浙江工商大学

虚拟仿真实验指导书

课程名称：Python+智能群体机器人虚拟仿真实验

授课教师：Python课程组

2023 年 11 月

实验一:基本语法练习

实验要求:

掌握 Python 基础语法知识，包括变量定义，选择控制，多分支选择，嵌套选择，while 循环结构，for 循环结构，函数使用以及智能机器人函数使用等内容。

步骤 1 变量定义

实验内容:

变量：可以发生变化的数据，变量名称应该符合标识符的命名要求，且不能使用系统关键字。

1. 变量名的命名规范和注意事项:

（1）变量名只能是字母、数字或下划线的任意组合

（2）变量名的第一个字符不能是数字

（3）关键字不能声明为变量名。

2. 变量的赋值

变量的赋值可以使用如下语法:

变量=表达式

变量 1= ... =变量 n=表达式

变量 1,...,变量 n=表达式 1,...,表达式 n

赋值语句的语义是把左边表达式的值赋值给右边的变量。其中=是赋值符号,而非等号。变量的类型是由其所赋值的表达式的数据类型所决定,这就是 Python 所支持的动态数据类型。

3. 变量的保存与删除

在程序运行过程中,变量的值是临时存储在内存单元中,一旦程序运行结束,或者关闭计算机,则变量中的数据就会丢失。因此可以把有用的数据永久保存到指定的文件中。对不再使用的数据,则可以将其删除。Python 支持内存的自动管理,对于内存中没有任何变量引用的数据,系统会自动将其删除。如果需要人工删除,则可以使用 del 语句来完成。

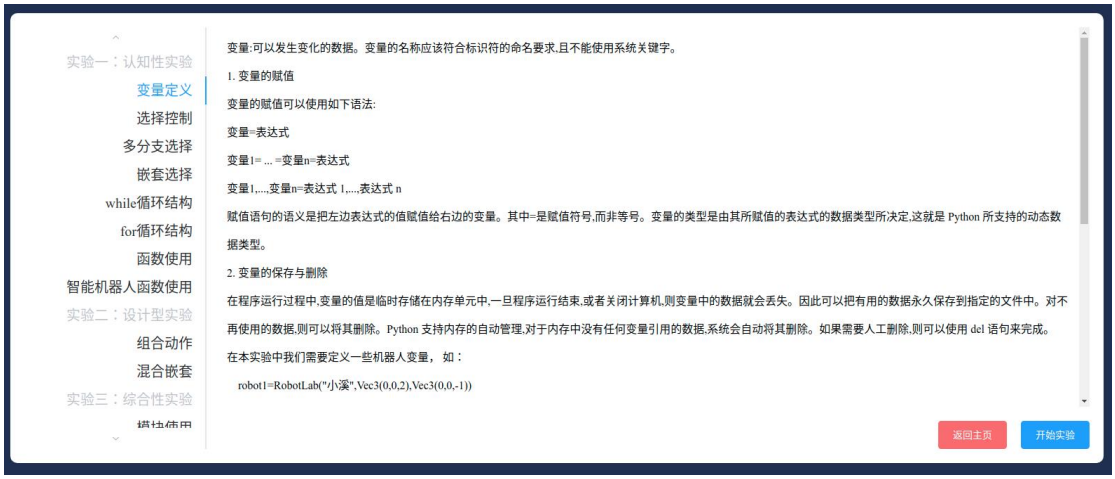

图 1 变量定义步骤描述

在本实验中我们需要定义一些机器人变量， 如:

robot1=RobotLab("小溪",Vec3(0,0,2),Vec3(0,0,-1))

上面这段代码就定义了一个有初始朝向和初始位置的机器人。

完整代码如下:

#encoding=utf-8

```
# 库文件中导入两个类
from RobotLab import RobotLab,Vec3
# 初始化机器人的姓名，位置和朝向
# 位置 用一个 3 维坐标来表示空间位置，对应到屏幕上 屏幕向右为 x 正方向，屏幕向上为 y 正方向,屏幕向里为 z 正方向
# 朝向 用一个 3 维向量来表示机器人的朝向,坐标系统和位置的坐标系统一样,比如现在的(0,0,-1)表示屏幕向外的朝向
robot1=RobotLab("小溪",Vec3(0,0,2),Vec3(0,0,-1))
# 调用机器人的鞠躬动作
robot1.bow()
```

上述代码首先从系统定义好的模块中导入类，然后定义机器人变量，并调用机器人的鞠躬动作，最后执行机器人的结束语句。

同学们可以仿照示例代码定义有其它初始朝向和位置的机器人。

图 2 变量定义示例程序及运行结果

示例代码中展示了如何初始化机器人的昵称、位置以及朝向，并调用机器人鞠躬动作。点击运行按钮之后，界面右边出现单机器人完成鞠躬动作。通过修改代码，机器人可以改变初始位置即初始朝向。

步骤 2 选择控制

实验内容：

Python 条件语句是通过一条或多条语句的执行结果（True 或者 False）来决定执行的代码块。选择结构类型包括单分支、二分支和多分支等。

单分支 if 的结构(选择是否执行语句块):

```
if 表达式:语句
if 表达式:
    语句块
```

根据表达式的值,选择执行语句(块)。即:如果表达式成立(True),则执行语句(块)，否则,不执行语句(块)

双分支 if 的结构：

if 表达式：语句

else:

