实习报告

张佳奕　罗晟原

摘要：本篇实习报告主要介绍了本组编写 micro:bit 小游戏《Inverse》的创意、思路、主要过程、结果以及可能的后续工作。报告首先对作品进行定性，然后回顾方案建构之初创意的来源，以及大致方案形成的过程，之后对于方案的实现进行了大致的梳理，并分析重要的几段代码，最后总结整个开发过程，对于后续的进一步优化提出几种可能的方案，并介绍小组分工。

一、选题及创意介绍

本小组作品的定位是：运行于 micro:bit 上的单机游戏。

本小组作品的核心玩法是：倾斜 micro:bit 以左右移动小球，按下左右两个按键来发动两种技能，最终的目标是到达一个指定的终点。

创作作品，一定有最初的灵感来源，或是一个核心主题。所有工作都是围绕着这个主题进行不断地修饰和优化，并最终形成一个完整的架构。在填写报名表单的时候，我们选定了“反转”这个词作为主题。当时我们对于整个作品最终的样子以及如何实现并没有很明确的构思，仅仅是觉得这个主题有点意思，或许能探索出一点新东西出来。

我们最初的设想有两个方向，均是基于对“反转”这一主题最直接的思考。一种设想是反转重力，这是一些跑酷游戏或闯关游戏使用的经典套路；另一种设想是反转障碍物与可行走的通路，这一设想则主要源于常见的“突破或绕过障碍以到达终点”的游戏目的。然而我们很快遇到瓶颈：对于前者而言，反转重力的创意不仅过于老套，而且在 micro:bit 有限的显示能力下难以对玩家进行清晰的提示；而对于后者而言，则几乎是赋予了游戏主角无限穿墙的能力，这让整个闯关过程变得过于简单而失去趣味。

我们最终的选择是两种设想某种意义上的“结合”。我们仍然选择保留重力系统，但是将“反转重力”这一相对抽象的概念具化为“上下翻转”。游戏主角并不拥有跳跃的能力，但是可以无限制地反转障碍物与道路。最终的游戏目的仍然是到达一个已经标定的终点，但是具体位置在游戏开始时并不可知。这样一来，

整个游戏的难度中等，属于可以被接受的范围。

在已经确定的玩法的基础上，我们又对主题进行了进一步的思考：反转，反转什么？反转上下，反转障碍与道路都是反转，但是这仅仅局限于游戏内。我们还可以反转操作，让原本操纵玩家向左移动的操作变成向右，让原本朝向下方的重力变成向上。这样的思路最终体现于游戏中第二阶段的关卡。

最后，我们希望赋予这个游戏一段简单的剧情，以让整个游戏不至于显得太过单调和乏味，而剧情的原点仍然是“反转”这一主题。在剧情中，这款游戏具有一种神秘的力量，能够让一个人的精神被困在游戏里，唯一的逃脱方式是找到下一位受害者，反转二者的位置，从而让自己离开。最终，玩家可以在三种结局中做出选择：是困于游戏，还是暂时逃脱，抑或是破坏这个游戏来终结这一轮回。

二、设计方案和硬件连接

显示与操作均使用 micro:bit 自带的组件实现，没有搭载额外模块。使用到的组件主要有：显示器、两侧按钮、加速度传感器、麦克风以及 logo 引脚。

三、实现方案及代码分析

实现方案的过程中，以下几个变量是比较重要的。

Inverse.py

```
from microbit import *
##############################
#INVERSE
#关卡：一个嵌套列表，列表的每一个元素是从上往下的行，每一个元素的元素是行内从左到右的组件
#空地—0；终点—3；球—6；墙体—9
#当前位置：一个含两个元素的列表
#standard：倾斜角度的标准值，可调
#level_index：当前关卡
##############################
```

关卡是一个嵌套列表，每一个元素都是一个列表，它给出了这一行所有的组件。数字 0、3、6、9 既代表组件本身，又代表显示亮度，这样做主要是为了方便拼合和显示。

当前位置是一个二元数组(行数，列数)。Standard 变量主要是为了调整使小球开始滚动时的倾斜角度，没有什么实际的意义。Level_index 变量表示当前处于第几关。

在实现方案时，我们主要遇到两个问题。对于两个问题的解决最终都体现在了代码里。

第一个问题是 micro:bit 的显示能力有限，不能够展现出关卡的全貌。我们选择的解决方案是始终以游戏主角所在的位置作为显示的中心，展示主角周围的 25 个格子的情况。例外情况是主角位于地图的边上或角上，此时将会显示最边上或最角上的 25 个格子。

