

Microbit: 生命游戏进阶

Game of Life PLUS

王子宸 化学与分子工程学院

石城玮 生命科学学院

2023 年 5 月 14 日

目录

0 摘要	2
1 选题及创意介绍	2
2 设计方案与硬件连接	3
2.1 设计方案	3
2.2 硬件连接	4
3 实现方案与代码分析	4
3.1 实现方案	4
3.1.1 模式选择	4
3.1.2 mod1 与 mod2 的主流程实现	5
3.1.3 mod1: 简单模式	6
3.1.4 mod2: 复杂模式	7
3.2 函数定义	8
3.2.1 draw_universe、draw_rev_universe 与 blink_pixel	8
3.2.2 evolve_1 与 get_neighbours_count	8
3.2.3 evolve_1、get_neighbours_diff 与 get_neighbours_cc	9
3.2.4 pixel_set_1 与 pixel_set_2	11
3.3 初始化设置	14
3.3.1 current_universe	14
3.3.2 mod2 参数设置	14
3.4 主程序	14

3.4.1	引入	14
3.4.2	游戏模式选择	15
3.4.3	游戏模式介绍	15
3.4.4	mod1: 简单模式	16
3.4.5	mod2: 复杂模式	18
4	后续工作展望	21
4.1	简单模式的改进	21
4.2	复杂模式的改进	23
4.3	单片机的改进	23
5	小组分工合作	23

0 摘要

生命游戏是由英国数学家康威自主研发的一款开放世界冒险游戏。受到其启发，我们小组在原版生命游戏规则的基础上进行了一定的扩展，加入了更多的元素从而进一步模仿了生物的多样性，在此基础上，不同的元素遵循不同的基本变化原则，从而可以实现多种多样的变化，甚至在经过适当的组合以后，可以用于进行计算。我们采用了抽象，模拟，组合的方式，在 5×5 的 Micro:bit 上实现了这一系统。其结果深刻揭示了“涌现特征”的原理，系统的功能多于每一个组分的功能之和，在某种程度上复现了生命从简到繁的演化。

1 选题及创意介绍

生命游戏 (Game of Life)，是由英国数学家约翰霍顿康威于 1970 年设计的细胞自动机。这是一个零玩家游戏，这意味着它的进化是由它的初始状态决定的，不需要进一步的输入。人们通过创建初始配置并观察它如何演变来与生命游戏互动。它是图灵完备的，可以模拟通用构造函数或任何其他图灵机。

生命游戏的宇宙是一个无限的二维正交网格，由方形细胞组成，每个细胞都处于两种可能的状态之一：生存或者死亡。每个像元都与其八个相邻的像元相互作用，这些像元是水平、垂直或对角线相邻的像元。在每个步骤中，每个细胞按照以下规则进行转换：

1. 任何具有两个或三个活邻居的活细胞都能存活下来。
2. 任何具有三个活邻居的死细胞都成为活细胞。
3. 所有其他活细胞在下一代死亡。同样，所有其他死细胞都保持死亡状态。

初始模式构成了系统的种子。第一代是通过将上述规则同时应用于种子中的每个细胞来创建

的，无论是活的还是死的。每一代都是前一代的纯函数。这些规则继续反复应用，以创造下一代。

考虑一种更为复杂的情形，将生命游戏的生存细胞种类由一种扩展为 3 种，且每种细胞分别具有一定的内禀属性：独立度、竞争力与扩散系数：

1. **独立度**：用于衡量一种生存细胞对周围细胞的依赖度，若周围细胞总数小于依赖度，则细胞因孤独而死
2. **竞争力**：用于衡量一种生存细胞，其周围的拥挤程度，如果周围细胞的总竞争力大于一常数，则细胞因拥挤而死
3. **扩散系数**：用于衡量死亡细胞周围每种细胞的总扩散系数，按照竞争力由大到小的顺序，优先诞生竞争力大，且死亡细胞周围的该种竞争力的细胞的总扩散系数，大于该种竞争力细胞的独立度，则在死亡细胞的位置生成该种竞争力的细胞

通过这样的方式将生命游戏的细胞演化跟更为复杂、生动而有趣。

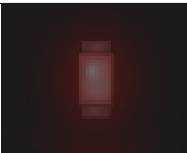
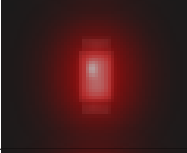
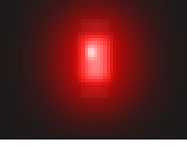
2 设计方案与硬件连接

2.1 设计方案

设计两种模式，分别对应简单生命游戏、复杂生命游戏：

- **简单生命游戏**：按照康威生命游戏规则运行
- **复杂生命游戏**：含有三种生存细胞与一种死亡细胞，基于独立度、竞争力、扩散系数与扩散阈值演化的生命游戏

