勇往直前

数学科学学院 汪晨阳 2200010865

物理学院 朱诚君 2200011485

摘要：这是一个简单易上手的跑酷，在逐渐加快的节奏中躲避不断前来的障碍物，在小小的显示屏上感受不断冲刺的快感吧！适合没事的时候玩一玩，是对大脑的一个很好的锻炼!

一.选题及创意介绍

由于上学期我们并未参与开源程序的设计，所以对外接硬件设备的了解很少，所以我们考虑只使用模拟网站上的 5*5 的显示屏来实现一些功能。但如果过于复杂的功能用如此小的显示屏来展示实在是有些困难，因此我们去思考一些简单的小游戏，于是快节奏、轻松而适合低像素呈现的跑酷游戏成为了我们最终的选择。我们回忆了小时候在游戏厅中游玩的赛车游戏，它很适合作为这种只有两种操作输入（左或右）的游戏的模型，于是在了解 micro:bit 确实可以实现我们的想法之后，我们设计出了这款朴素的跑酷游戏，难度会越来越高并不断累计分数，不断提高的分数可以给予玩家成就感!

二.设计方案和硬件连接

在设计上，利用不同深浅度的光点来表示玩家所在的位置和障碍物所在的位置。为了防止图像过于复杂以至于出现无解的情况，在每一行只设置一个障碍物，并随机出现，玩家仅出现在底线位置，以障碍物不断接近底线反过来表现玩家的前进。玩家需要不断躲避前来的障碍物，在躲避障碍物的同时，分数将会累积。如果代表玩家的光点

触碰到代表障碍物的光点，则游戏结束并播放分数来评估玩家当局的成绩。为了实现游戏的丰富性，在分数累积的同时会逐渐增大难度，共分三级，来让玩家感受到挑战更高难度的乐趣。作为补偿，分数积累的速度会越来越快。因为实现想法的指导，所以并未连接任何硬件，只要在 micro:bit 网站模拟器上就可以实现所有功能了。

三.实现方案和代码分析

```
from microbit import *
import random
import music

score = 0
tick=0
pressed=False
player_pos = 2  # 玩家光点的初始位置，初始位置是屏幕中央
queue=[-1,-1,-1,-1,-1]
tune1=['G3:1','A3:1','C4:1','A3:1','E4:3','E4:3','D4:5',
      'G3:1','A3:1','C4:1','A3:1','D4:3','D4:3','C4:3','B3:1','A3:1',
      'G3:1','A3:1','C4:1','A3:1','C4:4','D4:2','B3:3','A3:2','G3:3','D4:4','C4:4']
music.play(tune1)
```

引用所需的模块，其中 random 用于随机生成障碍物，music 用于播放音乐。随后进行初始化，用 score 记录所得分数，tick 记录行动节点，pressed 记录按键的状态，queue 记录障碍物的位置，tune1 用于播放开场音乐。

```
while True:
    if tick==40:
        tick=0
        # 随机生成光点的位置
        led_pos = random.randint(0, 4)
        queue.append(led_pos)
        queue.pop(0)
```

当行动节点数达到 40 时，执行下面的生成障碍物命令。由于一共有五行，于是在 0，1，2，3，4 中随机生成一个数，作为障碍物的坐标，随后类似于队列的出入列操作进行位置的存储。

```
        #检测是否出现无解情况
        if led_pos==4:
            l=0
            for i in range(3,-1,-1):
                if queue[i]==4-i:
                    l+=1
                else:
                    break
            if l > 1:
                led_pos = random.randint(0, 3)
        if led_pos == 0:
            l=0
            for i in range(1,5):
                if queue[i]==i:
                    l+=1
                else:
                    break
            if l > 1:
                led_pos = random.randint(1, 4)
```

如果障碍物的位置出现并不合理的情况，如连续的一串位置可能会出现无法躲避的情况，于是进行检测：每当生成一个位于边缘的障碍物时，如果此前已有至少两个连续延伸到边缘的障碍物，此时就会可能无解，所以将这个障碍物的生成改动至其他位置来规避这种情况，从而防止无解。

```
        # 移动光点
        for i in range(5):
            for j in range(5):
                if i == led_pos and j == 0:
                    display.set_pixel(i, j, 5)  # 新光点
                elif  i==player_pos and j == 4:
                    display.set_pixel(i, j, 9)  # 玩家光点
                else:
                    display.set_pixel(i, j, 0)  #清理屏幕
        for i in range(5):
            if queue[i]>-1:
                display.set_pixel(queue[i], 4-i, 5)#设置旧的光点
```

