Introduction to artificial intelligence

Academic year 2018-2019

Pacman Part 2 : Minimax agent

Bolland Julien - Gilson Maxence

(s161622 - s162425)

1 Adversarial Search Problem

1.1 Initial State

First of all, the state of an agent is determined by the position of the pacman and of the ghost ,their direction, the position of the food dots as well as the pacman's score. A state can be called the initial state if no food dot has been eaten yet and if the score is null. In this second part of the project, it is necessary to know that the pacman moves before the ghost.

1.2 Player(s)

There are two different players playing one at a time. The pacman starts and then it is the ghost's turn and so on.

1.3 Description of the Agent's Actions

1.3.1 Pacman's Actions

Concerning the pacman's actions, it is able to go north, south, west and east. When the pacman goes over a food dot, it automatically eats it. Furthermore, if it encounters a wall, it won't move.

1.3.2 Ghost's Actions

As the pacman's actions, the ghost is able to go north, south, west and east. However it selects its movement depending on the chosen policy. Here are three differents policies.

- *Dumby* : The ghost rotates on itself in a counterclockwise fashion until it can go on its left.
- *Greedy* : The ghost selects the next position that is the closest to the pacman.
- *Smarty* : The ghost selects the next position which leads to the shortest path towards the pacman.

1.4 Transition Model

The pacman starts at a state s(t) and undergoes an action in order to go to its successor state which would be s(t+1). However this is true only if the undergone action is acceptable. Therefore if the pacman encounters a wall, the actions isn't acceptable and the state won't be updated. The exact same transition model is followed by the ghost agent except that its actions will depend on the policy chosen.

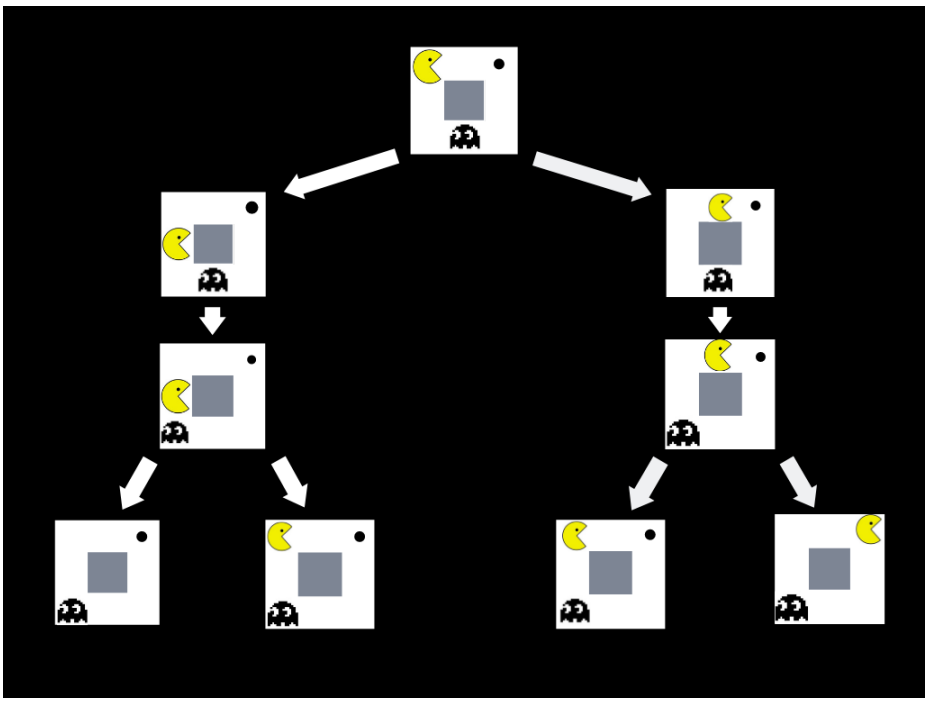
1.5 Terminal Test

We have two functions in order to know if the game is over or not : *isWin()* checks if all the food dots are eaten and *isLose()* checks if the pacman's and the ghost's position coincide.

1.6 Utility Function

The utility function is also known as the payoff. Therefore, this function sends back the final score of the pacman when he wins the game.

