Micro-bit 走迷宫设计

——王天顺

摘要：

利用 micro：bit 的可视化，我进行了迷宫的生成以及自动化走迷宫的程序设计，虽然这只是对 micro：bit 的小灯进行的设计，但是加上光敏感应之后，可以把现实中的复杂情景转换成有限的迷宫，从而带动机器人进行无人作业。

选题与创意

在如今的智能时代，许多曾经的难题也应该随着技术的进步迎刃而解比如人类难以进入的极端环境，微小的领域，这些地方也许都只能由机器人代劳，所以，设计机器人的智能程序是一项很重要的任务。

设计思路与硬件

使用 Micro:Bit 内置的加速度计来实现机器人在迷宫中移动。机器人可以通过检测其当前位置来决定下一步该往哪里走。具体来说，我们用以下方法进行设计：

1. 定义迷宫：定义一个二维数组来表示迷宫地图，并将起点和终点标记为特殊值。
2. 初始化机器人位置：将机器人初始位置保存在全局变量中。
3. 获取当前位置：通过 Micro:Bit 内置的加速度计获取机器人当前位置。
4. 判断下一步该往哪里走：根据机器人当前位置，以及迷宫地图中的墙壁位置，计算出下一步应该往哪里走。
5. 移动机器人：根据下一步的移动方向，控制机器人进行前进或旋转。
6. 检查是否到达终点：如果机器人到达了终点，则结束程序；否则，返回第 3 步继续执行。

硬件方面，暂时只使用 micro：bit 进行模拟，而没有超声波探测，光敏感应等设备。

代码实现

以下是一个 Micro:Bit 走迷宫程序的实现：

```
# Imports go at the top
from microbit import *
import random as rd
#利用 prim 算法生成完美迷宫
#
dd = [[-2, 0],[2, 0],[0, -2],[0, 2]]
class maze:#定义迷宫类，用于测试
    def __init__(self,row,col):
        self.map=None
        self.row=row
        self.col=col

    def getrow(self):
        return self.row
    def getcol(self):
        return self.col
    def createMaze(self):
        maze=[[0 for i in range(self.col)] for j in range(self.row)]
        check = []
        firstrow = rd.randrange(1, self.row - 2, 2)
        firstcol = rd.randrange(1, self.col - 2, 2)
        maze[firstrow][firstcol] = 1
        check.append([firstrow, firstcol])
        while len(check):
            c = rd.choice(check)
            nears = []
            conditions = []
            for d in dd:
                conditions.append([c[0] + d[0], c[1] + d[1]])
            for condition in conditions:
                if condition[0] >= 1 and condition[0] <= self.row - 2 and
```

```
condition[1] >= 1 and condition[1] <= self.col - 2:
                    nears.append([condition[0], condition[1]])
            for n in nears.copy():
                if maze[n[0]][n[1]]:
                    nears.remove(n)
            for block in nears:
                if block[0] == c[0]:
                    if block[1] < c[1]:
                        maze[block[0]][c[1] - 1] = 1
                        maze[block[0]][block[1]] = 1
                        check.append([block[0], block[1]])
                    else:
                        maze[block[0]][block[1] - 1] = 1
                        maze[block[0]][block[1]] = 1
                        check.append([block[0], block[1]])
                else:
                    if block[0] < c[0]:
                        maze[c[0] - 1][block[1]] = 1
                        maze[block[0]][block[1]] = 1
                        check.append([block[0], block[1]])
                    else:
                        maze[block[0] - 1][block[1]] = 1
                        maze[block[0]][block[1]] = 1
                        check.append([block[0], block[1]])
            if not len(nears):
                check.remove(c)
        k=rd.randint(1,self.row-2)
        maze[1][k] = "s"
        while True:
            c = rd.randint(1, self.col - 2)
            if maze[self.row - 2][c]:
                maze[self.row - 2][c] = "e"
                break

        return maze
dic={}



m=maze(7,7)#可以生成 7✖7 或者更大的迷宫，为了方便测试，生成 7✖7 的迷宫
row=m.getrow()
col=m.getcol()
m=m.createMaze()
que=[]
```

```
for j in range(row):
    if m[1][j] == "s":
        start=[1,j]
        break
que.append(start)
themaze=[[False for j in range(row)]for i in range(col)]

for j in range(col):
    if m[row-2][j] == "e":
        end=[row-2,j]
        break

move=[[0,1],[0,-1],[1,0],[-1,0]]

for i in range(1,row-1):
    for j in range(1,col-1):
        if m[i][j]==0:
            dic[(i,j)]=0
        elif m[i][j]==1:
            dic[(i,j)]=1


def DfsPerStep():
    global que
    position=que.pop()
    if position==end:
        return position,True
    themaze[position[0]][position[1]]=True
    for i in range(4):
        if 0<position[0]+move[i][0]<row and
0<position[1]+move[i][1]<col \
                and
themaze[position[0]+move[i][0]][position[1]+move[i][1]]==False\
                and
m[position[0]+move[i][0]][position[1]+move[i][1]]==1 \
                or
[position[0]+move[i][0],position[1]+move[i][1]]==end:

que.append([position[0]+move[i][0],position[1]+move[i][1]])
    if que==[]:
        return position,True
    return position,False
```

```
while True:
        po,flag=DfsPerStep()
        m[po[0]][po[1]]=2
        for k in dic.keys():
            display.set_pixel(k[0]-1,k[1]-1,dic[k]*4)
        display.set_pixel(po[0]-1,po[1]-1,9)
        display.set_pixel(end[0]-1,end[1]-1,7)
        sleep(1000)
        if flag==True:
            break

display.show(Image.HAPPY)



# Code in a 'while True:' loop repeats forever
```

后续应用

虽然已经成功地实现了一个简单的迷宫游戏，但还有很多改进空间。一些可行的扩展包括：

改进算法：改进控制算法，使小车更加智能，可以在更复杂的环境中行驶。

硬件升级：使用更高级的硬件组件，如激光传感器、摄像头等，以增强小车的感知能力。

游戏场景扩展：将游戏场景从单一的迷宫扩展到其他类型的场景，如追逐、避障以及救援等等。

小组分工

由我独自完成所有项目，所以内容这么贫瘠。

总结

在这个 Micro:Bit 走迷宫程序中，我们可以使用内置的加速度计来实现机器人在迷宫中移动。通过检测机器人当前位置，以及迷宫地图中的墙壁位置，我们计算出下一步该往哪里走，并控制机器人进行前进或旋转。这个程序虽然简单，但也显现出了机器人未来的发展前景，以及人工智能的磅礴伟力。