INFO8006: Project 2 - Report

Mohamed-Anass Gallass - s174796
Yassine Maziane - s184384

November 8, 2020

1 Problem statement

a. The **initial state** consists in the Pacman position (x,y), the Ghost position and a boolean matrix representing the positions of the N dots in the map.

$$s = (P_{position}, G_{position}, F_N)$$

As initial, we consider that no dot has been eaten yet with the Pacman and the Ghost positions at their starting state:

$$s_0 = (P^0_{position}, G^0_{position}, F^0_N)$$

The **player function** is defined like this:

$$\text{player}(s) = \begin{cases} 0 & \text{if it's the Pacman's turn to make a move} \\ 1 & \text{if it's the Ghost's turn to make a move} \end{cases}$$

We can indeed consider the game as a round by round game where the Ghost plays right after the Pacman. The first player to play is Pacman.

The **legal actions** are defined by 4 potential actions: North, South, East, West. The legality of each action must respect the border of the map.

The **transition model:** gives the new state s' reached after the player in state s has done a legal action a: $s' = result(s, a)$

The **terminal test** indicates if all the dots were eaten by Pacman or Pacman was eaten by Ghost.

The **utility function:** gives the score at terminal state:

$$\text{utility}(s, player) = \begin{cases} \text{Score} & \text{if p = Pacman = 0} \\ -\text{Score} & \text{if p = Ghost = 1} \end{cases}$$

b. We have 2 players: Pacman and Ghost. Pacman wants to maximize its utility function utility$(s, p = 0)$ and seeks to maximize its final score by eating dots as quickly and efficiently as possible. Ghost wants to maximize its utility function utility$(s, p = 1)$ and tries to minimize the final score by eating Pacman. We can translate this as:

$$\begin{cases} \text{utility}(s, p = 0) = \text{Score} \\ \text{utility}(s, p = 1) = -\text{Score} \end{cases}$$

and

$$\text{utility}(s, p = 0) + \text{utility}(s, p = 1) = 0$$

2 Implementation

a. Completeness is not guaranteed because Pacman and Ghost may go through a previous state more than once. So there can be cycles and the tree is not finite.
To guarantee completeness, cycles must be avoided, i.e. previous states can no longer be revisited. If we do this, the tree will be finite.

b. ***Leave empty.***

c. ***Leave empty.***

d. We decide to definie our cutoffTest as follow:

$$\text{cutoffTest}(state, depth) = \begin{cases} True & \text{, if depth} = 4 \text{ or state is a terminal state} \\ False & \text{, else} \end{cases}$$

Hminimax0 :
Intuitively, we want to make Pacman as close as possible to its closest food dot and as far as possible from Ghost. We have therefore chosen to implement this evaluation function:

$$\text{eval0(state)} = Score + Dist_{P-F} + Dist_{P-G}$$

with:

- $Dist_{P-F}$ = the Manhattan Distance between Pacman and the nearest food dot.
- and $Dist_{P-G}$ = the Manhattan Distance between Pacman and Ghost.

Hminimax1 :
For our second Hminimax, we are looking for a new path such that Pacman goes through the closest food, then through the food closest to the one he just ate and so on. Each of these small distances is then added in the variable $fullPathFodd$ and our new evaluation function eval1($state$) becomes:

$$\text{eval1(state)} = Score + Dist_{P-F} + Dist_{P-G} - fullPathFood$$

Hminimax2 :
For Hminimax2, we keep the same evaluation function as for Hminimax1 but we consider a greater depth of 5 in the cutoffTest. So we have :

$$\text{cutoffTest}(state, depth) = \begin{cases} True & \text{, if depth} = 5 \text{ or state is a terminal state} \\ False & \text{, else} \end{cases}$$

$$\text{eval2(state)} = Score + Dist_{P-F} + Dist_{P-G} - fullPathFood$$

3 Experiment

a. The bar plots in terms of score, time performances and number of expanded nodes can be found below:

b. The summary of the results obtained (for the large layout) can be found below:

Experimentally, we notice Pacman wins against all ghost types, whatever the layout. It also seems that the Hminimax1 algorithm is the most efficient and optimal. Indeed, it returns the same result as the Hminimax2 in with a lower compilation time and it returns better scores that Hminimax0 with less nodes exploration.