语句块

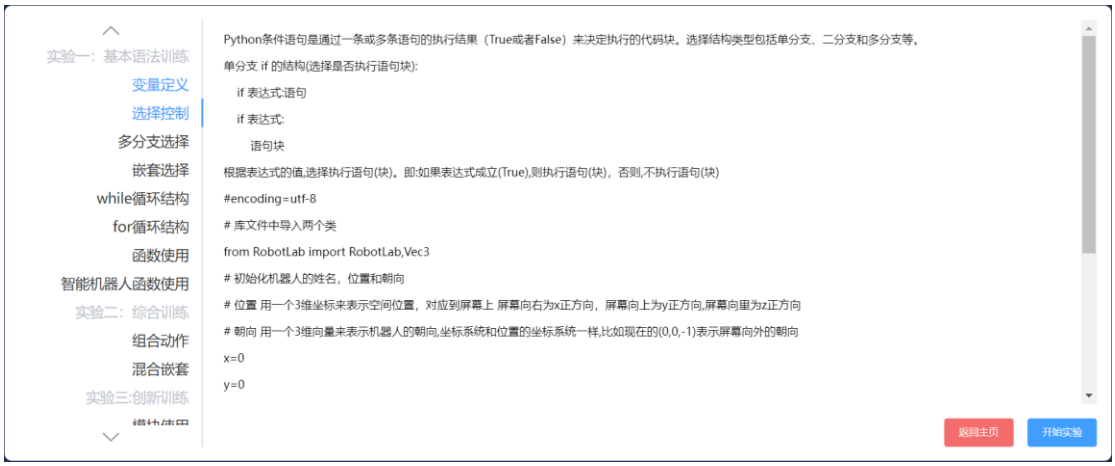

图 3 选择控制步骤描述

示例完整代码如下:

```
#encoding=utf-8
# 库文件中导入两个类
from RobotLab import RobotLab,Vec3
# 初始化机器人的姓名，位置和朝向
# 位置 用一个 3 维坐标来表示空间位置，对应到屏幕上 屏幕向右为 x 正方向，屏幕向上为 y 正方向,屏幕向里为 z 正方向
# 朝向 用一个 3 维向量来表示机器人的朝向,坐标系统和位置的坐标系统一样,比如现在的(0,0,-1)表示屏幕向外的朝向
x=0
y=0
z=2
robot1=RobotLab("小溪",Vec3(x,y,z),Vec3(0,0,-1))
# 调用机器人的鞠躬动作
if(x==y):
    robot1.bow()
# 整个程序结束
RobotLab.End()
```

在上面这个程序中，我们首先设置一些变量的值,然后初始化一个机器人,并根据变量 x 和 y 的值来选择是否调用机器人的鞠躬动作，在这里的话机器人会执行鞠躬动作。

在本实验中可以利用选择结构来更加灵活得控制机器人。 同学们可以修改以上示例中 x 和 y 的值，观察得到的结果。

图 4 选择控制示例程序及运行结果

示例代码展示了如何通过选择结构来控制机器人是否执行鞠躬动作，如图 4 所示。

步骤 3 多分支选择

实验内容:

Python 条件语句是通过一条或多条语句的执行结果（True 或者 False）来决定执行的代码块。选择结构类型包括单分支、二分支和多分支等。其中多分支选择是单分支选择的扩展。

多支 if 的结构(多个语句块中选择 1 个):

if 表达式 1:

语句块 1

elif 表达式 2:

语句块 2

……

elif 表达式 n:

语句块 n

[else:

语句块 m]

根据表达式 1，……，表达式 n 的值，选择执行语句块 1，……，语句块 n 或语句块 m 中的一个。即：首先判断表达式 1，如果成立(True)，则执行语句块 1（执行完束）；否则继续判断表达式 2，如果成立，则执行语句块 2（执行完结束）；否则依次继续判断 3，……，表达式 n。如果表达式 n 成立，则执行语句块 n（执行完结束），否则（所有表达式均不成立），执行语句块 m。

图 5 多分支选择步骤描述

示例完整代码如下:

```
#encoding=utf-8
# 库文件中导入两个类
from RobotLab import RobotLab,Vec3

x=1
y=0
z=1
# 初始化机器人的姓名，位置和朝向
# 位置　用一个 3 维坐标来表示空间位置，对应到屏幕上 屏幕向右为 x 正方向，屏幕向上为 y 正方向,屏幕向里为 z 正方向
# 朝向　用一个 3 维向量来表示机器人的朝向,坐标系统和位置的坐标系统一样,比如现在的(0,0,-1)表示屏幕向外的朝向
robot1=RobotLab("小溪",Vec3(x,y,z),Vec3(0,0,-1))
if(x==y):
    # 调用机器人的鞠躬动作
    robot1.bow()
elif(x==z):
    # 调用机器人的俯卧撑动作
    robot1.pushUp()
else:
    # 调用机器人的开怀大笑动作
    robot1.laugh()
# 整个程序结束
RobotLab.End()
```

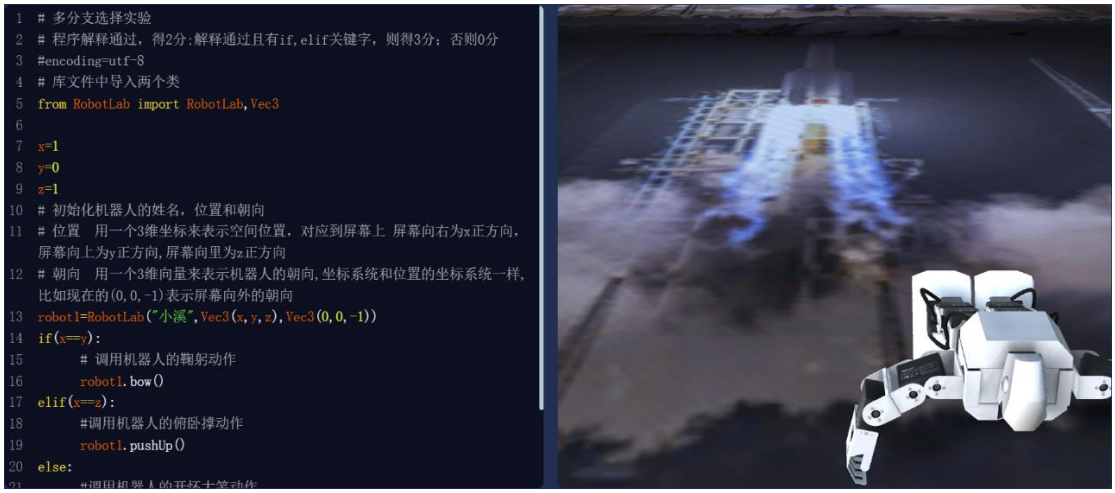

图 6 多分支选择示例程序及运行结果

示例代码展示了如何使用多分支选择控制机器人完成不同的动作。

步骤 4 嵌套选择

实验内容:

把 if 结构作为语句放入 if...else...的语句块中使用,从而在一个 if 结构中又套用了另一个 if 结构,形成 if 嵌套。由于 Python 允许用户嵌套多层,因此使用 if 嵌套给程序设计带来了方便。

if 嵌套的结构形式如下:

```
……
if 表达式 1:
if 表达式 2:
    语句块 1
else:
    语句块 2
else:
    if 表达式 3:
    语句块 3
else:
    语句块 4
……
```

if 嵌套可以根据多个表达式的值,选择执行相应的语句块。

图 7 嵌套选择步骤描述

示例完整代码如下:

```
#encoding=utf-8
# 库文件中导入两个类
from RobotLab import RobotLab,Vec3
import  random

y=0
# 随机变量[-1,0]
x=random.randrange(-1,1)
# 随机变量[0,4]
z=random.randrange(0,5)
# 初始化机器人的姓名，位置和朝向
# 位置  用一个 3 维坐标来表示空间位置，对应到屏幕上 屏幕向右为 x 正方向，屏幕向上为 y 正方向,屏幕向里为 z 正方向
# 朝向  用一个 3 维向量来表示机器人的朝向,坐标系统和位置的坐标系统一样,比如现在的(0,0,-1)表示屏幕向外的朝向
robot1=RobotLab("小溪",Vec3(x,y,z),Vec3(0,0,-1))
# 调用机器人的鞠躬动作
if(x==-1):
    if(z==2):
        # 招手
        robot1.waveHands()
    elif(z==1):
        # 俯卧撑
        robot1.pushUp()
    else:
        # 鞠躬
        robot1.bow()
else:
    if(z==3):
        # 开怀大笑
```

```
            robot1.laugh()
        else:
            # 左踢球
            robot1.leftKickBall()
    # 整个程序结束
    RobotLab.End()
```