为了实现这一功能，我们编写了 present 函数，它在整段代码中有非常重要的地位。

Inverse.py

```
16 def present(input_level, input_current_position):
17     current_map = []
18     for row in input_level:
19         current_map.append([])
20         for number in row:
21             current_map[-1].append(number)
22     current_map[input_current_position[0]][input_current_position[1]] = 6
23     y = input_current_position[0]
24     x = input_current_position[1]
25     if x <= 1:
26         x = 2
27     if x >= len(current_map[0]) - 2:
28         x = len(current_map[0]) - 3
29     if y <= 1:
30         y = 2
31     if y >= len(current_map) - 2:
32         y = len(current_map) - 3
33     image = "".join(map(str,current_map[y - 2][x - 2:x + 3])) + ":" +\
34             "".join(map(str,current_map[y - 1][x - 2:x + 3])) + ":" +\
35             "".join(map(str,current_map[y][x - 2:x + 3])) + ":" +\
36             "".join(map(str,current_map[y + 1][x - 2:x + 3])) + ":" +\
37             "".join(map(str,current_map[y + 2][x - 2:x + 3]))
38     display.show(Image(image))
39     return
```

Present 函数接收两个参数：本关地图和当前位置，没有返回值。变量 current_map 是应当被展示的 25 个格子，变量 x、y 表示此时展示的中心。当小球没有“靠边”时，(x, y)就等于小球的位置；当小球贴边时，就要做相应的修改。最后将列表拼合，展示出来。

第二个问题是 micro:bit 的存储能力有限，移动代码必须写成一个函数，并且在函数中要能够改变外部变量的值（比如小球当前位置）。因此我们采用全局变量的方式，并写出了 move 函数。

Inverse.py

```
40
41  #移动函数，当倾斜方向没有墙体或不是边缘时移动，可反转
42  def move(inversed):
43      if inversed == 0:
44          if current_position[0] < len(level) - 1 and \
45             level[current_position[0] + 1][current_position[1]] != 9:
46              current_position[0] = current_position[0] + 1
47          if standard <= accelerometer.get_x():
48              if current_position[1] < len(level[0]) - 1 and \
49                 level[current_position[0]][current_position[1] + 1] != 9:
50                  current_position[1] = current_position[1] + 1
51          elif accelerometer.get_x() <= -standard:
52              if current_position[1] > 0 and \
53                 level[current_position[0]][current_position[1] - 1] != 9:
54                  current_position[1] = current_position[1] - 1
55      else:
56          if current_position[0] > 0 and \
57             level[current_position[0] - 1][current_position[1]] != 9:
58              current_position[0] = current_position[0] - 1
59          if -standard >= accelerometer.get_x():
60              if current_position[1] < len(level[0]) - 1 and \
61                 level[current_position[0]][current_position[1] + 1] != 9:
62                  current_position[1] = current_position[1] + 1
63          elif accelerometer.get_x() >= standard:
64              if current_position[1] > 0 and \
65                 level[current_position[0]][current_position[1] - 1] != 9:
66                  current_position[1] = current_position[1] - 1
67      sleep(300)
68      present(level, current_position)
```

Move 函数只接收一个参数，它表示游戏是否已经进入第二阶段，操作已经反转。前后的两段代码并不是一次性写成的，而是做到第二阶段时才补上的第二段代码。Move 函数的大体思路还是比较简单的，竖直方向上是始终往下掉落直到下面是障碍，水平方向上是倾斜，如果没有障碍就移动。Move 函数调用了 present 函数，这样写代码时就更方便了。Move 函数设定为每 0.3 秒执行一次。