表 1: 三种生存细胞的特性

细胞类型 (对应显示亮度)	显示效果	细胞特性
3		能够快速的繁殖, 对其他细胞抑制力弱, 但是只有周围有不少细胞才能存活
6		繁殖慢, 但是可以独立生存, 且会抑制周围细胞的生长
9		繁殖速度, 周围最适细胞数量, 对周围细胞的抑制力均中等

2.2 硬件连接

由于生命游戏并不需要除去 Micro:bit 之外的硬件，因此，使用[Micro:bit 官网](#)提供的在线 Micro:bit 进行模拟仿真实验。

3 实现方案与代码分析

3.1 实现方案

3.1.1 模式选择

本程序分为两大模式，分别为：

1. 简单模式 **mod1**：实现康威生命游戏
2. 复杂模式 **mod2**：实现复杂生命游戏：

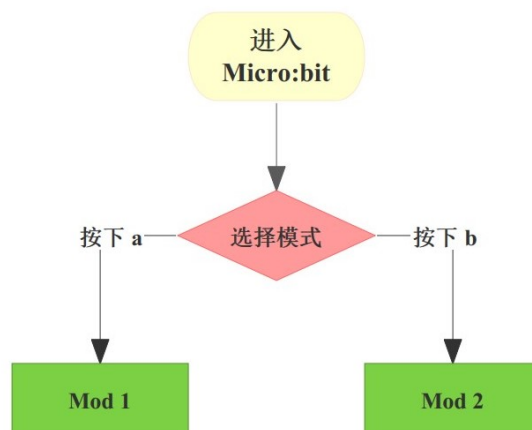


图 1: 简单模式 mod1 的实现流程

3.1.2 mod1 与 mod2 的主流程实现

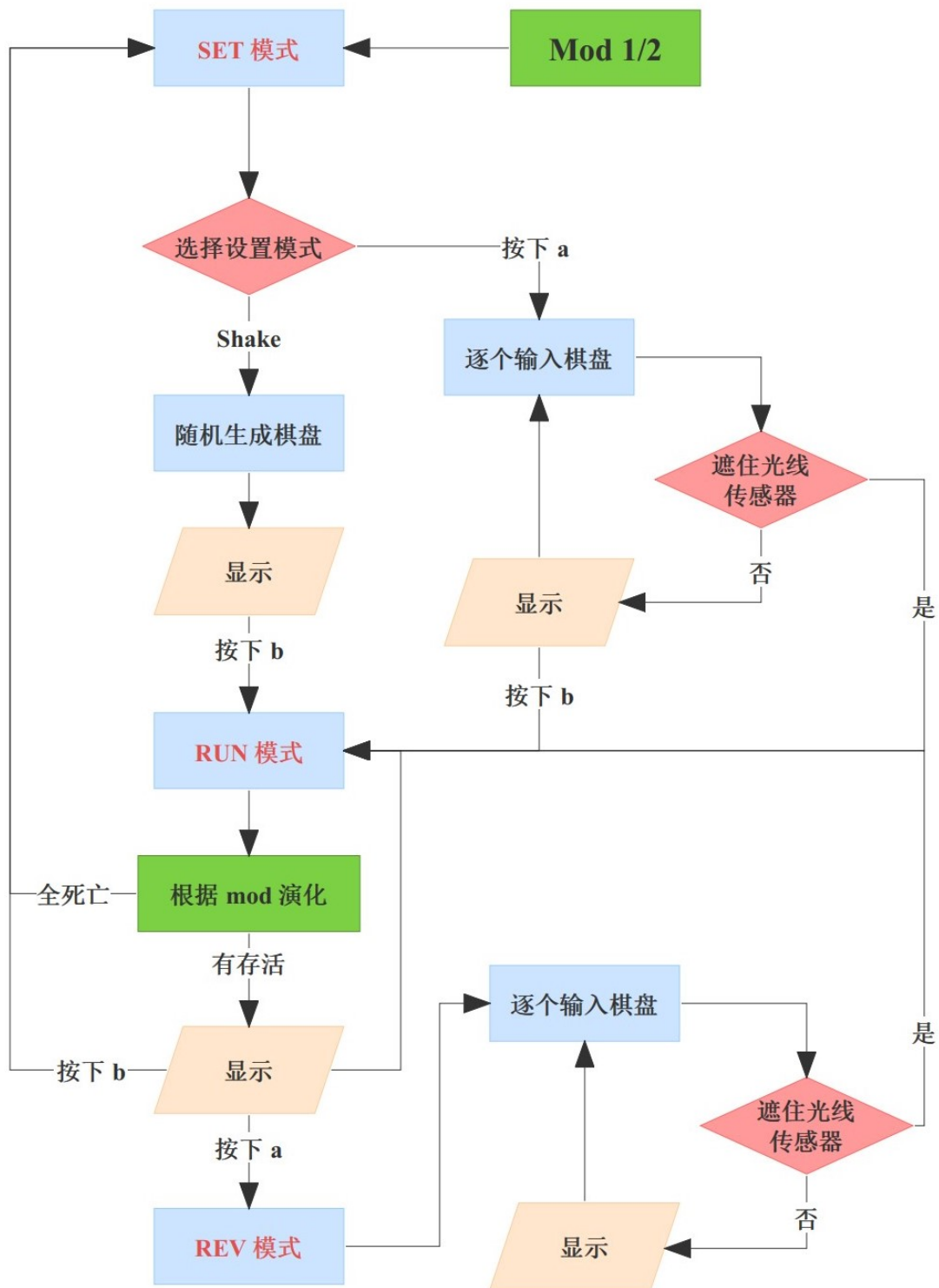


图 2: mod1 与 mod2 主流程实现

3.1.3 mod1: 简单模式

- 输入实现

