计算概论（B）大作业

小北的未名湖挑战

姓名：苏倩仪

学号：2400093012

院系：信息管理系

导师：甘锐 老师、孙海岳 助教

目录

[环境配置方案 3](#_Toc186734683)

[完成任务的思路 & 简单的代码实现方法 4](#_Toc186734684)

[game.py部分 4](#_Toc186734685)

[policy.py部分 11](#_Toc186734686)

[思考题 16](#_Toc186734687)

[附加要求（3.3） 16](#_Toc186734688)

[游戏音效（额外） 20](#_Toc186734689)

[实验数据 25](#_Toc186734690)

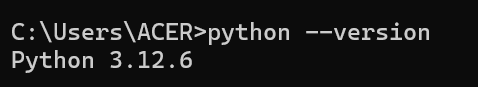
[总结 33](#_Toc186734691)

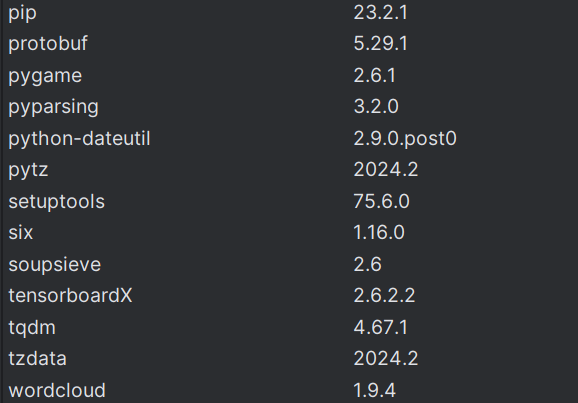
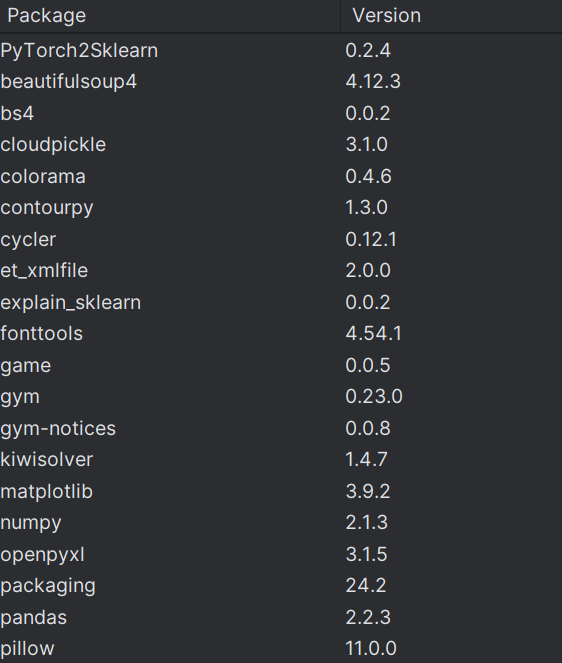
[反思与不足 33](#_Toc186734692)

[参考 34](#_Toc186734693)

# 环境配置方案

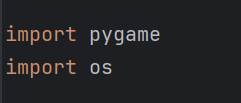
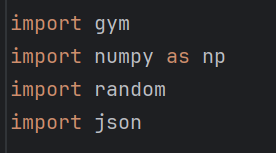
python版本和各种库的版本：

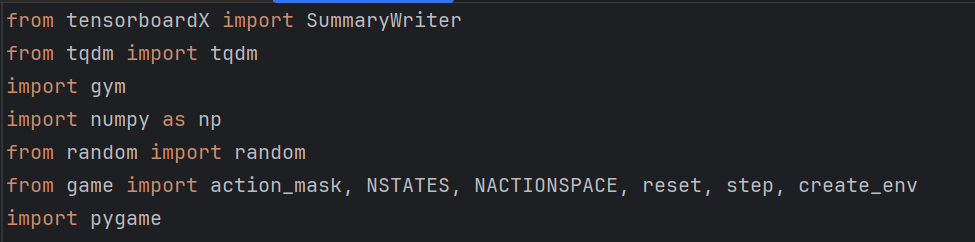




（但这其中有一些包是我自己的，不是每个包都有用到）

game.py中导入的库： 在后续有附加功能，另导入：



policy.py中导入的库：

# 完成任务的思路 & 简单的代码实现方法

## game.py部分

**先完善game.py中的代码（因为部分policy.py的代码需要game.py中的函数来实现）**

首先，先理解game.py已知的函数，并把各函数的意思记录下来以方便反复查阅。

动作集合：表示向上下左右移动；ACTIONS = [(1, 0), (-1, 0), (0, 1), (0, -1)]   
动作的空间数量：可以选择4个方向；NACTIONSPACE = 4  
地图的状态（位置）数量：由于是4\*4地图，因此有16个位置；NSTATES = 16  
地图形状：4\*4网格，也代表地图边界；MAPSHAPE = (4, 4)  
终点位置标识符：END\_STATE = 'G'  
起始位置标识符：START\_STATE = 'S'  
冰洞位置标识符：ICE\_HOLE\_STATE = 'H'  
安全位置标识符：SAFE\_STATE = 'F'

***create\_env()函数***：生成和初始化一个游戏环境并返回游戏的地图状态

代码意思：

1. 使用gym库创建一个名为FrozenLake-v1的游戏环境

2. 用env.reset()函数初始化环境，即将游戏重置为初始状态

3. 接着用env.render('ansi')以ANSI格式渲染环境，并返回当前游戏地图的字符串表示

4. 用list(filter(lambda x: x == 'F' or x == 'S' or x == 'H' or x == 'G', state)) 从字符串中提取指定的地图符号（如安全位置('F')、起点('S')、冰洞('H')和终点('G')）

5. 用state = [state[i \* 4: (i + 1) \* 4] for i in range(4)]将一维数组（地图字符串）重新分为 4x4 的二维列表表示游戏的地图结构，即游戏地图的状态

***Reset()函数***：根据给定的地图状态初始化游戏，并在同一个地图上随机生成玩家的出生点，并返回重置后的地图和出生点位置

1. 创建一个空列表safe用于储存所有安全位置的索引，然后遍历所有16个位置索引，并转化为二维坐标形式

2. if map[pos[0]][pos[1]] == 'S'表示如果当前位置为起始位置，就将其替换为安全位置 'F'，并将该索引添加到 safe 列表中，如果当前位置已为安全位置，也将其直接加入 safe 列表

3. new\_s = safe[random.randint(0, len(safe) - 1)]从 safe 列表中随机选择一个索引新的起始位置，其中，randint函数用于从给定的范围之间随即返回一个整数

4. new\_pos = state\_int2tuple(new\_s)将新的位置索引转换为 (row, col)的二维坐标形式

5. map[new\_pos[0]][new\_pos[1]] = 'S'在新位置标记为’S’（起始位置） 6. return [new\_s, map]最后返回新的地图和新的起始位置

***state\_int2tuple()函数***：将状态state从int类型转换为tuple类型，即将以

[[0, 1, 2, 3],

[4, 5, 6, 7],

[8, 9, 10, 11],

[12, 13, 14, 15]]

储存的坐标形式从整数转化为（x，y）的二维形式，例如当state=5，其二位坐标形式为（1，1）

***state\_tuple2int()函数***：与int2tuple相反，将state从tuple类型转换为int类型

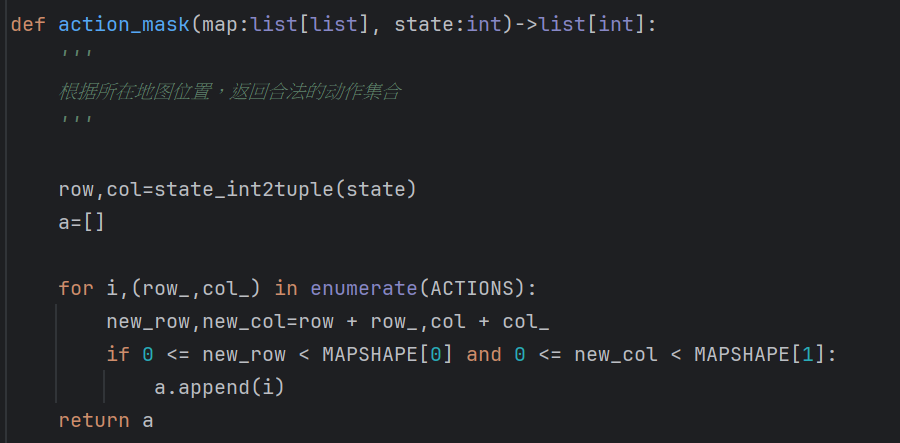
例如将（2，1）中的第0个位置（数字2）乘以4加第1个位置（数字1）等于9，即上述列表中的第9个状态

**理解了这上述已有的函数后，就可以开始补全其余的函数，思路：**

***action\_mask()函数***：将位置转为二维坐标→检查动作是否合法→返回所有合法动作

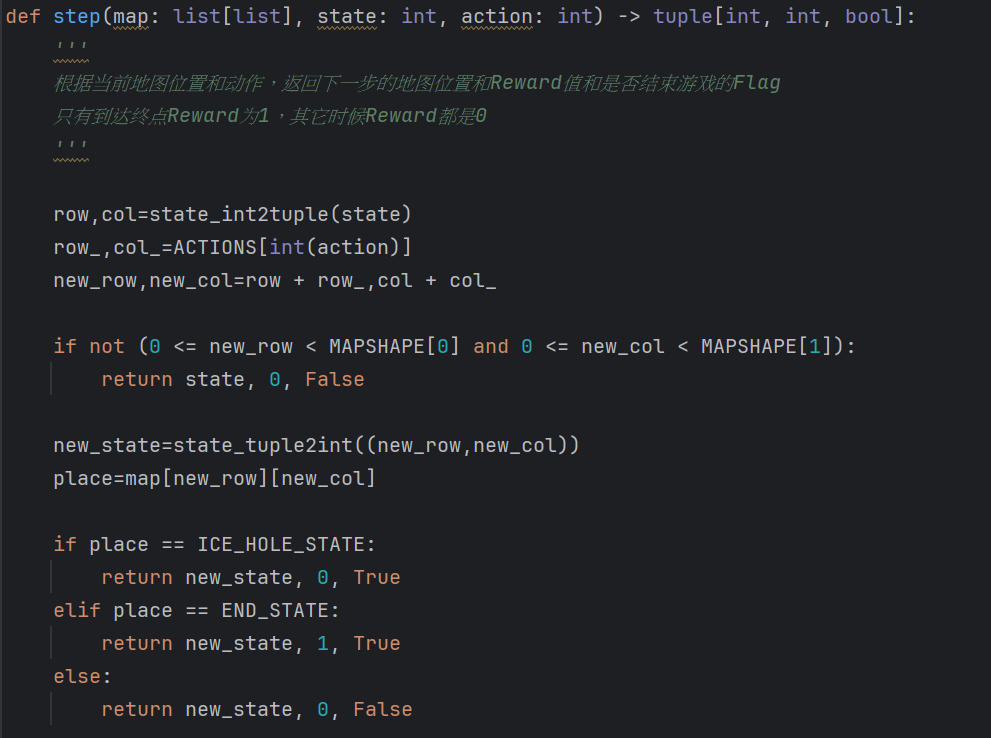
1. 首先将整数储存的坐标位置调用state\_int2tuple(state)转化为二维坐标形式
2. 然后检查输入的动作（ACTIONS = [(1, 0), (-1, 0), (0, 1), (0, -1)]）是否合法，即新位置的行和列必须在4\*4的地图（MAPSHAPE = (4, 4)）范围内，如果新位置（旧位置+输入动作对应的动作集合）和行和列没有超出地图范围，就使用列表a将合法动作储存
3. 最终输出当前状态（位置）可以执行的所有合法动作

例如，当初始位置state=0（左上角），将其转换为 (0, 0)；然后检查所有动作：动作0（左）: 越界；动作1（右）: 合法；动作2（上）: 越界；动作3（下）: 合法；最终会返回 [1, 3]。



***step()函数***（模拟游戏操作）：位置转为二维坐标→计算新位置→判断动作是否超出地图边界→用整数表达的位置形式判断玩家走到不同位置的结果（返回位置状态state，是否得到reward，是否游戏结束的flag）

1. 同样将当前位置从整数转换为二维坐标形式（用state\_int2tuple(state)函数）
2. 从ACTIONS中提取动作所对应的位移，并基于当前状态位置来计算执行动作后的新位置
3. 先考虑超出地图边界的情况（即动作不合法），用if not语句判断，如果新位置的行和列大于地图范围或小于等于0即超出地图范围；如果超出则返回保持当前位置，reward=0，Flag=False（无效动作，游戏继续）
4. 如果动作合法，则将新位置转化为整数，然后提取新位置，查看新位置是什么类型的位置标识符并判断其结果；例如当该位置为“洞”（H）则返回新位置，reward=0，Flag=True（掉入洞中，游戏结束）；或是当该位置为“终点/结束位置”（G）则返回新位置，reward=1，Flag=True（拿到奖励，游戏结束），而其他位置为安全位置（F, S），同样返回新位置，reward=0，Flag=False（游戏继续）

