LAB2 实验报告

袁雨 PB20151804

一、实验环境

使用Anaconda管理Python环境，使用Python 3.6 版本。

二、实验要求

本次实验只需改动并提交mylmpl.py文件，完成实验无需阅读其他代码文件。请勿在mylmpl.py中增加import其他模块，否则会造成测试失败。

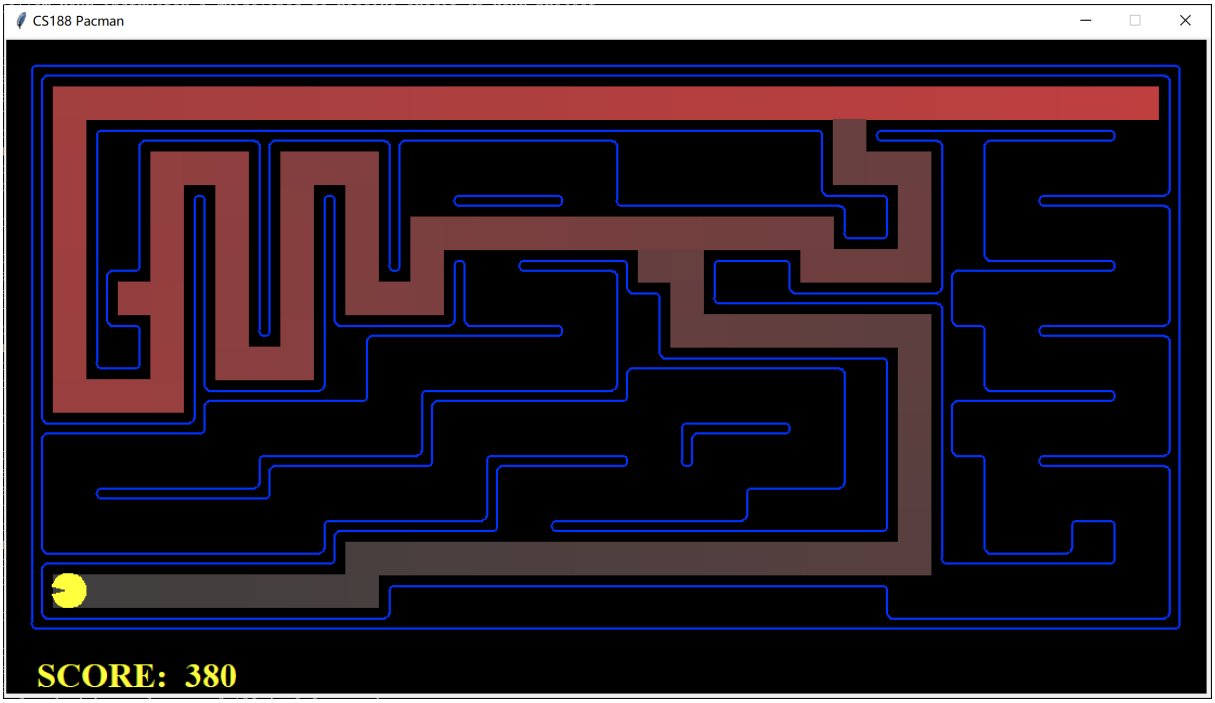
本次实验需要你按照给定框架补完4个算法，分为Search和Multi-Agent两类。具体而言，Search 的目标是吃豆人在没有鬼的干扰下寻找食物；Multi-Agent 的目标是在鬼的干扰下选择最优行动去吃食物。在Search中需要你实现BFS算法和A*算法, Multi-Agent 类需要实现minimax算法和alpha-beta剪枝。