mod 1
逐个输入棋盘：

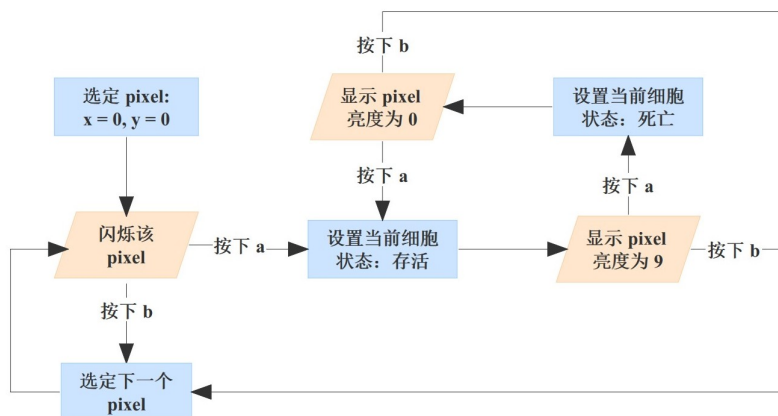


图 3: mod1 的输入实现

mod 1 细胞 演化

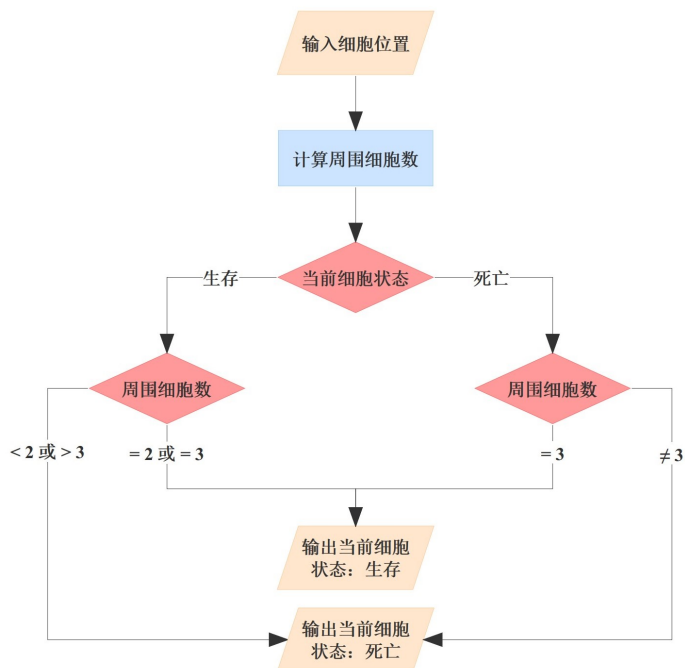


图 4: mod1 的细胞演化实现

3.1.4 mod2: 复杂模式

- 输入实现

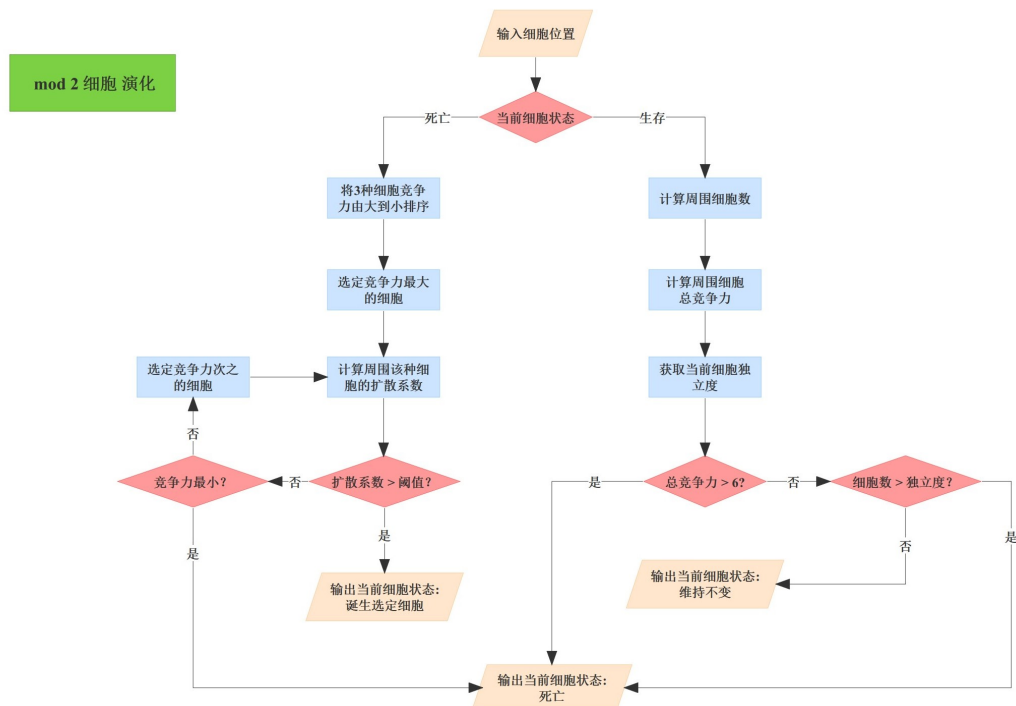


图 5: mod2 的输入实现

- 细胞演化实现

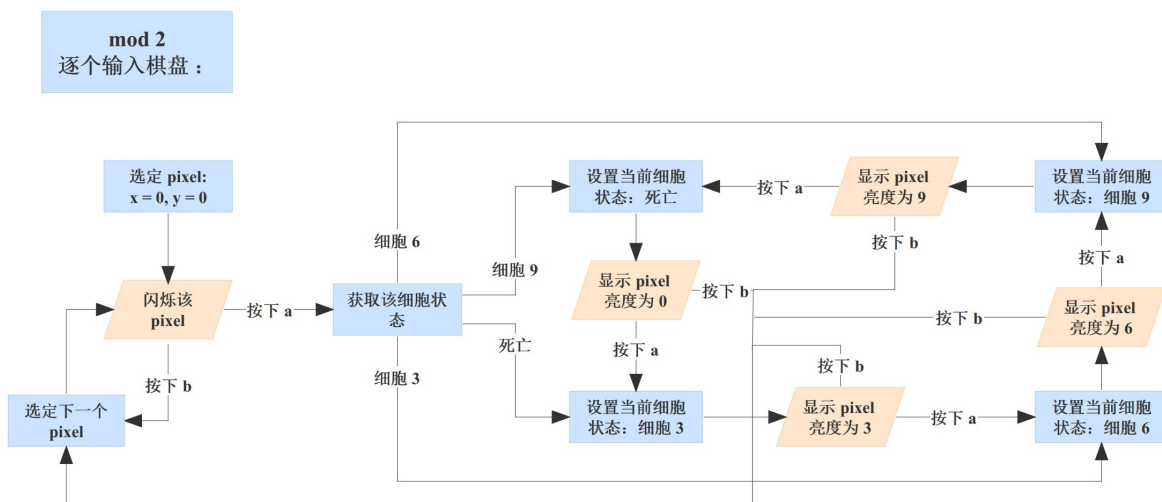


图 6: mod2 的细胞演化实现

3.2 函数定义

3.2.1 draw_universe、draw_rev_universe 与 blink_pixel

- draw_universe: 用于将棋盘内所有像素点显示

```
[1]: def draw_universe(universe):  
    for y in range(1, 6):  
        for x in range(1, 6):  
            display.set_pixel(x-1, y-1, universe[x + y * 7])
```

- draw_rev_universe: 用于棋盘内除去指定像素点, 将其余所有像素点显示, 用于 mod1 和 mod2 的 REV 模式

```
[2]: def draw_rev_universe(universe, xx, yy):  
    for y in range(1, 6):  
        for x in range(1, 6):  
            if not (x == xx and y == yy):  
                display.set_pixel(x-1, y-1, universe[x + y * 7])
```

- blink_pixel: 用于控制指定像素点闪烁一次

```
[3]: def blink_pixel(x, y):  
    display.set_pixel(x, y, 6)  
    sleep(500)  
    display.set_pixel(x, y, 0)  
    sleep(500)
```

