人工智能基础 实验一

PB18151866 龚小航

**【实验目的】**

在给出的代码框架上完成吃豆人游戏的决策过程，需要完成静态目标下的广度优先搜索与 搜索算法实现寻找食物，在此基础上更进一步在多agent环境中实现算法与剪枝，吃完所有食物同时避开鬼。

**【实验内容】**

本次实验有2个部分，分别是Search和Multiagent。具体而言，Search的目标是吃豆人仅仅是寻找食物；Multiagent的目标是吃完所有食物，同时避开鬼。抽象而言，Search实现的静态查找算法，Multiagent的问题是在有对手的情况下做出下一步决策使自己的利益最大化。Search部分需要实现BFS算法和A\*算法。Multiagent部分需要实现minimax算法和alpha-beta剪枝。

**【实验过程】**

一 、BFS算法实现静态搜索食物。只需将给出的深度优先搜索代码中的栈结构改为普通队列结构。

伪代码如下所示：

文本, 信件

描述已自动生成

将其写为python代码：

**def** myBreadthFirstSearch**(**problem**):**

# YOUR CODE HERE

visited **=** **{}** #存储访问过的节点

frontier **=** util**.**Queue**()** #队列结构存储待访问节点

frontier**.**push**((**problem**.**getStartState**(),** **None))**

**while** **not** frontier**.**isEmpty**():**#还有待访问节点时继续

#取出下一步访问的节点，判断是否为目标节点，返回或扩展

state**,** prev\_state **=** frontier**.**pop**()**

**if** problem**.**isGoalState**(**state**):** #找到目标节点，返回

solution **=** **[**state**]**

**while** prev\_state **!=** **None:**

solution**.**append**(**prev\_state**)**

prev\_state **=** visited**[**prev\_state**]**

**return** solution**[::-**1**]**

**if** state **not** **in** visited**:** #在BFS下，拓展该节点

visited**[**state**]** **=** prev\_state

**for** next\_state**,** step\_cost **in** problem**.**getChildren**(**state**):**

frontier**.**push**((**next\_state**,** state**))**

**return** **[]**

二、实现 静态搜索算法。和类似，存储待访问节点的数据结构为优先队列，权值为，g存储从起始节点开始到当前待访问而未访问节点的实际消耗，在将某个节点插入优先队列的时刻生成。从优先队列取出数据时每个节点都已经有域。

**def** myAStarSearch**(**problem**,** heuristic**):** #预估函数h已经从外部给出，调用即可

# YOUR CODE HERE

visited **=** **{}** #已访问节点列表

frontier **=** util**.**PriorityQueue**()** #优先队列存储待访问节点

frontier**.**\_\_init\_\_**()**

#优先队列结构：同时插入状态与权值，权值为 g+h，初始节点 g=0

frontier**.**push**(((**problem**.**getStartState**(),**0**),** **(None,**0**)),**

