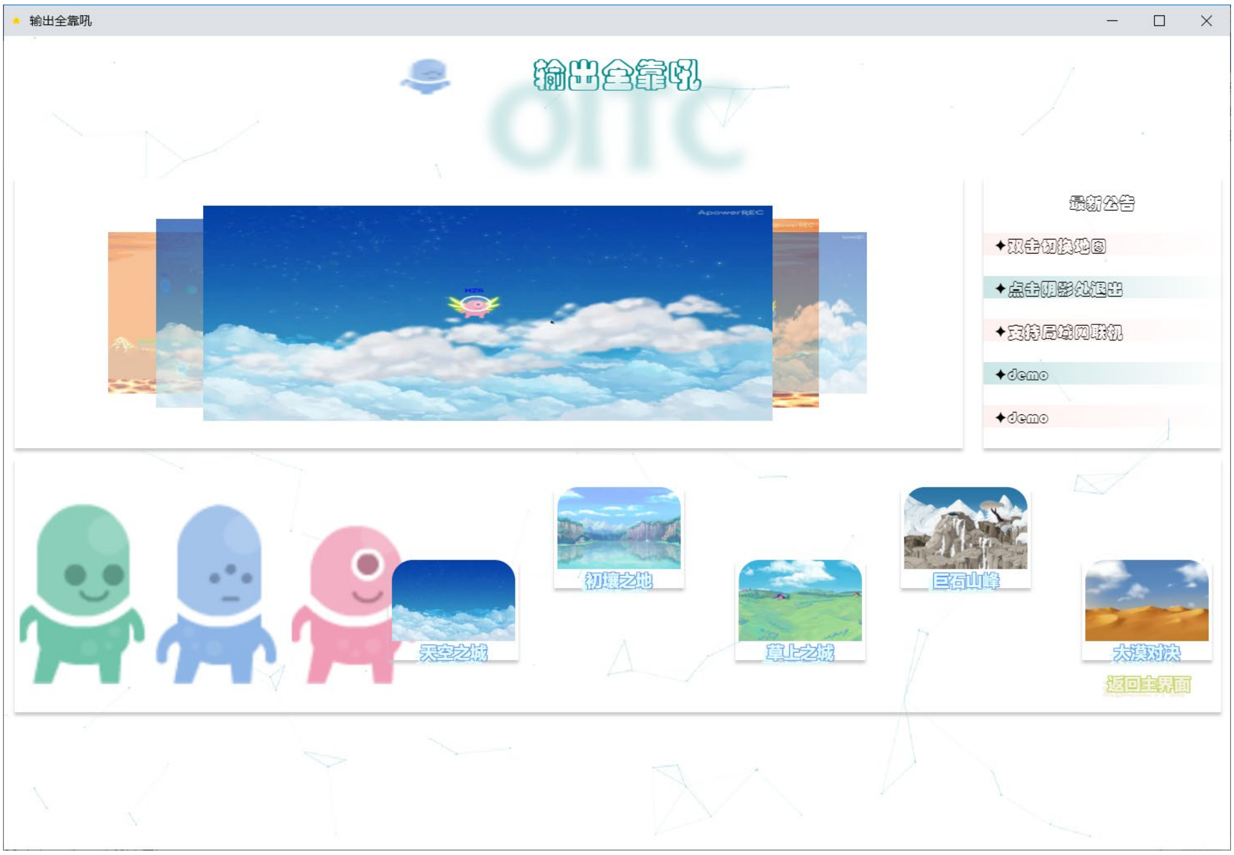
② UI 主界面



③ 技能选择及录制声音

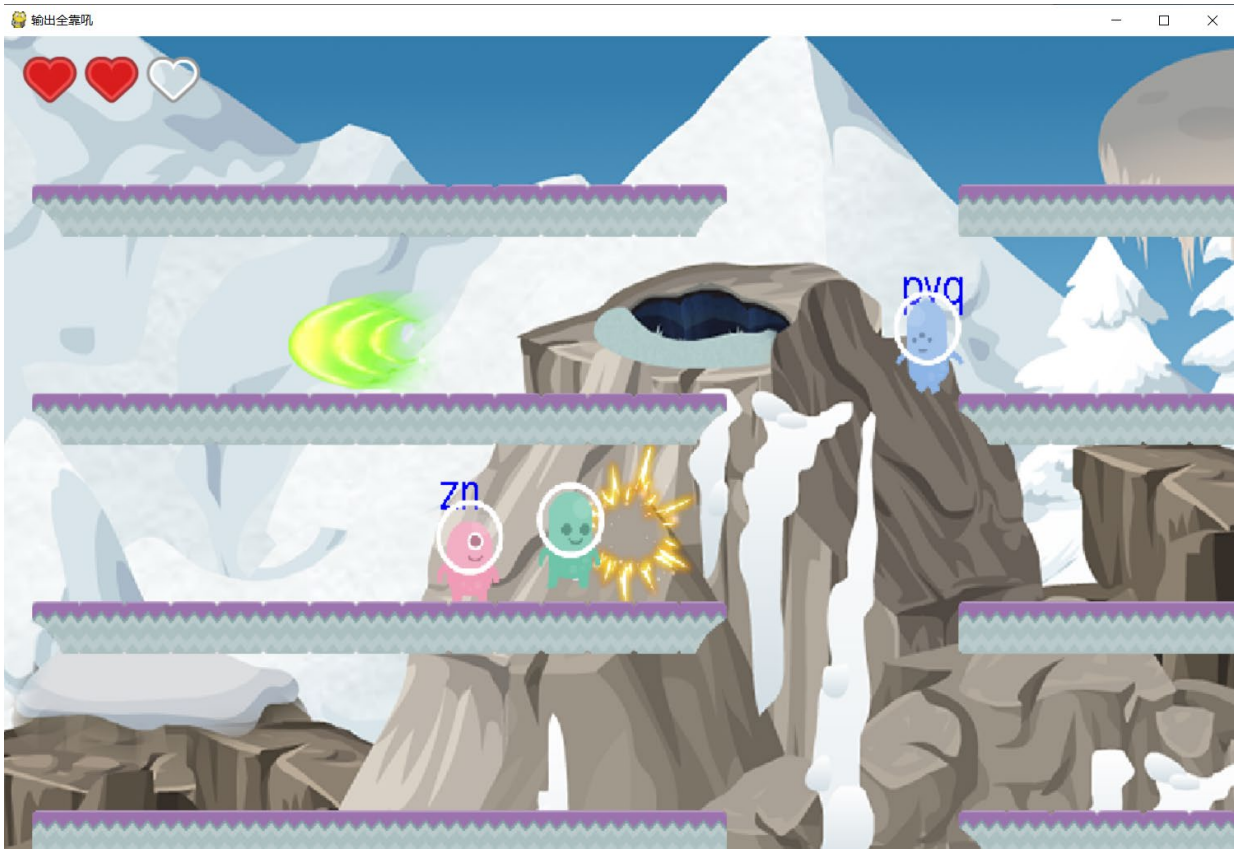

输出全靠吼
输出全靠吼
最新公告
✦双击切换地图
✦点击阴影处退出
✦支持局域网联机
✦demo
✦demo
设置技能
返回主界面
暗影身法

④ 地图选择


输出全靠吼
输出全靠吼
最新公告
✦双击切换地图
✦点击阴影处退出
✦支持局域网联机
✦demo
✦demo
天空之城
初始之地
草上之城
巨石山峰
大漠对决
返回主界面

⑤ 人物选择

⑥ 游戏画面



6. 调试分析记录

（软件开发调试过程中遇到的问题及解决过程；核心算法的运行时间和所需内存空间的量化测定；符合实际情况的数据测试，算法及功能的改进设想等）

遇到的问题：

① 局域网联机模式下不同电脑上的游戏画面不同步

解决过程：

在最初的联机测试中，不同电脑上的子弹飞行速度以及玩家位置有明显的不同步问题，特别是自己控制的角色在跳跃起来时在其它玩家的画面中出现向下抖动的现象。

经过排查，我们发现由于不同性能的电脑运行速度不同，而 pygame 中的画面更新与代码运行速度有关，这就导致了电脑性能较好的电脑运行速度快，子弹位置变化频率高，导致子弹在画面中飞行速度大。为了解决这个问题，我们在代码中调用了 pygame.time.Clock()模块，主动限制了代码运行帧率，这就会让性能较好的电脑上的运行速度和性能较差的电脑一样。