1.7 Game Tree



This game tree represents a ghost with the *dumby* policy. First of all, it is the pacman's turn to move. Then it is the ghost turn. However as it follows a *dumby* policy it will only go on its left. As we can see in the bottom left corner, we are in the case *isLose()*. Furthermore, in the bottom right corner, we are in the case *isWin()*. At the end of each branch, a score would be given.

2 Performances

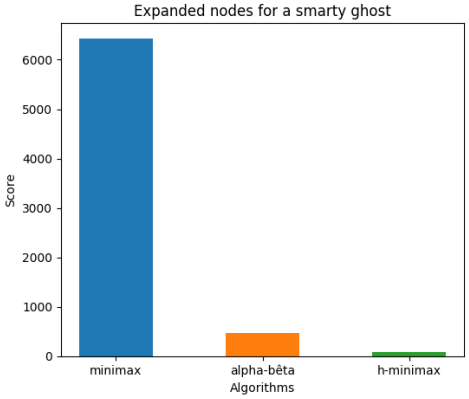


FIGURE 1 – Number of expanded nodes for a smarty ghost

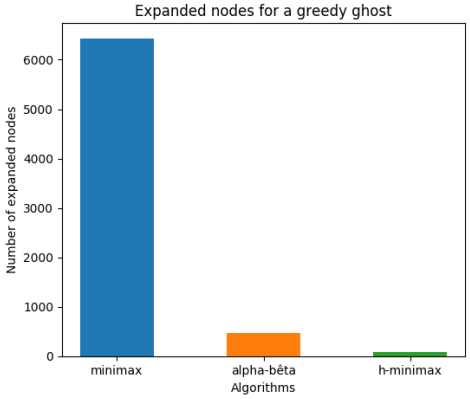


FIGURE 2 – Number of expanded nodes for a greedy ghost

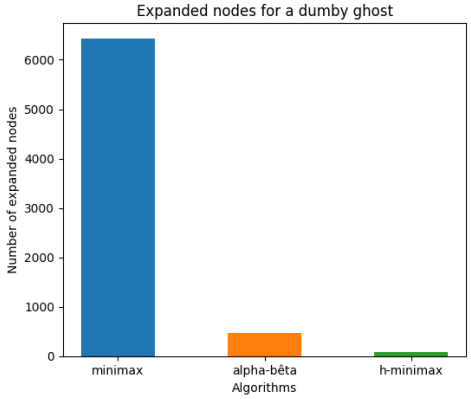


FIGURE 3 – Number of expanded nodes for a dumby ghost

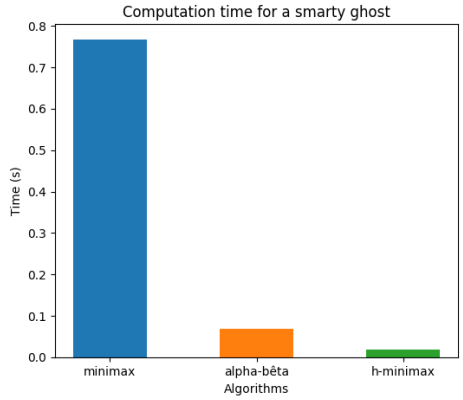


FIGURE 4 – Total computation time for a smarty ghost

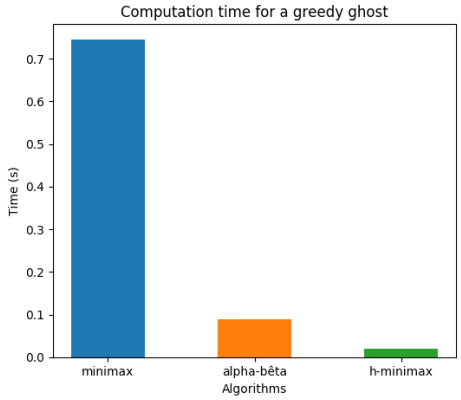


FIGURE 5 – Total computation time for a greedy ghost

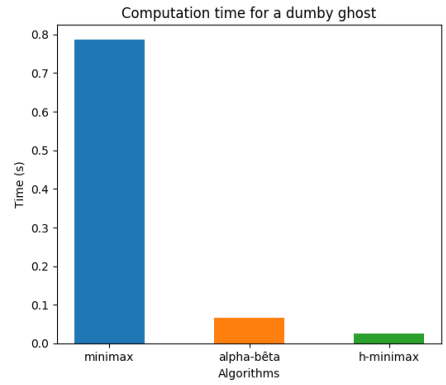


FIGURE 6 – Total computation time for a dumby ghost

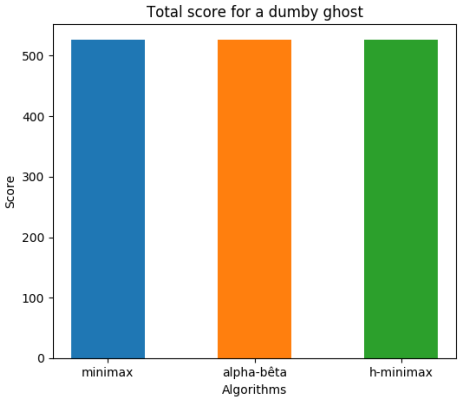


FIGURE 7 – Total score for a dumby ghost

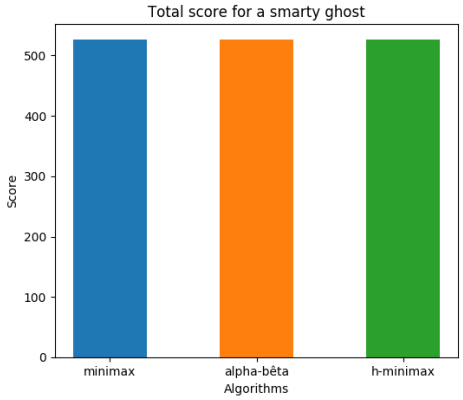


FIGURE 8 – Total score for a smarty ghost

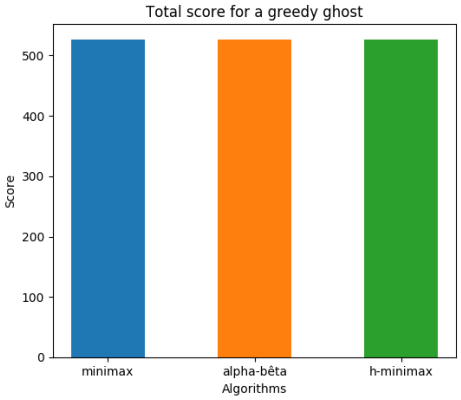


FIGURE 9 – Total score for a smarty ghost

3 Conclusion

3.1 Minimax agent

This algorithm is the least optimal in term of running time and also concerning the number of expanded nodes, the latter being very high compared to the others algorithms. Indeed, as the algorithm checks all the nodes of the tree, there is a lot of nodes we have to expand. If we try to run minimax on the medium or on the large layouts, we exceed the maximum recursion limit. Therefore we can't discuss on its performance concerning these two different layouts. For the small one, we can see on figures 4 to 6 that even if the computation time is lower than 1 second, it is still much higher than the other algorithms. This conclusion fits for the 3 ghosts agents. Finally, it always finds the best score, which is 526 for the small layout.