三、实验算法、结果与分析

1.Search

（1）DFS算法

```
(pytorch3.6) D:\pythonProject\AI\LAB2\search>python autograder.py -q q1
Starting on 6-29 at 18:44:17

Question q1
===========
*** PASS: test_cases\q1\graph_backtrack.test
***     solution:               ['1:A->C', '0:C->G']
***     expanded_states:        ['A', 'D', 'C']
*** PASS: test_cases\q1\graph_bfs_vs_dfs.test
***     solution:               ['2:A->D', '0:D->G']
***     expanded_states:        ['A', 'D']
*** PASS: test_cases\q1\graph_infinite.test
***     solution:               ['0:A->B', '1:B->C', '1:C->G']
***     expanded_states:        ['A', 'B', 'C']
*** PASS: test_cases\q1\graph_manypaths.test
***     solution:               ['2:A->B2', '0:B2->C', '0:C->D', '2:D->E2', '0:E2->F', '0:F->G']
***     expanded_states:        ['A', 'B2', 'C', 'D', 'E2', 'F']
*** PASS: test_cases\q1\pacman_1.test
***     pacman layout:          mediumMaze
***     solution length: 130
***     nodes expanded:         146

### Question q1: 4/4 ###


Finished at 18:44:17

Provisional grades
==================
Question q1: 4/4
------------------
Total: 4/4
```




```
(pytorch3.6) D:\pythonProject\AI\LAB2\search>python pacman.py -l mediumMaze -p SearchAgent --frameTime 0
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 130 in 0.0 seconds
Search nodes expanded: 146
Pacman emerges victorious! Score: 380
Average Score: 380.0
Scores:        380.0
Win Rate:      1/1 (1.00)
Record:        Win
```

(2) BFS算法

广度优先搜索算法的思想是：从图中某顶点v出发，在访问了v之后依次访问v的各个未曾访问过的邻接点，然后分别从这些邻接点出发依次访问它们的邻接点，并使得先被访问的顶点的邻接点先于后被访问的顶点的邻接点被访问，直至图中所有已被访问的顶点的邻接点都被访问到。如果此时图中尚有顶点未被访问，则需要另选一个未曾被访问过的顶点作为新的起始点，重复上述过程，直至图中所有的顶点都被访问到为止。

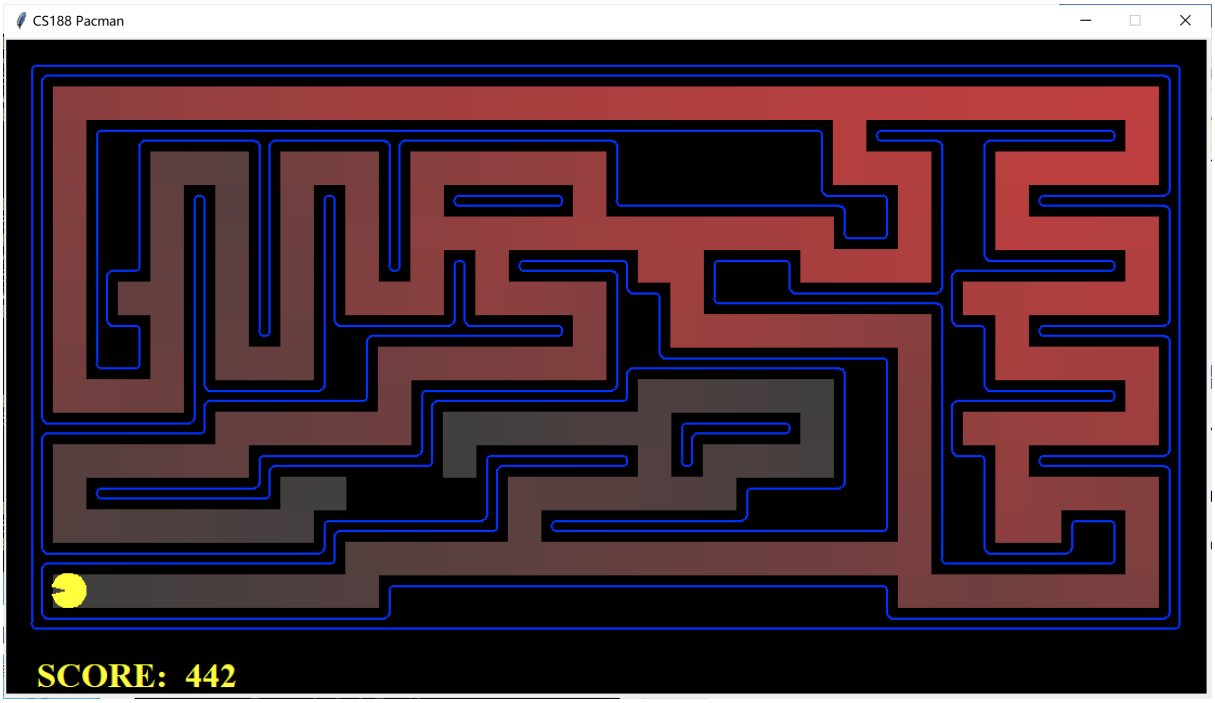
通过`myBreadthFirstSearch`函数实现广度优先搜索的功能。

由于搜索过程中会重复访问到部分节点，设置字典`visited = {}`标记该节点是否被访问过，其中键为当前节点`state`，值为当前节点的父节点`prev_state`。使用队列`frontier = util.Queue()`进行操作，每次将当前`state`与它的`prev_state`组合成元组进出队，先进先出。

