砖头派对（Python 版）实习报告

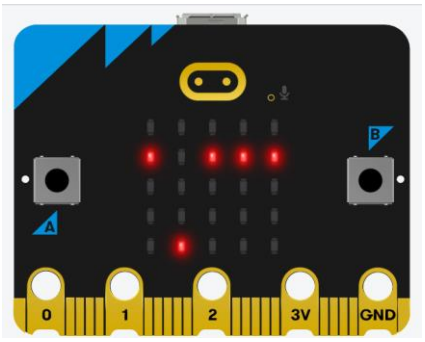
生命科学学院 廖宇瑄 2200012108

摘要：《糖豆人：终极淘汰赛》中的《砖头派对》小游戏机制有趣，易于上手，游戏性强，受到玩家喜爱。本报告介绍了对《砖头派对》小游戏的机制进行简化和修改，使其适合 micro:bit 仿真器的环境，并使用 Python 语言编程，最终在 micro:bit 仿真器上实现了简易版的《砖头派对》小游戏的方法。

1 选题及创意介绍

在《糖豆人：终极淘汰赛》中，《砖头派对》是一个经典的游戏项目：玩家操控代表自己的糖豆人在平台上左右移动，躲避前方迎面而来的各种障碍物，一段时间后没有被障碍物击落，仍然停留在平台上的玩家晋级。作者受此启发，结合 micro:bit 的特性，开发出了《砖头派对（Python 版）》这款小游戏。

(a) 《糖豆人：终极淘汰赛》中的《砖头派对》游戏截图



(b) 《砖头派对（Python 版）》运行截图

图 1 两款游戏的运行截图

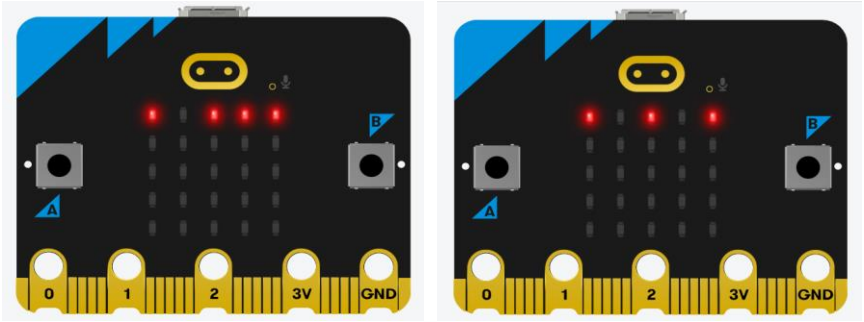
2 设计方案和硬件连接

2.1 设计方案

《糖豆人：终极淘汰赛》中的《砖头派对》（以下简称为原版）分为以下几个要素：玩家控制角色的移动、障碍物的生成和移动、玩家淘汰或晋级的判定、背景画面和音效。下面将分别介绍《砖头派对（Python 版）》（以下简称为 Python 版）中对于这几个要素的设计和简化。

玩家控制角色的移动是游戏的核心之一。在原版中，玩家控制的糖豆人可以在固定的大平台上前后左右自由移动。在 Python 版中，玩家所操控的对象变为 5×5 显示屏上的一个像素点，通过发光来指示其位置。考虑到 micro:bit 只有两个按键，实现二维移动较困难和只有 5 排像素点，玩家作出反应的时间较短两个因素，Python 版限定玩家只能在显示屏最后一排的 5 个像素点通过按下两侧的按钮实现向对应方向的一维移动。受到条件的限制（详见实现方案），在本游戏中，玩家的位置仅在每次躲避障碍物后显示，也增加了游戏的难度和趣味。

障碍物的生成和移动也十分重要。在原版中，障碍物有多种形状，对应的三维参数不同。在 Python 版中，虽然有 5×5 显示屏以及每个像素点有可调节的亮度，能够表示三维的信息，但经过测试亮度差异并不明显，因此我们放弃了三维障碍物的设定，将整个游戏限定在二维进行。此时主要的障碍物结构为有一个或两个开口的“墙”，如图 2 所示。障碍物在最上方一行，即最远离玩家所在位置处生成，每隔固定的时间向下靠近玩家所在的位置，直至移动到最后一行，进行是否与玩家发生碰撞，即淘汰和晋级的判定。



（a）有一个开口的“墙”　　（b）有两个开口的“墙”