而自己控制的角色在其他玩家画面发生抖动现象则是由于自己的角色每帧都在给服务器发送自己角色的位置坐标，而在其它玩家的游戏中，这个角色还有自己的重力加速度计算，也就是说这个角色有两次向下的位置变化，一次是在自己玩家电脑上向下移动后发送给服务器去更新其他玩家画面中该角色的坐标，而其他玩家的游戏中这个角色还有自己的加速度，因此会出现其他玩家角色抖动的现象。将角色向下重力下落的逻辑中我们设置了判断该角色是否是这台电脑用户控制的角色，否则不更新该角色的由于重力加速度导致的位置变化。

② 启动游戏发生闪退

解决过程：

在开发后期成员间开发的内容进行整合测试时，发现启动声音音量检测和语音识别的线程时，游戏有几率发生不报错但是直接结束整个游戏的现象。

经过排查，这个问题发生的原因与 python 进程管理有关，我们将启动声音音量检测和语音识别两个进程的时间做了间隔处理，并且让两个进程分别启动了之后玩家才可进入游戏，在这期间游戏显示“正在加载请稍等”的画面优化 UI，之后游戏闪退的现象就不再发生。

核心算法量化测试：

使用 python 的 psutil 模块查看运行前和运行中的内存情况；

```
35  def getMemCpu():
36      data = psutil.virtual_memory()
37      total = data.total #总内存,单位为byte
38      free = data.available #可用内存
39      memory =  "Memory usage:%f kb"%(data.used/1024)+" "
40      cpu = "CPU:%0.2f"%psutil.cpu_percent(interval=1)+"%"
41      return memory+cpu
```

使用 python 的 time 模块查看运行过程耗时；

① Floyd 最短路径算法

运行过程测试结果：

```
当前时间::
1577601591.4438014
当前cpu和内存使用情况
Memory usage:5331588.000000 kb CPU:16.40%
算法进行中cpu和内存使用情况
Memory usage:5336920.000000 kb CPU:20.80%
当前时间::
1577601593.4541733
```

由此获取到 Floyd 算法内存占用大约为 5336920kb-5331588kb=5332kb
CPU 占用率为 20.80%-16.40%=4.40%
运行过程耗时为 93.4541733s-91.4438014s=2.0103719s
大约为 2010 毫秒

② DTW 语音识别算法

```
按下‘Enter’开始确认指令
当前时间::
1577601049.1293511
当前cpu和内存使用情况
Memory usage:4951252.000000 kb CPU:0.20%
录音开始
100%|
100%|
                                                  | 3/3 [00:00<00:00, 23.05it/s]
算法进行中cpu和内存使用情况
Memory usage:4952596.000000 kb CPU:3.30%
特征距离值:
2984.791001981937
当前时间::
1577601053.5142605
```

