INFO8006 : Project - Part 2

Roumache Grégoire - 20162533 and Stoquart Colin - 20161140

November 22, 2018

1 Formulation of the Game as an Adversarial Search Problem

An adversarial search problem is a kind of search problem (also called game), with some special characteristics :

- There is the initial state s_0 of the game which is defined by its initial position, the positions of the walls, the food dots, the ghost position and the score which is initially set to zero.
- We need a function $player(s)$ that defines which player (either the ghost or pacman) has the move in state s. In the initial state, pacman has the right to play first. pacman and the ghost will play in turns. To implement this, we have add to each node a Boolean variable "*MaxPlayer*" that is *true* if it's pacman turn, *false* if not.
- The set of actions pacman and the ghost is allowed to perform consists in going in the directions that are not obstructed by a wall. When pacman, or the ghost, does this action, he reaches a new state. This set is always a subset of $\{North, South, East, West, Stop\}$.
- For a given action (i.e. a direction), the transition model consists of updating the position of pacman, the position of the ghost, its score and possibly the distance between dots and our agent (for our evaluation function in H-Minimax). We also update *MaxPlayer* to $\neg$*MaxPlayer*. This gives us all the characteristics of the resulting state.
- Our function terminal-test will return *true* if we are in a terminal test. We have two terminal states :
  - We can win if pacman eats all the food dots and doesn't get killed by the ghost.
  - We can lose if pacman gets killed before having eaten all the food dots.

  Whenever a terminal state is reached, it means the game is over.
  For H-Minimax, we have to replace this terminal-test with a cutoff-test. This function is the same as before except that it will also return *true* if the depth is greater than or equal to four, *i.e* if we have generated more than two successive successors for pacman and and the both ghost successors corresponding to its.
- We have to define a utility function that gives a numerical value at the end of the game. If the outcome is win, the output of the function is the score in the terminal state, if it's a loss, the output is $-\infty$. With this definition, we are not really in a zero-sum game. But we want to avoid the case where pacman commits suicide. Indeed, if there's no winning state (in the assumption of an optimal ghost), it should be preferable for pacman to commits suicide in order to avoid the score to decrease too much. There isn't this problem in the case where a winning state exists and we are well in a zero-sum game.
  Once again, we have to change this definition for H-Minimax. We replace this utility function with an evaluation function. This evaluation function returns the current score of the state minus the minimum of the distance between pacman and all dots except in a losing end where it returns $-\infty$. Note that this evaluation function is the same as the utility in a terminal state (seeing that distance to food is equal to zero). So the order of all this state is preserved.

In conclusion, we can describe the game tree as follow, the nodes are the game states with the initial state being the root of the tree. The actions the agents can perform are the edges connecting the nodes.

2 Results

In figure 1, 2 and 3 (at the end of the report), we present the results of our algorithms depending on the kind of ghost it faces.

3 Discussion

As we look at the graphs, we notice that on Fig 1, the final score of H-Minimax is superior to the other agents. H-Minimax also achieve its task (winning the game) with less resources, i.e. by using less computation time (Fig 2) and expanding a smaller amount of nodes (Fig 3). The second best algorithm is Minimax with alpha-beta pruning. This leaves us with the worst algorithm of them all, Minimax.
But this actually make sense. We had to first program Minimax which solves the problem then we implemented Minimax with alpha-beta pruning to make it faster and expand less nodes. Finally, we made H-Minimax that wins the game even faster because it doesn't actually explore the entire game-tree. However, this last solution is not optimal and it could lose whereas it was possible to win.

In order to know the value that H-Minimax has to associate with a given state (once it has reached the arbitrary depth limit in the exploration of the game-tree), we had to design a heuristic. The heuristic we chose is the current score minus the Manhattan-distance between the closest food and pacman. Well, actually the utility-function works like this :

1. When the maximum depth of the exploration of the game-tree is reached, the algorithm can't make recursive call anymore.
2. The algorithm then returns : utility = score - distance.

We can see two obvious improvements to make on H-Minimax :

1. Chose more carefully the *depth* at which the algorithm stops exploring the game-tree. We didn't take it randomly without any consideration but we didn't explored all possibilities either. There probably is an optimal depth to chose that can balance the advantages (fast algorithm) and disadvantages (not 100% optimal) of this method.
2. Chose a better *heuristic* function. Again, we considered several different heuristics and chose the one that looked optimal. But is it really optimal ? There may be a better heuristic that achieves better results than the one we chose.
   For example, we try to add the distance between pacman and the ghost and we try different weight before each term of the sum but without significant success.