图 2 两种障碍物的示意图

当障碍物移动到最后一行时，如何判断玩家晋级或是淘汰？原版中，没有来得及躲避的糖豆人会被打落平台，掉入下方的水里即宣告淘汰。Python 版的淘汰判定则更为简单：只要玩家与移动到最后一行的障碍相碰即为失败。若没有碰撞到，则障碍物消失，并在第一行重新生成新的障碍物，继续游戏。

对于背景画面和音效，由于 micro:bit 条件的限制，无法进行非常完善的设计。目前只在开始游戏前的倒计时、每次躲避障碍物成功/失败以及最后祝贺通关三个部分加入了对应的提示音效。

2.2 硬件连接

本游戏只需要连接 micro:bit 即可，所有的显示、音效、按键响应均使用 micro:bit 上的对应设备。

3 实现方案及代码分析

3.1 实现方案

由于游戏每一次躲避障碍物的流程是相同的，我们采用一个循环来进行主程序。每一循环中包括如下内容：

生成障碍物之前，系统随机选择一行中的一或两个位置作为墙上的开口，其余位置为障碍，产生障碍列表和空位列表。对于障碍列表中的每一个位置，只需点亮对应的像素点，即可表示障碍物的位置，从而实现在屏幕上生成障碍物。

障碍生成后，每隔一段时间后将其下一行的对应像素点亮，同时熄灭原本的像素点，即可在视觉上实现障碍物“移动”的效果。

由于 micro:bit 暂时不支持多线程（没有内置 threading 库），对于玩家是否按下按键不能实现实时检测，因此只能通过每个障碍物下落到最底部后，检测总按键次数的方式实现位置的改变，变换关系为：新的位置=原有位置-左按键按下的次数+右按键按下的次数。这样的限制导致我们不能在游戏中每一时刻对于玩家按键改变位置这一操作作出及时响应，因此在游戏设计上，规定玩家的位置只能在每次躲避障碍物后显示。显示方式为点亮对应位置的像素点，一段时间后熄灭。

当障碍物移动到最后一行时，检测玩家位置是否在最初生成的空位列表中，若是则说明成功躲避障碍，游戏继续；反之则游戏失败，跳出循环结束游戏。

当游戏通关（完成预定数目的关卡）或是失败后，会对游戏结果进行输出，包括“游戏结束”提示和完成的关卡数。若完整通关，还会使用内置的 music 模块播放特殊音效。

3.2 代码分析

```
from microbit import *
import random
import music
import time

#准备开始，倒计时三下后提示障碍物移动方向
display.show(3,delay=2000)
music.play(['c:7','r:3'])
display.show(2,delay=2000)
music.play(['c:6','r:3'])
display.show(1,delay=2000)
music.play(['c:6','r:3'])
display.show(Image('00900:'
                   '90909:'
                   '09990:'
                   '00900:'
                   '00000')) #屏幕上显示下箭头，代表障碍物移动方向
music.play(['c5:8'])
display.clear()

#参数设定、显示初始位置
waittime=500 #每一步操作的等待时间
current=random.choice([0,1,2,3,4])
display.set_pixel(current,4,9)
time.sleep_ms(2*waittime)
```

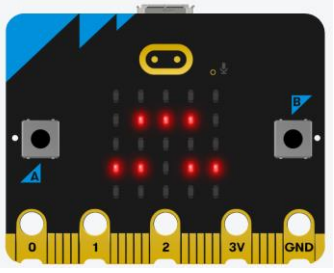
```
display.clear() #随机选择初始位置，在屏幕上显示一段时间后消失
score=0
level=20 #总关卡数

#主程序，每一关进行一次循环
for i in range(level):
    #每一关生成一个障碍物,生成对应障碍物的障碍位置列表和空位置列表，空位置可能有 1 或 2 个
    blocklist=[0,1,2,3,4]
    blanklist=[random.choice(blocklist) for _ in \
                                        range(random.randint(1,2))]
    blocklist=list(set(blocklist)-set(blanklist))

    #障碍物在第一行生成并不断下落
    y=0
    while y<=4:
        for block in blocklist:
            display.set_pixel(block,y,9)
        time.sleep_ms(waittime)
        for block in blocklist:
            display.set_pixel(block,y,0)
        time.sleep_ms(waittime)
        y+=1

    #检测按键次数左右移动位置，若在一关之内总位移超出了范围，会被墙壁挡住，位置停在 0 或 4，防止 index out of range 报错。
    current -= (button_a.get_presses()-button_b.get_presses())
    current=min(4,current)
    current=max(0,current)

    #每次显示当前位置，若增加难度，可将代码删去
    display.set_pixel(current,4,9)
    time.sleep_ms(waittime)
    display.clear()

    #检测是否发生碰撞，若是则游戏结束，播放失败语音；若否则游戏进入下一关，+1 分并播放得分语音。
    if current in blanklist:
        score+=1
        music.play(['c:1','e:1','g:1','c5:1'])
    else:
        display.set_pixel(current,4,9)
        time.sleep_ms(500)
        display.clear()
```

```
            display.show(Image.NO)
            music.play(['c5:1','g4:1','e:1','c:1'])
            sleep(1000)
            display.scroll('GAME OVER')
            display.scroll('SCORE:')
            display.show(score)
            break

        #增加游戏难度，每 5 关之后下落速度增加
        waittime=500-50*(score//5)

#检测是否通关，通关播放胜利音乐，显示分数
if score==level:
    sleep(1000)
    display.show(Image.YES)
    music.play(music.RINGTONE)
    sleep(1000)
    display.scroll('YOU WIN')
    display.scroll('SCORE:')
    display.show(score)
```