首先应用`problem.getStartState()`找到一个特定位置的豆，将其与它的前一个节点即None组合成元组入队。如果队伍不空，就将队首的`state`和`prev_state`出队，如果该`state`是有效的目标，则放入结果集`solution`，如果该`state`的`prev_state`不为空，就将`prev_state`追加到`solution`，将`prev_state`更新为原`prev_state`的`prev_state`，即`visited[prev_state]`，最后返回`solution[::-1]`，即solution的最后一个值到第一个值，即路径。如果该`state`未被访问过，将`visited[state]`更新为`prev_state`，即保存`state`的上一个节点。并依次将所有后续节点`next_state`与该`state`组成的元组入队。

BFS的大致搜索思路与DFS一致，只是搜索的次序发生了变化。对应地，BFD与DFS的图搜索算法一致，但涉及到的具体的数据结构不同，DFS是栈，按照压栈顺序的逆序进行搜索；BFS是队列，按照入队顺序的顺序进行搜索。

```
(pytorch3.6) D:\pythonProject\AI\LAB2\search>python autograder.py -q q2
Starting on 6-29 at 16:15:07

Question q2
===========
*** PASS: test_cases\q2\graph_backtrack.test
***     solution:               ['1:A->C', '0:C->G']
***     expanded_states:        ['A', 'B', 'C', 'D']
*** PASS: test_cases\q2\graph_bfs_vs_dfs.test
***     solution:               ['1:A->G']
***     expanded_states:        ['A', 'B']
*** PASS: test_cases\q2\graph_infinite.test
***     solution:               ['0:A->B', '1:B->C', '1:C->G']
***     expanded_states:        ['A', 'B', 'C']
*** PASS: test_cases\q2\graph_manypaths.test
***     solution:               ['1:A->C', '0:C->D', '1:D->F', '0:F->G']
***     expanded_states:        ['A', 'B1', 'C', 'B2', 'D', 'E1', 'F', 'E2']
*** PASS: test_cases\q2\pacman_1.test
***     pacman layout:          mediumMaze
***     solution length: 68
***     nodes expanded:         269

### Question q2: 4/4 ###


Finished at 16:15:07

Provisional grades
==================
Question q2: 4/4
------------------
Total: 4/4
```




```
(pytorch3.6) D:\pythonProject\AI\LAB2\search>python pacman.py -l mediumMaze -p SearchAgent -a fn=bfs --frameTime 0
[SearchAgent] using function bfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 68 in 0.0 seconds
Search nodes expanded: 269
Pacman emerges victorious! Score: 442
Average Score: 442.0
Scores:        442.0
Win Rate:      1/1 (1.00)
Record:        Win
```