图 8 多分支选择示例程序及运行结果

示例程序通过判断随机变量 x 和 z 的位置来确定机器人最终执行动作。

步骤 5 while 循环结构

实验内容:

while 循环的结构:

while 表达式:

循环体

[else:

语句块]

执行过程:

根据表达式的值,选择循环执行循环体,或者执行语句块(可以省略)。如果表达式成立(True),则执行循环体,当执行完之后,就返回到 while 处,再次判断表达式是否成立,重复上述操作,直到表达式不成立(False),转向执行 else 的语句块。

说明:

(1) 避免出现死循环。即:表达式的值永远成立(例如:1)。

(2) 循环体也是语句块。

(3) else 部分可以省略

图 9 while 循环结构步骤描述

示例完整代码如下:

```
#encoding=utf-8
# 库文件中导入两个类
from RobotLab import RobotLab,Vec3
x=0
y=0
z=5
# 初始化机器人的姓名，位置和朝向
# 位置　用一个 3 维坐标来表示空间位置，对应到屏幕上 屏幕向右为 x 正方向，屏幕向上为 y 正方向,屏幕向里为 z 正方向
# 朝向　用一个 3 维向量来表示机器人的朝向,坐标系统和位置的坐标系统一样,比如现在的(0,0,-1)表示屏幕向外的朝向
robot1=RobotLab("小溪",Vec3(x,y,z),Vec3(0,0,-1))
# 调用机器人的鞠躬动作
i=1
while(i<4):
# 鞠躬
    robot1.bow()
    i=i+1
# 整个程序结束
RobotLab.End()
```

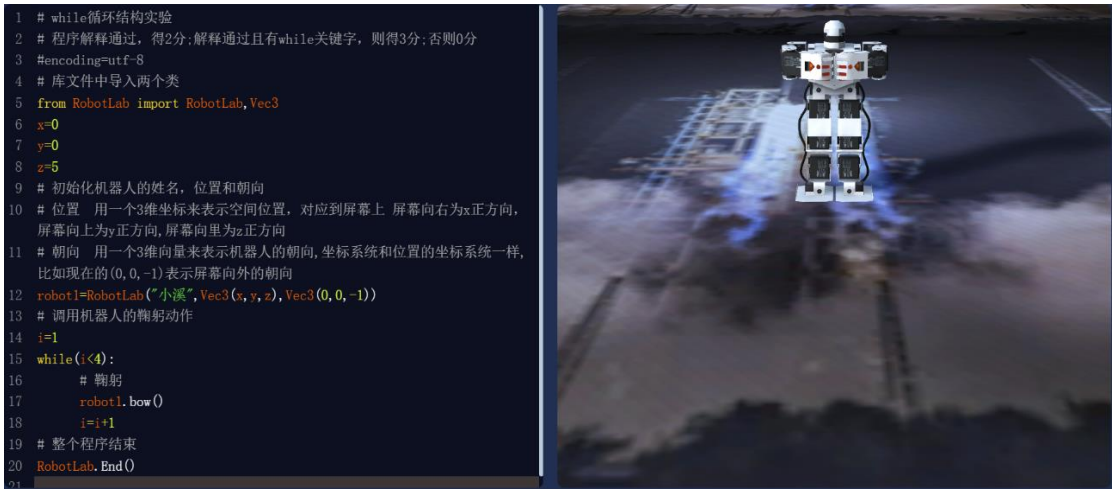

图 10 while 循环结构示例程序及运行结果

示例程序通过 while 循环来控制机器人鞠躬的次数。

步骤 6 for 循环结构

实验内容:

for 循环的结构:

for 变量 in 序列:

循环体

[else:

语句块]

执行过程:根据变量在序列中的值,选择循环执行循环体,或执行语句块(可以省略)。如果变量的值在序列中存在,则执行循环体,当执行完之后,就返回到 for 处,再次判断变量的值在序列中是否存在,重复上述操作,直到遍历完序列中的值之后,转向执行 else 的语句块。

说明:序列可以是 range()、列表、元组、集合或字典。

图 11 for 循环结构步骤描述

示例完整代码如下:

```
#encoding=utf-8
# 库文件中导入两个类
from RobotLab import RobotLab,Vec3
x=0
```

```
y=0
z=5
# 初始化机器人的姓名，位置和朝向
# 位置　用一个 3 维坐标来表示空间位置，对应到屏幕上 屏幕向右为 x 正方向，屏幕向上为 y 正方向,屏幕
# 向里为 z 正方向
# 朝向　用一个 3 维向量来表示机器人的朝向,坐标系统和位置的坐标系统一样,比如现在的(0,0,-1)表示屏幕
# 向外的朝向
robot1=RobotLab("小溪",Vec3(x,y,z),Vec3(0,0,-1))
# 循环 5 次
for i in range(5):
    # 调用机器人的鞠躬动作
    robot1.bow()
# 整个程序结束
RobotLab.End()
```

在上述代码中，我们先定义了一些变量，初始化机器人，然后利用 for 循环结构来控制机器人执行鞠躬动作的次数，在该示例中机器人执行了 5 次鞠躬动作。

在本实验中 for 循环结构也主要用来控制机器人动作的次数，在某些时候比 while 循环结构写法更加简介且清晰。

同学们可以修改 range()函数中的值，或者利用其它方式来控制机器人，并观察得到的结果。

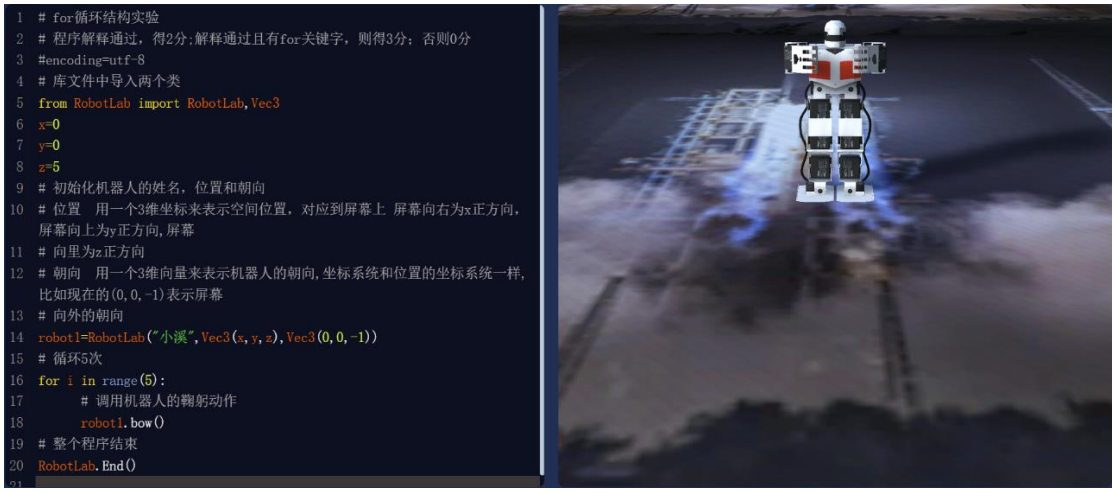

图 12 for 循环结构示例程序及运行结果

步骤 7 函数使用

实验内容:

