Lab1：简单游戏策略

林泽春　2200012868

2023 年 9 月 26 日

目录

§1 实验目的

1. 熟悉 Linux 系统及基本操作；
2. 了解使用 Python 语言进行硬件开发的基本方法；
3. 了解简单的游戏策略及实现；
4. 使用树莓派的 GPIO 借口实现控制硬件和简单的猜拳游戏及其策略

§2 实验内容

2.1 Linux 系统的基本操作与指令

Linux 系统有如下常用命令行指令来操作文件：

更改当前目录：cd dir

创建目录：mkdir dir

删除目录：rmdir dir

列出当前目录文件：ls

将 file1 复制到 file2：cp file1 file2

删除文件：rm file

显示文本文件内容：more file 或 less file 或 cat file

同时，在 bash 中按下 <tab> 键，可以补全命令，很大地提升效率

2.2 利用 Python 语言控制硬件的方法

2.2.1 RPi.GPIO 的基本使用

树莓派 RPi.GPIO Python 模块提供了 GPIO 相关的操作包括 GPIO 接口的配置、输入及输出等。下面的语句实现模块的导入：

```
1 import RPi.GPIO as GPIO
```

目前有 BOARD 编号系统和 BCM 编号系统两种方式来对树莓派上的 IO 引脚进行编号。在使用 GPIO 时必须指定使用哪种编号方式，指定方式如下：

```
GPIO.setmode(GPIO.BOARD) # 采用BOARD编号
# 或 者
GPIO.setmode(GPIO.BCM) # 采用BCM编号
```

在使用 GPIO 输入输出前，需要对每个用于输入或输出的引脚配置通道，配置方式如下：

```
# 配 置 channel 指 定 的 通 道 为 输 入 通 道
# channel 与 使 用 的 编 号 方 式 对 应
GPIO.setup(channel , GPIO.IN, GPIO.PUD_UP)
# 配 置 channel 指 定 的 通 道 为 输 出 通 道
GPIO.setup(channel , GPIO.OUT)
# 配 置 channel 指 定 的 通 道 为 输 出 通 道， 且 规 定 通 道 初 始 输 出 高 电 平
GPIO.setup(channel , GPIO.OUT, initial=GPIO.HIGH)
```

在设置为输入状态的时候，如果外部电路没有连接上拉电阻，可以通过 GPIO.PUD_UP 参数设置 GPIO 的内部上拉电阻有效，以保证输入端在没有接入信号的时候有确定的输入值（1）。

完成通道配置后，就可以通过通道读取 GPIO 引脚的值或者设置 GPIO 引脚的输出状态：

```
GPIO.input(channel) # 读 取 GPIO 引 脚 的 值
# 引 脚 的 值: 0/GPIO.LOW/False 或 者 1/GPIO.HIGH/True.
GPIO.output(channel , state)
# 设 置 GPIO 引 脚 的 输 出 状 态 为 state
# State 的 值: 0/GPIO.LOW/False 或 者 1/GPIO.HIGH/True.
```

RPi.GPIO 模块的脉宽调制（PWM）功能基本使用方式如下：

```
# 创 建 一 个 通 道 channel 的 PWM 实 例
pwm = GPIO.PWM(channel , frequency )
# 启 动 PWM， 并 指 定 占 空 比 dc， dc 的 范 围 0.0~100.0
pwm.start(dc)
```

```
# 更改PWM脉冲重复的频率为frequency
pwm.ChangeFrequency(freq)
# 更改PWM的占空比为dc
pwm.ChangeDutyCycle(dc)
# 停止PWM
pwm.stop()
```