(3) $A*$算法

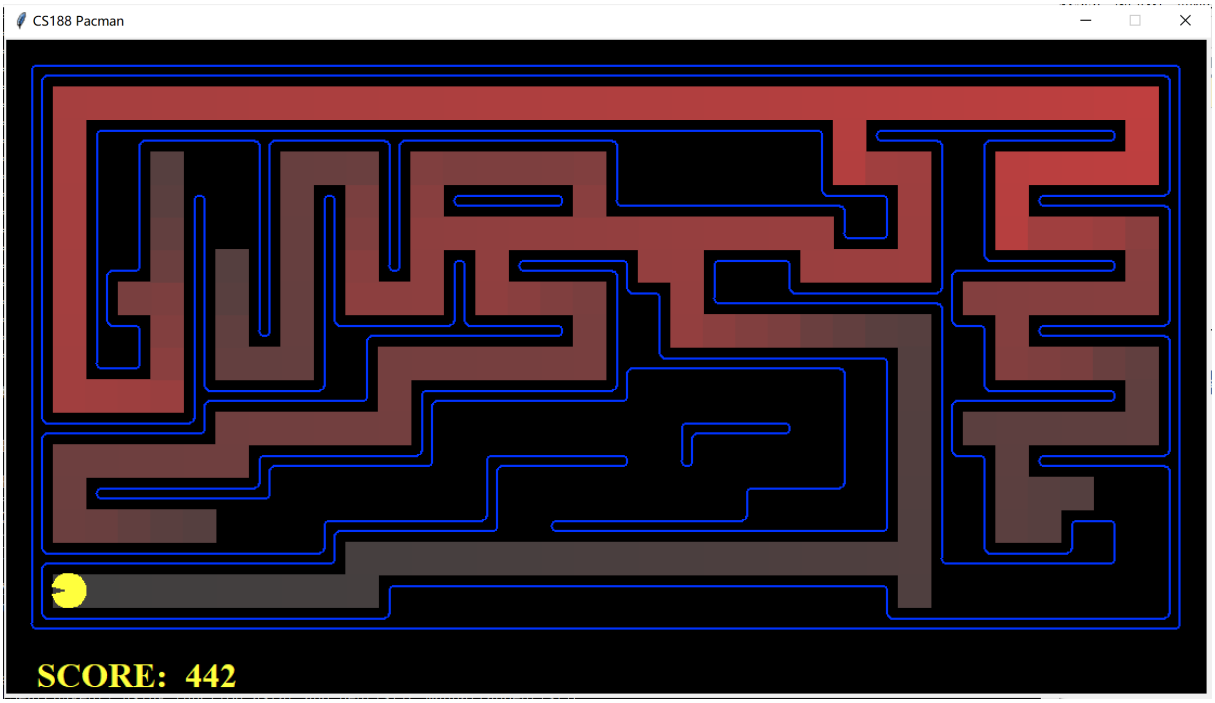
$A*$算法是扩展代价最低的节点，而不是深度最浅的节点。$f(n) = g(n) + h(n)$，其中$f(n)$是从初始状态经由状态n到目标状态的代价估计，$g(n)$是在状态空间中从初始状态到n的实际代价，$h(n)$是从状态n到目标状态的最佳路径的估计代价（对于路径搜索问题，状态就是图中的节点，代价就是距离）。

通过`myAStarSearch`函数实现$A*$搜索的功能。

创建字典`visited`，存储`state`与`prev_state`键值对；字典`g`，存储`g[state]`；字典`f`，存储`f[state]`。创建优先队列`frontier = util.PriorityQueue()`，值越小就会越先出队。首先使用`problem.getStartState()`获得`start_state`，使用`heuristic(problem.getStartState()) + 0`获得`h`值，即`start_priority`，将`(start_state, None), start_priority`入队。设置初值`g[start_state] = 0`。

大致思路与BFS算法类似，有改动的地方：①队列不空时，值越小就会越先出队。②如果该`state`未被访问过时，除了将`visited[state]`更新为`prev_state`，还要依次更新`state`的所有后续节点`next_state`的`g`与`f`值，即`g[next_state] = g[state] + step_cost`，`f[next_state] = g[next_state] + heuristic(next_state)`。最后将`(next_state, state), f[next_state]`入队。

```
(pytorch3.6) D:\pythonProject\AI\LAB2\search>python autograder.py -q q3
Starting on 6-29 at 18:02:50

Question q3
===========
*** PASS: test_cases\q3\astar_0.test
***     solution:               ['Right', 'Down', 'Down']
***     expanded_states:        ['A', 'B', 'D', 'C', 'G']
*** PASS: test_cases\q3\astar_1_graph_heuristic.test
***     solution:               ['0', '0', '2']
***     expanded_states:        ['S', 'A', 'D', 'C']
*** PASS: test_cases\q3\astar_2_manhattan.test
***     pacman layout:          mediumMaze
***     solution length: 68
***     nodes expanded:         221
*** PASS: test_cases\q3\astar_3_goalAtDequeue.test
***     solution:               ['1:A->B', '0:B->C', '0:C->G']
***     expanded_states:        ['A', 'B', 'C']
*** PASS: test_cases\q3\graph_backtrack.test
***     solution:               ['1:A->C', '0:C->G']
***     expanded_states:        ['A', 'B', 'C', 'D']
*** PASS: test_cases\q3\graph_manypaths.test
***     solution:               ['1:A->C', '0:C->D', '1:D->F', '0:F->G']
***     expanded_states:        ['A', 'B1', 'C', 'B2', 'D', 'E1', 'F', 'E2']

### Question q3: 4/4 ###


Finished at 18:02:50

Provisional grades
==================
Question q3: 4/4
------------------
Total: 4/4
```




```
(pytorch3.6) D:\pythonProject\AI\LAB2\search>python pacman.py -l mediumMaze -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic --frameTime 0
[SearchAgent] using function astar and heuristic manhattanHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 68 in 0.0 seconds
Search nodes expanded: 221
Pacman emerges victorious! Score: 442
Average Score: 442.0
Scores:        442.0
Win Rate:      1/1 (1.00)
Record:        Win
```