函数能提高应用的模块性，和代码的重复利用率。

函数包括函数名、参数和函数体等。即:

```
def 函数名([形参[,...,形参]]):
    函数体
```

def:函数定义标识。def、函数名和冒号(:)是必项的,不能没有。

形参:形式上的参数(变量)。在设计函数时,形参可以没有确定的值,也可以不指定确定的

类型,但是不能影响后续的运算(隐含了满足后续运算的数据类型)。省略形参的函数称为无参函数,否则称为有参函数。

函数体:具有特定功能的语句块。如果函数体只用一个 pass 语句,则称为空函数。在函数体中可以使用“return [表达式[,表达式[,...,表达式]]]”,返回表达式的值,并作为函数值;如果省略表达式或者省略 return,则返回 None(空值);return 可以返回多个值(默认:元组)。

函数调用的主要因素:参数和返回值等。即:

函数名([实参[,...,实参]])

实参:实际使用时,拥有确定值的参数(表达式)。在调用函数时,实参必项拥有确定的值和确定的类型。特别强调:实参的个数要与形参的个数相等,顺序必项相一致,而且各实参的数据类型与相应的形参的数据类型要一致。即:个数相等、顺序一致和类型一致等三要素。

返回值:调用函数后,带回一个或多个值。特别提醒:返回值的个数及其类型。

匿名函数与调用:

定义匿名函数:可以利用 lambda 在一个语句行内,把一个表达式定义为一个函数进行使用。即:

Lambda [形参 1[,形参 2,...,形参 n]]:表达式

调用匿名函数:可以在定义匿名函数时,直接赋值给一个函数对象,然后就可以按照函数的使用方法调用函数即可。即:

函数对象= Lambda [形参 1[,形参 2,...,形参 n]]:表达式

变量=函数对象([实参 1[,实 2,...,实参 n]])

匿名函数:函数直接(或间接)调用函数本身。即:在函数体内部调用函数自身。

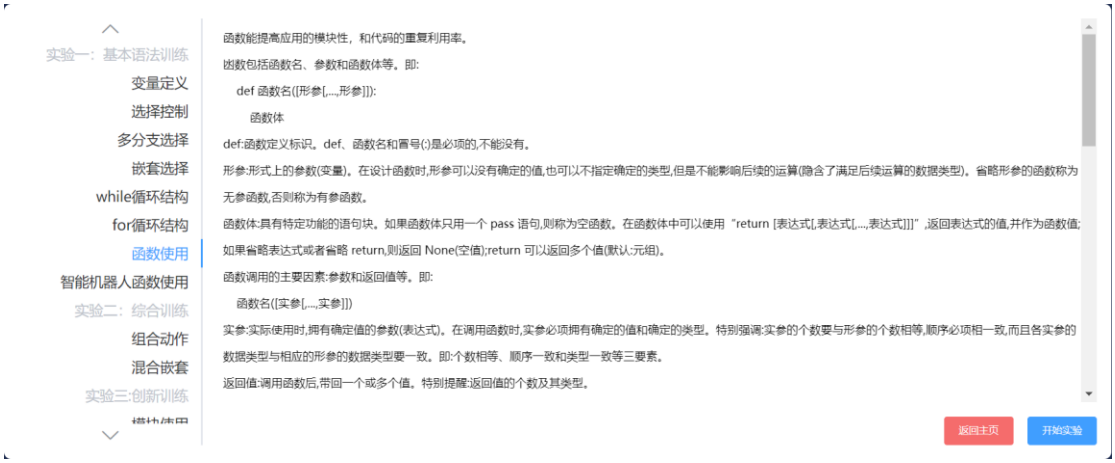

图 13 函数使用步骤描述

示例完整代码如下:

```
#encoding=utf-8
# 库文件中导入两个类
from RobotLab import RobotLab,Vec3
#一些变量
x=0
y=0
z=3
#函数定义
def act(robot):
```

```
        robot.waveHands()
        robot.pushUp()
        robot.bow()
# 初始化机器人的姓名，位置和朝向
# 位置　用一个 3 维坐标来表示空间位置，对应到屏幕上 屏幕向右为 x 正方向，屏幕向上为 y 正方向,屏幕向里为 z 正方向
# 朝向　用一个 3 维向量来表示机器人的朝向,坐标系统和位置的坐标系统一样,比如现在的(0,0,-1)表示屏幕向外的朝向
robot1=RobotLab("小溪",Vec3(x,y,z),Vec3(0,0,-1))
#调用函数，传入参数 robot1
act(robot1)
#机器人结束
RobotLab.End()
```

为了复用动作序列，我们定义了名为 act 的函数，该函数的形参是机器人，在函数体里定义了机器人的动作序列，这里的话机器人先执行招手，然后做俯卧撑，最后再做鞠躬动作。然后我们初始化机器人 robot1，调用 act 函数使得机器人执行函数中定义好的动作序列。