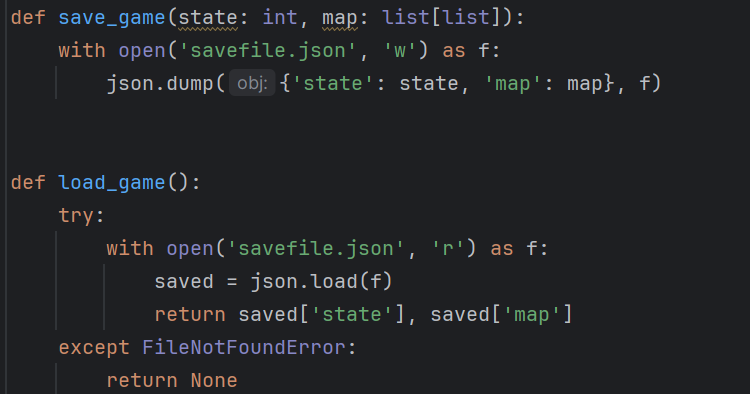


**到这里动作和返回状态都设置好了，接下来我选择按照作业要求进行游戏储存和加载的设置**

***save\_game函数()***：导入json库（用于存档和读档的设置）→创建用于存档的文件→将当前游戏进度保存为JSON格式并存入文件

1. 首先import json库，用以将游戏的当前状态 (state) 和地图 (map) 存储到一个与代码同目录的文件中
2. 用with函数创建一个”savefile.json“（文件名）及能写入（”w“：write mode写入模式）的文件，并将其赋值为f
3. 用json.dump()函数将游戏当前的状态（玩家位置）state和地图map的字典对象转换为JSON格式，并写入f文件中

通过这个函数玩家可以存档当前游戏进度，如果中途退出有存档的话可以在以后继续当前游戏进度。

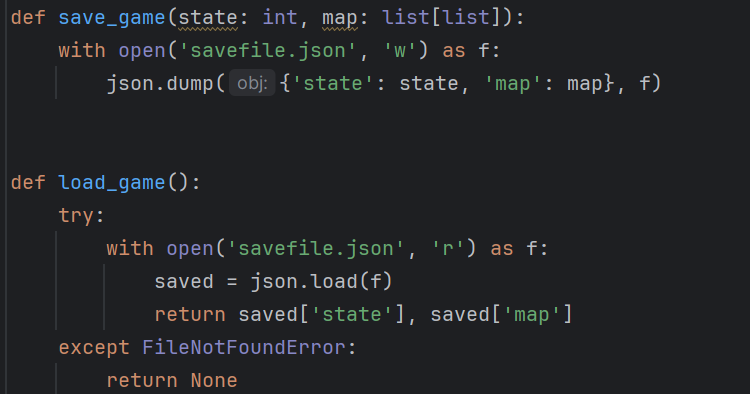


接着，再定义一个可以从存档中读取游戏进度的load\_game()函数用以恢复之前的游戏进度，

***load\_game函数()***：打开之前存档的文件→提取之前储存的游戏进度

1. 首先用try函数尝试用只读模式（“r”，read mode）打开之前存档的文件“savefile.json”，并同样赋值为f
2. 如果文件存在（没有异常），则用json.load()函数解析该文件的内容，并提取之前存储的玩家位置state和地图map状态
3. 接着用except函数处理异常情况：如果没有存档（文件不存在），则返回None，再次运行游戏，游戏会从头开始

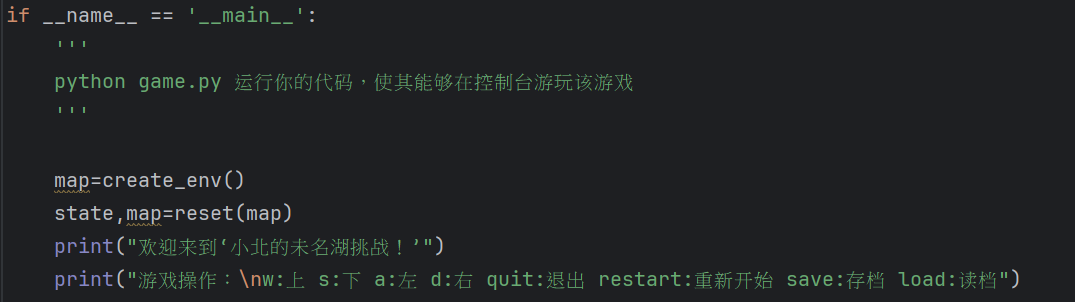
用这个函数能加载所储存的游戏存档，并从之前退出的位置继续游戏。



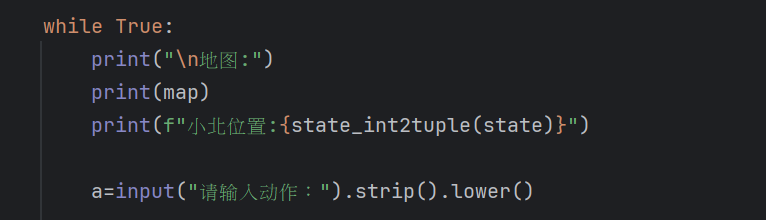
接下来，就可以在if \_\_name\_\_ == '\_\_main\_\_':写入运行代码：

生成并初始化新地图和游戏状态→循环+处理用户输入→先处理特殊情况（退出、重新开始、存档、读档）→再处理游戏操作（移动、是否拿到奖励或掉入湖中、无效操作、超出地图范围、重启等等）

1. 首先用create\_env()来创建一个新的游戏地图，并调用reset()来重置游戏，返回游戏开始时的初始状态state和地图map



1. 接下来用无限循环while来不断运行游戏，并在每次循环中打印当前的地图状态和玩家当前位置，接着用input()来获取输入如wasd分别代表上下左右移动，quit代表退出，restart代表重启游戏，save和load代表保存和加载以及其他的无效输入等等，在这里我选择用strip和lower函数来统一输入格式，应对用户多输入了空格或使用大写的情况



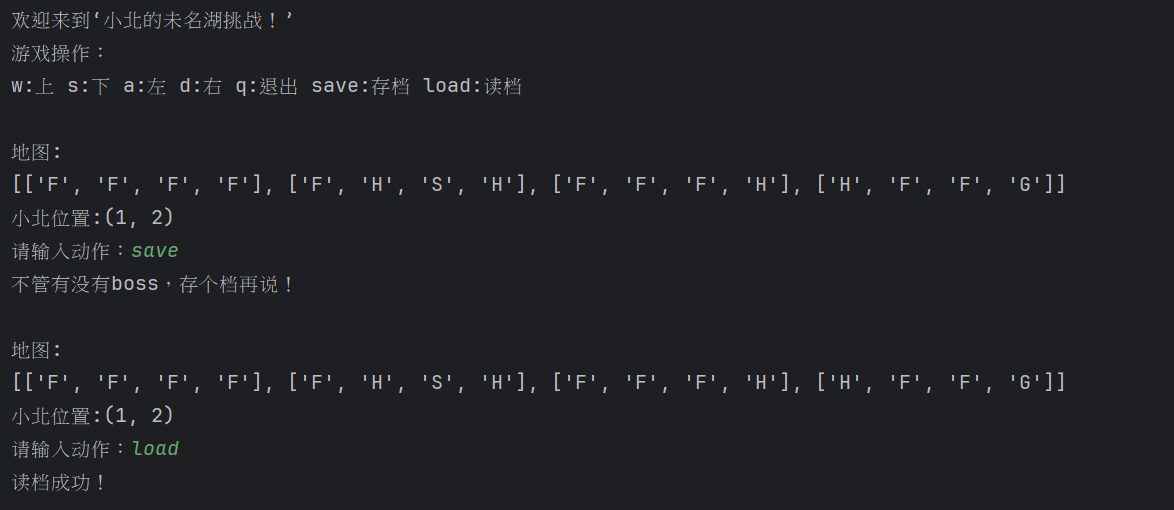
1. 先处理退出、重启和保存及加载的情况，当用户输入“quit”即退出游戏，用break推出整个循环；当输入“restart”，则重置地图的游戏状态，并返回新的游戏地图的玩家起始位置状态；当输入“save”或“load”，则调用save\_game和load\_game函数，在调用load\_game函数时也需要处理文件存在或异常的情况，如调用load\_game()后看是否能读取到文件，如果为True则读取存档中的状态state和地图map，否则打印相应提示信息，最后用continue来退出当前循环



1. 接着进行动作指令的处理，将“w”，“a”，“s”，“d”四个字母用字典映射到对应的action\_mask动作索引，如果输入的动作在该字典中，保存该动作索引并验证该动作是否在action\_mask中合法。如果合法，则用step函数计算动作执行后的新位置state、奖励reward和游戏是否结束的flag（end）。当end=True，即游戏结束，用reward是否=1来判断是获得奖励还是掉入冰洞，并打印相应信息，然后用break退出循环；当输入不合法（超出地图范围）或无效（即输入没有在选项中）则返回相应的提示信息



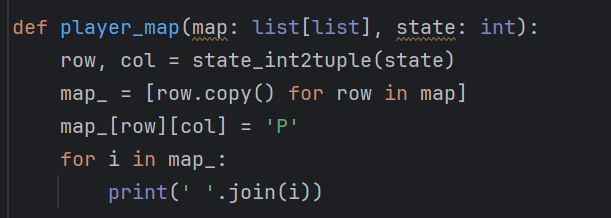
到这一步，运行代码能在控制台中看见基本的游戏功能如：以列表储存的地图及玩家的当前位置，以及能实现根据玩家输入的方向来移动、当前位置能随着动作改变、有退出、保存、加载、 重启等功能，且输入后如果游戏未结束会出现下一行让玩家继续输入，直到满足结束条件。这时的游戏在控制台显示为：

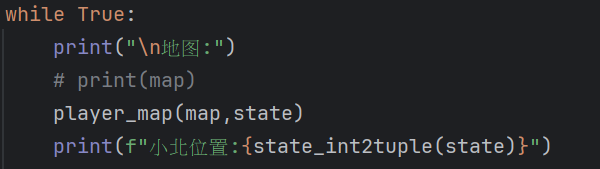


但以列表储存的地图不够直观，且也看不到玩家位置，容易造成混淆，因此我想定义一个函数用于实现在控制台的游戏中加入显示玩家当前在地图上的位置的标记“P”，并将地图列表转化为4\*4的格式：

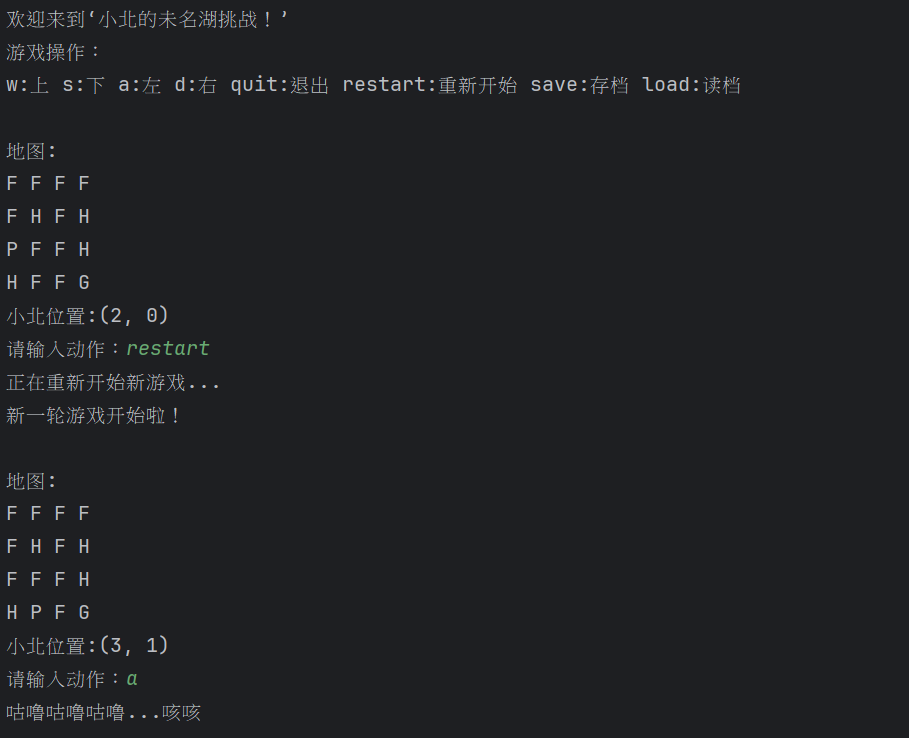
***player\_map()函数***：位置转为二维坐标→复制地图副本→将副本中的玩家位置做另外标记（“P”）

1. 首先将玩家位置转换为二位坐标形式
2. 然后用copy函数按行复制一个地图的副本以确保原始地图不会被修改
3. 再将地图中玩家当前位置标记为“P”，并用join函数打印出地图的每一行



接着在if \_\_name\_\_ == '\_\_main\_\_':中将原本地图替换为用player\_map()函数定义的地图：  


这样，代码运行后能获得一个较美观的4\*4地图显示，而且在每一次移动中，会输出一个新地图来显示每次玩家位置“P”的移动：



## policy.py部分

**接下来，对policy.py进行完善，在开始之前，同样先简单理解已有的代码函数：**

***test\_pi()函数***：用来测试策略的表现（成功率=成功次数 / 总回合数）

1. 首先初始化count，用于记录成功到达终点的次数

2. 然后循环给定的回合次数，在每次的循环中调用reset函数重置环境并返回初始的状态ob和地图map

3. 接着根据当前状态，在已有策略pi中选择动作a，然后调用step函数，返回新的状态 ob、奖励 rew 和回合是否结束的flag（done）

4. 如果回合结束done==True，且奖励rew==1，代表成功到达终点，增加count+=1，并退出循环，否则不计数，继续下一个回合

5. 最后返回成功率（成功次数 / 总回合数）

***Policy\_iteration\_MC函数()***：基于蒙特卡洛方法的迭代，通过不断优化策略来找到最优策略，并使用可视化工具将其记录

1. 首先设置pi为一个全零的向量（np.zeros函数），表示所有状态下都选择动作 0，随着迭代会更新成更优的策略（pi=new\_pi）；Iteration则用于记录迭代次数，每次+1

