作品实习报告

Microbit 作品：钢琴块儿

作者：徐正贤 袁帅

摘要

我们小组的作品是通过 Microbit 自带的 5*5 LED/光线传感器、加速度传感器与实体按键，将前几年大火的手机游戏《钢琴块儿》的玩法在轻巧便携的 python 开源硬件上实现。玩家可以在软件中选择不同模式与乐曲，进入游戏后通过屏幕的指示击打相应按键或进行其他操作，模拟演奏出音乐。

一、选题及创意介绍

小组成员在对 Microbit 的功能与外型进行分析后，发现其适合手持的外型与早期的游戏掌机十分相像，因此小组萌生了开发移植 Microbit 游戏的想法。在经过挑选后，我们选择了前段时间大火的手机游戏《钢琴块儿》，并结合 Microbit 的特性进行改动，加入了新的功能玩法与创意。

二、设计方案和硬件连接

当玩家按下 logo 键时，主题音乐响起，并进入游戏的 UI。游戏系统的操作由 logo 键、A 键、B 键进行，分别代表确定，向左，向右。

首先，我们为游戏设计了模式选择的功能。模式选择有两种：休闲娱乐模式、计时挑战模式。由 AB 键进行模式切换，logo 键进行模式选择。

1.休闲娱乐模式（classic）

顾名思义，即没有任何游玩压力，让玩家能够练习娱乐的模式，玩家在此模式可以不受时间限制地游玩曲目，如果曲目全程无失误，则显示通关。

2.计时挑战模式（speed）

在规定时间内让玩家尽可能打击更多次数，并记录分数的模式，期间需要玩家没有失误，否则游戏也会立即结束。在游戏结束后会显示玩家的分数，因此有一定的竞技性。

其次，我们为游戏设计了歌曲选择的功能。玩家在选择模式之后，就可以选择相应的歌曲，并进入游戏。同样的，AB 键进行乐曲的切换，logo 键进行乐曲的选择。玩家在游戏结束后，显示屏对结果进行显示，此时按下 logo 键，可以重新开始游戏。

进入游戏，画面上会显示出两个轨道，分别对应 microbit 的 A 键与 B 键，上面有需要击打的音符，玩家要按顺序正确的击打音符，每击打成功一个音符，便会有对应的音乐声响起（声音调用提前录入进去的乐谱）。同时游戏中也有模拟钢琴踏板的音符（一行全亮），玩家需要摇晃 microbit，通过加速传感器获取摇晃的信号，实现音符的打击。

因为游戏是在 Microbit 上独立运行、操作、游玩的，因此没有设计额外的硬件连接方案。但我们也大致构想了游戏未来的一些外接设计，详情请看下文【四、后续工作展望】部分。

三、实现方案及代码分析

```
# Imports go at the top
from microbit import *
import music
import gc
import random
```

此处为所需要使用的库。

```
# abridged version of Prelude, for testing purposes
preludeShort = [
    'c4:1', 'e', 'g', 'c5', 'e5', 'g4', 'c5', 'e5', 'c4', 'e', 'g', 'c5', 'e5', 'g4', 'c5', 'e5',
    'c4', 'd', 'a', 'd5', 'f5', 'a4', 'd5', 'f5', 'c4', 'd', 'a', 'd5', 'f5', 'a4', 'd5', 'f5',
    'b3', 'd4', 'g', 'd5', 'f5', 'g4', 'd5', 'f5', 'b3', 'd4', 'g', 'd5', 'f5', 'g4', 'd5', 'f5',
]
```

此处为所使用歌曲的乐谱，格式为 microbit 中 music.play 所能识别的列表形式。

播放音符 更少

您可以使用乐谱来演奏曲调：

```
import music

music.play(['c', 'd', 'e', 'c'])
```

默认情况下，它在第 4 个八度音阶中播放音符。您可以指定其他八度音阶。

此示例在第 3 个八度音阶中演奏音符 C、D、E、C：

```
import music

music.play(['c3', 'd3', 'e3', 'c3'])
```

您可以在音符名称后使用冒号和数字来选择每个音符的播放时间。在这里，音符 G4 播放的时间是其他音符的两倍：

```
import music

music.play(['e4:4', 'f4:4', 'g4:8'])
```

可以使用音符名称"r"来添加休止符，从而添加您指定的任何时长的停顿。

您可以通过在音符名称中添加"b"来使用降号音符，在音符名称中添加"#"来使用升号音符。这会演奏降 A、自然 A和升 A：

```
import music

music.play(['ab', 'a', 'a#'])
```