（4）比较

从图形上看，DFS选择的路径与搜索过的状态较深，BFS较广，$A*$介于二者之间。

DFS算法在搜索过程中，先尽量向深处（离出发点更远的位置）搜索，再回溯并搜索其它分支。BFS算法在搜索过程中，先尽量搜索兄弟节点，再进入更深层的节点继续搜索。$A*$算法是最有效的直接搜索算法，使用公式进行预处理，节省了大量不必要的搜索路径，提高了效率。

2.Multi-Agent

（1）minmax算法

极大极小算法从当前状态计算极大极小决策。它使用了递归算法计算每个后继的极大极小值，自上而下一直前进到树的叶节点，然后随着递归回溯通过搜索树把极小极大值回传。

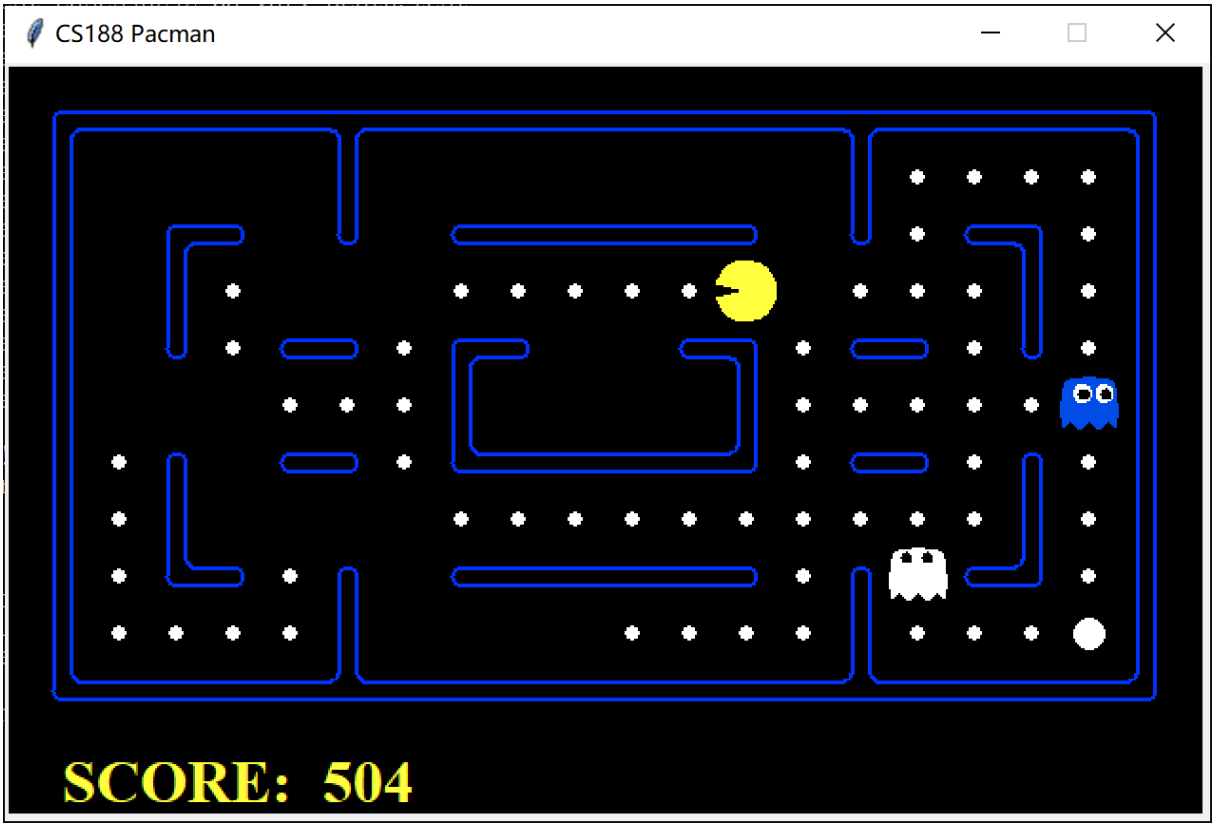
外部函数调用`getNextState`获得`best_state`，通过`minimax`函数实现极大极小策略求得。