一般来说，程序到达最后都需要释放资源，这样可以避免偶然损坏树莓派。释放程序中使用的引脚如下：

```
GPIO.cleanup()
```

注意：GPIO.cleanup() 只会释放掉脚本中使用的 GPIO 引脚，并会清楚设置的引脚编号规则。

2.2.2 Python 的异常捕获机制

Python 的异常捕获机制可以让程序具有更好的容错性，当程序运行出现意外情况时，系统就会自动生成一个 Error 对象来通知程序，从而实现将“业务实现代码”和“错误处理代码”分离，提供更好的可读性。异常捕获机制的代码结构如下：

```
try:
  #业务实现代码
   . . .
except Error1:
  #错误处理代码
   . . .
finally:
  #不管是否发生异常，一定会执行的代码
   . . .
```

2.3 简单的游戏策略

2.3.1 随机策略

就像人类有时会通过抛硬币来决定一些事情，计算机通过产生随机数的方法来进行决策是最简单而往往也很有效的方法。比如在猜拳游戏中，如果一方采用完全随机的出拳策略，理论上胜负也同样随机，因此会有 50% 左右的胜率，不算好也不算差。

但目前大部分随机数发生器都是伪随机序列，虽然看起来分布比较均匀，却是通过公式产生，可以通过一定的算法来预测。经典的随机数产生方法是线性同余法（Linear Congruence Generator，LCG），其基本公式如下：

$$X_{n+1} = (aX_n + c) mod m \tag{1}$$

线性同余法定义了三个常数：乘量 a、增量 c 和模数 m，Xi 为生成的随机数序列。为了保证产生的随机数可以获得最大周期（不重复长度），这三个常数应该符合：

1. m 与 c 互质

2. m 的所有质因数都能整除 a 1

3. 若 m 是 4 的倍数，a 1 也是 4 的倍数

4. a、c 都是正数，还要比 m 小

对于比较简单的情况，游戏一方仅初始化一次随机数（X_0 的取值，即随机数的种子），游戏另一方就有可能猜到这个种子，并正确预测出后续产生的全部随机数，所谓的随机性也就完全丧失了。为了获得更好的随机数，可以采用更好的算法，或者采用可以产生真随机数的物理硬件来实现。

2.3.2 根据历史数据进行决策

在多次重复游戏中，游戏历史数据可以作为分析对象，简单的历史模式匹配就可以成为很好的策略：当各方面条件都相同时，采用一个过去曾经成功的动作获胜的概率会比较大。

而如果考虑到对方也会分析历史，那么曾经采用的动作有可能被针对，相对要降低选择的概率。

部分多次重复的游戏运行过程可以抽象成马尔可夫链：从一个游戏状态到另外一个状态的转换状态概率可以视为常数。这样经过几轮游戏之后，就可以积累一定的数据，并根据统计信息计算状态转移的概率。

假设模型的状态空间为 S_i,i=1, $\cdots$,m，动作空间为 A_i,i=1, $\cdots$,n。则在特定状态下采取某种动作的概率可以表示为 $P(A_i|S_j)$。对于多阶情况则可以表示为：$P(A_i)|S_{j1}, S_{j2}, \cdots, S_{jk}$。

对于猜拳游戏，根据双方的出拳状态，一共有 9 种可能，那下一次的出拳概率可以根据这 9 种情况采取不同的概率应对。而对方的出拳概率可以根据历史统计资料获得。上面只是考虑了一阶的情况，对于高阶情况则需要考虑更多的历史状态。

2.3.3 利用深度学习进行决策

前面的方法都是利用了人类的经验来设计策略，能不能通过人工学习的方式获得更好的策略呢？这也是目前人工智能研究中的重要分支，在很多领域也已经获得了很好的结果。例如著名的 Alpha-Zero 就可以完全通过学习，在围棋游戏中打败人类顶尖高手。