3.2 Minimax with alpha-beta pruning

This algorithm prunes nodes of the game tree, which allows less expanded nodes and a better computation time than the minimax without pruning. Indeed, on the small layout, instead of 6423 nodes for minimax, we obtain 479 for alphabeta pruning, for all ghosts agents. The computation time is than divided by approximatively 8. Concerning the medium and large layouts, the maximum recursion is also exceeded, therefore no conclusion can be done concerning the performances on these two layouts.

3.3 H-Minimax

This algorithm is even better than the alpha-beta pruning in terms of expanded nodes and computation time. Moreover, it can compute a solution on all the three different layouts. Concerning the small layout, it does not differ from exactly the other algorithms. On the medium layout, the behaviour of the pacman changes when the ghost agent changes. It proceeds to the same actions for the smarty and greedy agents, which returns the best score. However it does not return the optimal score and expands more nodes for a dumby ghost. For the large layout, the algorithm isn't optimal because of the irrationnality of the ghosts, especially for the dumby one.

Concerning our implementation, we chose for the cutoff function to give a maximum depth which can't be exceeded by the recursion. In our case, we chose a maximum depth of 6, because we had to make a compromise between the score and the expanded nodes for all layouts and all agents. In order to check which action will lead to a maximisation of the score, we chose to use the A-Star algorithm. Instead of returning the best path as we did in project 1, we return the distance between pacman and the nearest food dot. Then we substract it to the score the recursion returns. We use exactly the same heuristic as in project 1.