为了表示障碍物的移动，将玩家所在位置的光点亮度设为 9，障碍物

所在位置的光点亮度设为 5，其他位置的光点亮度设置为 0，来表现屏幕上所有物体的位置。

```
        # 计分规则
        if score < 20:
            sleep(300)
            score += 1
        elif score < 80:
            sleep(100)
            score += 2
        else:
            sleep(0)
            score += 3

        #记录时间
        tick += 1
```

在障碍物移动时进行计分，分数线性增长，且随着分数基数增长，分数积累会越来越快，而停顿的时间也越来越长，来减少玩家反应的时间，提高游戏的难度，以体现难度变化。最后增加行动节点，来表示障碍物的一次移动。

```
    else:
        # 移动玩家光点
        if button_a.is_pressed() and player_pos > 0:
            if pressed==False:
                player_pos -= 1
        elif button_b.is_pressed() and player_pos < 4:
            if pressed==False:
                player_pos += 1
        pressed=button_a.is_pressed() or button_b.is_pressed()
```

在行动节点未达到预设值时，玩家可以自由移动。左键向左，右键向右，符合认知。为了防止按压一次按键多次移动的情况，对按压的情况进行记录，如果连续两次行动节点按键都被按压，则判断为一次按压时间过长，不予记录为两次按压。事实上，将一个行动节点的时间设置的足够短（10ms），可以防止在人类的反应速度下在两个行动节点之间连按两次而检测不到。

```
        # 碰撞检测
        if queue[0] == player_pos:
            for k in range(3):
                for i in range(5):
                    for j in range(5):
                        display.set_pixel(i, j, 0)
                sleep(500)
                for i in range(5):
                    for j in range(5):
                        display.set_pixel(i, j, 9)
                sleep(300)
            tune2=['A3:4','B3:2','E4:2','D4:4','C4:4','B3:4','B3:2','B3:2',
                'D4:4','C4:2','B3:2','A3:2','A3:1','C5:2','B4A3:2','C5:2',
                'B4A3:2','C5:2']
            music.play(tune2)
            display.scroll("SCOre:%d"%score )
            break               # 如果玩家光点和光点重合，游戏结束，并闪光提示，播放音乐
```

如果玩家所在的位置与障碍物所在的位置重合，则判断为碰撞，游戏失败，进行结束播报：先让屏幕闪烁三次来给予玩家视觉刺激，提醒游戏的结束；随后播放结束音乐，随后播放所得的分数，让玩家得到自己本局表现的评估。

```
        # 保持原本光点不动
        for i in range(5):
            for j in range(5):
                if  i==player_pos and j == 4:
                    display.set_pixel(i, j, 9)  # 玩家光点
                else:
                    display.set_pixel(i, j, 0)  #清理屏幕
        for i in range(5):
            if queue[i]>-1:
                display.set_pixel(queue[i], 4-i, 5)#设置旧的光点

        #记录时间
        tick += 1
        sleep(10)
```

如果没有碰撞，则仍然进行屏幕上原有光点的显示，来保持视觉表现的连贯性，防止割裂感。最后，记录行动节点，并设置每个节点的持续时间 10ms，来进行尽可能细致的按键操作的检测。

四.后续工作展望

由于 micro:bit 模拟器中只有两个可以操作的按键(我们当时认为，虽然貌似并非如此)，这限制了我们对游戏功能多样化的实现，最终

游戏的形式相当简单，不过这也使得游戏更加容易上手。在可以连接更多外置硬件的条件下，我们希望可以增加按键的数量，但也无需过多，只需实现四个方向的移动。障碍物现在将会从四个方向生成，但一次仍然只会生成一个，来防止小小的屏幕显得过于拥挤来降低玩家的游戏体现。随后玩家可以四个方向移动来躲避障碍物，这将极大提高游戏的丰富度，使得操作更加多样化。但障碍物方向的增多会使得游戏难度相对降低，我们将设计更多的难度梯度来让游戏得以得到更好的分层，从而让不同水平的玩家都能从中得到乐趣。这并不需要很多的额外工作，但确实会好玩很多。

五.小组分工合作

我们小组由来自数学科学学院的汪晨阳和来自物理学院的朱诚君组成。在共同构思出游戏的初步思路后，鉴于我们掌握的关于micro:bit 的知识较少，在模拟网站上学习了初步的知识后，汪晨阳同学编写了游戏的主体部分，包括障碍物的实现机制以及玩家与障碍物的移动机制等；朱诚君同学则就程序的想法实现提出了自己的看法，并结合多次的模拟测试，对程序的参数进行了调整，使得游玩体验得以优化，且实现了游戏中的音乐模块的设计，来为游戏增强趣味性。我们二人分工明确，最终得到了这个想法简单，实现也很朴素，但可玩性较高的游戏。