2.4 实验题目的实现

2.4.1 GPIO 输出

1. 实现 LED 指示灯的闪烁

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM) # 设置采用BCM编号

light  = 26
GPIO.setup(light, GPIO.OUT, initial = GPIO.LOW) # 初始化引脚

try:
    while True:
```

```
        GPIO.output(light, GPIO.HIGH) # 引脚设置为高电位，LED点亮
        time.sleep (0.2)
        GPIO.output(light, GPIO.LOW)
        time.sleep (0.2)  # 利用time.sleep()达到闪烁的效果
except KeyboardInterrupt:
    GPIO.cleanup() # 按下ctrl+C时释放引脚
finally :
    GPIO.cleanup() # 确保释放引脚
```

2. 利用 PWM 功能实现呼吸灯

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM) # 设置采用BCM编号

light  = 26
GPIO.setup(light, GPIO.OUT) # 初始化引脚
p = GPIO.PWM(light, 50) # 创建一个PWM实例
p.start (0)  # 指定初始占空比为0

try :
    while True:
        for dc in range(0, 101, 2): # 由暗变亮，通过增加占空比实现
            p.ChangeDutyCycle(dc)
            time.sleep (0.02)
        for dc in range(100, -1, -2): # 由亮变暗，通过降低占空比实现
            p.ChangeDutyCycle(dc)
            time.sleep (0.02)
except KeyboardInterrupt:
    pass
    p.stop()
    GPIO.cleanup() # 按下ctrl+C时释放引脚
finally :
    pass
    p.stop()
    GPIO.cleanup() # 确保释放引脚
```

2.4.2 按键输入

利用按键控制 LED 灯明灭，没按下一次按键就切换 LED 灯的状态

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM) # 设置采用BCM编号

key = 20
light  = 26
lighter  = True # 借助布尔值lighter判断目前LED灯的明暗变化方向
GPIO.setup(light, GPIO.OUT)
GPIO.setup(key, GPIO.IN, GPIO.PUD_UP) # 初始化引脚
p = GPIO.PWM(light, 50) # 创建PWM实例
p.start (0)

def my_callback(ch): # 定义my_callback函数，在按键时调用
    global  lighter
    if  lighter  == True:
        lighter  = False
    else:
        lighter  = True
GPIO.add_event_detect(key, GPIO.RISING, callback = my_callback, bouncetime = 200) # 设置发
        生指定事件时刻的回调函数，参数中bouncetime用于消抖

try: # 控制lED灯明灭的代码逻辑与呼吸灯的实现相同
    while True:
        if  lighter  == True:
            for  dc in  range(0,  101,  2):
                p.ChangeDutyCycle(dc)
                time.sleep (0.05)
            p.ChangeDutyCycle(0)
        elif  lighter  == False:
            for  dc in  range(100,  -1,  -2):
                p.ChangeDutyCycle(dc)
                time.sleep (0.05)
```

```
            p.ChangeDutyCycle(100)
finally :
    p.stop()
    GPIO.cleanup() # 确保释放引脚
```

2. 单击按键使 LED 灯进入闪烁模式，再次单击按键闪烁频率加倍，双击按键停止闪烁

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM) # 设置采用BCM编号

key = 20
led  = 26
freq  = 1
t  = 0
stop  = False
light  = False

GPIO.setup(led, GPIO.OUT, initial = GPIO.LOW)
GPIO.setup(key, GPIO.IN, GPIO.PUD_UP)

def my_callback(ch): # 定义my_callback函数,在按键时调用
    global t
    global stop
    global freq
    global light
    if stop == False and GPIO.input(led) == GPIO.LOW: # 判断初始状态，开始闪烁
        light  = True
    else:
        freq  = freq  / 2 # 每次单击闪烁频率加倍
    if (time.time() - t) < 0.5: # 若两次点击时间小于0.5s，则判断为双击
        stop  = True
        light  = False
    else:
        t  = time.time()
```

```
GPIO.add_event_detect(key, GPIO.RISING, callback = my_callback, bouncetime = 50) # 回调函
        数，将bouncetime设置得较短防止因消抖时间过长影响双击的判断

try:
    while True:
        if light:
            GPIO.output(led, GPIO.HIGH)
            time.sleep(freq)
            GPIO.output(led, GPIO.LOW)
            time.sleep(freq)
        elif stop:
            GPIO.output(led, GPIO.LOW)
            freq = 2
except KeyboardInterrupt:
    GPIO.cleanup() # 按下ctrl+C时释放引脚
finally:
    GPIO.cleanup() # 确保释放引脚
```