2. 通过调整epsilon来控制策略评估时的探索程度；如果diff\_p为真，调用函数 decay\_f 调整ϵ，否则使用公式来调整；随着迭代次数增加，ϵ会逐渐减小，从而减少随机探索，变得更多利用当前策略

3. 然后调用 compute\_qpi\_MC()函数（仍需完善），并使用蒙特卡洛算法计算其当前策略的动作价值函数Q，根据估计，通过argmax（一种np 中用于返回最大值所在位置索引的方法，axis=1 表示按行查找最大值）返回能够使动作价值函数最大的动作，并更新策略

4. 接着检查策略是否发生变化，如果旧策略与新策略相同，则返回最终策略

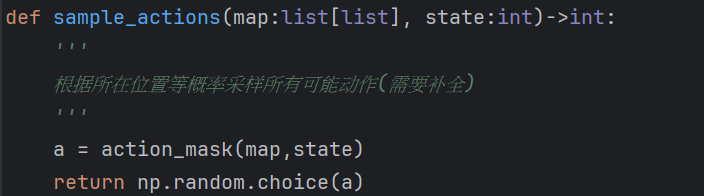
5. 调用test\_pi函数来测试该策略的表现，并记录到writer（用于记录和可视化训练过程的工具，在代码中用TensorBoardX中的Summary Writer来记录并储存策略的测试结果）中，其中writer.add\_scalar用于记录标量数据，第一个参数 '/MC3/{name}'用于指定记录的分类路径、result用于记录的数值、global\_step=iteration则指定当前的迭代次数，用于在图表中标记横坐标，可以将 result 和迭代次数的关系存储起来，并使用工具（如 TensorBoard）进行可视化

6. 之后将每次迭代后的新策略调用test\_pi进行测试，并将结果同样记录到writer中

简单理解了上述函数后，根据作业原要求，完善代码：

**1. sample\_actions5函数以地图 map, 当前状态 state 作为输入, 根据等概率采样, 返回一个合法的动作。**

***补全sample\_action()函数***：已知这段代码需要从当前地图map和状态state中根据等概率采样随机选择一个合法的动作并返回；因此可以调用前面game.py中的action\_mask函数，筛选出当前位置的所有合法动作后，从中通过np.random.choice来随机返回一个合法动作，补全后的代码如下图所示：



**2. compute\_qpi\_MC6的 greedy 部分, 需要你实现 *ϵ − greedy* 采样算法. 该算法的想法是, 我以 1 *− ϵ*的概率选择当前的最优动作 (即你的策略), 同时以 *ϵ* 的概率在所有合法动作中随机选择. 该策略作为一种简单但是广泛应用的方法, 十分优秀的平衡了探索与利用的过程。**

首先，先了解***compute\_qpi\_MC()函数***中已有的代码部分：

1. 用numpy中的zeros函数来初始化矩阵，表达为np.zeros(shape, dtype=float)，shape是要生成的数组或矩阵的形状，在这段代码中，shape是一个元组 (rows, cols)来表示二维数组，而dtype指定了数组中元素的类型，通常默认为浮点数类型

* Q = np.zeros((NSTATES, NACTIONSPACE), dtype=np.float32) 是一个二维数组（矩阵），其中NSTATES表示环境中状态（位置）的总数，NACTIONSPACE表示每个状态的合法动作数量，dtype=np.float32则表示Q 矩阵中的每个值是32位浮点数，用来存储每个状态下不同动作的价值
* N = np.zeros((NSTATES, NACTIONSPACE), dtype=np.int64)而N和Q矩阵一样，也是一个二维数组（矩阵），但dtype=np.int64表示了N矩阵中的每个值是64位整数，用来统计状态和动作对被访问的动作次数

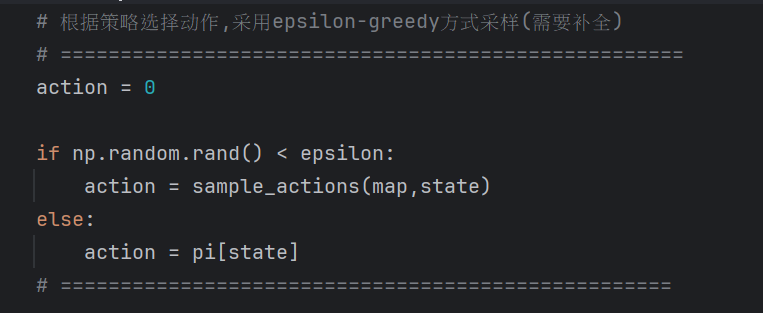
例如，Q[state][action]表示从某个状态state执行某个动作action时的价值，而N[state][action]则是从状态state执行动作action被访问的次数

1. 接着，进行num\_episodes次循环（代码中是1000次），用来模拟多次回合的蒙特卡洛算法
2. 用reset来重置环境，即重新开始一个新回合，并初始化状态（位置）和地图
3. 用episode列表来储存每一轮回合中的state，action，reward

接着***补全epsilon-greedy策略***：以 1 *− ϵ*的概率选择当前的最优动作, 同时以 *ϵ* 的概率在所有合法动作中随机选择

1. 首先用np.random.rand()生成一个介于 [0, 1) 区间内的随机数，当epsilon大于这个随机数，调用sample\_action()函数随机选择动作（探索阶段）
2. 反之就从已有的pi list当前策略中选择最优动作pi[state]（利用阶段）

这样设计是为了平衡探索和利用当前的策略，探索阶段（随机数<epsilon）是为了使用随机动作来更全面地探索所有可能的状态和动作，从而发现新的、更优的策略，帮助算法跳出局部最优解；而利用阶段（随机数>=epsilon）则是利用当下已学到的最优策略，减少随机选择的浪费，当*ϵ*减小时，算法会逐渐偏向于选取当前策略下的最优动作，补全代码如下图所示：

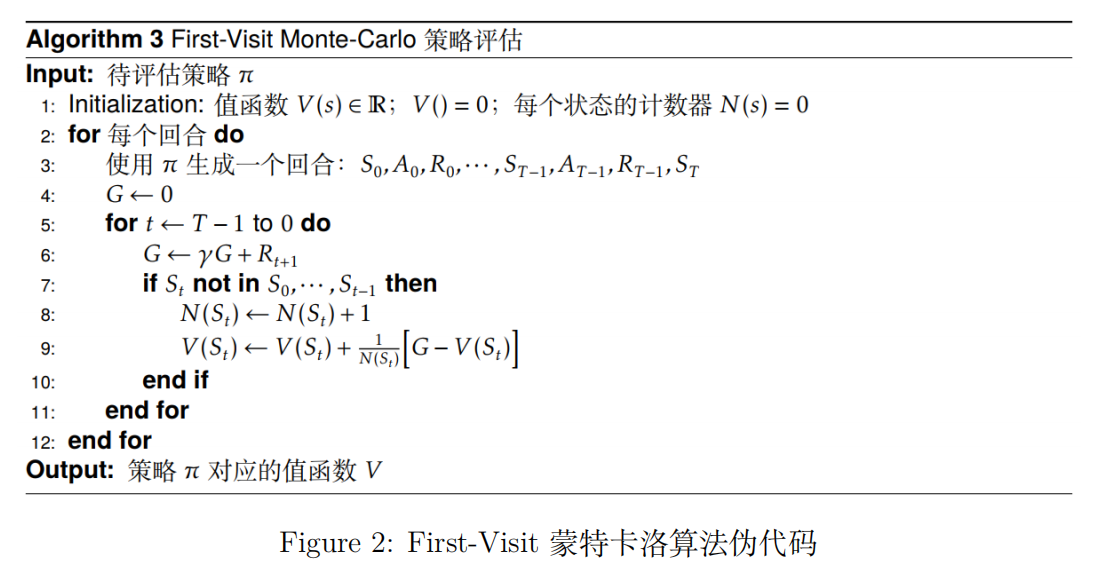


（同样是***compute\_qpi\_MC()函数***中接下来已有的代码部分）

1. 然后调用step函数，获得新的状态state，是否获得奖励reward和游戏回合是否结束的flag（done），并使用episode列表来储存每个回合的状态state、动作action和奖励reward
2. 当done=True（即游戏结束）时，也用episode列表来记录终点的状态、动作、奖励，其中由于游戏已结束，所以用None来标识终止，即没有动作可以选择，然后通过break终止循环，回合终止
3. 最后如果游戏未结束就将state更新为next\_state来处理下一个状态，并储存进episode列表中
4. 首先，visited是一个由于记录状态和动作对是否已经被访问过的集合，以确保不发生重复；并初始化累计回报，G=0
5. 接着从回合结束开始往前遍历episode中的状态、动作和奖励

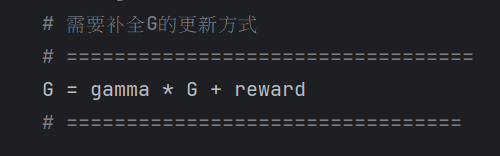
接下来到需要补全的另一段代码，要求：

**3. compute\_qpi\_MC7部分需要你按照提到的伪代码实现具体的 Reward 的更新, 动作价值函数 (Q 函数) 的更新，其中伪代码如下图：**



***对G（累计回报）的更新：***

1. 根据伪代码，可以看到G可以表达为折扣因子 γ（代码中折扣因子gamma=0.95）\*G累计回报+Reward奖励（从后往前遍历），并从终点状态往回计算每个状态的累计回报
2. 在代码中可以表示为G=gamma\*G+reward，其中gamma\*G是前一状态的回报的折扣，reward 是当前的回报

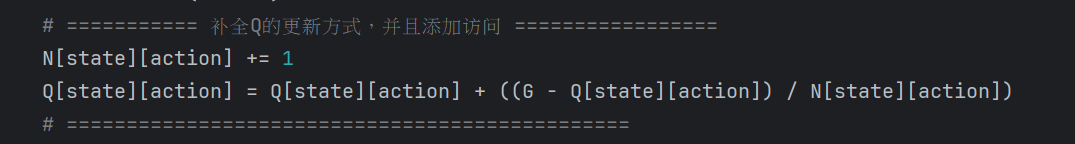


（同样是***compute\_qpi\_MC()函数***中接下来已有的代码部分）

1. 接下来，如果当前回合结束，则不再处理动作（if action is None:）
2. 接着用sa表示（state，action）的元组，如果该元组没有被访问过，就将其添加到visited集合中，避免重复

接下来***对N[state][action]和Q[state][action]（Q函数）进行补全***（用于更新当前状态下的动作价值），在补全算法前需将状态state和动作action转换为整数形式用以计算：

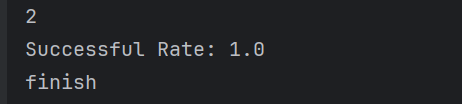
1. 从以上伪代码中可以看到N（St）= N（St）+1，即每一次访问时N（St）都会增加一次计数，因此在代码中可以表达为N[state][action]+=1
2. 而伪代码中也标识出了V（St）=V（St）+1/N（St）[G-V（St）]，这是蒙特卡洛中更新动作价值函数的算法，相当于把过去的平均值和这次的结果结合，更新为一个新的平均值，在我们的代码中可以表示为Q[state][action] = Q[state][action] + ((G - Q[state][action]) / N[state][action])
3. 这个算法在循环中会重复多次，直到价值函数足够准确，最后返回计算后的动作价值函数Q



最后在if \_\_name\_\_ == '\_\_main\_\_':中：（不需补全但可以修改用以调试，所以在这里只理解）

1. 首先调用了create\_env()函数来初始化地图
2. 然后通过SummaryWriter为模型训练添加TensorBoard日志来训练可视化，并将训练过程中产生的数据存储到 ./result 文件夹中
3. 接着调用pi = policy\_iteration\_MC(map, 0.95, decay=0.1, writer=writer, name=f'basic')执行蒙特卡洛算法来得出最优策略，在这里设置了不同参数如折扣因子gamma=0.95、衰减速率decay=0.1、writer 用于记录训练日志、name='basic' 为日志标记
4. 最后调用test\_pi函数来测试最终策略的成功率，并打印finish表示程序结束

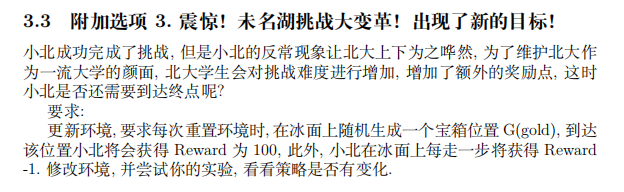
最后运行这段程序时，会看到一个数字和一个successful rate，分别代表收敛后的策略数量以及策略的成功率。