利用函数可以复用代码，使得代码结构更加清晰。比如有多个机器人想要执行相同的动作序列，为了不重复编写动作序列代码，我们把该代码写在函数中。

同学们可以仿照上述示例编写多个机器人的舞蹈序列,并尝试其它传参方式。

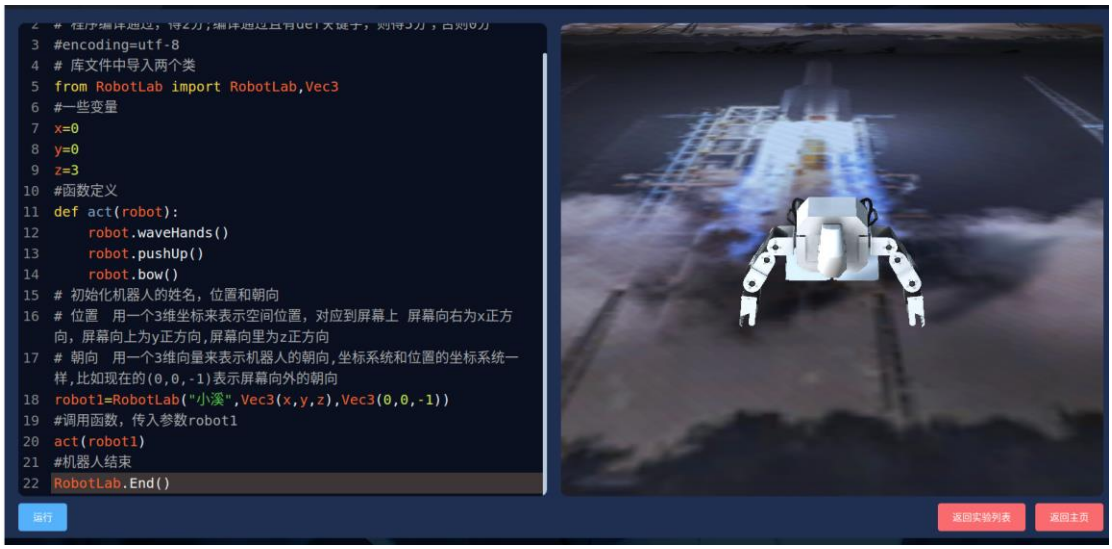

图 14 函数使用-示例代码及运行结果

步骤 8 智能机器人函数使用

实验内容:

本实验涉及了 24 种机器人基础动作，具体如下:

waveHands()#挥手

bow()#鞠躬

pushUp()#俯卧撑

turnRight()#右转

turnLeft()#左转

forward()#向前走

backward()#向后走
squat()#下蹲
rollForward()#前翻滚
rollBackward()#后翻滚
icanfly()#大鹏展翅
laugh()#开怀大笑
leftArmLift()#左臂抬起
rightArmLift()#右臂抬起
leftHandLift()#左手抬起
rightHandLift()#右手抬起
leftArmForward()#左臂向前
rightArmForward()#右臂向前
leftKickBall()#左踢球
rightKickBall()#右踢球
leftFootSideway()#左侧移
rightFootSideway()#右侧移
waistLeft()#左扭腰
waistRight()#右扭腰

图 15 智能机器人函数使用步骤描述

上一节中我们已经讲解了函数的定义以及调用，系统中定义好的 24 个动作函数不需要传入参数，直接调用即可。

示例完整代码如下:

```
# 库文件中导入两个类
from RobotLab import RobotLab,Vec3
# 初始化机器人的姓名，位置和朝向
# 位置　用一个 3 维坐标来表示空间位置，对应到屏幕上 屏幕向右为 x 正方向，屏幕向上为 y 正方向,屏幕向里为 z 正方向
# 朝向　用一个 3 维向量来表示机器人的朝向,坐标系统和位置的坐标系统一样,比如现在的(0,0,-1)表示屏幕向外的朝向
robot1=RobotLab("小溪",Vec3(0,0,2),Vec3(0,0,-1))
#循环一次
for i in range(1):
```

```
        #左扭腰
        robot1.waistLeft()
        #左侧移
        robot1.leftFootSideway()
    #机器人结束
    RobotLab.End()
```

在上述代码中先初始化机器人，然后在循环中调用机器人的动作序列。

图 16 智能机器人函数使用-示例代码及运行结果

示例代码展示了机器人的两种函数使用，如图 16 所示，学生可自行实现其他多种函数的使用。

实验二:综合训练

实验要求:

实验要求学生利用已学习的 Python 语法知识和智能机器人函数，完成初步组合设计。

步骤 9 组合动作 1)

实验内容:

利用智能机器人基础动作函数，自行进行组合。要求至少调用了一个机器人动作函数。

步骤 10 组合动作 2)

实验内容:

利用智能机器人基础动作函数，自行进行组合，掌握多动作组合实现方法。至少调用了四个机器人动作函数。之前我们单个机器人只执行了简单的动作序列，现在我们想要编写更加复杂的动作序列，比如让机器人跳一段比较好看的舞蹈，这不仅需要考虑动作间的协调性，还需要同学们丰富的想象力。

图 17 组合动作步骤描述

示例程序如下:

```
# 库文件中导入两个类
from RobotLab import RobotLab,Vec3
# 初始化机器人的姓名，位置和朝向
# 位置 用一个 3 维坐标来表示空间位置，对应到屏幕上 屏幕向右为 x 正方向，屏幕向上为 y 正方向,屏幕向里为 z 正方向
# 朝向 用一个 3 维向量来表示机器人的朝向,坐标系统和位置的坐标系统一样,比如现在的(0,0,-1)表示屏幕向外的朝向
robot1=RobotLab("小溪",Vec3(0,0,2),Vec3(0,0,-1))
# 循环 1 次
for i in range(1):
    # 左扭腰
    robot1.waistLeft(1)
    # 左侧移
    robot1.leftFootSideway()
    # 左转
    robot1.turnLeft()
    # 左踢球
    robot1.leftKickBall()
    # 左臂抬起
    robot1.leftArmLift()
    # 右臂抬起
    robot1.rightArmLift()
    # 左手抬起
    robot1.leftHandLift()
    # 右手抬起
    robot1.rightHandLift()
    # 下蹲
    robot1.squat()
    # 后翻滚
    robot1.rollBackward()
    # 右转
```

```
        robot1.turnRight()
        # 鞠躬
        robot1.bow()
    # 整个程序结束
    RobotLab.End()
```

图 18 组合动作示例程序及运行结果

示例代码比较简单，就是通过组合不同的动作来生成机器人的动作序列。同学们可以参照示例程序组合不同的动作，编写更加有趣或者更加优美的舞蹈动作。

步骤 11 混合嵌套 1)

实验内容:

综合利用 if,for,while 等语句实现机器人复杂动作。

步骤 12 混合嵌套 2)

实验内容:

合利用 if、for、while 等语句实现机器人复杂动作的混合嵌套控制；掌握混合嵌套语句的实现。利用 Python 允许的多局嵌套,可以方便的实现复杂的数据处理。

混合嵌套的结构形式如下:

```
……
if 表达式 1:
    while 表达式 2:
        语句块 1
        if 表达式 3:
            语句块 2
            for x in 对象:
                语句块 3
                if 表达式 4:
                    语句块 4
                语句块 5
        else:
            语句块 6
else:
```