自己的agent为吃豆人，对手agent为鬼。当前状态`state`停止，游戏结束或`depth==0`，agent所走步数为0是终止条件，返回`None，state.evaluateScore()`。`best_state`初始化为$None$，若轮到己方agent，`best_score`初始化为$-float('inf')$，若轮到对方agent，`best_score`初始化为$float('inf')$。

$$state\begin{cases}轮到己方\,agent & 递归\_,score=self.minimax(child,depth)得到child的最佳分数，更新best\_score，best\_state\\ 轮到对方\,agent & \begin{cases}child轮到己方\,agent & 递归\_,score=self.minimax(child,depth-1)，更新best\_score\\ child轮到对方\,agent & 递归\_,score=self.minimax(child,depth)，更新best\_score\end{cases}\end{cases}$$

注意鬼到人时，agent已走完一轮，递归时depth-1。最后返回`state`的所有`child`中的`best_state`与`best_score`。

```
(pytorch3.6) D:\pythonProject\AI\LAB2\multiagent>python autograder.py -q q2 --no-graphics
Starting on 6-30 at 0:43:14

Question q2
===========

*** PASS: test_cases\q2\0-eval-function-lose-states-1.test
*** PASS: test_cases\q2\0-eval-function-lose-states-2.test
*** PASS: test_cases\q2\0-eval-function-win-states-1.test
*** PASS: test_cases\q2\0-eval-function-win-states-2.test
*** PASS: test_cases\q2\0-lecture-6-tree.test
*** PASS: test_cases\q2\0-small-tree.test
*** PASS: test_cases\q2\1-1-minmax.test
*** PASS: test_cases\q2\1-2-minmax.test
*** PASS: test_cases\q2\1-3-minmax.test
*** PASS: test_cases\q2\1-4-minmax.test
*** PASS: test_cases\q2\1-5-minmax.test
*** PASS: test_cases\q2\1-6-minmax.test
*** PASS: test_cases\q2\1-7-minmax.test
*** PASS: test_cases\q2\1-8-minmax.test
*** PASS: test_cases\q2\2-1a-vary-depth.test
*** PASS: test_cases\q2\2-1b-vary-depth.test
*** PASS: test_cases\q2\2-2a-vary-depth.test
*** PASS: test_cases\q2\2-2b-vary-depth.test
*** PASS: test_cases\q2\2-3a-vary-depth.test
*** PASS: test_cases\q2\2-3b-vary-depth.test
*** PASS: test_cases\q2\2-4a-vary-depth.test
*** PASS: test_cases\q2\2-4b-vary-depth.test
*** PASS: test_cases\q2\2-one-ghost-3level.test
*** PASS: test_cases\q2\3-one-ghost-4level.test
*** PASS: test_cases\q2\4-two-ghosts-3level.test
*** PASS: test_cases\q2\5-two-ghosts-4level.test
*** PASS: test_cases\q2\6-tied-root.test
*** PASS: test_cases\q2\7-1a-check-depth-one-ghost.test
*** PASS: test_cases\q2\7-1b-check-depth-one-ghost.test
*** PASS: test_cases\q2\7-1c-check-depth-one-ghost.test
*** PASS: test_cases\q2\7-2a-check-depth-two-ghosts.test
*** PASS: test_cases\q2\7-2b-check-depth-two-ghosts.test
*** PASS: test_cases\q2\7-2c-check-depth-two-ghosts.test
*** Running MinimaxAgent on smallClassic 1 time(s).
Pacman died! Score: 84
Average Score: 84.0
Scores:        84.0
Win Rate:      0/1 (0.00)
Record:        Loss
*** Finished running MinimaxAgent on smallClassic after 2 seconds.
*** Won 0 out of 1 games. Average score: 84.000000 ***
*** PASS: test_cases\q2\8-pacman-game.test

### Question q2: 5/5 ###


Finished at 0:43:16

Provisional grades
==================
Question q2: 5/5
------------------
Total: 5/5
```




```
(pytorch3.6) D:\pythonProject\AI\LAB2\multiagent>python pacman.py -p MinimaxAgent -l mediumClassic --frameTime 0
Pacman emerges victorious! Score: 1479
Average Score: 1479.0
Scores:        1479.0
Win Rate:      1/1 (1.00)
Record:        Win
```