From small_adv, we can't conclude any significant different between the ghost agents. But, what is interesting is to compare the score of alphabeta in medium_adv. In this case, the score of pacman is 517 versus "greedy" and "smarty" ghost and it makes an infinite part versus "dumby". Here, we clearly see that alphabeta has a good behaviour versus optimal (or sub-optimal) opponents. This is because alphabeta assumed to play versus an optimal opponents, *i.e* it will consider the "worst case" and be very careful. So pacman will never get killed but will take a very long time (in this case, an infinite time) to finish the game if the ghost is not optimal.
Of course, this reasoning could also be applied to Minimax seeing that it choose the action to perform in the same way. However, because Minimax doesn't make it in an efficient way, it takes too times and we can't directly observed it.

An other observation that we could make is when we compare the behaviour of alphabeta and H-Minimax in large_adv. The first one will get killed and the second will win. This feel us rather strange and we realise that, the reason why alphabeta get killed, is that it doesn't find any winning terminal state. It means that, in the way we have implemented our optimal ghost, the ghost will always kill pacman. In other word : "in the worst case, it's not possible to win". So the only thing that alphabeta tries to do is to go farther from the ghost and finally get killed because it has reached an impasse.
In the other hand, H-Minimax only try to maximise its score and to reduce the distance between all dots.

