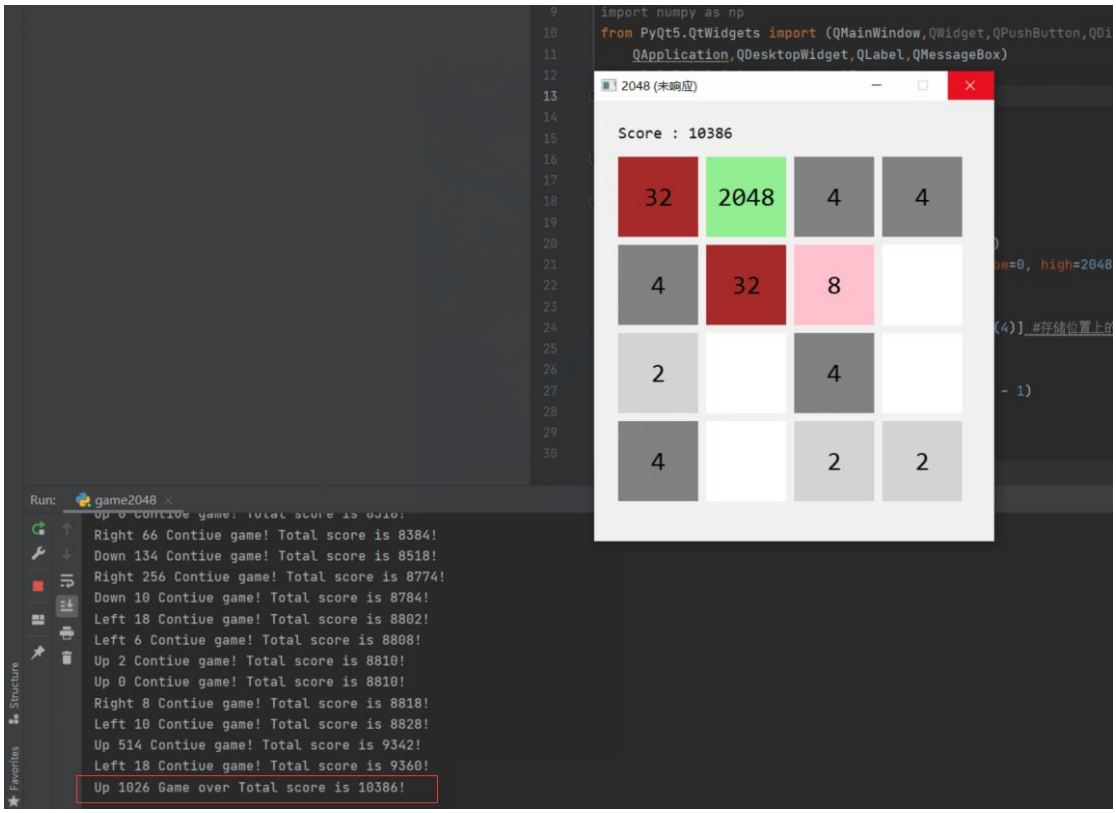
PA2 实验报告

191300073 杨斯凡 191300073@smail.nju.edu.cn

一、实验结果



我注释掉了结束后关闭的代码，就可以不退出。

二、具体思路

这次的问题是 2048 游戏，框架代码将游戏限制到 4x4 的方格上，玩家可以采取上下左右四个行动，相同的数字格子会合并。但是，当采取行动之后，下一个 2 的生成位置是随机的，这就导致采取动作之后的下一个状态未知。

而在阅读完框架代码后，我对给定的接口 API 有了一个初步的了解，之后我使用了课上讲的蒙特卡洛树搜索来进行最佳行动的选取。

我定义了两个类，一个是蒙特卡洛树的节点类，存储了每个节点的状态，Q，N，这个节点的子节点和父节点。另一个是整个蒙特卡洛树，其中各个函数的作用如下：

caculate_value：计算每个节点的上置信界，如果没有探索，则会返回正无穷（使用了 1e50 模拟正无穷）。

expand_choose：是算法中的展开，选择两步，首先选择行动，然后把对应行动的子节点展开。

Evalute：使用了 rollout 方法，默认策略是随机策略，到达预设深度（10）后使用当前评分进行评价。

Reverse_update：沿着当前节点的父节点向上直到根节点，沿路更新节点的 Q 和 N。

三、具体实现

计算上置信界，使用了 PPT 上的公式，如果未探索则返回正无穷，其中我将 c 设置为 100 .

```python
def caculate_value(self,node,action):
    if node.child[action] == None:
        return infinity
    else:
        return node.child[action].Q + c * math.sqrt(math.log(node.N) / node.child[action].N)
```

$$Q(s,a)+c\sqrt{\frac{\log N(s)}{N(s,a)}}$$

评价：我使用了均匀随机的 rollout 方法，深度设置为 10，达到深度 10 之后，选择当前的得分作为值返回。

```python
def evalute(self,node,env):
    value=env.score
    for i in range(d):
        action = random.randint(0, 3)
        obs, rew, done, info = env.step(action)
        if done==True:
            break
        value = env.score
    return value
```

反向更新：从当前节点一直向上遍历，直到遍历到根节点位置，更新每个节点的 Q 和 N

```python
def Reverse_update(self,node,value):
    while node != None:
        node.N += 1
        node.Q = node.Q + (value - node.Q) / node.N
        node = node.father
```

展开选择：首先拷贝当前的所有状态进行计算，然后从根节点开始进行选择最佳的动作，每次模拟做出行动后的回报，直到叶节点返回当前状态。

```
def expand_choose(self):#因为初始状态已经初始化过,所以先选择
    temp=copy.deepcopy(env)
    node=self.root

    while 1:
        action=0
        max_value = -infinity
        for i in range(4):
            now_value=self.caculate_value(node,i)
            if now_value > max_value:
                action=i
                max_value=now_value

        if node.child[action]==None:
            obs, rew, done, info = temp.step(action)
            child_node = Node(temp.state)
            child_node.father = node
            node.child[action] = child_node
            return child_node, temp
        else:
            node = node.child[action]
            obs, rew, done, info = temp.step(action)
```

主循环:

首先获取当前的时间，以保证 1s 内做出选择，每次搜索重置整个搜索树，以当前状态作为新的根节点搜索。循环体在 1s 内会完成选择，展开，评估和反向更新。然后再从根节点的子节点中选择值最大的行动以作为最优行动。

```
def Play(self,state,env):
    self.root = Node(state)
    self.env = copy.deepcopy(env)
    t = time.time()
    t=int(round(t * 1000))
    search_time=0
    while time_bound > int(round(time.time() * 1000))-t:
        new_node, new_env = self.expand_choose()
        value = self.evalute(new_node, new_env)
        self.Reverse_update(new_node, value)
    action = 0
    max_value = -infinity

    for i in range(4):
        if self.root.child[i] != None:
            value = self.root.child[i].Q
            if value > max_value:
                max_value = value
                action = i

    self.env.step(action)
    self.root = self.root.child[action]
    self.root.father = None
    return action
```

但是在实验的过程中有很多的 bug。

首先是在快产生 2048 的时候，会报错 KeyError，这个是框架代码的问题，和助教沟通后，我将代码进行了修改

```
for num in range(1, 15):
    self.color_dict[2 ** num] = col[num - 1]
```

相应的颜色也做出了修改。

第二个问题就是，右上角的得分只会显示 2 位，和助教沟通后发现是 linux 和 windows 的问题，我将字体调小后，就可以显示 4 位。

四、运行方式

直接运行即可