Music.play 的规则如上。

```
# contains string name, note list, and bpm for each music
musicList = [('PRELUDE', prelude, 80), ('SHORT PRELUDE', preludeShort, 80),
             ('NYAN CAT', nyan, 300), ('SHORT NYAN CAT', nyanShort, 300)]
```

一个存放音乐相关信息的列表。方便后面选择音乐时进行调用。

```
class Queue:
    def __init__(self):
        self.items = []

    def enqueue(self, item):
        return self.items.insert(0, item)

    def dequeue(self):
        return self.items.pop()

    def isEmpty(self):
        return self.items == []

    def size(self):
        return len(self.items)

    # def display(self):
    #     print(self.items)
```

需要在后续使用的 Queue 类

```
class Game:
    # initialize board (5x5 grid)
    def __init__(self):
        self.screen = [0,0,0,0,0] * 5

    # "shift" the grid down by removing bottom row and adding new row to the top
    def add_line(self, list):
        self.screen = list + self.screen[:20]

    def get_screen(self):
        return self.screen
```

Class Game 实现了游戏画面的基本运行逻辑。初始的游戏画面为 5x5 的空白画面。当执行 add_line 的时候，则是将最下一行的五个元素删除（通过切片的方式），并添加新的一行在最上端，以此来实现‘音符’向下掉落的效果。Get_screen 返回当前的游戏画面。

```
class Line:
    # line (containing "piano tile") with a key value associated to it
    def __init__(self):
        self.line = []
        self.key = -1

    # generate a random "piano tile" and associated key
    # key: 1 = left tile; 2 = right tile; 0 = both tiles
    def gen_line(self):
        x = random.randint(0,100)
        if x <= 49:
            self.line = [0,9,0,0,0]
            self.key = 1
        elif x >= 50 and x <= 99:
            self.line = [0,0,0,9,0]
            self.key = 2
        elif x == 100:
            self.line = [9,9,9,9,9]
            self.key = 0

    # same as gen_line(), except no double tiles (for SPEED mode to remove rng)
    def gen_line2(self):
        x = random.randint(0,1)
        if x == 0:
            self.line = [0,9,0,0,0]
            self.key = 1
        elif x == 1:
            self.line = [0,0,0,9,0]
            self.key = 2


    def get_line(self):
        return self.line
    def get_key(self):
        return self.key
```

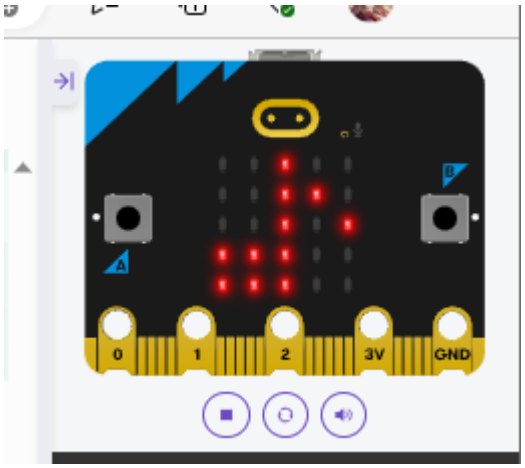
Class line 实现了以行为单位的音符的生成。一个初始的 line 为空列表，其对应的 key 为异常值-1。gen_line 生成每一行的音符，首先通过 random 方法随机生成 0-100 的自然数，并设定范围分配相应的值（1 为 A 键，2 为 B 键，0 为加速传感器中的 shake）。Getline、

getkey 分别返回该行音符与对应需要按的 key 值。分别定义了两个 gen_line，对应两个不同的模式。因为考虑到 shake 相对于按键更耗费时间，且具有随机性，会影响 speed 模式分数的公平性。因此 shake 按键只在 classic 模式生成。

```
# on a filled grid (tiles in all rows), "shifts" the tiles down, creating a new one in the process
# uses a Queue to keep track of keys for each line of tiles
def cont():
    line.gen_line()
    game.add_line(line.get_line())
    keyQ.enqueue(line.get_key())
    display.show(Image(5,5,bytearray(game.get_screen())))
```

Cont 函数进行了完整的一次音符下落的操作，即让所有音符平移下来，创建一行新音符放在最上面，并显示在屏幕上。Cont2 与 Cont 原理相同，仅是为了区分不同的 mode，在此不多做赘述。

```
# start up
display.show(Image.MUSIC_QUAVER)
music.set_tempo(bpm=60)
music.play(music.POWER_UP)
```