# 思考题

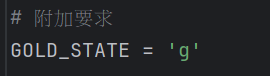
## 附加要求（3.3）

完成了基本要求后，我选择做附加要求3.3：



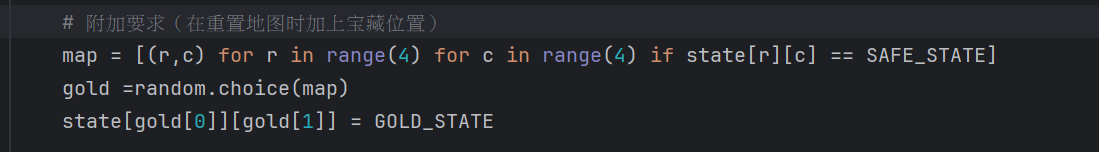
**同样从game.py开始：**

先在代码一开始加上GOLD\_STATE=’g’的标签，以方便在后续调用。

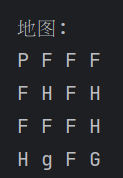


然后，需要在***create\_env()函数***中，在重置地图时随机选择一个安全位置并加入宝藏位置：

1. 先生成一个元组用于提取所有安全位置“F”
2. 从上述的map中随机选择一个位置坐标代表随机选定的宝藏位置
3. 然后将该位置坐标设置为GOLD\_STATE

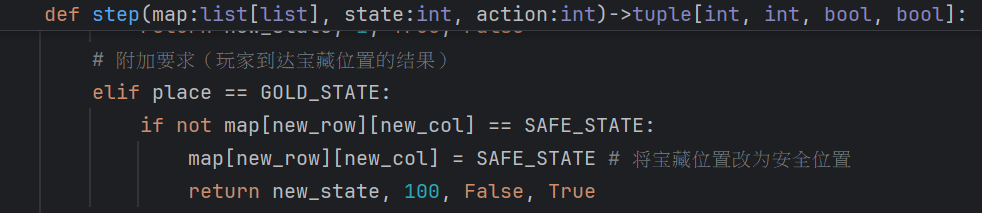


此时，地图将会是：



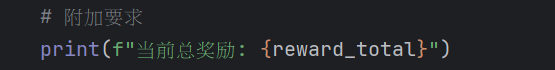
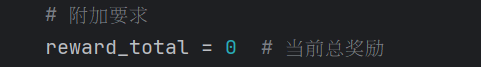
接着在***step()函数***中加上当当前位置==GOLD\_STATE时，返回该位置的结果：state=玩家处于宝藏位置，reward=100，结束游戏的flag=False（获得宝藏不代表结束游戏）

1. 且因为我想在玩家到达该GOLD\_STATE位置后不能再次访问该位置获得reward+100的奖励，所以当玩家抵达该位置时，这个位置会变成“F“安全位置，再次抵达不能再获得reward奖励
2. 然后加上第四个布尔值表示是否找到宝藏的flag（只有当当前位置==GOLD\_STATE才==True，其他都为==False），以便在后续能对用户移动到该位置时做出reward+100的反应；代码如下：



（但是后来我用这个运行policy.py时好像出现死循环了，而且有一些游戏操作也不正确T\_T经过一系列（不知道对不对的）调整后，我发现在if \_\_name\_\_ == '\_\_main\_\_':另外判断是否到达宝藏位置可以让policy.py正常运行（指不陷入死循环），所以这步只是用于判断是否到达宝藏位置和将已经过的宝藏位置改为普通的安全位置）

最后在if \_\_name\_\_ == '\_\_main\_\_':中加上，



1. 用于储存并在每次循环中打印当前总奖励

**接下来处理用户的输入与其对应的奖励的状态：**

1. 首先当输入为“restart”，当前总奖励需要一并重置为0：

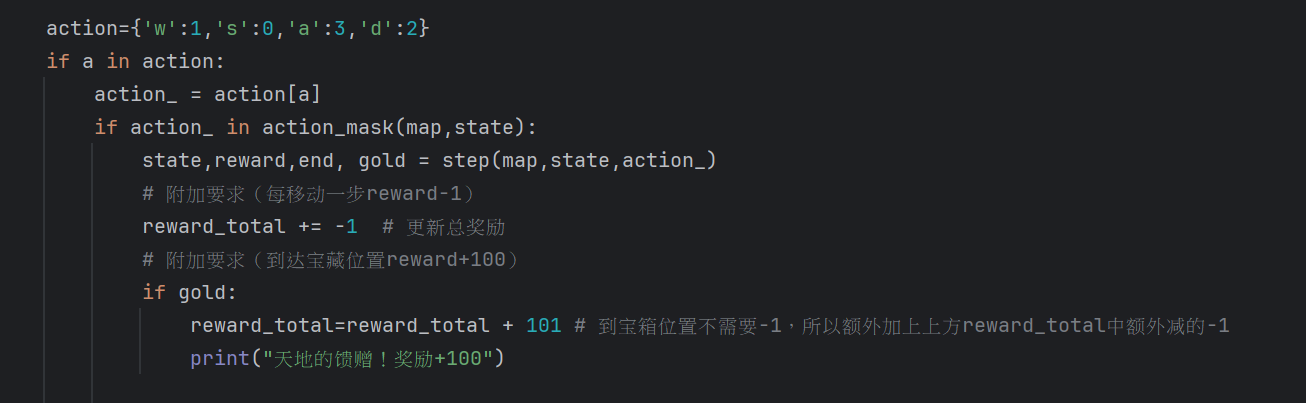


1. 当输入为”save”或”load”时，除了保存和读取玩家位置和地图和进度，需要再加上当前总奖励的进度，因此在save\_game()和load\_game函数中也需要再加上reward的参数：



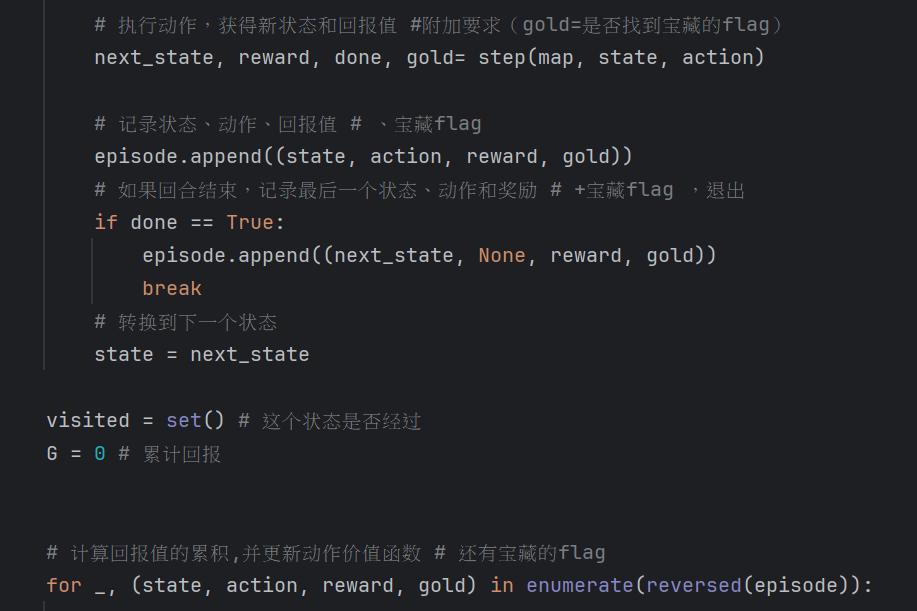


1. 接下来，对用户的每一次移动进行处理，当用户每移动一步，reward-1，但当到达宝藏位置则+100。但由于用户的每一次移动会-1，导致用户抵达宝藏时，reward+100前会先-1，变成只+99，而我想要用户到宝箱位置不需要-1，所以额外补上移动中所减少的1，因此是reward+101：

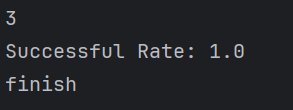


**到这里附加要求在game.py中已完成，接下来处理policy.py中对应的改动：**

***compute\_qpi\_MC()函数***：因为game.py中的step()函数额外加了一个布尔值，所以在policy.py中与step()函数有关的代码都需额外加上一个是否找到宝藏的flag：



完成后，运行policy.py中的代码后会看到：

（几次运行都是差不多的结果）

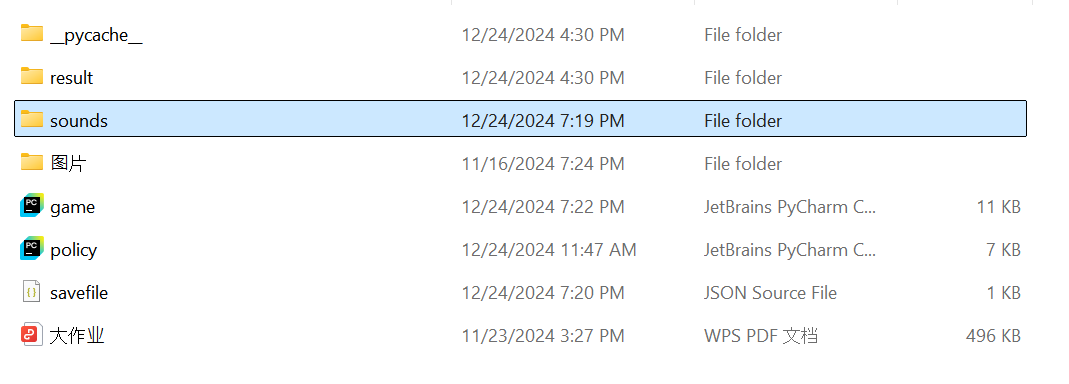
发现与原来的结果并没有太大区别，代表策略没有太多变化。

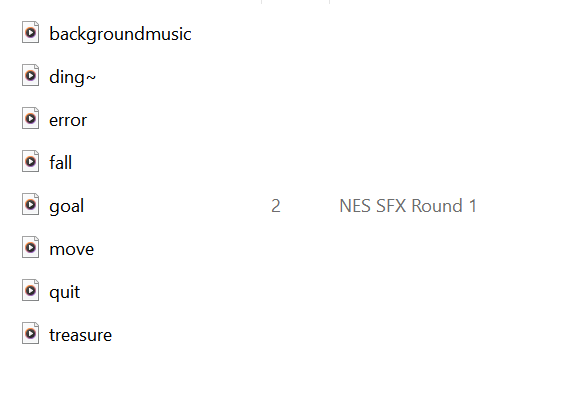
## 游戏音效（额外）

某天我在浏览CSDN时偶然发现原来python还能加入音效！所以我打算在冰湖游戏中加入这个功能来增加玩家的沉浸式体验~。

**实现：pygame.mixer模块**

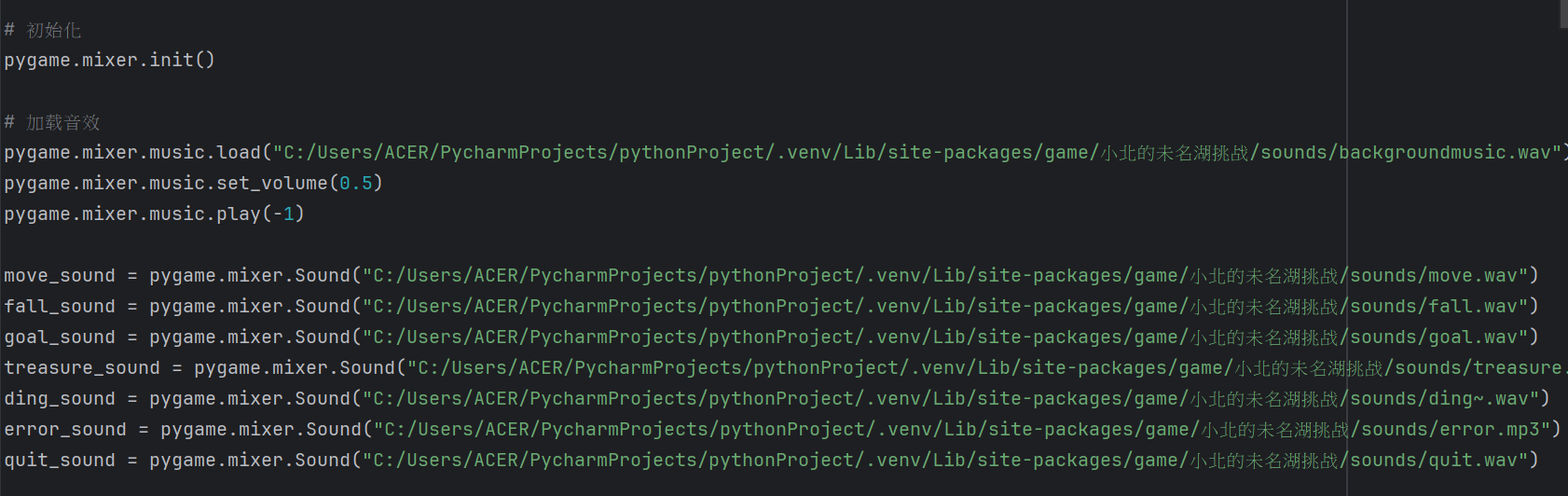
首先需要导入pygame库，然后将想要作为游戏音效的音频放到和代码同样的目录下，这里我创建了一个文件夹sounds来储存所有的音效（所有音效皆从Freesound网站下载）





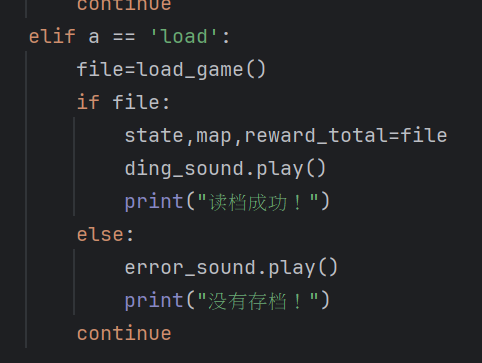
下载完后，对音效进行初始化，并设置和播放背景音乐（backgroundmusic.wav）：