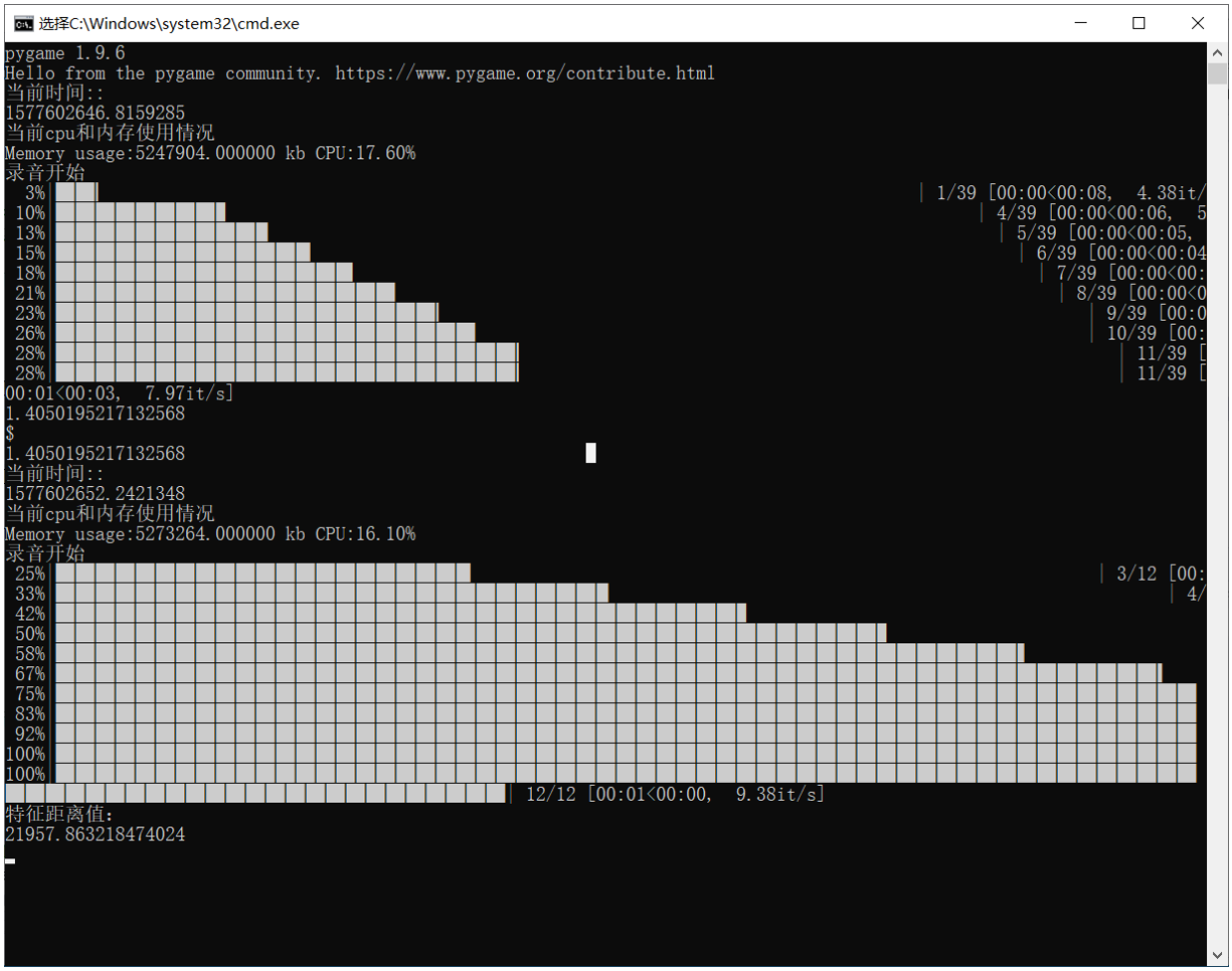
由此获取到 DTW 算法内存占用大约为 4952596kb-4951252kb=1344kb
CPU 使用率大约为 3.30%-0.20%=3.10%
运行耗时大约为 1577601053.5142605s-1577601049.1293511s=4.3849094s
大约为 4385 毫秒

实际测试：

① 自动寻路测试：

右下角是玩家角色，左上角名为“pyq”的是电脑 AI 角色，图中有 0,1,2,4 四个点。玩家角色处在点 2 附近，AI 角色当前处在点 4 附近，要想攻击到玩家角色，就得和玩家角色的垂直距离减小，因此自动寻路算法得出当前最短路径就是 4->0 这条，也就是 AI 角色垂直下落即可。