3.2.2 evolve_1 与 get_neighbours_count

- evolve_1: 用于 mod1 中的演化
 - 使用 get_neighbours_count 计算周围细胞数
 - 若细胞状态为生存:
 - * 周围细胞数为 2 或 3, 保持生存
 - * 周围细胞数小于 2 或大于 3, 死亡
 - 若细胞状态为死亡:
 - * 周围细胞数等于 3, 诞生细胞
 - * 周围细胞数不等于 3, 保持死亡


```
[4]: def evolve_1(universe):
    next_universe = universe[:]
    for y in range(1, 6):
        for x in range(1, 6):
            count_neighbours = get_neighbours_count(universe, x, y)
            if not universe[x + 7 * y]:
                if count_neighbours == 3:
                    next_universe[x + 7 * y] = 9 # 新生命的诞生
            else:
                if count_neighbours < 2:
                    next_universe[x + 7 * y] = 0 # 生命因孤独而死亡
                elif count_neighbours > 3:
                    next_universe[x + 7 * y] = 0 # 生命因拥挤而死亡
                else:
                    pass

    return next_universe
```

- get_neighbours_count: 用于计算指定细胞周围, 状态为生存的总细胞数

```
[5]: def get_neighbours_count(universe, x, y):
    count_neighbours = 0
    for dy in [-1, 0, 1]:
        for dx in [-1, 0, 1]:
            if dx == 0 and dy == 0:
                pass
            elif universe[(x + dx) + 7 * (y + dy)] == 0:
                pass
            else:
                count_neighbours += 1
    return count_neighbours
```

3.2.3 evolve_1、get_neighbours_diff 与 get_neighbours_cc

- evolve_2: 用于 mod2 中的演化
 - 若细胞状态为死亡
 - * 使用 get_neighbours_diff 返回细胞演化后的状态
 - 若细胞状态为生存

- * 使用 `get_neighbours_cc` 返回细胞周围的总细胞数与总竞争度
 - 若总细胞数小于细胞独立度 `dependence`, 细胞因孤独而死
 - 若总竞争度大于等于 6, 细胞因拥挤而死

```
[6]: def evolve_2(universe):
    next_universe = universe[:]
    for y in range(1, 6):
        for x in range(1, 6):
            if not universe[x + 7 * y]:
                next_universe[x + 7 * y] = get_neighbours_diff(universe, x, y)
                # 新生命的诞生
            else:
                count_neighbours, compet_neighbours =
                get_neighbours_cc(universe, x, y)
                if count_neighbours < dependence[universe[x+7*y]]:
                    next_universe[x + 7 * y] = 0 # 生命因孤独而死亡
                elif compet_neighbours >= 6:
                    next_universe[x + 7 * y] = 0 # 生命因拥挤而死亡
                else:
                    pass

    return next_universe
```

- `get_neighbours_diff`: 用于根据扩散系数 `diffusion` 计算细胞扩散结果
 - 将三种细胞 (3,6,9) 按照竞争力 `competitiveness` 排序
 - 按竞争力从大到小的顺序, 依次选定该种细胞
 - 计算当前细胞周围的细胞中, 该种细胞的总个数
 - 若该种细胞总个数大于该种细胞独立度 `dependence`, 诞生新的该种细胞
 - 若小于独立度, 选定竞争力次之的细胞, 重复上述过程
 - 若始终无法大于独立度, 则保持死亡状态

```
[7]: def get_neighbours_diff(universe, x, y):
    compet_values = [value for value in competitiveness.values()]
    compet_values.sort(reverse = True)
    for value in compet_values:
        key = reverse_competitiveness[value]
        diff_count = 0
```

```

    for dy in [-1, 0, 1]:
        for dx in [-1, 0, 1]:
            if dx == 0 and dy == 0:
                pass
            elif not universe[(x + dx) + 7 * (y + dy)]:
                pass
            elif universe[(x + dx) + 7 * (y + dy)] == key:
                diff_count += 1
            else:
                pass
        if diff_count >= diffusion[key]:
            return key
    return 0

```

- `get_neighbours_cc`: 用于返回指定细胞周围, 状态为生存的总细胞数, 并根据对应细胞竞争力 `competitiveness` 的和返回总竞争度

```

[8]: def get_neighbours_cc(universe, x, y):
    count_neighbours, compet_neighbours = 0, 0
    for dy in [-1, 0, 1]:
        for dx in [-1, 0, 1]:
            if dx == 0 and dy == 0:
                pass
            elif not universe[(x + dx) + 7 * (y + dy)]:
                pass
            else:
                cur = universe[(x + dx) + 7 * (y + dy)]
                count_neighbours += cur // 3
                compet_neighbours += competitiveness[cur]
    return count_neighbours, compet_neighbours