1. 先用pygame.mixer.init()初始化游戏音效
2. pygame.mixer.music.load()加载音效
3. pygame.mixer.music.set\_volume()设置音量
4. pygame.mixer.music.play（-1：用于循环播放）循环播放音乐
5. 接着给每种音效用pygame.mixer.Sound进行赋值，以方便后续调用



接着就可以对玩家动作设置对应的音效：

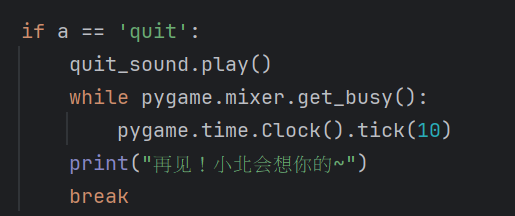
（如输入quit播放quit音效、输入restart、save、load播放ding~音效，其中当输入load时如果找不到存档则播放error音效）

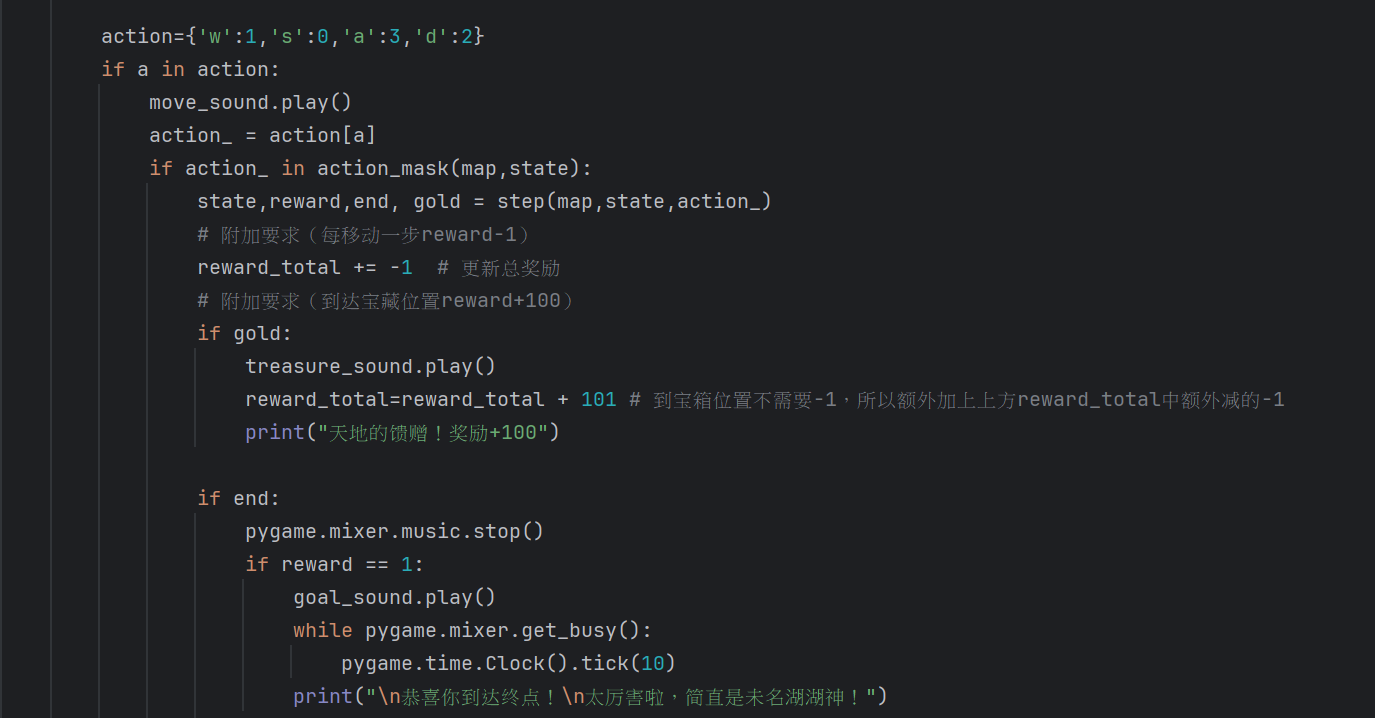


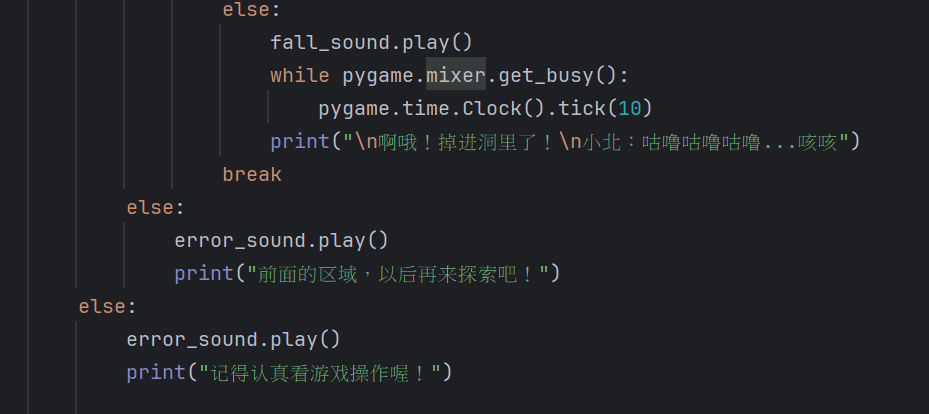
（每次移动播放一次move音效，找到宝藏播放treasure音效，当游戏结束先用pygame.mixer.music.stop()来停止背景音乐，然后在游戏结束的情况下当到达终点播放goal音效，掉入冰洞播放fall音效；而超出地图范围的动作和非法操作则播放error音效）

但到这一步我发现当输入的动作对应着游戏结束（quit、掉入冰洞、到达终点）时，会因为程序直接结束导致相应的音效无法被触发。

1. 因此需要加上pygame.mixer.get\_busy()来检查当前是否有任何音效正在播放（音效还在播放返回True，否则返回False）
2. 接着用pygame.time.Clock().tick(10) 限制循环频率，即每秒检查10次是否播放完成，接下来的代码改为：





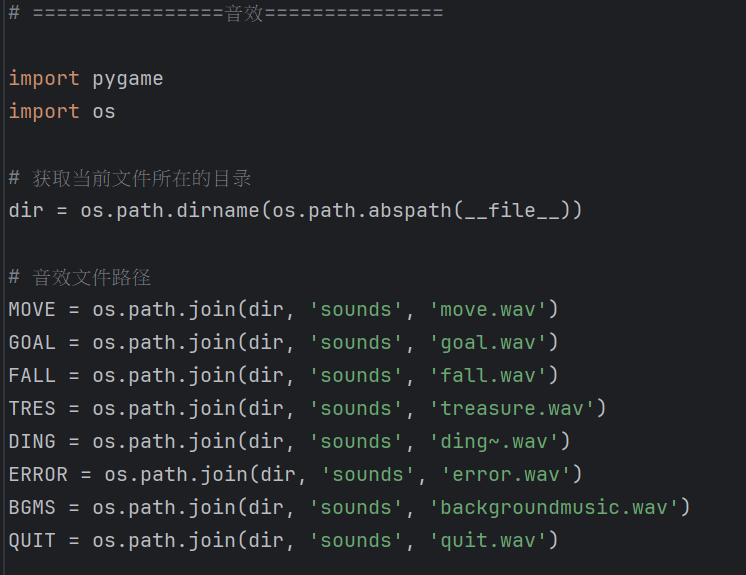


这样当换成不同设备就也可以正常播放音效啦！（本人发给了几位朋友来帮忙运行这段代码，都是可以正常播放）

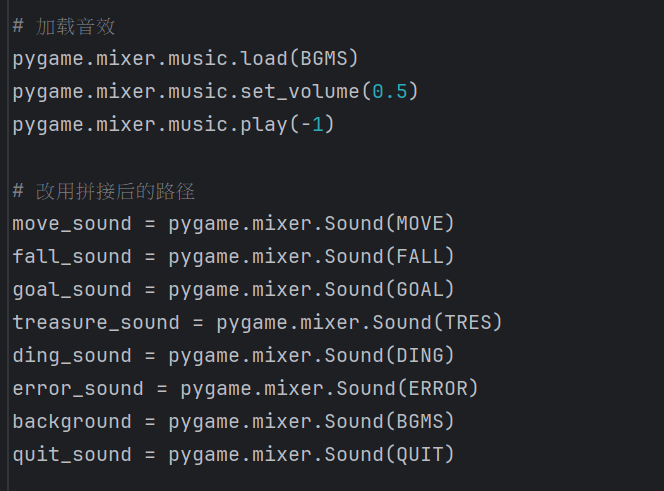
到这里所有游戏音效就可以正常播放了，但是接下来我又发现代码中使用的都是我自己的目录下的绝对路径，当用不同设备来运行时可能因为找不到路径而出错，因此相对路径再进行拼接成绝对路径。

**实现：os模块**

1. 首先导入os库用于获取文件路径
2. 用os.path.abspath(\_\_file\_\_)将文件转换为绝对路径
3. 然后用os.path.dirname()来获取文件路径的“父目录”，这样最后dir变量就保存了当前脚本所在的目录路径
4. 接下来使用os.path.join拼接路径，找到目录下的各个音频并将其赋值，代码如下：



然后将上方pygame.mixer中原本用到绝对路径的参数改为设置好的参数：



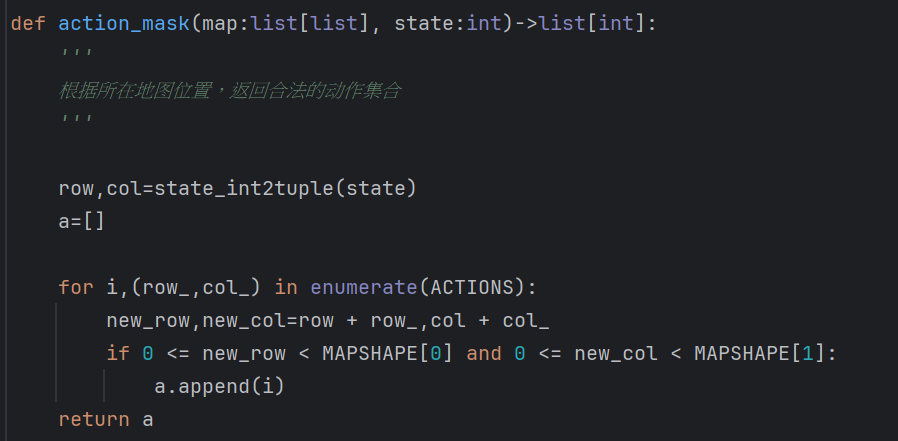
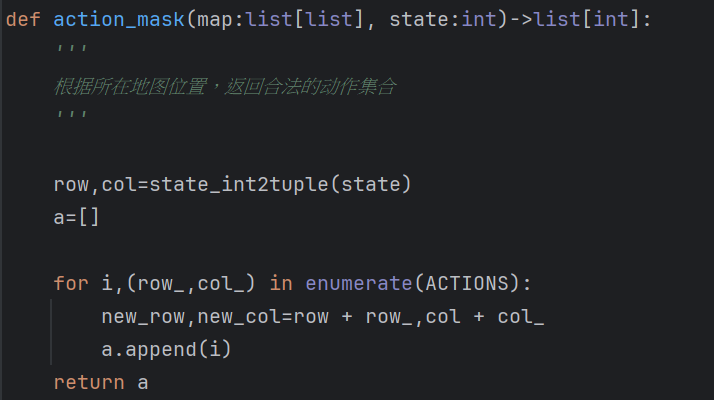
这样当在不同设备运行代码时，也会有游戏音效了。

之后运行policy.py时，由于会调用前面game.py中的函数使游戏的背景音乐也会播放（但不影响程序运行），所以我在policy.py中的if \_\_name\_\_ == '\_\_main\_\_':用pygame.mixer.music.stop()来停止音乐。

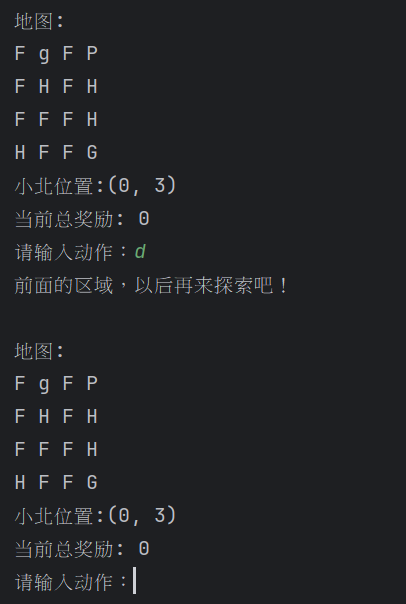
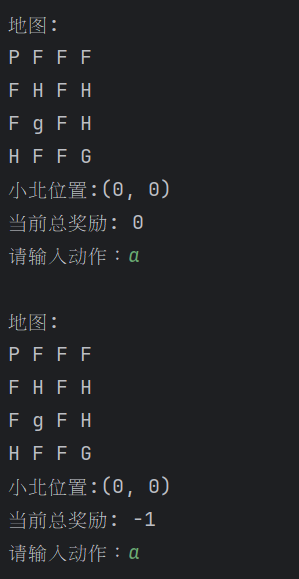