(2) alpha-beta剪枝

alpha-beta剪枝可以应用于任何深度的树，一般原则是：考虑在树中某处的结点n，选手选择移动到该结点。如果选手在n的父结点或者更上层的任何选择点有更好的选择m，那么在实际的博弈中就永远不会到达n。所以一旦发现关于n的足够信息（通过检查它的某些后代），能够得到上述结论，我们就可以剪裁它。

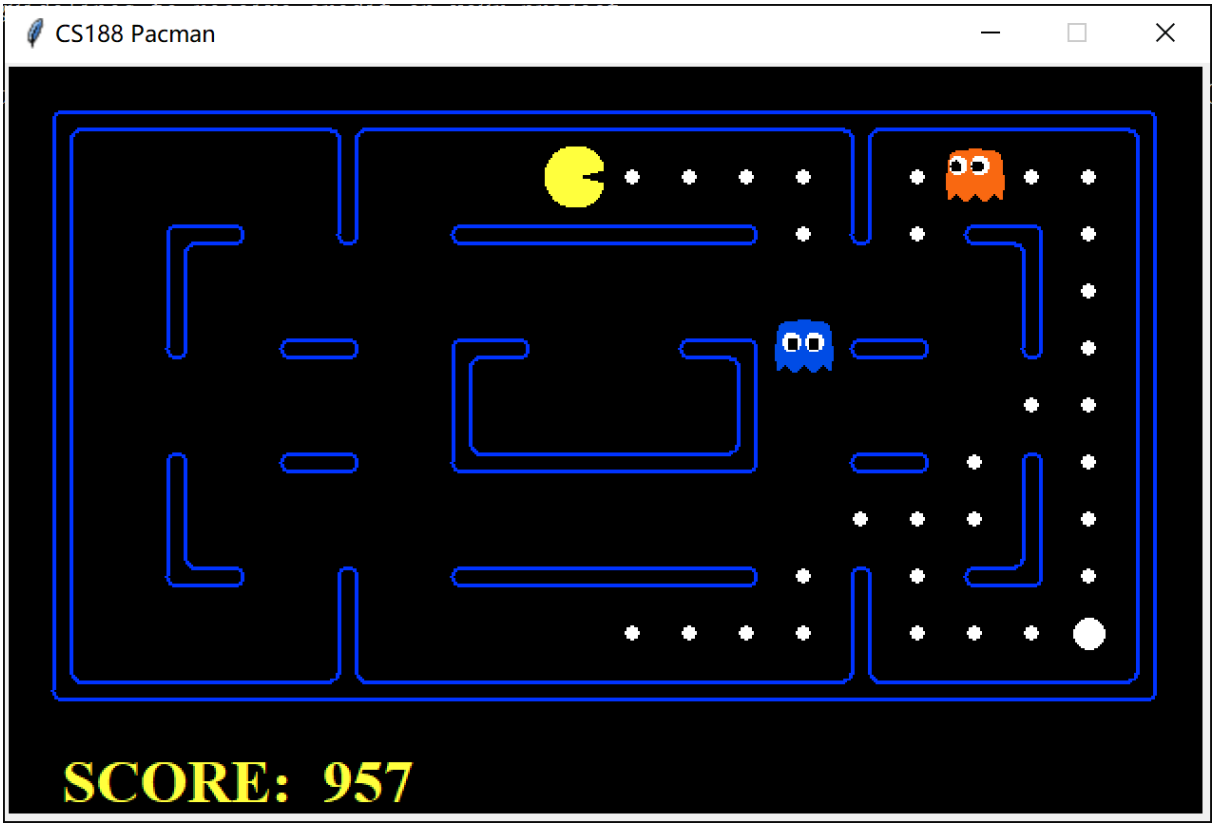
因为极小极大搜索是深度优先的，所以任何时候只需考虑树中某条单一路径上的结点。alpha=到目前为止路径上发现的MAX的最佳(即极大值)选择；beta=到目前为止路径上发现的MIN的最佳(即极小值)选择。alpha-beta搜索中不断更新alpha和beta的值，并且当某个结点的值分别比目前的MAX的alpha或者MIN的β值更差的时候剪裁此结点剩下的分支(即终止递归调用)。

