Introduction to artificial intelligence. Part 2: Minimax agent

Benjamin DELVOYE (s154317) - Delphine MASSART (s151138)

Novembre 2018

1 Formalisation of the game

We can formalise the game as follow:

- **Initial state:**
  - initial position of pacman
  - initial position of ghost
  - initial position of the food dots within the maze
  - The first player is pacman (pacman represents MAX)
- **Function player:**
  - player(s_0) = MAX
  - player(s_i)=
    - MAX if player(s_{i-1})=MIN
    - MIN if player(s_{i-1})=MAX
- **Description of the legal actions:**
  Depending on the presence of walls or not,

$$action(s) = \{West, East, North, South\}$$

- **Transition model:**

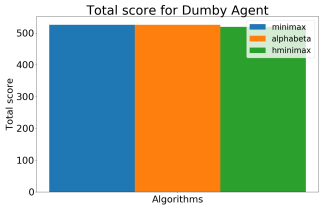
`result(pacmanPosition, ghostPosition direction, foodMap)`
`=`
`updated(pacmanPosition), updated(ghostPosition), updated(foodMap), updated(player)`

- **Terminal test:**
  - win: no more food
  - lose: pacman killed by ghost
- **Utility function:**
  The score of pacman in $s_{terminal}$

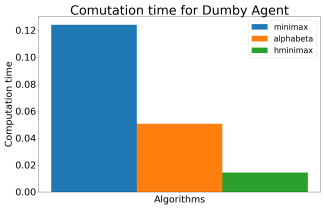
2 Comparison between agents

2.1 Dumby

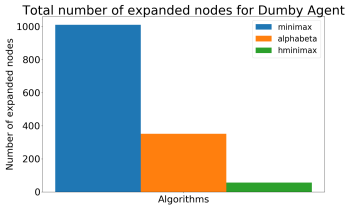
	Final score	Total computation time	Total number of expanded nodes
minimax	526.0	0.12416291236877441	1011
alphabeta	526.0	0.05069613456726074	352
hminimax	519.0	0.014436960220336914	57



(a) Final score



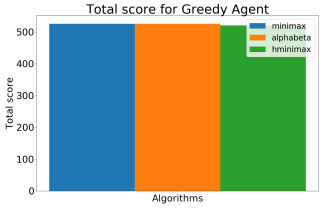
(b) Computation Time



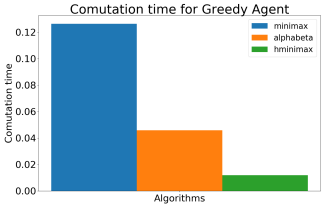
(c) Number of expanded nodes

2.2 Greedy

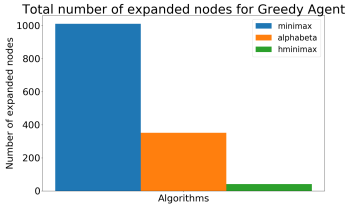
	Final score	Total computation time	Total number of expanded nodes
minimax	526.0	0.12649774551391602	1011
alphabeta	526.0	0.045945167541503906	352
hminimax	521.0	0.011970996856689453	41



(a) Final score



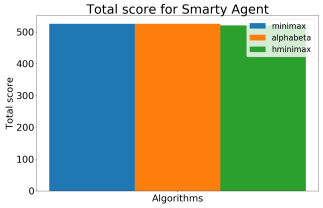
(b) Computation Time



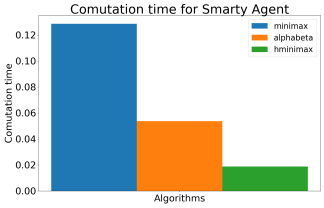
(c) Number of expanded nodes

2.3 Smarty

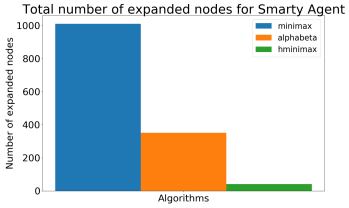
	Final score	Total computation time	Total number of expanded nodes
minimax	526.0	0.12872743606567383	1011
alphabeta	526.0	0.05370664596557617	352
hminimax	521.0	0.018778085708618164	41



(a) Final score



(b) Computation Time



(c) Number of expanded nodes

3 Performance and limitations of the agents

- **Minimax:**
  For the total score, total computation time and total number of expanded nodes, we have the same values in regards of different agents. If we try to resolve this problem on the medium and large maps, we reach a recursion depth too large for the computer. It's due to the importance of size of research space and thus, of the total number of expanded nodes.