# 实验数据

*action\_mask()函数：*

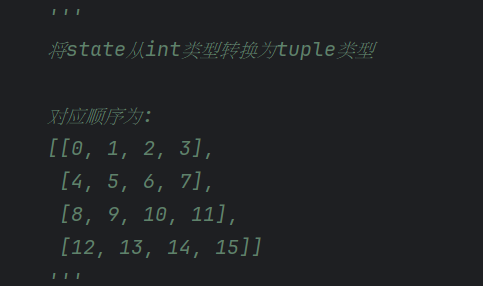
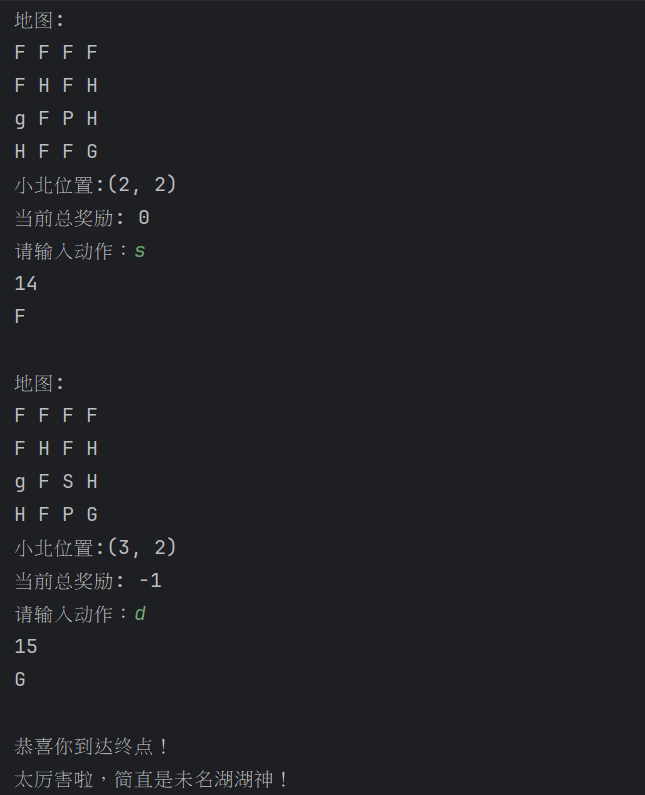


在这个函数中，一开始由于没有考虑到超出地图范围的情况，导致当玩家输入超出地图范围时没有打印相关提示信息，后来加入“当前总奖励”时更是在超出地图范围时也会-1，因此再后来进行了调整，代码由上方左图变为右图，对应的输出也从下方作图更新为右图。



*step()函数：*

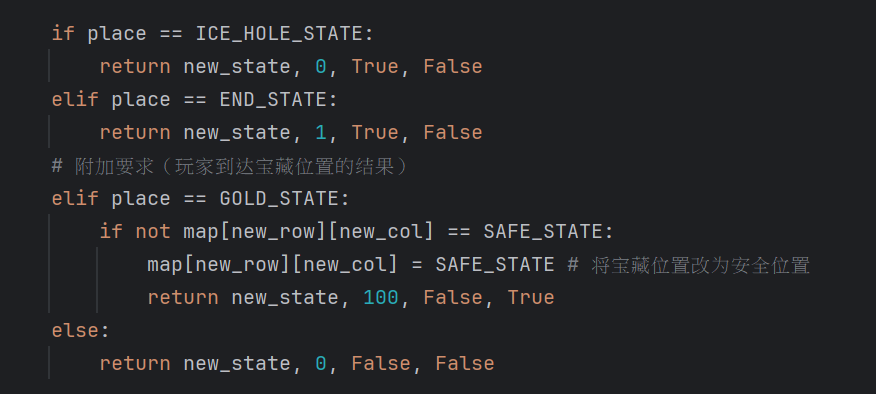
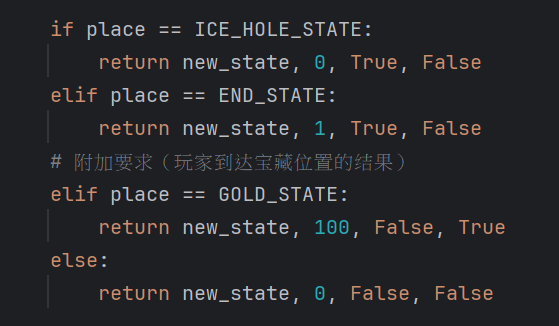
为了验证每次玩家移动的位置是否正确，一开始在先step函数里加上print(new\_state)，可以看到玩家每次移动后位置的整数形式，以此确定每次移动的位置以及其返回的结果是否正确



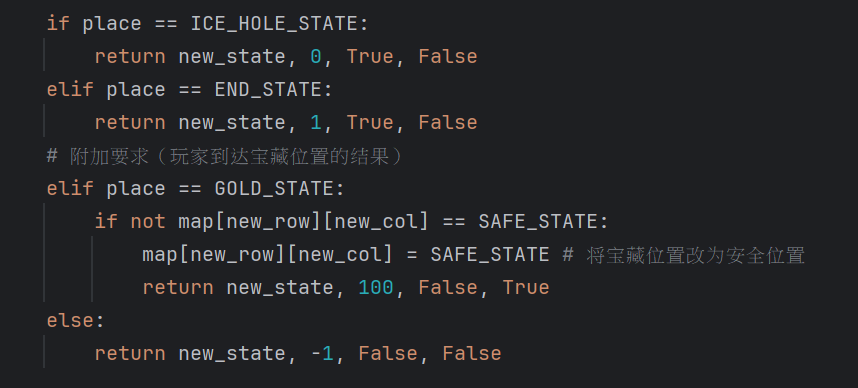
以上述例子为例，当玩家输入s（向下移动）后，位置（new\_state）和对应的地图标记（place）变为14，F；再向右移动一次变为15，G，即到达终点。由此可以确认代码的运行符合我们的期望，对应的返回结果（step函数返回的new\_state，reward，游戏是否结束的flag，是否获得宝藏的flag）也正确。

接下来可以尝试不同输入是否能正常执行（如当移动到地图范围外或非法输入时找不到而不输出对应的new\_state和place等等）；通过几次尝试，代码的运行都符合我的预想。

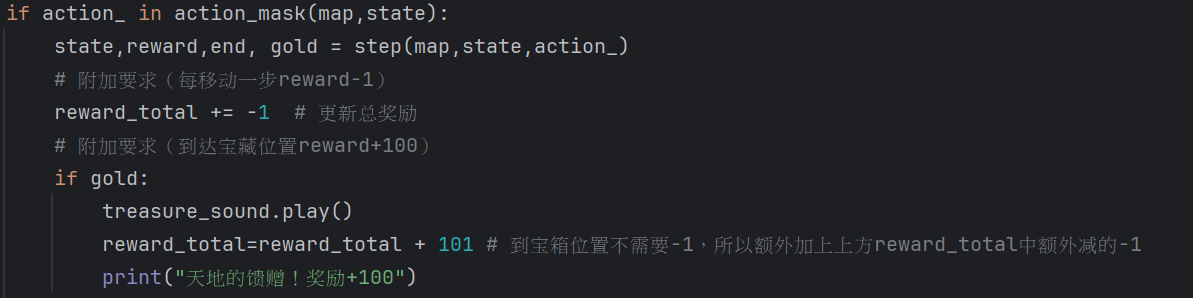
但后来加入宝藏位置时发现玩家可以反复经过宝藏位置造成当前总奖励不断累积，因此在访问宝藏位置的动作代码下加入让已经过的宝藏位置变为安全位置的代码，代码由下方左图变为右图：



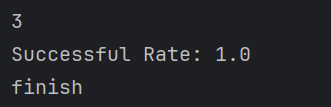
但这里，一开始加入宝藏位置时我的代码原本是：（即如果当前位置是安全位置则reward-1）



但用这个来运行policy.py后，程序貌似出现了死循环，运行了很久都没有出现结果。因此我将其安全位置返回的reward变回0，改为在if \_\_name\_\_ == '\_\_main\_\_':中宝藏位置的结果进行判断：