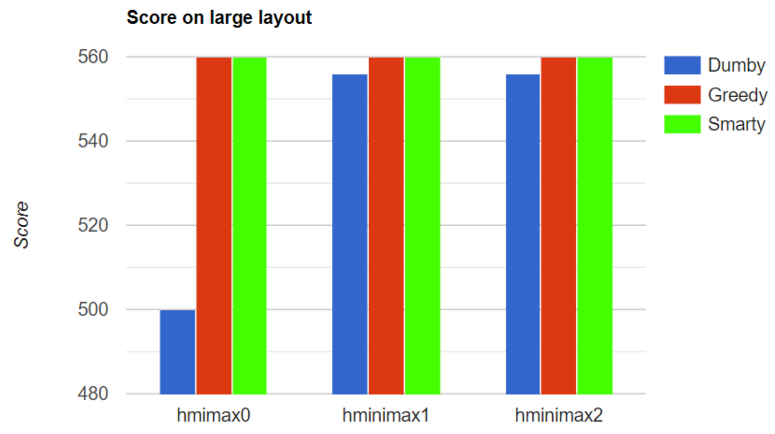

Figure 1: Hminimax : score

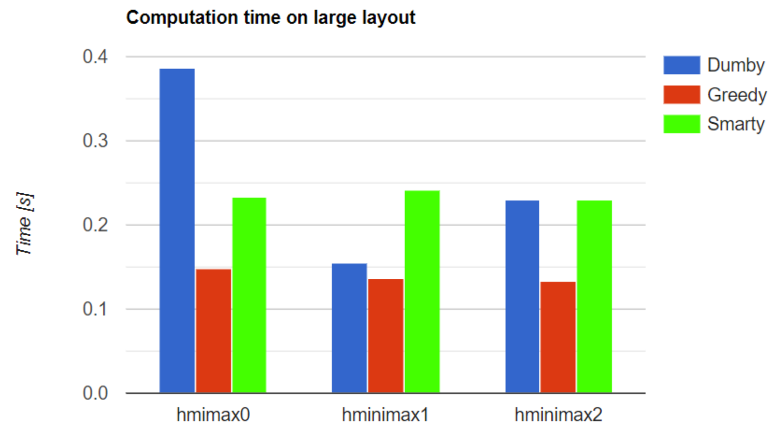

Figure 2: Hminimax : time performances

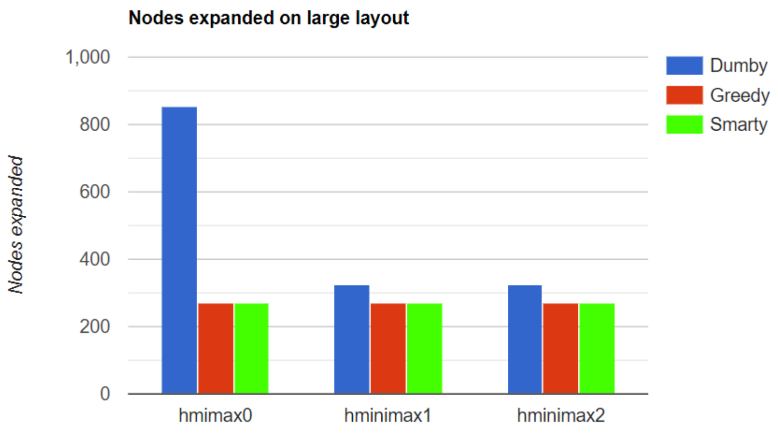

Figure 3: Hminimax : expanded nodes

		Score	Time (sec.)	Node
Hminimax 0	dumby	500	0.386	855
	greedy	560	0.149	270
	smarty	560	0.233	270
Hminimax1	dumby	556	0.155	325
	greedy	560	0.136	270
	smarty	560	0.242	270
Hminimax2	dumby	556	0.230	325
	greedy	560	0.133	270
	smarty	560	0.230	270

Figure 4: summary of the results of cutoff-test/evaluation function pairs, according to the type of ghosts.

c. In order to manage multiple Ghosts, each agent must have its own utility function, i.e. each agent tries to maximize its utility function independently of the utility functions of others. So Hminimax should return a tupple of all those values.

   The game can still be described as a zero-sum game if, for n Ghosts, we define utility functions as follows:

$$\text{utility}(s, player) = \begin{cases} \text{Score} & \text{if player = Pacman} \\ -\frac{\text{Score}}{n} & \text{if player = Ghost} \end{cases}$$

   with

$$\text{utility}(s, Pacman) + \sum_{i=1}^{n} \text{utility}(s, Ghost_i) = 0$$