② DTW 算法进行语音识别测试：



两次说话录制两段语音，采取它们的特征值，计算出两段语音的距离为 21957

改进设想：

在现有基础上进一步优化游戏的各方面功能，例如把局域网联机拓展为广域网联机，进一步拓展服务器功能，将游戏打造成在线对战游戏。

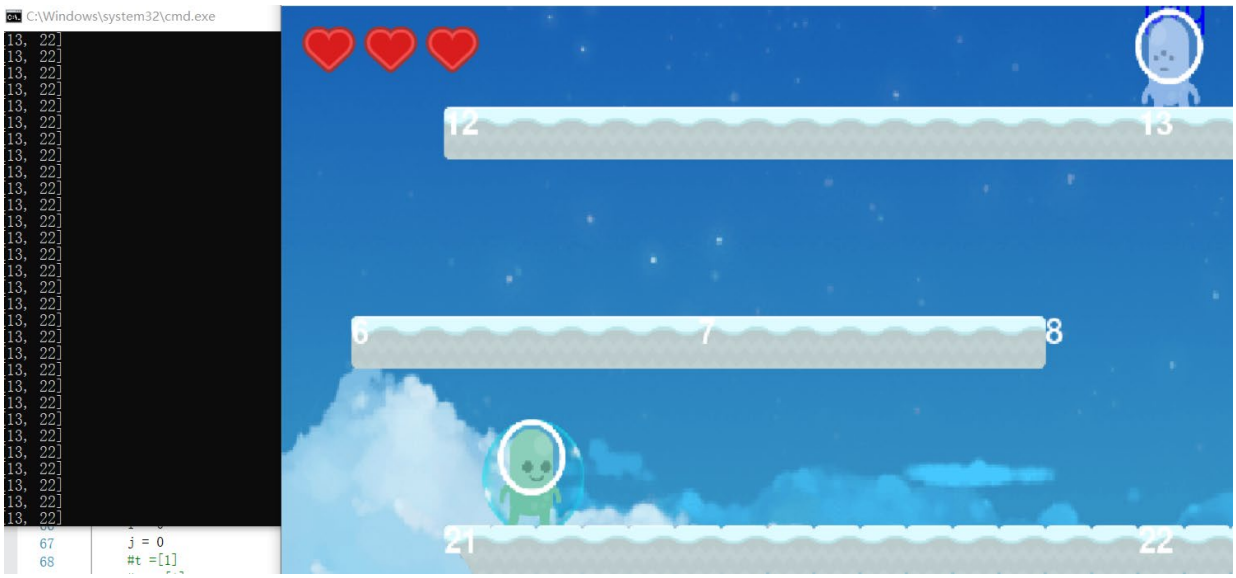
把机器学习引入到电脑 AI 角色的动作中，而不只用 Floyd 算法帮助其寻路，让 AI 更加地智能化。同时完善游戏的其他内容，例如增加地图数量，增加角色种类，增加技能种类，完善游戏画面等等。