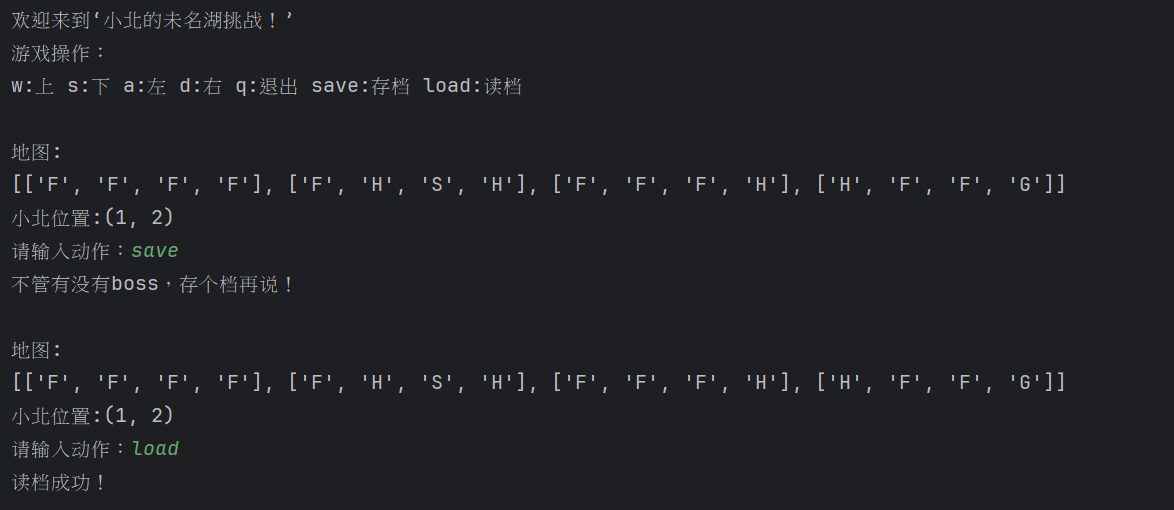


调整完后，policy.py就可以正常运行（没有死循环）了，其结果和之前和相差不大：

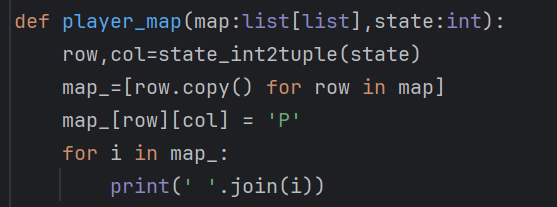
（其中一个结果，其中迭代的次数可能有所不同）

*player\_map()函数：*

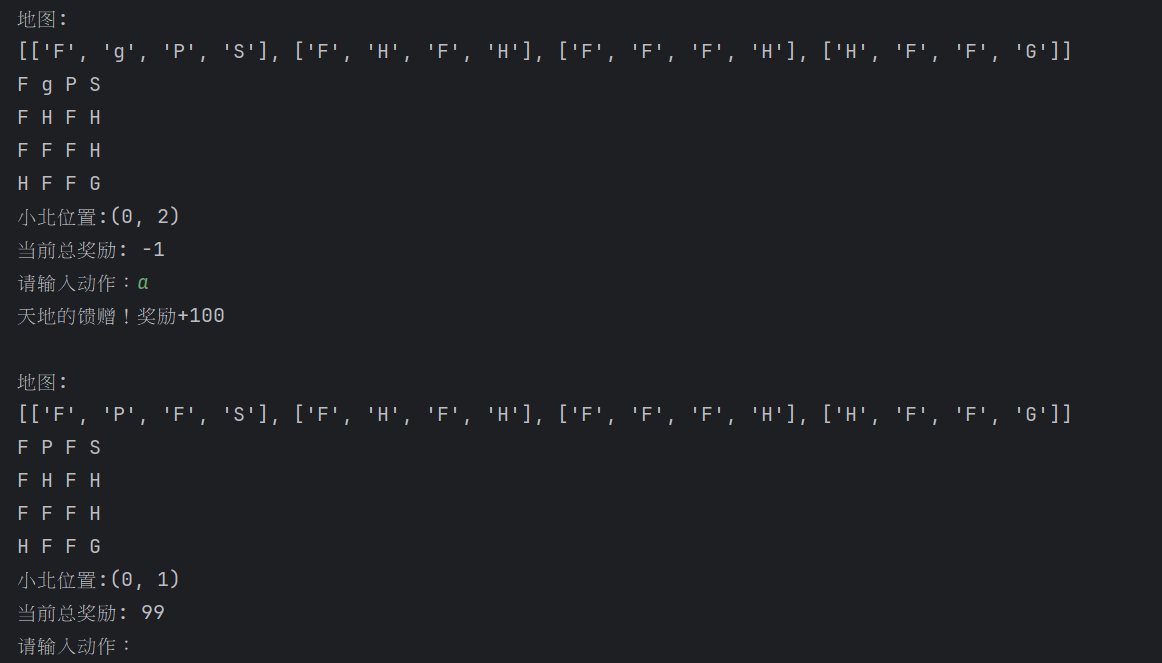
一开始还没定义player\_map()函数时，运行代码游戏显示为：



因为非常不直观且也无法从地图上看出玩家位置，所以另外定义了这个函数来实现我想要的功能（4\*4的地图形式和玩家位置显示）：



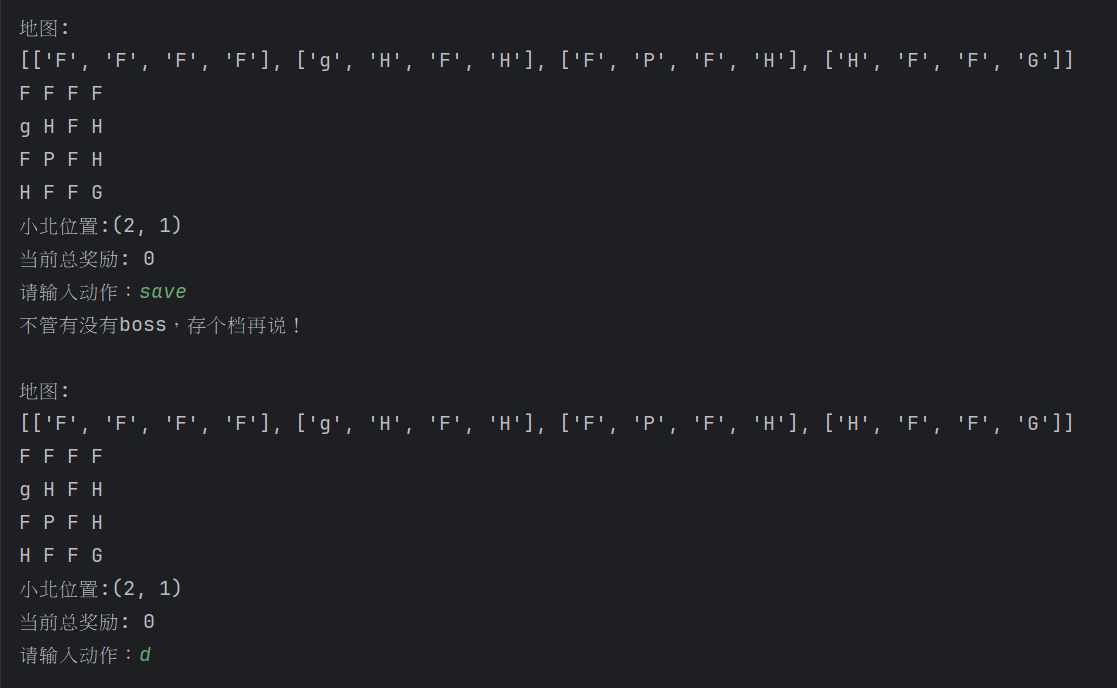
代码主要由将玩家位置state设为“P”和使用了join函数来实现功能，先在player\_map函数里加上print(map\_)，可以查看每次玩家移动后的位置及地图是否正确，接着就可以将该函数应用到if \_\_name\_\_ == '\_\_main\_\_':中：



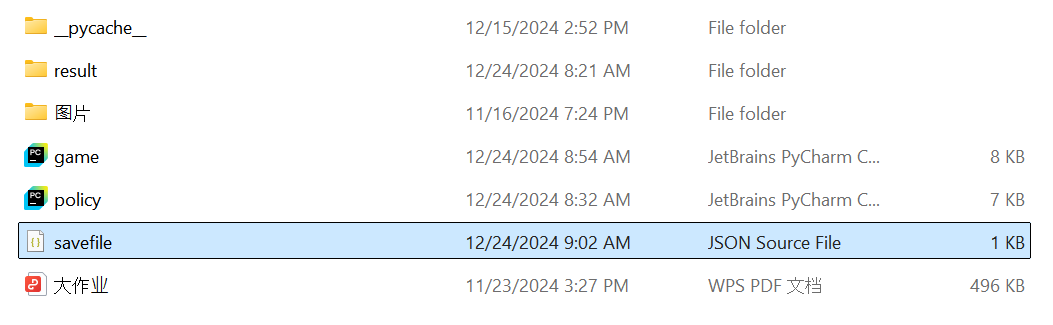
以上述例子为例，可以看到每次生成的地图会先以列表形式储存，当玩家进行移动操作时，对应的列表也会变化（“P”位置随玩家输入变化），且地图最终通过player\_map函数以4\*4的形式输出，到这里可以验证此时代码运行符合我的预想。

接下来同样可以尝试不同输入是否符合预想（如移动到地图范围外或非法输入时地图和玩家位置不变等等），通过几次运行和尝试，其结果也符合预想。

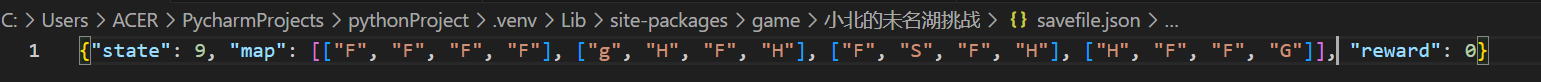
*存档及读档功能：*



首先运行代码并输入“save”来储存游戏，此时在存放文件的目录中可以看到当前游戏状态已被储存进一个名为“savefile”的JSON文件：

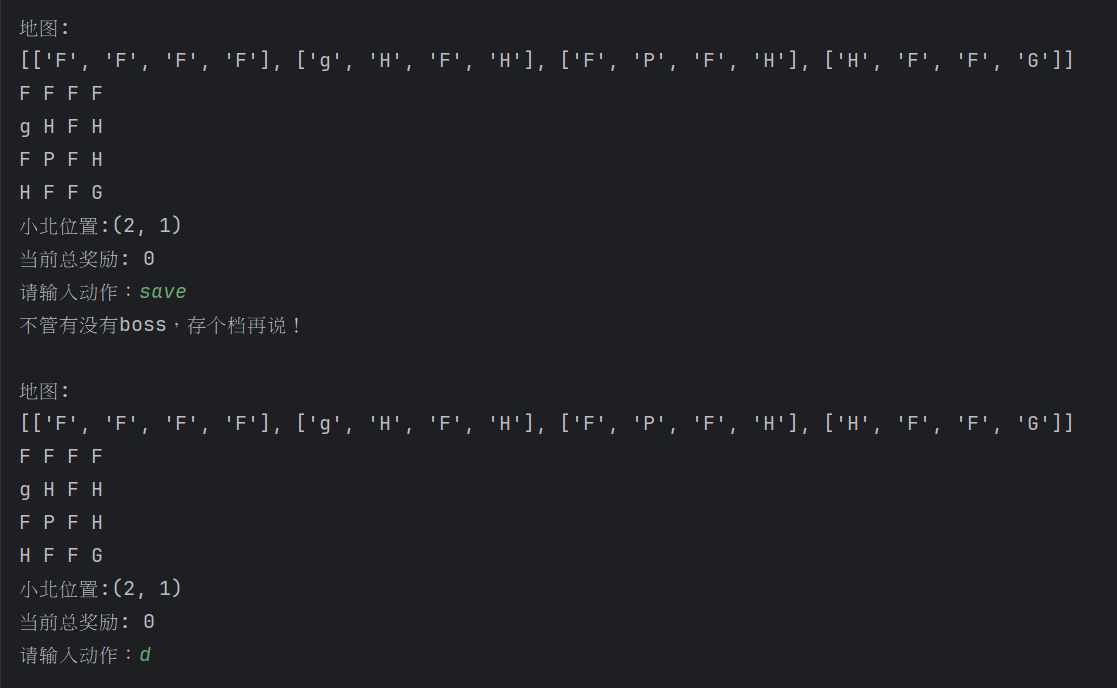


点开后：



可以看到储存的信息是之前我们想要储存的游戏进度（即state=以整数形式表达的第9个位置，map=对应的储存地图，reward=当前奖励状态0），到这里可以确认代码的存档功能可以正常执行。

下一步当进行读档操作时，代码应把这段信息读取并将其输出到控制台中，首先我输入了一个“d”向右移动来改变当前玩家位置以检查读档功能是否能将游戏进度返回到上次的存档：



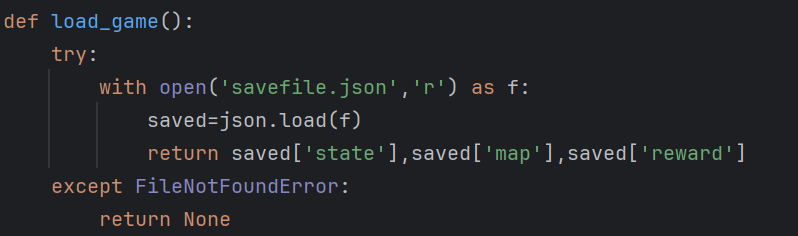
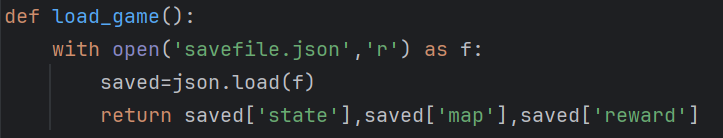
（此时小贝位置：（2，2），当前总奖励=-1）

接着对存档功能进行检查：

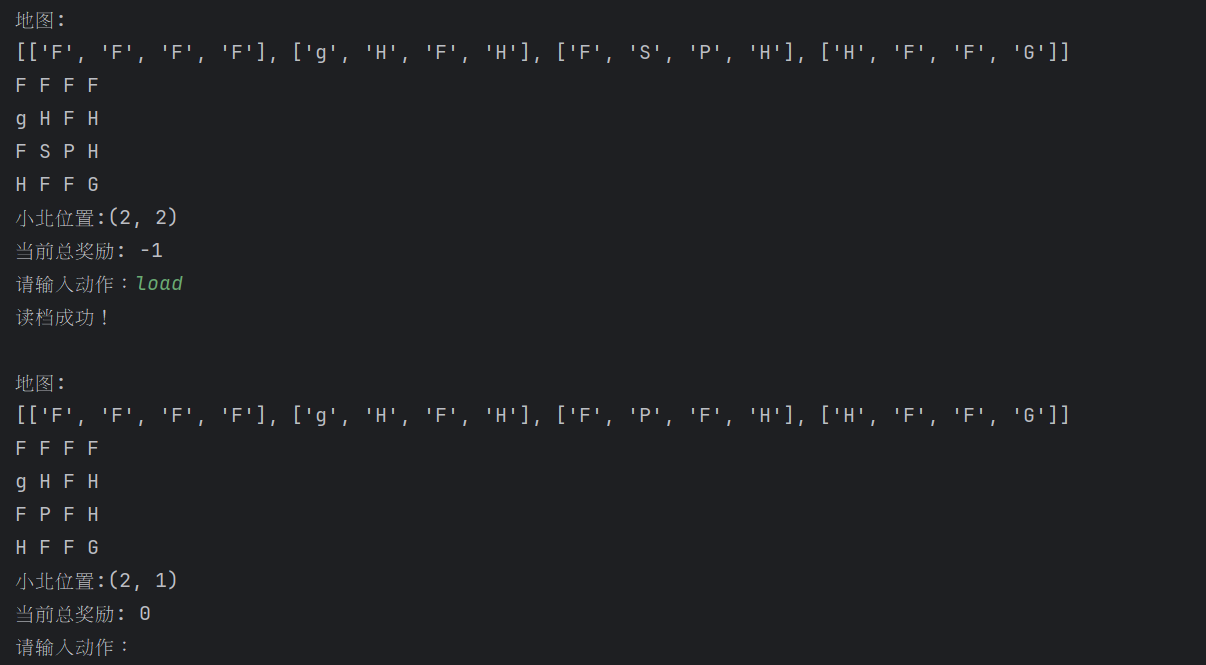
由于一开始忽略了找不到存档的情况，导致在没有存档的情况下程序会报错：



因此对代码进行修改，用try-except来处理异常情况，代码由下方左图变为右图：



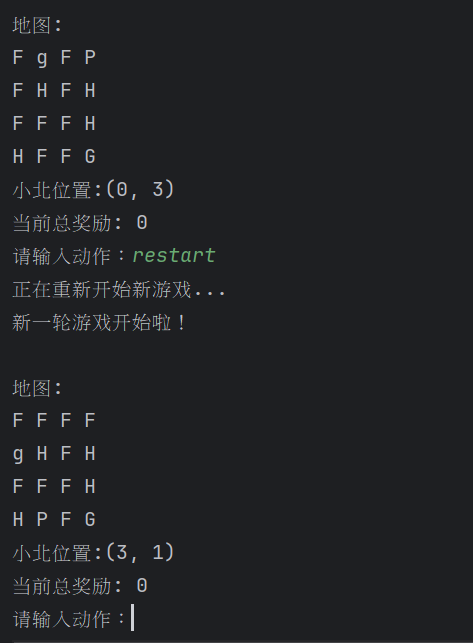
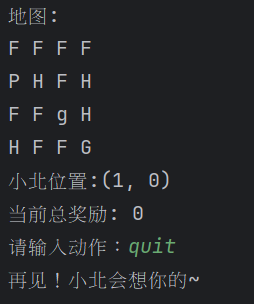
接着，再一次对存档功能进行检查，



可以看到输入“load”进行读档后，控制台中会出现我们之前保存下来的游戏进度（小北位置：（2，1），当前总奖励=0），证明了代码到这里能正常实现其读档功能。

接下来也能通过尝试不同的输入来看存档和读档的结果是否符合预想（如移动到宝藏位置后能正常存档和读档，这一轮游戏结束后在下一轮还能读取同样的进度等等），同样地在通过几次尝试后的代码运行都符合我的预期。

