CSE537 Artificial Intelligence

Assignment-2 Report

ReflexAgent, Minimax Agent and AlphaBetaAgent

Author: Vishal Nayak

SBU-ID: 109892702

Email:

vishal.nayak@stonybrook.edu

vnayak@cs.stonybrook.edu

Contents

1. ReflexAgent

Reflex agent scores the state in order to choose the 'action' to be taken. Higher the score, better it is.

Observations:

1) Farther the distance to active ghosts, higher the priority.
   Score is directly proportional to distance to ghosts.
2) Lesser the distance to food, higher the priority.
   Score is inversely proportional to distance to food.
3) Lesser the distance to power capsule, higher the priority.
   Score is inversely proportional to distance to power capsules.
4) Lesser the distance to scared ghosts, higher the priority.
   Score is inversely proportional to distance to scared ghosts.
5) Actions other than Directions.STOP can have slightly better priority
   Score should be decreased if the action is Directions.STOP.

I have used all the above metrics to decide on the score of the state.

Result:

python pacman.py -p ReflexAgent -l openClassic -n 10 –q
Win Rate: 10/10 (1.00)
Trace file: Traces/ReflexAgent.txt

2. MinimaxAgent

Depth of the search tree will be depth multiplied by number of agents, including pac-man. The depth corresponding to pac-man's move will be the max node and the rest are all min nodes, one for each ghost. The limit of the tree is initially set for (depth * num_of_agents). The remainder, represented by (height % num_of_agents) will provide the hint necessary to identify the behavior of nodes.

If the remainder is zero, it is a max node, min node otherwise.

The minimax values received are 9, 8, 7, -492 as mentioned in the instructions page.

Stats for each depth are as follows:

MinimaxAgent:
Depth 1: Win Rate: 612/1000 (0.61)
Depth 2: Win Rate: 693/1000 (0.69)
Depth 3: Win Rate: 368/1000 (0.37)
Depth 4: Win Rate: 655/1000 (0.66)

3. AlphaBetaAgent

The minimax algorithm implemented above is modified slightly to accommodate alpha and beta values. If alpha is less than beta, the node is explored, pruned otherwise. Implementation is straightforward once the minimax agent is working well.
Stats for each depth are as follows:

AlphaBetaAgent

Depth 1: Win Rate: 622/1000 (0.62)
Depth 2: Win Rate: 685/1000 (0.69)
Depth 3: Win Rate: 345/1000 (0.34)
Depth 4: Win Rate: 613/1000 (0.61)

4. Plot of Statistics

Observations for MinimaxAgent vs AlphaBetaAgent:

1) Expanded nodes per game is reduced if alpha beta pruning is used.
2) Winning rate remains the same even after using alpha beta pruning. This is because the result of the minimax value is unaffected by alpha beta pruning.
3) Running time per game does down since the size of the game tree is cut off due to alpha beta pruning.

Remarkable improvement is in terms of performance.

Temporarily modified pacman.py to measure the running time:

```
import timeit
start_time = timeit.default_timer()
args = readCommand( sys.argv[1:] )
runGames( **args )
running_time = timeit.default_timer() - start_time
print 'Time taken: ' + str(running_time)
```