```

3.2.4 pixel_set_1 与 pixel_set_2

- `pixel_set_1`: 用于在 `mod1` 中设置指定像细胞 (像素点) 的状态
 - 若无任何操作, 待设置状态的细胞对应像素点闪烁
 - 每按下一次 `a` 键, 更新一次状态, 使之与之前状态不同, 规则如下:
 - * 若之前状态为死亡, 更新后状态为生存
 - * 若之前状态为生存, 更新后状态为死亡

- 在中途任意时刻按下 b 键，固定当前细胞的状态，进入下一个细胞的编辑中
- 在中途任意时刻遮挡感光元件，固定之前设置的所有状态，直接跳入 RUN 模式中，这样可以节省大量设置时间

```
[9]: def pixel_set_1(x, y, light):
    while not button_a.is_pressed():
        blink_pixel(x-1, y-1)
        light = (display.read_light_level() < 30)
        if button_b.is_pressed() or light:
            break
    else:
        is_pressed_a = True
        sleep(500)
        while is_pressed_a:
            cur = current_universe[x + 7 * y]
            current_universe[x + 7 * y] = 9 if cur == 0 else 0 if cur == 9 else cur
            # 循环设置 0 与 9
            display.set_pixel(x-1, y-1, current_universe[x + 7 * y])
            sleep(500)
            while True:
                light = (display.read_light_level() < 30) # 通过光线传感器
                来判断是否需要退出
                if button_b.is_pressed() or light:
                    is_pressed_a = False
                    sleep(500)
                    break
                if button_a.is_pressed():
                    is_pressed_a = True
                    sleep(500)
                    break
    return light
```

- pixel_set_1: 用于在 mod1 中设置指定像细胞（像素点）的状态
 - 若无任何操作，待设置状态的细胞对应像素点闪烁
 - 每按下一次 a 键，更新一次状态，使之与之前状态不同，规则如下：
 - * 若之前状态为死亡，更新后状态为细胞 3
 - * 若之前状态为细胞 3，更新后状态为细胞 6

- * 若之前状态为细胞 6，更新后状态为细胞 9
- * 若之前状态为细胞 9，更新后状态为死亡
- 在中途任意时刻按下 b 键，固定当前细胞的状态，进入下一个细胞的编辑中
- 在中途任意时刻遮挡感光元件，固定之前设置的所有状态，直接结束设置，这样可以节省大量设置时间

```
[10]: def pixel_set_2(x, y, light):
    while not button_a.is_pressed():
        blink_pixel(x-1, y-1)
        light = (display.read_light_level() < 30)
        if button_b.is_pressed() or light:
            break
    else:
        is_pressed_a = True
        sleep(500)
        press_a_count = 1
        while is_pressed_a:
            press_a_count %= 4
            current_universe[x + 7 * y] = press_a_count * 3    # 通过统计 A 键
按下次数来确定当前 LED 灯的状态
            display.set_pixel(x-1, y-1, current_universe[x + 7 * y])
            sleep(500)
            while True:
                light = (display.read_light_level() < 30)
                if button_b.is_pressed() or light:
                    is_pressed_a = False
                    sleep(500)
                    break
                elif button_a.is_pressed():
                    is_pressed_a = True
                    press_a_count += 1
                    sleep(500)
                    break
    return light
```

3.3 初始化设置

3.3.1 current_universe

- 初始化 7×7 生命群落
 - 中间 5×5 的部分用于像素显示、演化计算
 - 周边其余细胞始终保持死亡状态
 - 对于特定像素 (x,y)
 - * 在计算时使用 `current_universe[x + 7 * y]` 调用
 - * 在显示时使用 `display.set_pixel(x-1, y-1, current_universe[x + 7 * y])` 显示

```
[11]: current_universe = [0] * 49
```

3.3.2 mod2 参数设置

- diffusion: 扩散系数, 用于衡量无生命细胞诞生生命的生命力
- competitiveness: 竞争力
 - 若细胞生存, 周边的总竞争度 >6 则该细胞死亡
 - 若细胞死亡, 优先诞生竞争度强的细胞
- dependence: 对于对应的细胞, 周围细胞数少于这个数值则这个细胞死亡
- reverse_competitiveness: 用于 `get_neighbours_diff` 函数的计算中, 简化计算过程

```
[12]: diffusion =      {3:2, 6:4, 9:3}
competitiveness = {3:1, 6:3, 9:2}
dependence =      {3:4, 6:0, 9:2}
reverse_competitiveness = {value:key for key,value in competitiveness.items()}
```