语句块 7

……

图 19 混合嵌套步骤描述

利用混合嵌套控制机器人的示例程序如下:

```
#encoding=utf-8
# 库文件中导入两个类
from RobotLab import RobotLab,Vec3
import random
y=0
# 随机变量[-1,0]
x=random.randrange(-1,1)
# 随机变量 [0,4]
z=random.randrange(0,5)
# 初始化机器人的姓名，位置和朝向
# 位置 用一个 3 维坐标来表示空间位置，对应到屏幕上 屏幕向右为 x 正方向，屏幕向上为 y 正方向,屏幕向里为 z 正方向
# 朝向 用一个 3 维向量来表示机器人的朝向,坐标系统和位置的坐标系统一样,比如现在的(0,0,-1)表示屏幕向外的朝向
robot1=RobotLab("小溪",Vec3(x,y,z),Vec3(0,0,-1))
# 循环 3 次
for i in range(3):
    # 调用机器人的鞠躬动作
    if(x==-1):
        if(z==1):
            # 招手
            robot1.waveHands()
        elif(z==2):
            # 俯卧撑
            robot1.pushUp()
        else:
            # 鞠躬
            robot1.bow()
else:
```

```
            if(z==3):
                # 开怀大笑
                robot1.laugh()
            else:
                # 左踢球
                robot1.leftKickBall()
    # 整个程序结束
    RobotLab.End()
```

在上述程序中，我们先定义一些变量，然后初始化机器人，之后在 for 循环结构中编写控制机器人动作的程序，通过随机变量 x 和 z 的值来判断机器人将要执行的动作。

图 20 混合嵌套示例程序及运行结果

通过混合嵌套，我们可以编写更加复杂的代码。同学们可以任意修改示例代码，如实现:根据随机变量 z 的值执行不同的动作序列(一段舞蹈动作)，实现多个机器人的控制等，并观察得到的实验结果。

实验三:创新训练

实验要求:

掌握模块的使用，利用模块完成机器人控制和初始化操作，并在此基础上完成机器人群体舞蹈设计。

步骤 13 模块使用

实验内容:

模块是包含变量、函数和类定义的程序。一个模块通常包含若干函数。在 Python 中，不但可以直接使用系统提供的标准模块,而且可以设计适合实际应用的用户定义模块。

1. 创建模块

创建模块就是创建一个 Python 程序文件。即:把变量(赋值语句)、函数定义和类定义等写入一个程序文件。

在完成模块文件的建立之后,需要把模块文件(如:imod.py)放到 Python 的一个搜索路径(默认:...Python36\lib)中。

2. 使用模块

使用模块,可以使用 import,把模块导入到当前的程序中。即:

import 模块[,模块,...,模块]:导入一个(或多个)模块。

import 模块 as 新名:按照新名导入模块。

from 模块 import *:导入模块。使用时,可以省略模块名前缀。

from 模块 import 对象:从模块中导入对象。

from 模块 import 对象 as 新名:按照新名,从模块中导入对象。

其实在我们之前的代码中就已经包括了模块的使用，在编写控制机器人程序中，我们首先需要从系统定义好的模块中导入一些类，如机器人类。

实验要求：掌握模块的使用，利用模块完成机器人控制和初始化操作，并在此基础上完成机器人群体舞蹈设计。

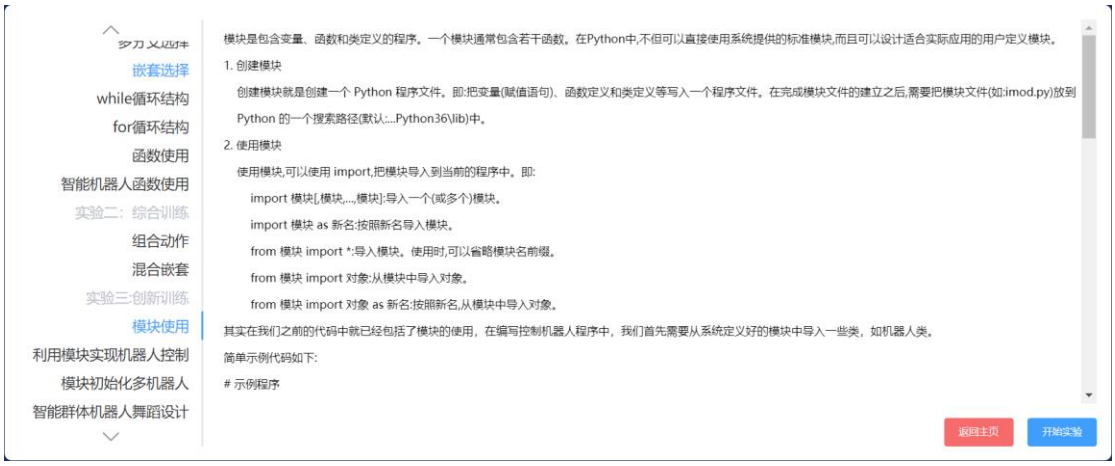

图 21 模块使用步骤描述

完整示例代码如下:

```
# 示例程序
# 库文件中导入两个类
from RobotLab import RobotLab,Vec3
# 初始化机器人的姓名，位置和朝向
# 位置　用一个 3 维坐标来表示空间位置，对应到屏幕上 屏幕向右为 x 正方向，屏幕向上为 y 正方向,屏幕向里为 z 正方向
# 朝向　用一个 3 维向量来表示机器人的朝向,坐标系统和位置的坐标系统一样,比如现在的(0,0,-1)表示屏幕向外的朝向
robot1=RobotLab("小溪",Vec3(0,0,2),Vec3(0,0,-1))
# 循环 1 次
for i in range(1):
    # 左扭腰
    robot1.waistLeft(1)
    # 左侧移
    robot1.leftFootSideway()
    # 左转
    robot1.turnLeft()
    # 左踢球
    robot1.leftKickBall()
    # 左臂抬起
    robot1.leftArmLift()
```

```
        # 右臂抬起
        robot1.rightArmLift()
        # 左手抬起
        robot1.leftHandLift()
        # 右手抬起
        robot1.rightHandLift()
        # 下蹲
        robot1.squat()
        # 后翻滚
        robot1.rollBackward()
        # 右转
        robot1.turnRight()
        # 鞠躬
        robot1.bow()
# 整个程序结束
RobotLab.End()
```