  We decided to apply a sort of optimisation on the *minimax* algorithm while avoiding cycle. When we expand the nodes, we check if the one actually dug has already been visited. Then, if it has been visited, we evaluate the node and verify if it got a smaller score smaller than the previous one. If it's the case we stop exploring, knowing that subtree has been evaluated earlier with a better score.

- **Alphabeta:** This algorithm have a great total score with less total of expanded nodes and so, less computation time than *minimax*. In particular we observe that the number of expanded nodes, as the computation time, are reduced at least by half. This is thanks to a pruning method. However, like *minimax*, this algorithm doesn't work in medium and large maze for the same reasons.

- **Hminimax:**
  We tried some different ways to build our *eval* function and decide on our *cutoff* condition. From the begin we chose the *cutoff* test to be based solely on the depth of the recursion. We choose a depth of 3 after many tries. With this depth Pacman can see two states beyond and so, decide which way is the best.

  Our final *eval* computation is given by,

  $$eval = score + 100 * \frac{1}{foodMD + 1} + 50 * nbFood + \frac{1}{ghostMD + 1}$$

  With

  - score = Score of the current state
  - foodMD = Manathan distance between food and Pacman
  - nbFood = Number of food the which stay
  - ghostMD = Manathan distance between ghost and Pacman

  The score term gives a first rough approximation of the state but, as many state could have the same score, we need to refine our evaluation. To do so, we add the manathan distance of Pacman and minimize it. The food is the more important term in our evaluation, that is why we weight to 100. Same goes for the number eaten food, weighted to 50. To finish, we give a bonus to the states where the ghost isn't too far away from Pacman. Indeed, the manathan distance between Pacman and ghost isn't really to be maximized because sometimes it's more interesting to pass near the ghost

  It is operational for all the ghost agent on the medium maze, as for the large one excepting the *dumby* agent. Let's first discuss the working ones.

  - Medium:

	Final score	Total computation time	Total number of expanded nodes
Greedy	539.0	0.04209756851196289	111
Smarty	539.0	0.0628666877746582	108
Dumby	539.0	0.052172183990478516	141

    On the medium maze, we reach the same score for any ghost agent. But surprisingly enough the agent making us developing the more nodes is the *dumby* one. As *minimax* expect the ghost to take an optimal move at each recursion, it is probable that the dumb behaviour of the ghost leads Pacman to a move forcing him to step back later on.

– Large:

	Final score	Total computation time	Total number of expanded nodes
Greedy	560.0	0.11408615112304688	276
Smarty	560.0	0.22265291213989258	276
Dumby	$-\infty$	$+\infty$	$+\infty$

Regarding the *dumby* ghost on the large layout, we didn't find the right values to tune the parameters and allow our Pacman to cross the ghost. We though about increasing the depth of the *cutoff* test but Pacman would get stuck in a loop close to the walls, probably because of the proximity of the dot food across the wall, which the Manathan distance doesn't take into account. To tackle this problem we should have implemented a least path cost algorithm allowing Pacman to know the "real" distance to a food and thus have a more accurate evaluation of a *cutoff state*.
For the two other ones ghost agent, we reach the same score once again. We see that the greedy ones has a smaller computation time for the same number of expanded nodes.

4 Conclusion

To sum up, we see that *hminimax* isn't optimal for the small case in total score in comparison with other algorithms, but thanks to its low expanded nodes, it can develop a solution to medium and large maps. Its computation time is always under the others algorithms. In case of small search space, *alphabeta* is the best solution in term of computation time for an optimal score.