3.4 主程序

3.4.1 引入

- 滚动显示 “Game of Life” 游戏名, 同时播放一段魔性的音乐
- 进入简单模式与复杂模式的选择环节
- 提示: 按下 a 键, 进入 mod1; 按下 b 键, 进入 mod2

```
[ ]: display.scroll("Game of Life", wait = False) # Game of Life
music.play(music.PYTHON, wait = False)
sleep(11000)
music.stop()
sleep(1000)
```

```

display.scroll("mod")
sleep(500)
display.show("1")
sleep(1000)
display.show(Image.ARROW_W)
sleep(1000)
display.show("2")
sleep(1000)
display.show(Image.ARROW_E)
sleep(1000)

```

3.4.2 游戏模式选择

- 按下 a 键，进入 mod1
- 按下 b 键，进入 mod2

```

[ ]: while True:
    if button_a.is_pressed():
        game_mode = "mod1"
        display.show("1")
        sleep(1000)
        break
    elif button_b.is_pressed():
        game_mode = "mod2"
        display.show("2")
        sleep(1000)
        break

```

3.4.3 游戏模式介绍

- SET 模式：用于初始化设置每个细胞状态
 - 摇晃 Micro:bit，随机生成一个 5×5 的核心生命群落
 - 或按下 a 键，从左到右、从上到下逐个设置细胞状态
 - * 若在设置过程中遮挡感光元件，直接结束设置
 - 按下 b 键，进入 RUN 模式，生命群落开始演化
- RUN 模式：用于运行演化生命群落
 - 如果群落中无活的细胞，提示“DEAD”并退回 SET 模式
 - 若按下 a 键，进入 REV 模式

- 若按下 b 键，进入 SET 模式
- 若无操作，继续演化
- REV 模式：用于在运行过程中暂停修改
 - 显示除去带设置细胞外的所有细胞
 - 从左到右、从上到下逐个设置细胞状态
 - * 若在设置过程中遮挡感光元件，直接结束设置
 - 按下 b 键，进入 RUN 模式，生命群落开始演化

3.4.4 mod1: 简单模式

- 初始为 SET 模式
- 使用 pixel_set_1 设置生命状态
- 使用 evlove_1 的规则演化生命群落

```
[ ]: if game_mode == "mod1":
    mode = "SET"
    display.scroll(mode)
    while True:

        if mode == "RUN":
            if current_universe == [0] * 49:
                display.scroll("DEAD")
                mode = "SET"
                display.scroll(mode)

            current_universe = evolve_1(current_universe)
            draw_universe(current_universe)
            sleep(100)
            t_0 = running_time()
            while running_time() - t_0 < 1000:
                if button_a.is_pressed():
                    mode = "REV"
                    display.scroll(mode)
                    sleep(100)
                    break

                if button_b.is_pressed():
                    mode = "SET"
```



```

        display.scroll(mode)
        sleep(100)
        break

if mode == "SET":
    t_0 = running_time()
    while running_time() - t_0 < 10000:
        if accelerometer.current_gesture() == "shake":
            current_universe = []
            current_universe.extend([0]*7)
            for i in range(5):
                current_universe.append(0)
                for _ in range(5):
                    current_universe.append(randint(0, 1) * 9)
                current_universe.append(0)
            current_universe.extend([0]*7)
            draw_universe(current_universe)
            sleep(1000)

        if button_a.is_pressed():
            sleep(1000)
            light = False
            current_universe = [0] * 49
            for y in range(1, 6):
                for x in range(1, 6):
                    light = pixel_set_1(x, y, light)
                    sleep(100)
                    if light:
                        break
                if light:
                    break
            sleep(1000)

        if button_b.is_pressed():
            mode = "RUN"
            display.scroll(mode)

```

```

        sleep(100)
        draw_universe(current_universe)
        sleep(1000)
        break

if mode == "REV":
    light = False
    for y in range(1, 6):
        for x in range(1, 6):
            draw_rev_universe(current_universe, x, y)
            light = pixel_set_1(x, y, light)
            sleep(100)
            if light:
                break
        if light:
            break
    mode = "RUN"
    display.scroll(mode)

draw_universe(current_universe)
sleep(100)

```

3.4.5 mod2: 复杂模式

- 初始为 SET 模式
- 使用 pixel_set_2 设置生命状态
- 使用 evlove_2 的规则演化生命群落

```

[ ]: if game_mode == "mod2":
    mode = "SET"
    display.scroll(mode)
    while True:

        if mode == "RUN":
            if current_universe == [0] * 49:
                display.scroll("DEAD")
                mode = "SET"