在这两个核心函数的基础上，代码的其他部分虽然很长，但是都是简单的工作。两个反转函数分别实现关卡中 0 与 9 的互换和关卡列表的倒序；根据关卡数设计不同的嵌套列表和终点位置，然后再用一个 while 循环来检查是否已经到达终点等等。举两关为例：

```
156     if level_index == 3:
157         level = [[0,0,0,9,0,0,9,0,0,0,0,9,0,9,0],\
158                  [0,9,9,9,0,0,9,9,9,9,0,9,0,0,0],\
159                  [0,9,9,0,0,9,9,0,9,0,0,0,0,9,9],\
160                  [0,0,0,0,0,0,0,0,9,0,9,0,0,9,0],\
161                  [9,9,0,9,9,9,0,9,9,0,9,9,0,9,0],\
162                  [9,9,0,9,9,0,0,9,0,0,0,9,0,9,0],\
163                  [0,0,0,9,0,0,9,9,0,9,0,9,0,9,0],\
164                  [0,9,9,9,0,9,9,0,0,9,0,0,0,9,0],\
165                  [0,9,9,9,0,9,0,0,9,9,9,0,9,9,0],\
166                  [0,0,0,0,0,9,0,9,9,9,9,0,9,0,0],\
167                  [9,9,9,9,0,9,0,0,0,0,9,0,0,0,9],\
168                  [9,9,9,0,0,9,9,9,9,0,9,0,9,0,0],\
169                  [0,0,0,0,9,9,0,0,0,0,9,0,9,0,9],\
170                  [0,9,9,9,9,9,0,9,9,0,9,9,9,0,0],\
171                  [0,0,0,0,9,0,0,9,9,0,0,3,9,9,0]]
172         current_position = [0,2]
173         destination = [14,11]
174     if level_index == 4:
175         level = [[3,0,9,9,0],\
176                  [0,9,0,9,0],\
177                  [9,0,0,9,0],\
178                  [9,9,9,9,9],\
179                  [0,0,0,9,0]]
180         current_position = [4,4]
181         destination = [0,0]
```

在游戏的第一阶段和第二阶段，代码都是本质相同的，因此不再重复展示。

在游戏的第三阶段，我们还加入了一些解谜要素，并在剧情中给予提示。比如第一小关中，屏幕滚动文字“HELP ME”。首先玩家会通过屏幕上随着声音强度而变化的图案了解到这关与麦克风有关，然后屏幕上图案点与直线的变化提示玩家应当用声音发出“SOS”的摩尔斯电码来通过这一关卡。这样的小解密能够充分利用 micro:bit 自带的各种传感器，还与剧情相关联，并且有一定的趣味性。下面展示这一小关后半部分的代码。

```
sound_list = []
duration = 0
while True:
    if microphone.sound_level() < 30:
        if 1 < duration <= 10:
            sound_list.append(0)
            print(sound_list)
        elif duration > 10:
            sound_list.append(1)
            print(sound_list)
        clear()
        duration = 0
    else:
        duration = duration + 1
        if 1 < duration <= 10:
            display.show(Image("00000:00000:00900:00000:00000"))
        elif duration > 10:
            display.show(Image("00000:00000:99999:00000:00000"))
        sleep(100)
    if sound_list[-9:] == [0,0,0,1,1,1,0,0,0]:
        break
```

列表 sound_list 记录所有玩家输入的符号，duration 记录当前声音持续的时间，以 0.1s 为单位。当声音强度低于 30 时，如果上段声音持续时间超过 0.1s，认为是有效输入，不超过 1s 认为是“.”，超过一秒认为是“-”，然后重置 duration。当声音强度不低于 30 时，认为是正在输入，此时屏幕上会根据当前持续时间相应地提示“.”或“-”。

四、后续工作展望

可能的后续工作主要有以下两个方面。

一是扩大显示范围，增加关卡规模。如前所述，micro:bit 的显示器只能显示 25 个格子，这实际上极大地限制了关卡的最大规模。试想：一个 30×30 的迷宫，在只能看到周围 25 个格子，且事先不知道终点方向的情况下，怎么可能顺利通关呢？况且这还没有计入不断反转上下而需要的额外记忆力。事实上，作者自己在游玩上面代码展示的第 3 关（它是一个 15×15 的迷宫）时，已经感到非常吃力。而关卡规模小又让整个游戏变得单调，没有什么新花样，这一点是我们对于最终作品不太满意的一个方面。

二是优化显示能力，丰富游戏内容。25 个像素点，每个像素点虽然有 0~9 共 10 档亮度可调，但是实际应用中根本不可能充分利用 10 个亮度，这主要是因为相邻档数之间的亮度实在太过接近，难以清晰地展现出游戏中不同的组件。于是在成品中，我们只（主要）用到了四个亮度，这使得游戏内容较为单一。如果能够加入显示不同颜色的能力，那么就可以加入更多的游戏要素，比如机关、陷阱等等，让游戏变得更富于变化。

总而言之，micro:bit 受限的显示能力和存储能力确实为我们的编程造成了不少的麻烦。但是从另一个角度想，不断优化自己的代码，让游戏能够成功地在性能受限的情况下运行，又何尝不是一种乐趣呢？

五、小组分工合作

张佳奕：最初创意，海报制作，报告撰写；

罗晟原：思路扩展，代码实现，视频制作。