7. 运行结果与分析

（要有多组正确的实际测试数据，并给出相应运行结果截图，并对多组结果进行比较分析）

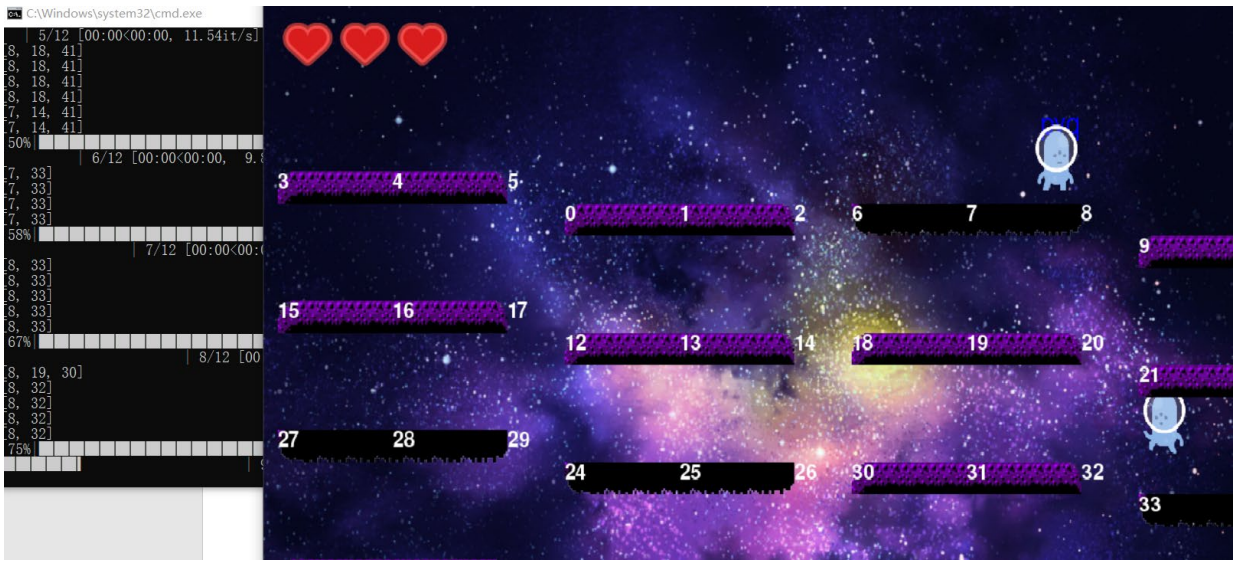
① Floyd 算法：

地图 1



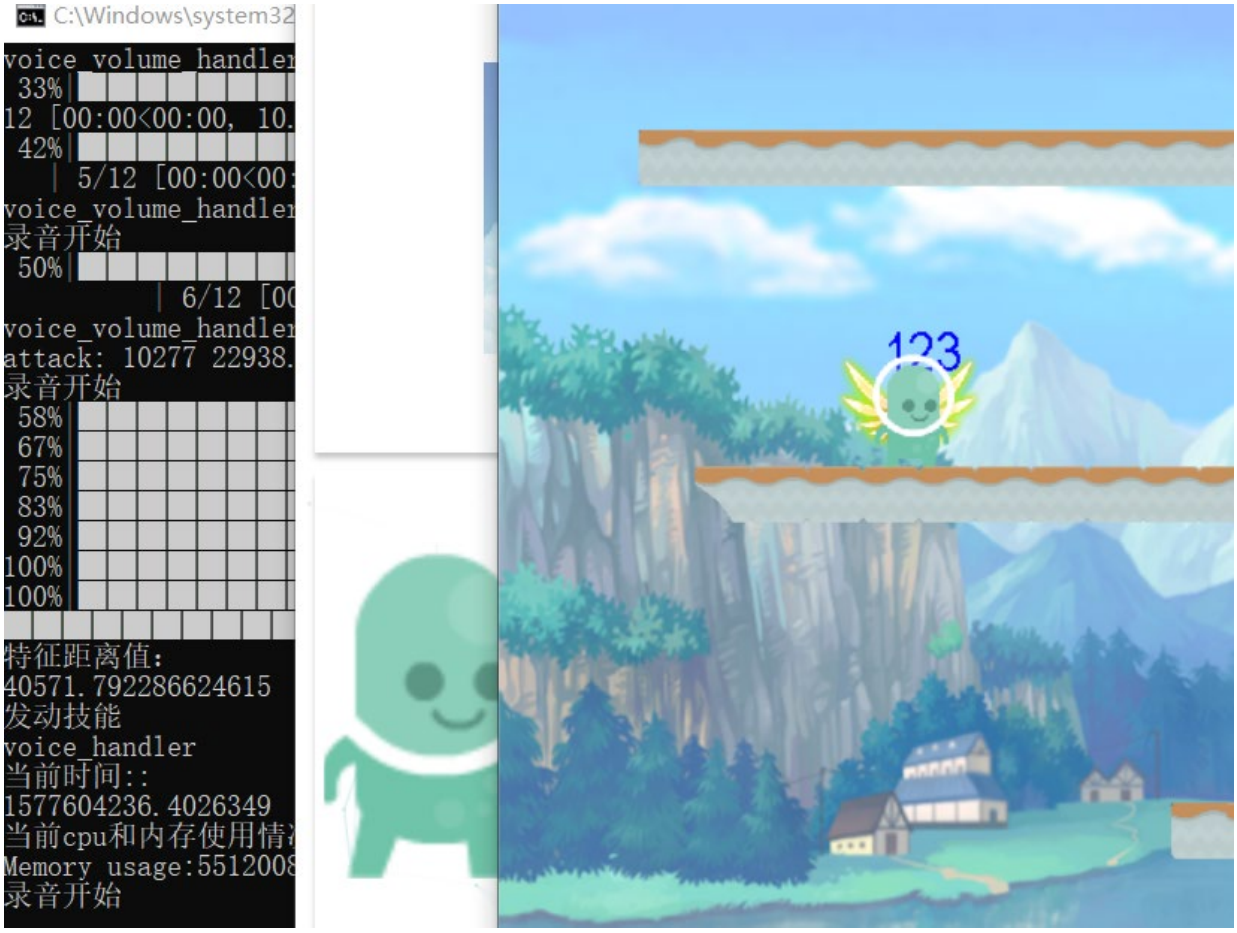
玩家在点 21，AI 角色需要从 13 走到 22 即可开始攻击。

地图 2



玩家在点 33 附近，AI 角色需要从 8 走到 32 即可开始攻击。

② 声音识别算法：



系统识别到玩家说出技能语音，发动“飞行”技能。

8. 教师指导建议及解决记录

（概述任课教师在开题指导、中期检查和软件验收环节中对课程任务的指导建议及各自所采取的相应解决措施和效果等）

8.1 开题指导及中期检查

①开题指导

指导建议：
将游戏由单一的声音发动攻击拓展出更多的声控动作。

解决措施：
我们采用语音识别的算法，让玩家既能通过声音发动普通攻击，还能通过说出特定的“咒语”来发动技能。

效果：
实现了三个可供玩家选择的技能，并且能让玩家自主录制声音后在游戏里说出这段声音即可发动自己选择的技能。

②中期检查

指导建议：
将游戏从只与电脑角色对战的单人模式拓展成能和真人联机对战的游戏。

解决措施：
通过学习 python 的网络编程，采用 socket 架构构建了一套适合游戏的数据包协议、服务器、客户端，并和之前已开发出的单人模式游戏结合，开发出了局域网联机的游戏。

效果：
游戏实现了局域网联机，并且实现了一定范围内寻找房间的功能，还能看到房间的地图以及房间已经有的玩家数量。

8.2 软件验收

指导建议：
将游戏开发成多人在线网络游戏。

解决措施：
进一步开发服务器，进一步学习有关网络的知识，将游戏服务器部署在购买的云服务器上，实现网络在线游戏。

效果：