0 **+** heuristic**(**problem**.**getStartState**()))**

**while** **not** frontier**.**isEmpty**():**

**(**state**,**g**),** **(**prev\_state**,**prev\_g**)** **=** frontier**.**pop**()**

#与BFS类似，从优先队列中取出下个待访问的节点，判断是否为目标，返回或扩展它

**if** problem**.**isGoalState**(**state**):**

solution **=** **[**state**]**

**while** prev\_state **!=** **None:**

solution**.**append**(**prev\_state**)**

prev\_state **=** visited**[**prev\_state**]**

**return** solution**[::-**1**]**

#扩展节点，同时将 gn 值赋予即将插入优先队列的待访问节点

**if** state **not** **in** visited**:**

visited**[**state**]** **=** prev\_state

**for** next\_state**,** step\_cost **in** problem**.**getChildren**(**state**):**

next\_g **=** g **+** step\_cost

frontier**.**push**(((**next\_state**,**next\_g**),** **(**state**,**g**)),**

next\_g **+** heuristic**(**next\_state**))**

**return** **[]**

三、极小极大算法实现多环境中的动态决策。伪代码如下所示：

表格

描述已自动生成

本次实验中还需要实现同时返回下一步最佳的状态，只需要在child中找出max或min的同时将对应的状态记录下来即可。代码如下所示：

**class** **MyMinimaxAgent():**

**def** \_\_init\_\_**(**self**,** depth**):**

self**.**depth **=** depth

#用于选择使用MAX方案还是MIN方案

**def** minimax**(**self**,** state**,** depth**):**

**if** state**.**isMe**():**

**return** self**.**maxv**(**state**,**depth**)**

**return** self**.**minv**(**state**,**depth**)**

#MAX节点操作

**def** maxv**(**self**,**state**,**depth**):**

beststate **=** **None**

bestv **=** **-float(**'inf'**)**

**if** state**.**isTerminated**():**

**return** **None,** state**.**evaluateScore**()**

**if** depth **==** 0**:**

**return** state**,** state**.**evaluateScore**()**

**for** child **in** state**.**getChildren**():**#MAX行动一次depth-1

st**,**sc **=** self**.**minimax**(**child**,** depth**-**1**)**

**if** sc **>=** bestv**:**

bestv **=** sc

beststate **=** child

**return** beststate**,**bestv

#MIN节点的操作

**def** minv**(**self**,**state**,**depth**):**

beststate **=** **None**

bestv **=** **float(**'inf'**)**

**if** state**.**isTerminated**():**

**return** **None,** state**.**evaluateScore**()**

**for** child **in** state**.**getChildren**():**#MIN行动一次depth不变

st**,**sc **=** self**.**minimax**(**child**,** depth**)**

**if** sc **<=** bestv**:**

bestv **=** sc

beststate **=** child

**return** beststate**,**bestv

#外部调用的接口，输入当前状态返回下一步的最佳行动状态

**def** getNextState**(**self**,** state**):**

best\_state**,** \_ **=** self**.**minimax**(**state**,** self**.**depth**)**

**return** best\_state

四、在极小极大算法的基础上实现 剪枝。类似MAXMIN算法，在其上增加每个节点的上下限 。伪代码实现如下所示：

表格

描述已自动生成

将其实现为python代码，如下所示：

**class** **MyAlphaBetaAgent():**

**def** \_\_init\_\_**(**self**,** depth**):**

self**.**depth **=** depth

#外部调用接口，输入当前状态返回最佳后继状态

**def** getNextState**(**self**,** state**):**

# YOUR CODE HERE

best\_state**,** \_ **=** self**.**alpha\_beta\_cut**(**state**,** self**.**depth**,**

**-float(**'inf'**),** **float(**'inf'**))**

**return** best\_state

#用于选择调用MAX还是MIN方法

**def** alpha\_beta\_cut**(**self**,** state**,** depth**,** alpha**,** beta**):**

**if** state**.**isMe**():**

**return** self**.**alpha\_beta\_cutmaxv**(**state**,**depth**,** alpha**,** beta**)**

**return** self**.**alpha\_beta\_cutminv**(**state**,**depth**,** alpha**,** beta**)**

**def** alpha\_beta\_cutmaxv**(**self**,**state**,**depth**,** alpha**,** beta**):**

beststate **=** **None**

bestv **=** **-float(**'inf'**)**

**if** state**.**isTerminated**():**

**return** **None,** state**.**evaluateScore**()**

**if** depth **==** 0**:**

**return** state**,** state**.**evaluateScore**()**

**for** child **in** state**.**getChildren**():**

st**,**sc **=** self**.**alpha\_beta\_cut**(**child**,** depth**-**1**,** alpha**,** beta**)**

**if** sc **>** bestv**:**

bestv **=** sc

beststate **=** child

**if** bestv **>** beta**:**

**return** beststate**,**bestv

alpha **=** **max(**alpha**,** bestv**)**

**return** beststate**,**bestv

**def** alpha\_beta\_cutminv**(**self**,**state**,**depth**,** alpha**,** beta**):**

beststate **=** **None**

bestv **=** **float(**'inf'**)**

**if** state**.**isTerminated**():**

**return** **None,** state**.**evaluateScore**()**

**for** child **in** state**.**getChildren**():**

st**,**sc **=** self**.**alpha\_beta\_cut**(**child**,** depth**,** alpha**,** beta**)**

**if** sc **<** bestv**:**

bestv **=** sc

beststate **=** child

**if** bestv **<** alpha**:**

**return** beststate**,**bestv

beta **=** **min(**beta**,** bestv**)**

**return** beststate**,**bestv

**【实验结果】**

运行test.sh测试文件，得到的结果如下所示。各测试结果均为PASS，符合预期。

文本

描述已自动生成