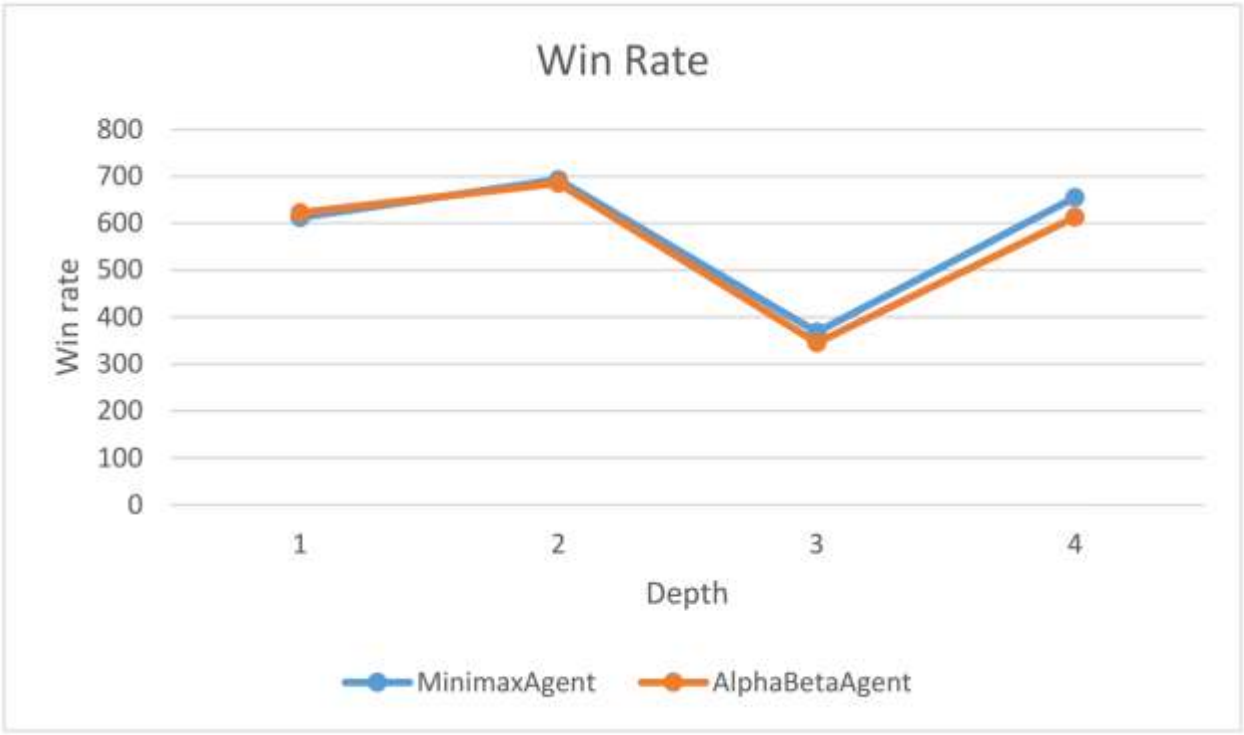
MinimaxAgent

Wins	RunningTime/Game	ExpandedNodes/Game
612	1.88	40
693	2.83	518
368	3.38	1370
655	2.28	3442

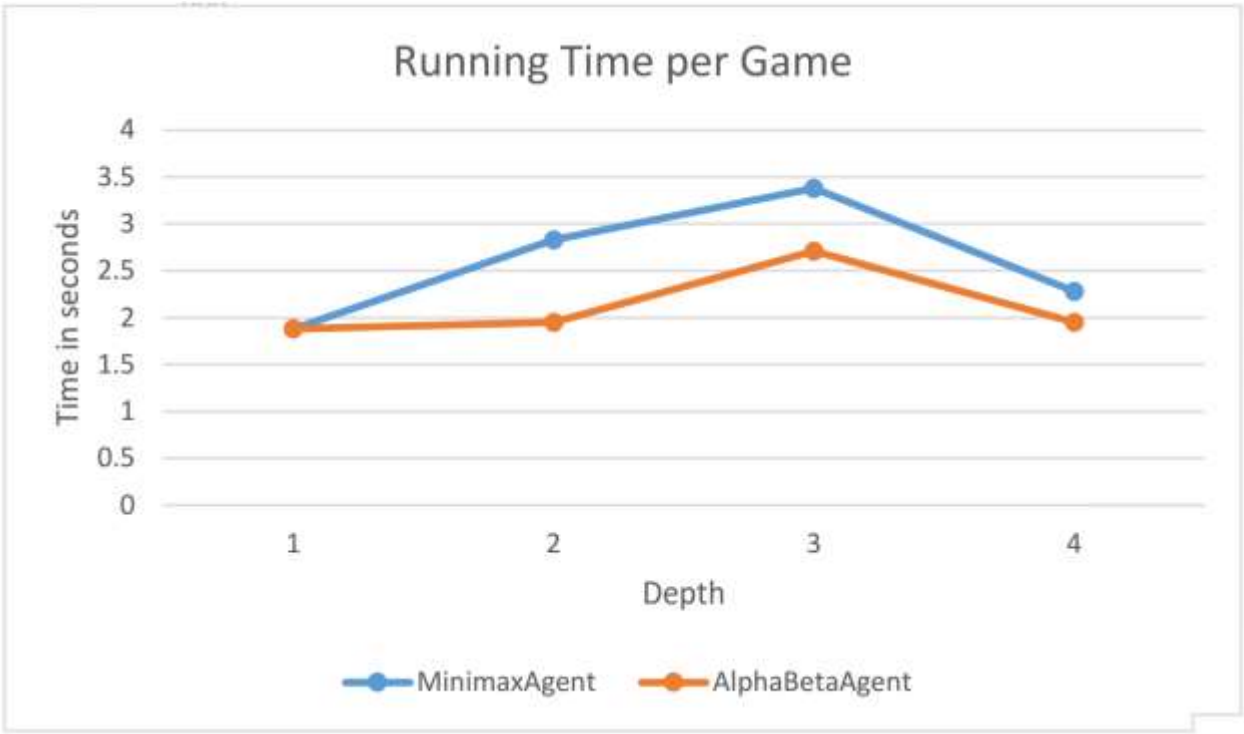
AlphaBetaAgent

Wins	RunningTime/Game	ExpandedNodes/Game
622	1.88	35
685	1.95	106
345	2.71	668
613	1.95	670

4.1. Stats for Winning Rate



4.2. Stats for Running time per Game



4.3. Stats for Expanded nodes per Game