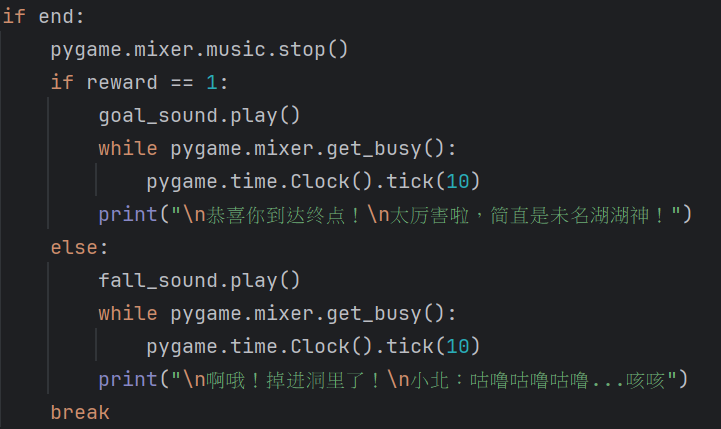
而在整体游戏中，通过各种输入如移动（上下左右）、退出、重启、保存、读档、非法输入等等，也可以看到对应的输出如地图状态、玩家位置、奖励状态、输入所对应的输出提示信息等等都符合预期，下面以退出和重启为例来看输入与对应输出：

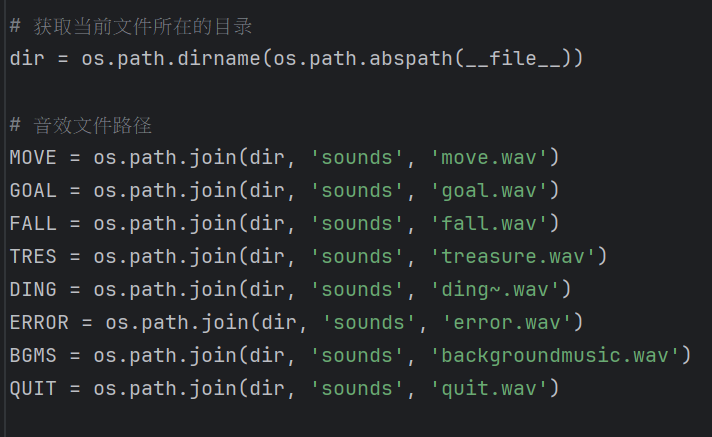


除了退出和重启两种操作，还可以通过不同输入来验证代码的运行是否符合预期。在尝试了其他各种不同的操作后，代码基本都符合我的预想。

*游戏音效：*

之后在处理游戏音效时，也发现了当输入对应着游戏结束时音效没有正常播放及音效文件路径的问题（上方已详细说明，这里做简短的阐述），我采取的措施是使用pygame.mixer.get\_busy()来检查音乐是否播放完成和pygame.time.Clock().tick(10)来循环检查的次数（每秒检查10次）来解决结束游戏的音效不能正常播放的问题，还有使用os库来在不同的设备查找音频文件的相对路径并拼接成绝对路径来解决可能找不到音频文件的问题。

<-解决结束音效播放问题

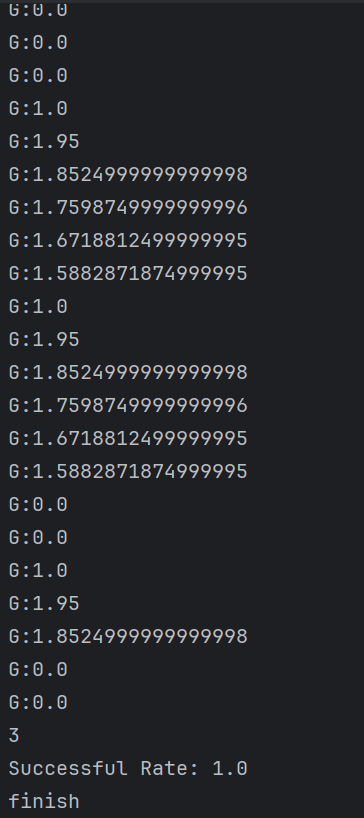
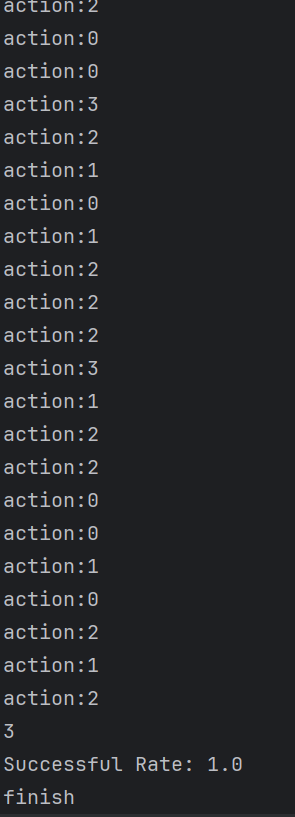
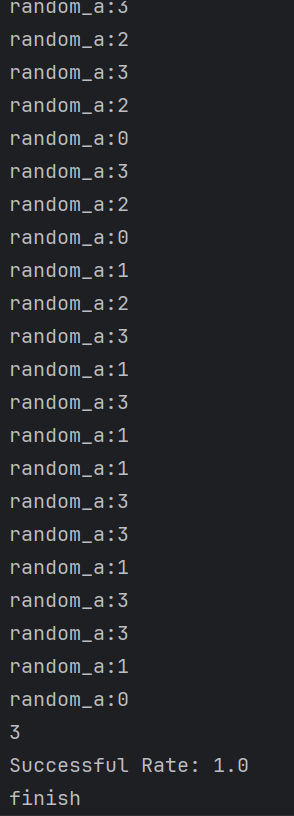
<-解决音频文件路径问题

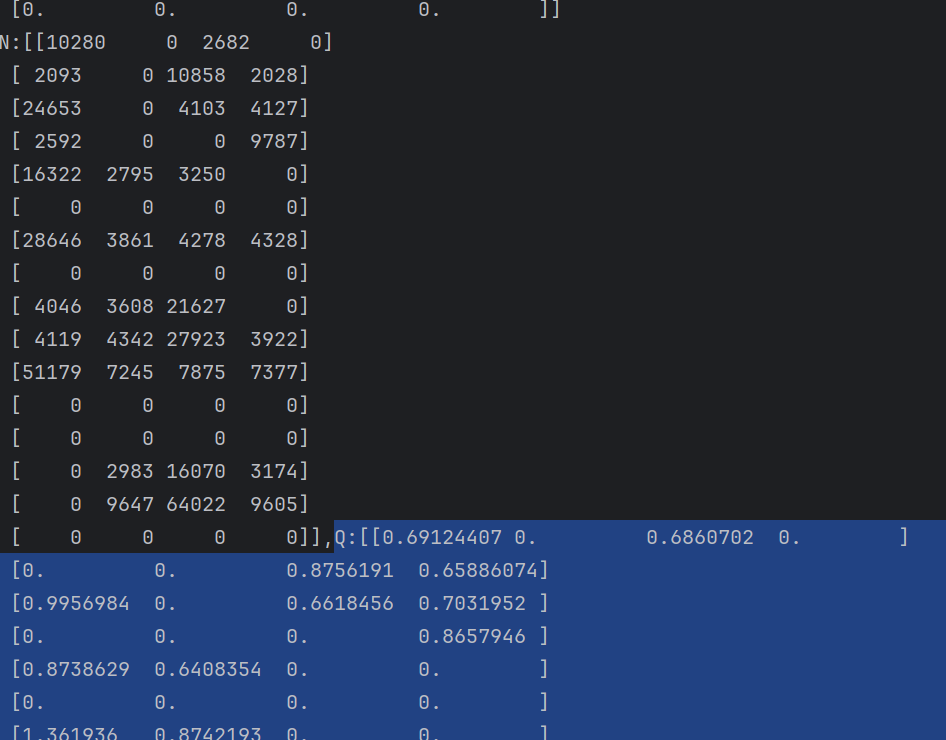
*policy.py中：*

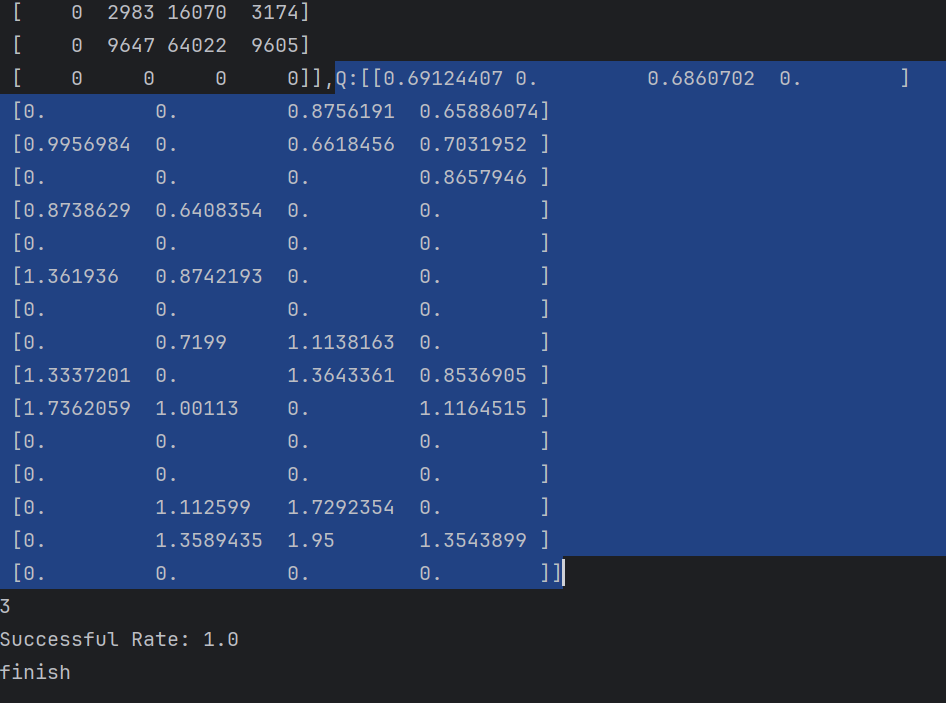
在policy.py中，我选择在sample\_actions()函数和compute\_qpi\_MC()函数中对需要补全的代码的各个返回值进行打印来观察代码的运行：

（random\_a：随机的合法动作，action：最优动作，G：累计回报，N：每次循环计数+1，

Q：动作价值函数）







（N和Q部分只举例一个（因为控制台只装得下一个QwQ），其中蓝色部分为Q）

在上方的结果中，其中random\_a是从game.py的action\_mask函数随机选择一个合法的动作并返回。action用epsilon-greedy策略，以 1 *− ϵ*的概率选择当前的最优动作, 同时以 *ϵ* 的概率在所有合法动作中随机选择，当*ϵ*减小算法会逐渐偏向选取当前策略下的最优动作。G则是每次的累计回报；N和Q会循环重复多次来更新当前状态下的动作价值，直到价值函数足够准确，其中N将每一次访问都+1计数，而Q是更新动作价值函数的算法，即把过去的平均值和这次的结果结合更新为一个新值，最后返回计算后的动作价值函数Q。通过这些运行结果，可以看出各函数与其中不同步骤中需要补全的代码的返回值。

# 总结

通过这个大作业，我学会在python中设计简单的游戏环境，也了解到了各种算法如蒙特卡洛和epsilon-greedy方式采样等等算法的概念以及它们之间的关系和应用，而且通过这次的作业我也认识了python的各种有趣的功能例如音频设置、存档和读档等等。此外，通过这次初步尝试的交互式程序设计，也让我对更复杂的python项目产生了兴趣及有了初步的认识，对未来不同的程序项目的学习奠定了基础。同时也要谢谢甘锐老师和助教老师们在这一学期的悉心指导，让身为电脑小白的我在这学期收获了各种python的相关知识。

# 反思与不足

在此次大作业中，我认为我有着以下不足之处：

1. 在编写和调整代码参数时忘了将每次的运行结果记录下来，导致后来在报告中写实验数据时大部分只能靠回忆，因此无法将详细的思考和调整过程体现出来，因此以后应将每次的调整参数都记录下来，方便以后的查阅。
2. 对于policy.py，只是依靠所给的伪代码和找到的参考资料来对空缺的代码进行补全，但对其中策略的算法还缺乏更深入的理解，导致对其分析不够深入，在未来应该对代码中的算法有更进一步的了解才能对程序进行优化。
3. 对代码中可能出现的问题并没有彻底理解和解决，例如我在调整前的step函数加入宝藏位置的操作时时，原本的代码虽然在game.py中可以正常运行，但在policy.py时好像出现了死循环，导致运行了很长时间都没有出现结果，对这个问题我选择另在if \_\_name\_\_ == '\_\_main\_\_':中对宝藏位置的操作结果进行判断，而没有对原本的问题进行解决；因此在未来应先了解代码问题出在哪里，才能从根本上解决问题。
4. 在附加要求中，修改环境后策略应是有变化的（因为：）但我完成代码后尝试运行几次代码都是与之前相差不大的结果，因此我认为在这点上应是有问题的，应该找出问题是什么才能得到正确的结果，并对其进行分析。

# 参考

1. <https://developer.aliyun.com/article/1400075>
2. <https://developer.aliyun.com/article/1400081>
3. <https://blog.csdn.net/chenmozhe22/article/details/81434081>
4. <https://blog.csdn.net/qq_41556318/article/details/86304765>
5. <http://www.360doc.com/content/18/1224/00/2005961_804010736.shtml>
6. <https://blog.csdn.net/lom9357bye/article/details/79285170>