```

```

        display.scroll(mode)

current_universe = evolve_2(current_universe)
draw_universe(current_universe)
sleep(100)
t_0 = running_time()
while running_time() - t_0 < 1000:
    if button_a.is_pressed():
        mode = "REV"
        display.scroll(mode)
        sleep(100)
        break

    if button_b.is_pressed():
        mode = "SET"
        display.scroll(mode)
        sleep(100)
        break

if mode == "SET":
    t_0 = running_time()
    while running_time() - t_0 < 10000:
        if accelerometer.current_gesture() == "shake":
            current_universe = []
            current_universe.extend([0]*7)
            for i in range(5):
                current_universe.append(0)
            for _ in range(5):
                current_universe.append(randint(0, 3) * 3)
            current_universe.append(0)
            current_universe.extend([0]*7)
            draw_universe(current_universe)
            sleep(1000)

        if button_a.is_pressed():
            sleep(1000)

```

```

        light = False
        current_universe = [0] * 49
        for y in range(1, 6):
            for x in range(1, 6):
                light = pixel_set_2(x, y, light)
                sleep(100)
                if light:
                    break
            if light:
                break
        sleep(1000)

    if button_b.is_pressed():
        mode = "RUN"
        display.scroll(mode)
        sleep(100)
        draw_universe(current_universe)
        sleep(1000)
        break

if mode == "REV":
    light = False
    for y in range(1, 6):
        for x in range(1, 6):
            draw_rev_universe(current_universe, x-1, y-1)
            light = pixel_set_2(x, y, light)
            sleep(100)
            if light:
                break
        if light:
            break
    mode = "RUN"
    display.scroll(mode)

draw_universe(current_universe)
sleep(100)

```

4 后续工作展望

4.1 简单模式的改进

- 可以预设一些类型特殊结构生命群落，分别通过一定接口调用

– 静物

* 块

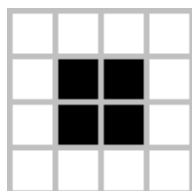


图 7: 块

* 面包

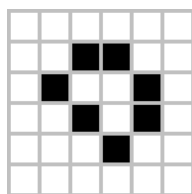


图 8: 面包

* 船

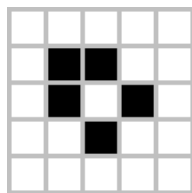


图 9: 船

* 桶

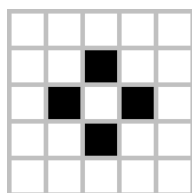


图 10: 桶

* 蜂巢

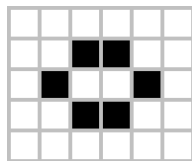


图 11: 蜂巢

— 振荡器

* 闪光灯

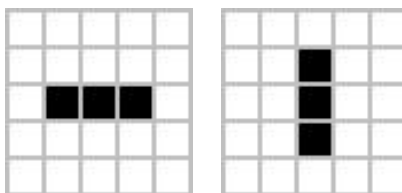


图 12: 闪光灯 (周期为 2)

* 蟾蜍

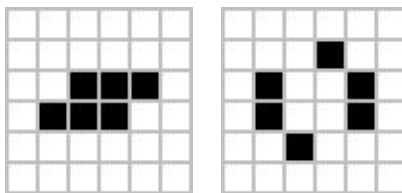


图 13: 蟾蜍 (周期为 2)

* 信标

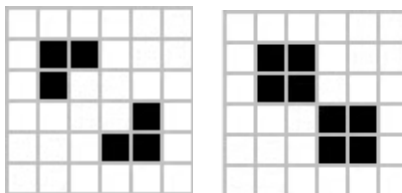


图 14: 信标 (周期为 2)

— 滑翔机

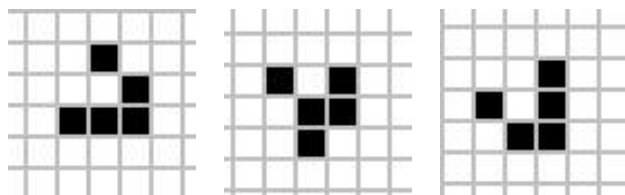


图 15: 滑翔机（每四步平移一格）

4.2 复杂模式的改进

- 可以拓展接口，使得参数可以自定义
- 可以探索更多的竞争力、独立度、扩散常数、竞争力阈值（用于判断因拥挤而死的阈值）的参数组合，使得游戏更具有可玩性
- 可以探索在特定参数下，复杂生命游戏的特殊结构，类似于简单生命游戏中的静物、振荡器、滑翔机等

4.3 单片机的改进

- 可以使用其他功能更为全面的单片机实现生命游戏，以及复杂生命游戏的参数自定义化，例如 AVR 单片机、PIC 单片机、MSP430 单片机等等

5 小组分工合作

表 2: 小组分工合作

姓名	工作
王子宸	源代码编写与调试，实习报告撰写，海报制作，视频剪辑
石城玮	游戏思路提供与参数设置，源代码调试与游戏测试，视频素材制作