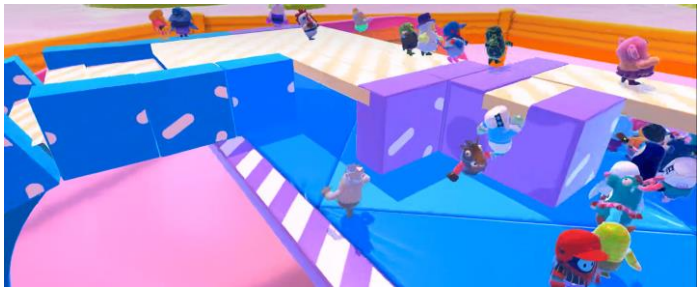
4 后续工作展望

4.1 障碍物

在原版游戏中，障碍物的形状千变万化，如图 3 所示。在 Python 版当中，我们还可以增加更复杂的障碍。但是如何在 micro:bit 中表示复杂障碍物，如何平衡游戏的难度还需要进一步的测试。



(a) (b) 短时间内多个障碍同时生成及其在 micro:bit 中的显示，但玩家操作时间太短难以过关



(c) 更加复杂的障碍物，但无法在 micro:bit 中显示

图 3 两种复杂障碍物

4.2 参数和机制调整

在代码中有以下几个参数可以进行调整：通关所需的总关卡数 level、每一步操作的等待时间 waittime、下落速度随关卡数增加的函数关系。这些参数可以根据玩家的情况随时进行调整，使游戏难度适中。

另外，还可以删除程序中每次显示当前位置的代码，让玩家需要时刻记住自己所在的位置，加大了游戏难度。

4.3 多线程

前面已经提到，受限于开发环境，不能进行多线程的操作。但是经过搜索和查找资料，在 micro:bit 平台上的 MakeCode editer 中，可以用 Scratch 和 Python 进行编译，实现了多线程操作。图 4 为代码截图，此程序在游戏的基础上添加了循环滚动播放的 BGM 和能够实时监测按键响应位置变化的功能，更加完善，游戏性更强。

(a)用 Scratch 编译的代码截图

```python
def on_button_pressed_a():
    myself.change(LedSpriteProperty.X, -1)
    basic.pause(200)
input.on_button_pressed(Button.A, on_button_pressed_a)

def on_button_pressed_b():
    myself.change(LedSpriteProperty.X, 1)
    basic.pause(200)
input.on_button_pressed(Button.B, on_button_pressed_b)

myself: game.LedSprite = None
basic.show_number(3)
music.play_sound_effect(music.create_sound_effect(WaveShape.NOISE,
        54,
        54,
        255,
        0,
        500,
        SoundExpressionEffect.NONE,
        InterpolationCurve.LINEAR),
    SoundExpressionPlayMode.IN_BACKGROUND)
basic.pause(500)
basic.show_number(2)
music.play_sound_effect(music.create_sound_effect(WaveShape.NOISE,
```

(b)Python 代码截图（部分）

图 4 在 MakeCode editor 上编译游戏程序的代码截图，
遗憾的是不能在课程要求的平台上实现，尽管编译语言一致。

5 小组分工合作

本项目由作者本人独立完成，创意来自游戏《糖豆人：终极淘汰赛》。

6 参考资料

游戏：《糖豆人：终极淘汰赛》

部分游戏实况图片来自网络。