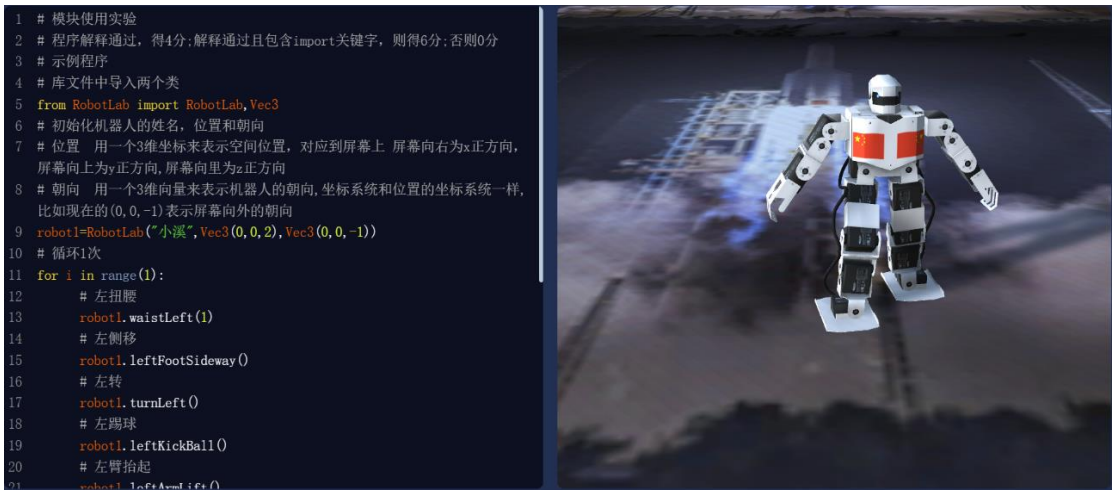

图 22 模块使用示例程序及运行结果

示例程序中通过 from...import...语句导入内容，如图 22 所示。

步骤 14：利用模块实现机器人控制 1）

实验内容：

学生通过本实验完成使用模块控制机器人的功能。

Python 模块(Module)，是一个 Python 文件，以 .py 结尾，包含了 Python 对象定义和 Python 语句。把相关的代码分配到一个模块里能让你的代码更好用，更易懂。模块让你能够有逻辑地组织你的 Python 代码段。模块能定义函数，类和变量，模块里也能包含可执行的代码。

关于模块的导入在上一个实验中已经介绍过了，在模块文件中函数，类和变量的定义和我们普通文件中是一样的。

步骤 15：利用模块实现机器人控制 2）

实验内容：

学生通过本实验完成使用模块控制机器人的功能，要求实现对机器人的多动作控制，动

作数大于 5。

步骤 16：利用模块实现机器人控制 3）

实验内容：

学生通过本实验完成使用模块控制机器人的功能，要求实现对机器人的多动作控制，动作数大于 10 且使用嵌套语句。

学生通过本步骤完成利用模块批量初始化机器人工程，如图 23 所示。

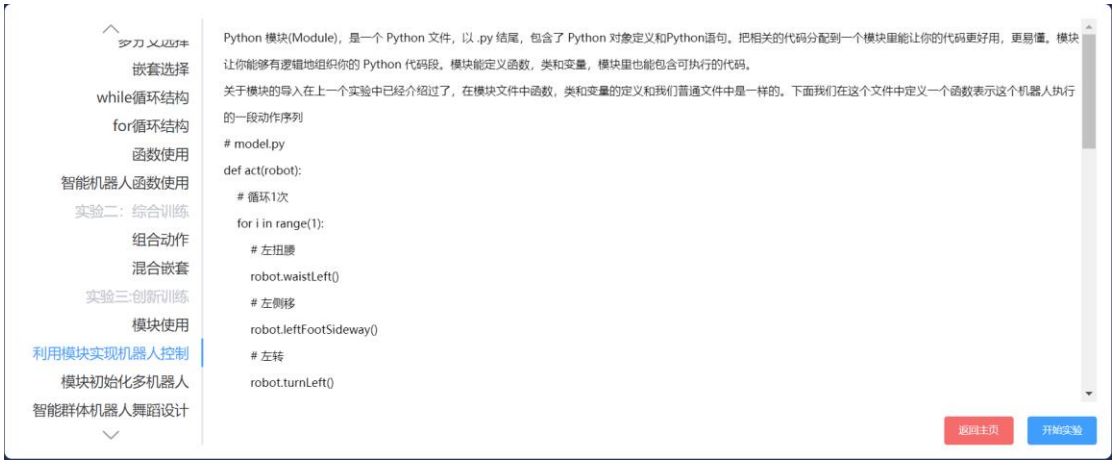

图 23 模块初始化多机器人步骤描述

下面我们在这个文件中定义一个函数表示这个机器人执行的一段动作序列

```
# model.py
def act(robot):
    # 循环 1 次
    for i in range(1):
        # 左扭腰
        robot.waistLeft(1)
        # 左侧移
        robot.leftFootSideway()
        # 左转
        robot.turnLeft()
        # 左踢球
        robot.leftKickBall()
        # 左臂抬起
        robot.leftArmLift()
        # 右臂抬起
        robot.rightArmLift()
        # 左手抬起
        robot.leftHandLift()
        # 右手抬起
        robot.rightHandLift()
        # 下蹲
        robot.squat()
        # 后翻滚
        robot.rollBackward()
```

```
        # 右转
        robot.turnRight()
        # 鞠躬
        robot.bow()

if __name__=='__main__':
    # 库文件中导入两个类
    from RobotLab import RobotLab,Vec3
    # 初始化机器人的姓名，位置和朝向
    # 位置　用一个 3 维坐标来表示空间位置，对应到屏幕上 屏幕向右为 x 正方向，屏幕向上为 y 正方向,屏幕向里为 z 正方向
    # 朝向　用一个 3 维向量来表示机器人的朝向,坐标系统和位置的坐标系统一样,比如现在的(0,0,-1)表示屏幕向外的朝向
    robot1=RobotLab("小溪",Vec3(0,0,2),Vec3(0,0,-1))
    # 调用上面定义好的动作函数
    act(robot1)
```

以上程序我们先定义了 act 函数，在里面我们设计了机器人的一些动作序列。然后在 if __name__=='__main__':中我们定义了一个机器人，并且调用了之前定义好的舞蹈动作，因为是在同一个文件中调用函数，所以不用加导入模块的语句。

图 24 模块实现多机器人控制示例程序及运行结果

步骤 17：模块初始化多机器人 1）

实验内容：

学生通过本步骤初步完成利用模块批量初始化机器人工程，要求机器人数目大于 2。

关于模块的定义以及使用之前都已经讲过了。接下来我们尝试一下用模块的方式来初始化多个机器人，生成机器人阵列。

```
# model.py
# 库文件中导入两个类
from RobotLab import RobotLab,Vec3
# 创建机器人的动作
# 定义一个函数，在函数里写机器人要执行的动作
# 函数传入的参数是 RobotLab 类型，是一个机器人
```

```
# 下面函数中机器人执行了几次挥手动作
def act1(robot):
    # 遍历 6 次
    for i in range(6):
        # 机器人执行挥手动作
        robot.waveHands()
# 定义了一个函数，传入参数和上面类似，只不过
# 这个函数中机器人执行的动作和上面不同
# 这个函数中机器人执行了几次挥手加下蹲动作
def act2(robot):
    # 遍历 3 次
    for i in range(3):
        # 机器人执行挥手动作
        robot.waveHands()
        # 机器人执行下蹲
        robot.squat()
# 创建机器人阵列
# 输入参数为阵列的长度和宽度
# width 阵列的宽度
# height 阵列的高度
def robotArray1(width,height):
    # 按宽度方向遍历
    for i in range(width):
        # 按高度方向遍历
        for j in range(height):
            # 初始化机器人的姓名，位置和朝向
            # 位置用一个 3 维坐标来表示空间位置，对应到屏幕上 屏幕向右为 x 正方向，屏幕向上为 y 正方向,屏幕
            # 向里为 z 正方向 在这里整个阵列 x 方向是以 0 为中心,y 方向上是固定为 0,z 方向上是以 0 为起点,
            # 朝向用一个 3 维向量来表示机器人的朝向,坐标系统和位置的坐标系统一样,比如现在的(0,0,-1)表示屏幕
            # 向外的朝向
            robot=RobotLab("hi",Vec3(i*2-width+1,0,j),Vec3(0,0,-1))
            # 让机器人执行动作 1
            act1(robot)
if __name__=='__main__':
    robotArray1(3,4)
```