玩家们下载了游戏的资源后，即可连接我们部署的服务器，与其他玩家一起在线对战。

9. 总结（收获与体会）

（如实撰写课程任务完成过程的收获和体会以及遇到问题的思考、程序调试能力的培养提升等相关内容；要求不少于 500 字，严禁雷同）

软件设计实践任务开始之前，我和团队的成员们都想做一款游戏，但是想做出一款好玩的游戏确实是很难的，不单单是技术上，更是玩法和游戏思维上。我们讨论了现在市场上面流行火爆的游戏的共同点，那就是能够让玩家有很深的沉浸感，能够给玩家们很大的反馈。再加上电影《头号玩家》带给我们的畅快体验，我们也想着做一款能够带给玩家很强的游戏体验、能够让玩家身临其境的游戏。于是便想到了用麦克风做一款声控游戏，这种声音交互式游戏之前也有过，但是通过声音来控制角色发射子弹，且子弹随着声音大小而改变其威力的游戏应该是没有的。

项目任务初期，我们规划好了游戏的软件系统架构，从前端 UI 到后端游戏内部逻辑，而最具特色的声音控制部分我们查阅了相关资料，计划学习机器学习中有关语音识别的知识。在不断地迭代开发中，程序也出现过大大小小的问题，有些问题是由于编写时的粗心大意，有些则是我们三个人坐在一起努力了一下午才勉强解决的难题。在这之中我们积累许多经验和教训，知道了 html 和 python 可以结合在一起的跨平台开发，知道开启多线程有可能会导致竞争危险的发生，知道了 socket 网络编程的基本原理。而更重要的是我们第一次把自己学到的知识使用得这么开心，玩游戏其实是很多人的爱好，以前我们也被老师家长常常念叨少玩游戏，但其实爱玩游戏本没有错，游戏带给人欢乐，游戏带给人生活的幸福感。游戏在我眼里就是一幅画卷，描绘着生活中也许永远也见不到的奇妙幻想，从某种程度上说游戏丰富了人生。这次我们团队结合着自身的兴趣，不断解决眼前的难题，在不知不觉中熟悉了一个项目从规划到开发到完整实现的整体流程，也对知识有了更深的理解和掌握。