2.4.3 实现猜拳游戏

```
import RPi.GPIO as GPIO
import time
import random #导入random库，方便之后使用random.choices函数以根据历史数据指定出圈策略

GPIO.setmode(GPIO.BCM) # 设置采用BCM编号

stone = 5
scissors = 6
cloth = 13
led = 26
select_stone = 0
select_scissors = 0
select_cloth = 0
select = -1

GPIO.setup(stone, GPIO.IN, GPIO.PUD_UP)
```

```
GPIO.setup(scissors, GPIO.IN, GPIO.PUD_UP)
GPIO.setup(cloth, GPIO.IN, GPIO.PUD_UP)
GPIO.setup(led, GPIO.OUT, initial = GPIO.LOW)

def my_callback(ch):
    global select_stone
    global select_scissors
    global select_cloth
    global select
    # 根据玩家选项，记录玩家出拳的历史数据，为指定策略提供依据
    if ch == stone:
        select_stone = select_stone + 1
        select = 0
    elif ch == scissors:
        select_scissors = select_scissors + 1
        select = 1
    elif ch == cloth:
        select_cloth = select_cloth + 1
        select = 2

# 按下不同按键的回调函数
GPIO.add_event_detect(stone, GPIO.RISING, callback = my_callback, bouncetime = 200)
GPIO.add_event_detect(scissors, GPIO.RISING, callback = my_callback, bouncetime = 200)
GPIO.add_event_detect(cloth, GPIO.RISING, callback = my_callback, bouncetime = 200)

try:
    print(push a button)
    while True:
        if select != -1:
            ai_select = random.choices([0, 1, 2], [select_scissors, select_cloth, select_stone],
          k = 1) # 根据历史数据，增加相应选项的权重，判断玩家出拳习惯，增加电脑获胜几率
            if ai_select [0] == 0:
                if select == 0:
                    print(your select is stone, aiś select is stone, the result is draw!)
                elif select == 1:
                    print(your select is scissors, aiś select is stone, you lose!)
```

```
52                  elif  select  == 2:
53                      print(your select is cloth, aiś select is stone, you win!)
54              elif  ai_select [0]  == 1:
55                  if  select  == 0:
56                      print(your select is stone, aiś select is scissors, you win!)
57                  elif  select  == 1:
58                      print(your select is scissors, aiś select is scissors, the result is draw!)
59                  elif  select  == 2:
60                      print(your select is cloth, aiś select is scissors, you lose!)
61              elif  ai_select [0]  == 2:
62                  if  select  == 0:
63                      print(your select is stone, aiś select is cloth, you lose!)
64                  elif  select  == 1:
65                      print(your select is scissors, aiś select is cloth, you win!)
66                  elif  select  == 2:
67                      print(your select is cloth, aiś select is cloth, the result is draw!)
68              select  = -1
69  except KeyboardInterrupt:
70      GPIO.cleanup() # 按下ctrl+C时释放引脚
71  finally :
72      GPIO.cleanup() # 确保释放引脚
```

§3 思考题

3.1 如何产生不同分布的随机数？

可以利用 numpy 等库中的函数

3.2 如果猜拳游戏的对手是电脑，什么样的策略更好？

需要根据电脑采用的策略具体确定，例如在本报告的实现中，电脑采用根据历史数据做出决策的方法，那么如果连续保持同一个选项太多次，就会降低胜率，那么就应该尽量平均且随机地选择出拳选项。如果电脑采用深度学习等策略，根据围棋等游戏的经验我们不妨认为由于电脑的算力已经远超人脑，学习电脑深度学习得出的策略是更好的选择。