上面的代码中我们在 act1 和 act2 中定义了机器人的动作序列，并在 robotArray1 中定义了机器人阵列，输入是阵列的宽度以及阵列的高度，然后我们在 if __name__=='__main__':中调 robotArray1 这个函数，如果是在其它文件中，我们需要加上导入模块的语句。

同学们可以参照以上例子定义机器人的阵列，并在 if __name__=='__main__'中去调用这些定义好的内容，生成不同队形的机器人阵列，并尝试定义生成不同队形的机器人阵

列函数(如:三角形，心形),观察得到的结果。

步骤 18：模块初始化多机器人 2）

实验内容：

学生通过本步骤进一步利用模块批量初始化机器人工程，要求机器人数目大于 5。

步骤 19：模块初始化多机器人 3）

实验内容：

学生通过本步骤进一步完善模块批量初始化机器人工程，要求机器人数目大于 10，如图 25 所示；示例代码中定义机器人阵列的函数，通过控制阵列的宽度和高度来控制该输出阵列的形状，如图 26 所示。

图 25 模块初始化多机器人步骤描述

图 26 模块初始化多机器人示例程序及运行结果

步骤 20：智能群体机器人舞蹈设计 1）

实验内容：

至此，我们已经掌握了设计智能群体机器人舞蹈的所有语法。通过定义类我们可以抽象一些对象，比如我们用到的机器人，就是通过类的方式来定义的；通过函数的方式我们可以复用一些代码，比如我们调用机器人的一些动作(waveHands 以及其它)；通过模块的方式我们可以更好的组织程序，比如我们可以在一个模块中定义好所有关于单个机器人的舞蹈动作，在另一个模块中定义好机器人的一些阵列，而在另一个模块中我们又可以定义一些其它内容。

所以同学们可以充分发挥想象，通过团队分工合作等方式来完成智能群体机器人舞蹈设计。

要求学生利用已有知识进行智能群体机器人舞蹈设计，并通过程序初步实现结果展示。要求机器人数目大于 2。

图 27 智能群体机器人舞蹈设计步骤描述

步骤 21：智能群体机器人舞蹈设计 2）

实验内容：

要求学生利用已有知识进行智能群体机器人舞蹈设计，并通过程序进一步实现结果展示。要求机器人数目大于 5，动作数大于 10。

步骤 22：智能群体机器人舞蹈设计 3）

实验内容：

要求学生利用已有知识综合进行智能群体机器人舞蹈设计，并通过程序实现结果展示。要求机器人数目大于 10，动作数大于 10。

下面我们简单列一段产生机器人舞蹈动作的代码。

```
#!/usr/bin/python3
#encoding=utf-8
# 库文件中导入两个类
from RobotLab import RobotLab,Vec3
# 创建机器人的动作
# 定义一个函数，在函数里写机器人要执行的动作
# 函数传入的参数是 RobotLab 类型，是一个机器人
# 下面函数中机器人执行了几次挥手动作
def act1(robot):
    # 遍历 6 次
    for i in range(2):
        # 机器人执行挥手动作
        robot.waveHands()
        # 左臂抬起
        robot.leftArmLift()
        # 右臂抬起
        robot.rightArmLift()
        # 左手抬起
```

```
            robot.leftHandLift()
    # 定义了一个函数，传入参数和上面类似，只不过
    # 这个函数中机器人执行的动作和上面不同
    # 这个函数中机器人执行了几次挥手加下蹲动作
    def act2(robot):
        # 遍历 3 次
        for i in range(2):
            # 机器人执行挥手动作
            robot.waveHands()
            # 机器人执行下蹲
            robot.squat()
            # 左手抬起
            robot.leftHandLift()
    # 创建机器人阵列
    # 输入参数为阵列的长度和宽度
    # width 阵列的宽度
    # height 阵列的高度
    def robotArray1(width,height):
        # 按宽度方向遍历
        for i in range(width):
            # 按高度方向遍历
          for j in range(height):
                # 初始化机器人的姓名，位置和朝向
                # 位置用一个 3 维坐标来表示空间位置，对应到屏幕上 屏幕向右为 x 正方向，屏幕向上为 y 正方向,屏幕向里为 z 正方向 在这里整个阵列 x 方向是以 0 为中心,y 方向上是固定为 0,z 方向上是以 0 为起点,
                # 朝向用一个 3 维向量来表示机器人的朝向,坐标系统和位置的坐标系统一样,比如现在的(0,0,-1)表示屏幕
                # 向外的朝向
                robot=RobotLab("hi",Vec3(i*2-width+1,0,j),Vec3(0,0,-1))
                # 让机器人执行动作 1
                act1(robot)
    # 和上面这个函数类似,创建机器人,只不过需要输入整个阵列的起始位置和要执行的动作
    # xstart 阵列 x 方向的起始位置
    # zstart 阵列 z 方向的起始位置
    # length 阵列的宽度
    # width 阵列的高度
    # act 该阵列中机器人要执行的动作
    def robotArray2(xstart,zstart,length,width,act):
        # 按宽度方向遍历
        for i in range(xstart,xstart+length):
            # 按高度方向遍历
            for j in range(zstart,zstart+width):
```

```
            # 初始化机器人的姓名，位置和朝向
            robot=RobotLab("hi",Vec3(i*2,0,j),Vec3(0,0,-1))
            # 让机器人执行动作
            act(robot)
if __name__=='__main__':
    # 阵列宽度为 3
    length=3
    # 阵列高度为 1
    width=1
    # 产生一个 3*1 的机器人阵列，z 方向起始点为 2,执行动作 2
    robotArray2(-int(length/2),2,length,width,act2)
    # 产生一个 3*1 的机器人阵列,z 方向起始点为 4,执行动作 1
    robotArray2(-int(length/2),4,length,width,act1)
    # 整个程序结束
    RobotLab.End()
```

在示例程序中定义两排机器人阵列，并分别执行不同的动作序列，如图 28 所示。

图 28 智能群体机器人舞蹈设计示例程序及运行结果