编程作业2实验报告

人工智能学院 181220076 周韧哲

用蒙特卡洛树搜索方法设计和实现会玩2048游戏的智能程序。要求程序每1秒执行一个行动，并可视化智能程序下2048游戏的过程。

- **算法实现**：

  在文件`MCTS.py`中创建了类Node和MCTS，游戏开始后，每次执行动作前，创建一个新的MCTS实例并初始化根节点root，然后在时间限制和迭代次数限制内训练，每一次迭代都是

  $$\text{forward\_search} \rightarrow \text{expand} \rightarrow \text{rollout} \rightarrow \text{backup}$$

  迭代完成后，返回一个最优动作，游戏执行该动作进入下一个状态，不断循环直到游戏结束。我并没有完全按照Slides上的伪代码实现，因为发现当有深度限制的时候游戏结果不太理想，而是按照Alphago论文*Mastering the game of Go with deep neural networks and tree search*（http://www.lamda.nju.edu.cn/IntroAI19/PDF/alphago2016.pdf，上学期俞扬老师人工智能导论课程提供的论文链接）中描述的思路写的。

  - class Node：

    Node有属性parent，last_action(父结点到达该结点所执行的动作)，avaliable_actions，Q、N初始化为0，分别代表奖赏值和访问次数。主要函数有：

    - `set_avaliable_actions(env)`：只在初始化根节点时调用，因为有的动作不会导致游戏局面发生改变，为了约简搜索空间而去掉根节点的无效动作。
    - `is_fully_expanded()`：判断结点是否完全扩展。
    - `get_unused_action()`：当结点未完全扩展时，随机返回一个未被扩展的动作
    - `calc_uct(c)`：根据公式

      $$\frac{Q}{N}+c\sqrt{\frac{2\ln N_{parent}}{N}}$$

      计算UCT值。
    - `select_child(c)`：在选择阶段，根据UCT值选择一个孩子
    - `select_best_action()`：返回$\frac{Q}{N}$最大的孩子节点所对应的动作。

  - class MCTS：

    MCTS有属性root，env，max_iter，simulate_env。主要函数有：

    - `forward_search()`：前向搜索函数，从根节点出发直到一个未被完全扩展的结点，同时在仿真环境中执行结点的动作，选择的时候c值设为动态的$\frac{Q}{N}$，而不是一个常数，因为考虑到游戏状态不同，结点UCT值的左项可能较大也可能较小，使用这一式子可以平衡探索与利用。

      ```
      def forward_search(self):
          self.simulate_env=copy.deepcopy(self.env)
          node=self.root
          while(node.is_fully_expanded()):
              node=node.select_child(node.Q/node.N)
              self.simulate_env.step(node.last_action)
          return node
      ```

- `expand(node)`：随机选择node的一个未被扩展的动作，生成新的孩子结点，并且在仿真环境中执行该动作。

```
def expand(self,node):
    action=node.get_unused_action()
    newNode=Node(node,action)
    node.children.append(newNode)
    node=newNode
    self.simulate_env.step(action)
    return node
```

- `rollout()`：随机执行动作直到游戏结束，返回rollout后与rollout前游戏分数的差值作为reward。

```
def rollout(self):
    score_before=self.env.score
    done=False
    while not done:
        action=random.randint(0,3)
        obs,res,done,info=self.simulate_env.step(action)
    score_after=self.simulate_env.score
    return score_after-score_before
```

- `backup(node,R)`：将奖励R反向传播，更新选择阶段形成的路径上的结点的N与Q。

```
def backup(self,node,R):
    while node != None:
      node.N+=1
      node.Q+=R
      node=node.parent
```

- `train(time_limit)`：在迭代次数范围内不断训练，若超出时间限制，则强制退出，最后返回根节点的最优动作。

```
def train(self,time_limit):
    t1=time.time()
    t2=time.time()
    for i in range(self.max_iter):
        node=self.forward_search()
        node=self.expand(node)
        R=self.rollout()
        self.backup(node,R)
        t2=time.time()
        if (t2-t1)>=time_limit:
            break
    if (t2-t1)<time_limit:
        time.sleep(time_limit-t2+t1)
    best_action=self.root.select_best_action()
    return best_action
```

- 游戏过程：

```
env=Game2048Env()
done=False
state=copy.deepcopy(env.state)
while not done:
    node=Node()
    node.set_avaliable_actions(env)
    agent=MCTS(node,env,max_iter)
    action=agent.train(time_limit)
    obs,rew,done,info=env.step(action)
```

- **实验**:
  - 实验环境：`cpu:intel(R) Core(TM) i5-8250U @ 1.60GHz, Ram:8GB, python3.7.0`

    参数设置：`max_ter=400, time_limit=1`

    运行：`python game2048.py`

    由于笔记本性能不够，在游戏前期通常跑不满400次迭代，前期的平均迭代次数在40-70左右，后期的平均迭代次数在100-400左右。所以理性情况是配置高，所有迭代都能在1s内跑完，游戏成功率也应该会提高。
  - 我进行了100次的随机实验，得到
    - 达到1024的比率：100%
    - 达到2048的比率：87%
    - 平均分数：9986.7
    - 平均步数：767

    当我设置time_limit更低比如0.5、0.8时，成功率偏低一些，从而可以看到蒙特卡洛树搜索的精髓：采样次数越多，就越能逼近最优解。给定2048游戏的一个状态，上下左右4个动作的成功率可以看作是某个概率分布，通过采样来逼近这个分布。

    实验结果还是比较让人满意的，附上游戏成功截图一张：