Start up 开机界面生成如上的一个音符图案，并播放音乐。

我们将这个游戏的结构分成几个 stage：stage0 为开机的界面，stage1 为选择歌曲界面，stage2 为选择模式界面。Stage3 为选择 speed 模式后的限制时间选择界面，stage4 为 classic 模式的游玩界面，stage5 为 speed 模式的游玩界面。接下来我将分 stage 对代码进行讲解。

```
while True:
    # start the game by pressing gold logo (activates upon release)
    # the while loop for when the gold logo is touched is important, as it is considered held down
    # being held down will activate later gold logo condition checks
    stage = 0
    if stage == 0 and pin_logo.is_touched():
        while pin_logo.is_touched():
            pass
        stage = 1

    # stage 1: select music from list of available music
    # button get_presses() to reset counter when playing through a second time
    if stage == 1:
        m = p = 0
        n = q = -1
        menu = musicSelection = True
        button_a.get_presses()
        button_b.get_presses()

        # display.scroll() works in background in loop, only want to call it once when necessary
        # otherwise will keep restarting in a while loop
        # this logic will be used a lot (m != n, p != q)
        while menu == True:
            if p != q:
                display.scroll('SELECT MUSIC', delay=65, loop=True, wait=False)
                q = p
            if button_a.get_presses() == 1 or button_b.get_presses() == 1:
                menu = False

        # select music from musicList, using m as index
        while musicSelection == True:
            if m != n:
                currentMusic = musicList[m]
                display.scroll(currentMusic[0], delay=65, loop=True, wait=False)
                n = m
            if button_b.get_presses() == 1:
                m = (m + 1) % len(musicList)
            if button_a.get_presses() == 1:
                m = (m + len(musicList) - 1) % len(musicList)
            if pin_logo.is_touched():
                while pin_logo.is_touched():
                    pass
                melody = currentMusic[1]
                musicSelection = False
                stage = 2
```

这个大 while 语句是为了进入游戏，并选择音乐。（即 stage0 与 stage1）我们使用有关 logo 的按键（pin_logo.is_touched()）的 while 语句来判断是否触发，具体实现方法详见注释。当玩家触发按键，即进入 stage1。

m = p = 0 n = q = -1 两句是为了后续语句判断的布尔值而设置。并无实际作用（没有想到更好的方案）button get_presses()是为了在后续重新游玩时，清除之前按键的记录。进入 stage1, 首先进入 menu 界面, 滚动'SELECT MUSIC', 直到按键触发使 menu 的布尔值为 false, 进入实际歌曲的选择。实际歌曲的选择则是通过对 m，n 的加法，整除（循环歌曲列表）操作进行的，当玩家选定歌曲，点击 logo（if pin_logo.is_touched()），则进入 stage2。

```
# stage 2: selecting gamemode
# two gamemodes: classic and speed
# code is very similar to stage 1
if stage == 2:
    m = p = 0
    n = q = -1
    menu = gamemodeSelection = True

    while menu == True:
        if p != q:
            display.scroll('SELECT GAMEMODE', delay=65, loop=True, wait=False)
            q = p
        if button_a.get_presses() == 1 or button_b.get_presses() == 1:
            menu = False

    while gamemodeSelection == True:
        if m == 0:
            if m != n:
                display.scroll('CLASSIC', delay=65, loop=True, wait=False)
                n = m
        if m == 1:
            if m != n:
                display.scroll('SPEED', delay=65, loop=True, wait=False)
                n = m
        if button_a.get_presses() == 1 or button_b.get_presses() == 1:
            m = (m + 1) % 2
        if pin_logo.is_touched():
            while pin_logo.is_touched():
                pass
            if m == 0:
                stage = 4
            else:
                stage = 3
            gamemodeSelection = False
```

stage2（游戏模式选择）的运作机理与 stage1 相同，在此不做赘述。

```
# stage 3: time limit selection if chose SPEED gamemode
# 6 available time limits: 5, 10, 15, 20, 25, 30 seconds
# again, uses very similar code to previous two stages, here directly uses m
if stage == 3:
    m = p = 0
    n = q = -1
    menu = timeSelection = True

    while menu == True:
        if p != q:
            display.scroll('SELECT TIME', delay=65, loop=True, wait=False)
            q = p
        if button_a.get_presses() == 1 or button_b.get_presses() == 1:
            menu = False

    while timeSelection == True:
        if m != n:
            display.scroll(str(m + 5), delay=65, loop=True, wait=False)
            n = m
        if button_b.get_presses() == 1:
            m = (m + 5) % 30
        if button_a.get_presses() == 1:
            m = (m + 25) % 30
        if pin_logo.is_touched():
            while pin_logo.is_touched():
                pass
            timeLimit = m + 5
            timeSelection = False
            stage = 5
```