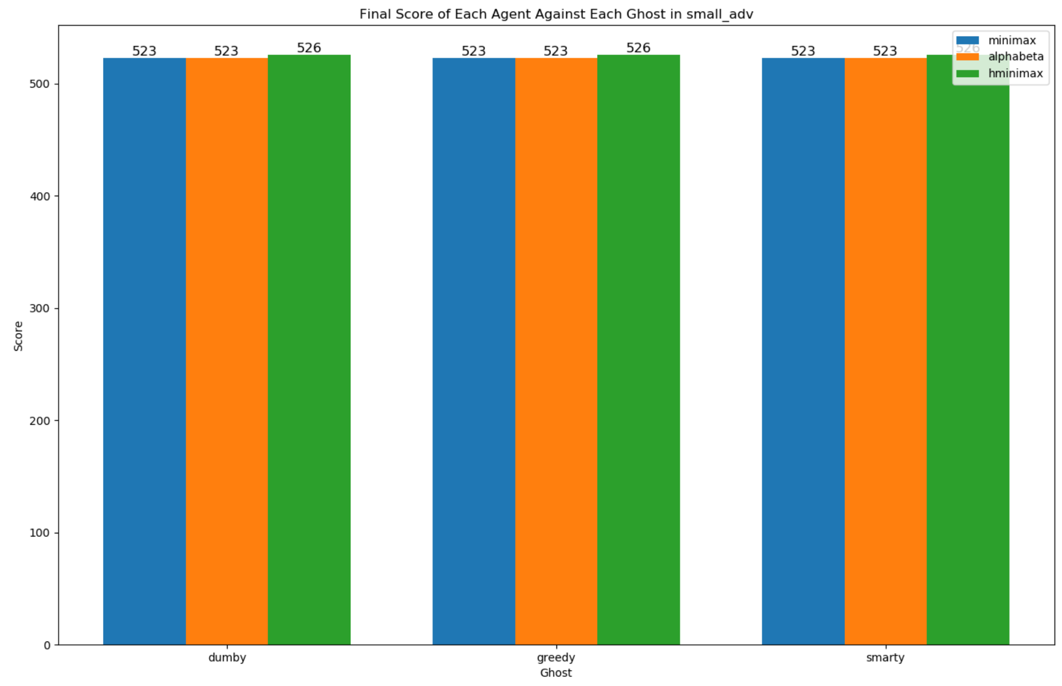

Figure 1: Score of different algorithm against the different ghosts

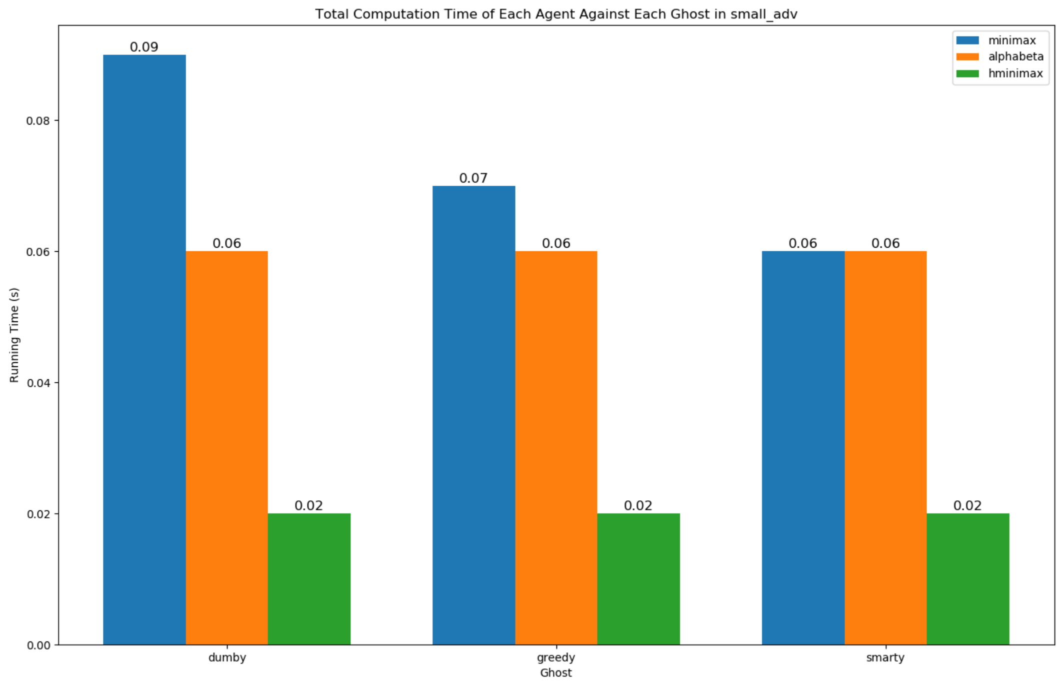

Figure 2: Computation time of different algorithm against the different ghosts

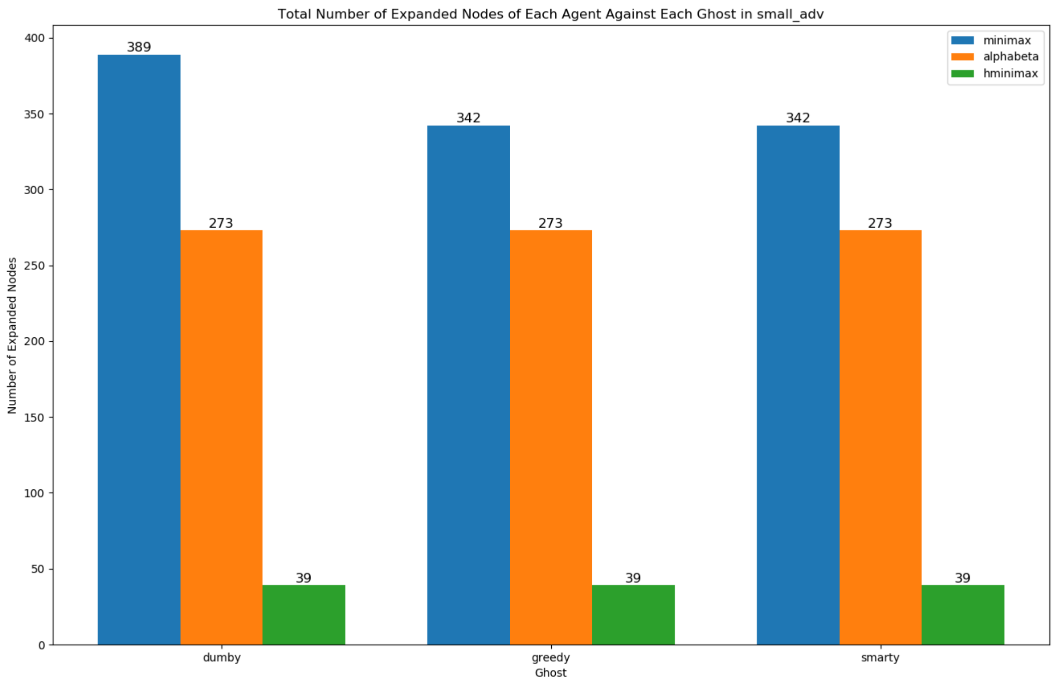

Figure 3: Number of node expanded for different algorithm against the different ghosts