通过`AlphaBeta`函数实现alpha-beta剪枝。

大致思路与极大极小算法类似，外部函数调用`getNextState`获得`best_state`，通过`AlphaBeta`函数实现极大极小策略求得，`alpha`初始化为$-float('inf')$，`beta`初始化为$float('inf')$。

但是在更新`best_score`的同时，如果`state`轮到己方agent，要将`beta`与`best_score`比较，若`beta<best_score`，则进行剪枝，返回当前的`best_state,best_score`。更新`alpha`值`alpha = max(alpha, best_score)`；若轮到对方agent，则将`alpha`与`best_score`比较，若`alpha>best_score`，则进行剪枝，返回当前的`best_state,best_score`。更新`beta`值`beta= min(beta, v)`。

```
(pytorch3.6) D:\pythonProject\AI\LAB2\multiagent>python autograder.py -q q3 --no-graphics
Starting on 6-30 at 0:23:37

Question q3
===========

*** PASS: test_cases\q3\0-eval-function-lose-states-1.test
*** PASS: test_cases\q3\0-eval-function-lose-states-2.test
*** PASS: test_cases\q3\0-eval-function-win-states-1.test
*** PASS: test_cases\q3\0-eval-function-win-states-2.test
*** PASS: test_cases\q3\0-lecture-6-tree.test
*** PASS: test_cases\q3\0-small-tree.test
*** PASS: test_cases\q3\1-1-minmax.test
*** PASS: test_cases\q3\1-2-minmax.test
*** PASS: test_cases\q3\1-3-minmax.test
*** PASS: test_cases\q3\1-4-minmax.test
*** PASS: test_cases\q3\1-5-minmax.test
*** PASS: test_cases\q3\1-6-minmax.test
*** PASS: test_cases\q3\1-7-minmax.test
*** PASS: test_cases\q3\1-8-minmax.test
*** PASS: test_cases\q3\2-1a-vary-depth.test
*** PASS: test_cases\q3\2-1b-vary-depth.test
*** PASS: test_cases\q3\2-2a-vary-depth.test
*** PASS: test_cases\q3\2-2b-vary-depth.test
*** PASS: test_cases\q3\2-3a-vary-depth.test
*** PASS: test_cases\q3\2-3b-vary-depth.test
*** PASS: test_cases\q3\2-4a-vary-depth.test
*** PASS: test_cases\q3\2-4b-vary-depth.test
*** PASS: test_cases\q3\2-one-ghost-3level.test
*** PASS: test_cases\q3\3-one-ghost-4level.test
*** PASS: test_cases\q3\4-two-ghosts-3level.test
*** PASS: test_cases\q3\5-two-ghosts-4level.test
*** PASS: test_cases\q3\6-tied-root.test
*** PASS: test_cases\q3\7-1a-check-depth-one-ghost.test
*** PASS: test_cases\q3\7-1b-check-depth-one-ghost.test
*** PASS: test_cases\q3\7-1c-check-depth-one-ghost.test
*** PASS: test_cases\q3\7-2a-check-depth-two-ghosts.test
*** PASS: test_cases\q3\7-2b-check-depth-two-ghosts.test
*** PASS: test_cases\q3\7-2c-check-depth-two-ghosts.test
*** Running AlphaBetaAgent on smallClassic 1 time(s).
Pacman died! Score: 84
Average Score: 84.0
Scores:        84.0
Win Rate:      0/1 (0.00)
Record:        Loss
*** Finished running AlphaBetaAgent on smallClassic after 1 seconds.
*** Won 0 out of 1 games. Average score: 84.000000 ***
*** PASS: test_cases\q3\8-pacman-game.test

### Question q3: 5/5 ###


Finished at 0:23:38

Provisional grades
==================
Question q3: 5/5
------------------
Total: 5/5
```




```
(pytorch3.6) D:\pythonProject\AI\LAB2\multiagent>python pacman.py -p AlphaBetaAgent -l mediumClassic --frameTime 0
Pacman emerges victorious! Score: 1664
Average Score: 1664.0
Scores:        1664.0
Win Rate:      1/1 (1.00)
Record:        Win
```