Stage3 选择 speed 模式的游戏限制时间。运作机理依然很接近前两个 stage。
需要注意时间选择时 timelimit 与 m 的不同：m 是为了方便计算而设置的变量，并非实际的

时间。最后要在 m 上+5，才是最终游玩时的时间（也是 display 上显示的时间）

```
# stage 4: begin playing CLASSIC gamemode for selected music
if stage == 4:
    game = Game()
    line = Line()
    keyQ = Queue()
    c = 0
    # tempo set at 400 to allow quick gameplay while maintaining note length
    music.set_tempo(bpm=400)

    # initialize the board with the first 5 rows of tiles
    for i in range(5):
        cont()
        sleep(300)

    stop = False
```

Stage4 为 classic 的游玩界面。因代码较长，分为几个部分讲述。
首先进入 stage4，建立几个需要使用的类，并先生成五行音符（玩家需要先击打音符才能生成下一个音符，因此最开始五个音符的生成是必要的）

```
#iterate through each note in the music list
for note in melody:
    if stop == True:
        break
    key = keyQ.dequeue()
    while True:
        if key == 1:
            if button_a.get_presses() == 1 and button_b.is_pressed() == 0:
                music.play(note)

                # prevent extra rows of tiles spawning after music ended
                if c < len(melody) - 5:
                    cont()
                    c += 1
                    # print(c)
                    break
                else:
                    game.add_line([0,0,0,0,0])
                    keyQ.enqueue(line.get_key())
                    display.show(Image(5,5,bytearray(game.get_screen())))
                    break
            elif button_b.get_presses() != 0:
                stop = True
                break
        elif key == 2:
            if button_b.get_presses() == 1 and button_a.get_presses() == 0:
                music.play(note)
                if c < len(melody) - 5:
                    cont()
                    c += 1
                    break
                else:
                    game.add_line([0,0,0,0,0])
                    keyQ.enqueue(line.get_key())
                    display.show(Image(5,5,bytearray(game.get_screen())))
                    break
            elif button_a.get_presses() != 0:
                stop = True
                break
        elif key == 0:
            if accelerometer.was_gesture('shake') and button_a.get_presses() == 0 and button_a.get_presses() == 0:
                music.play(note)
                if c < len(melody) - 5:
                    cont()
                    c += 1
                    break
                else:
                    game.add_line([0,0,0,0,0])
                    keyQ.enqueue(line.get_key())
                    display.show(Image(5,5,bytearray(game.get_screen())))
                    break
            elif button_a.get_presses() != 0 or button_b.get_presses() != 0:
                stop = True
                break
```

接下来迭代乐谱 list 中的每一个元素，生成音符。
当 stop 布尔值为 true 时，通过 key 值获取每一个音符对应的 key 值，并判断按键（或 shake）是否与 key 值相吻合，以此来达到判断玩家是否按到了正确按键的目的。当玩家按到正确按键，则消除旧音符，更新一行新的音符。
当玩家按到的按键与 key 值不符合，则使 stop 布尔值变为 true，并中断接下来的音符生成。需要注意的是因为 classic 模式音乐不循环，因此设置了一个判断音乐结束前音符不够生成五行音符时的特殊情况，详见代码 299-304。

```
344         # execute if failed midway through music
345         if stop == True:
346             music.set_tempo(bpm=60)
347             display.show(Image.SAD)
348             music.play(music.POWER_DOWN)
349         # execute if level completed successfully
350         else:
351             music.set_tempo(bpm=currentMusic[2])
352             display.show(Image.HAPPY)
353             music.play(melody)
```

当游戏结束后，通过 stop 布尔值来判断游戏进行的情况，输出不同的结果。

```
355 # stage 5: being playing SPEED gamemode for chosen music and time limit
356 if stage == 5:
357     game = Game()
358     line = Line()
359     keyQ = Queue()
360     c = 0
361     # tempo set at 400 to allow quick gameplay while maintaining note length
362     music.set_tempo(bpm=400)
363 
364     # initialize the board with the first 5 rows of tiles
365     for i in range(5):
366         cont2()
367         sleep(300)
368 
369     stop = False
370 
371     m = 0
372     n = -1
```

Stage5 为 speed 模式下的游戏界面。同样，因代码较长的缘故，分段进行说明。
前面的部分与 stage4 相同，仅有新变量 mn 是为了进行时间的计算。

```
374     # set up variables for timing
375     # timeElapsed indicates time passed after starting timer
376     # timer indicates whether or not the timer is running
377     # completed indicates whether or not the player has played through without failing
378     # inGame indicates in the game (endless, so will repeat music lists)
379     timeElapsed = 0
380     timer = completed = False
381     inGame = True
382     while inGame == True:
383         if stop == True:
384             break
385         # iterate through each note in the music list
386         for note in melody:
387             if stop == True:
388                 break
389             # start timer after pressing first tile
390             if m != n and c == 1:
391                 start = time.ticks_ms()
392                 timer = True
393                 n = m
394 
395             key = keyQ.dequeue()
396 
397             #keep playing as long as time elapsed doesn't go over time limit
398             while timeElapsed < timeLimit * 1000:
399                 # once timer is activated, constantly keep track of time elapsed from while loop
400                 if timer == True:
401                     timeElapsed = time.ticks_ms() - start
402                 if key == 1:
403                     if button_a.get_presses() == 1 and button_b.is_pressed() == 0:
404                         music.play(note)
405                         cont2()
406                         c += 1
407                         break
408                     elif button_b.get_presses() != 0:
409                         stop = True
410                         break
411                 elif key == 2:
412                     if button_b.get_presses() == 1 and button_a.get_presses() == 0:
413                         music.play(note)
414                         cont2()
415                         c += 1
416                         break
417                     elif button_a.get_presses() != 0:
418                         stop = True
419                         break
420             else:
421                 stop = True
422                 inGame = False
423                 completed = True
```

此处编写了 speed 的运作逻辑。timeElapsed 表示启动定时器后经过的时间，timer 表示定时器是否在运行，通过 timeElapsed 和 start 的时间记录相减（通过 time.ticks_ms()来实现计时）得到当前的游戏时间，并后续与 timelimit 进行比较。completed 表示玩家是否在没有失败的情况下完成了游戏。

需要注意的是，因为 speed 模式下歌曲是无限循环的，因此不像 stage4 一样，需要对 music len 少于 5 的情况进行特殊的安排。

```
426     # execute if failed midway through music
427     if completed == False:
428         music.set_tempo(bpm=60)
429         display.show(Image.SAD)
430         music.play(music.POWER_DOWN)
431     # execute if level completed successfully
432     else:
433         music.set_tempo(bpm=currentMusic[2])
434         display.scroll(str(c), delay=65, loop=True, wait=False)
435         music.play(melody)
```

最后根据游玩的情况进行结果的输出。因为游戏结构整体采用 stage 的分割形式，因此玩家

在游玩结束后，可以直接返回到先前的循环，再次开始新的游戏。

四、后续工作展望

我们的作品仍然有很多值得完善的地方，比起一个完整的项目，倒不如说更像是一个demo，有大量的新功能可以添加与挖掘。小组成员后续将对项目进行以下几个方向的开发：

1. 玩法的增加

目前游戏只有两个键与一个摇晃的操作，游戏画面也基本只用到了 2、4 轨道，玩法相对单薄。后续可能会结合 Microbit 的其他功能与传感器，进行更多玩法的创新。比如同时按下两个键演奏乐曲中的 chord，并结合加速度传感器的其他功能增加 shake 以外的音符。比如将 5x5 的屏幕利用起来，最左、右面的音符需要玩家手持 Microbit 向左、右倾斜再击打音符，甚至是通过 v2 新增的麦克风来实现唱歌击打音符……

2. 外部硬件连接

Microbit 支持使用 Chrome 或 Edge 网络浏览器在编辑器中访问串行控制台，使用计算机的键盘将信息输入到 Python 程序中。因此后续可以添加端口，使玩家只要将 microbit 连接到电脑，就可以通过串行控制台将想要演奏的歌曲便捷地录入到机器中（只要遵守 music.play 的格式）

此外，microbit 的底部边缘有金色引脚，可用于连接触摸传感器。后续可以使用触摸传感器代替原本的按键，让玩家更有演奏乐器的代入感。

3. 联网排名

当 Microbit 连接电脑时，通过串行控制台输出机器内记录的最高分数，并上传到服务器。玩家可以通过服务器查看自己的排名，提高游戏的竞技性。

五、小组分工合作

袁帅：软件编写、debug、视频拍摄、游戏设计、Poster 设计

徐正贤：游戏设计、曲谱编写、软件编写、实习报告编写