INFO8006: Project 2 - Report

Aldeghi Florian - s183157
Baguette Brice - s181482

September 28, 2023

1 Problem statement

a. **-Initial state** : Un état de jeu est une matrice reprenant la position de pacman, celle du fantôme, celle des murs ainsi que celle des nourritures. $s \in \{0, 1, 2, 3, 4\}^{m*n}$ ou m*n sont les dimensions de la matrice et les 5 chiffres représentent respectivement un mur, le vide, une nourriture, pacman et un fantôme. L'état initial est donc la matrice avec toutes les nourritures présentes à leur place, et pacman et le fantôme à leur place de départ.Le compteur poour savoir a qui c'est de jouer est initialement à 0.

-Player : Le player est un compteur qui s'incrémente à chaque déplacement, que ce soit pacman ou le fantôme. Quand il vaut 0 au début, c'est à pacman de jouer.$p \in \Re$, si $p\%2 = 0$, c'est à pacman sinon, c'est au fantôme.

-Action : Une action est un mouvement légal du fantôme ou de pacman (pas de déplacement dans un mur et en plus pour le fantôme, pas de demi tour). Après une action, l'état est modifié. $a \in \{North, South, East, West\}$

-Transition model :Un modèle de transition est un état s' qui est la matrice s modifiée après une action a sur un état s, $result(s, a) = s'$

-Terminal test :Le test de fin est un test qui vise à voir si la partie est terminée. Ici, c'est dans le cas ou le fantôme mange pacman ou que toutes les nourriture sont mangées. $terminal(s) = 1$ s'il n'y a plus aucune case '2' ou '3' ou '4' dans la matrice d'état s, sinon 0.

-Utility function: La fonction d'utilité est une fonction qui associe un score à un état de fin. Ici, on donne 500 si toutes les nourritures sont mangée, -500 si pacman est mangé, -1 par déplacement de pacman ce qui correspond à -p/2 arrondis au dessus ainsi que 5 points par nourriture mangée, ce qui correspond au nombre de 2 initial - le nombre de 2 final dans la matrice s, le tout multiplié par 5. Nous avons ainsi un score de fin.

b. Pour rendre le jeu comme un 'zero-sum game', on donnerait la valeur donné par utility(s) au joueur et la valeur opposée au fantôme, ainsi le but de chacun est de maximiser son score et le total des deux scores vaut toujours zéro (e.g. si le score de pacman est 10, celui du fantôme est -10).

2 Implementation

a. la 'completeness' de la fonction minimax est assurée à partir du moment où l'arbre qu'elle crée pendant le calcul est fini. Si l'arbre est infini, la fonction qui est récursive ne se fini jamais. Grâce à notre variable 'closed' qui retient les états dans lequel on est déjà passé pour ne pas s'y retrouver une seconde fois, ça empêche la création de boucle et donc assure que l'arbre soit fini.
La fonction minimax n'est donc pas complète dans le cadre du jeu pacman de base mais cette fonction peut devenir complète si on ajoute une variable qui retient les états et empêche les boucles.

b. ***Leave empty.***

c. ***Leave empty.***

d. **-Hminimax0**: La fonction cut-off regarde si soit on est dans un état de fin soit le compteur p vaut 4, cela veut dire qu'on regarde maximum 4 coups à l'avance ce qu'il se passerait.
La fonction évaluation ou 'évaluation' associe un score fictif à un état. Ce score vaut + ou - 500 en fonction de si l'état est une victoire (plus de nourriture) ou une défaite (mangé par le fantôme) pour le pousser à ne jamais perdre, on y ajoute la distance de manhattan entre le fantôme et pacman pour favoriser pacman à être loin du fantôme et éviter de perdre, on retire 50 fois le nombre de nourriture restante car c'est l'objectif principal du jeu et il faut donc favoriser pacman à manger des nourritures avant tout et enfin on retire 4 fois la distance de manhattan entre pacman et la nourriture la plus proche pour favoriser pacman à s'approcher d'une nourriture. Ce dernier facteur est bien inférieur à celui du nombre de nourriture restante car lorsque pacman mange une nourriture, la distance par rapport au plus proche va fortement augmenter et lui faire perdre des points, il faut donc que l'obtention de la nourriture soit supérieur au nombre de point qu'il va perdre avec la nouvelle distance.

-Hminimax1:La fonction cut-off regarde si soit on est dans un état de fin soit le compteur p vaut 6, cela veut dire qu'on regarde maximum 6 coups à l'avance ce qu'il se passerait.
la fonction évaluation ou 'évaluation' associe un score fictif à un état. Ce score est le même que pour hminimax0. La différence est donc le cut-off. Ce deuxième algorithme est moins performant que le premier car comme il va anticiper plusieurs coup à l'avance, il osera moins s'approcher des nourriture en pensant que le fantôme risque de le manger.

-Hminimax2:La fonction cut-off regarde si soit on est dans un état de fin soit le compteur p vaut 8, cela veut dire qu'on regarde maximum 8 coups à l'avance ce qu'il se passerait.
la fonction évaluation ou 'évaluation' associe un score fictif à un état. Ce score est au début identique à la fonction hminimax0 mais on y retire 2 fois la distance de manhattan entre pacman et la nourriture la plus éloignée car ça montre une distance qu'on sera au moins obligé de parcourir et donc des points perdus.

3 Experiment

a.

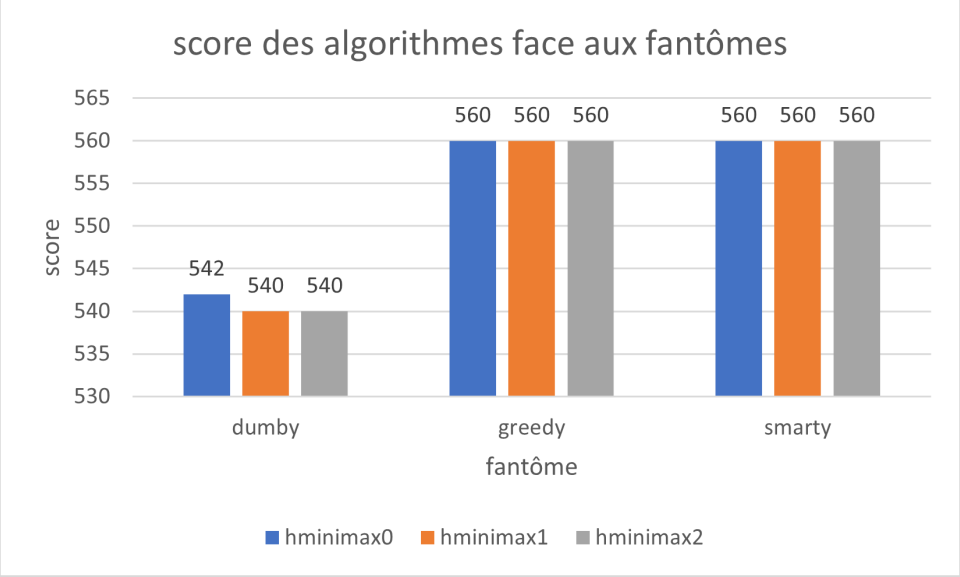

Figure 1: score

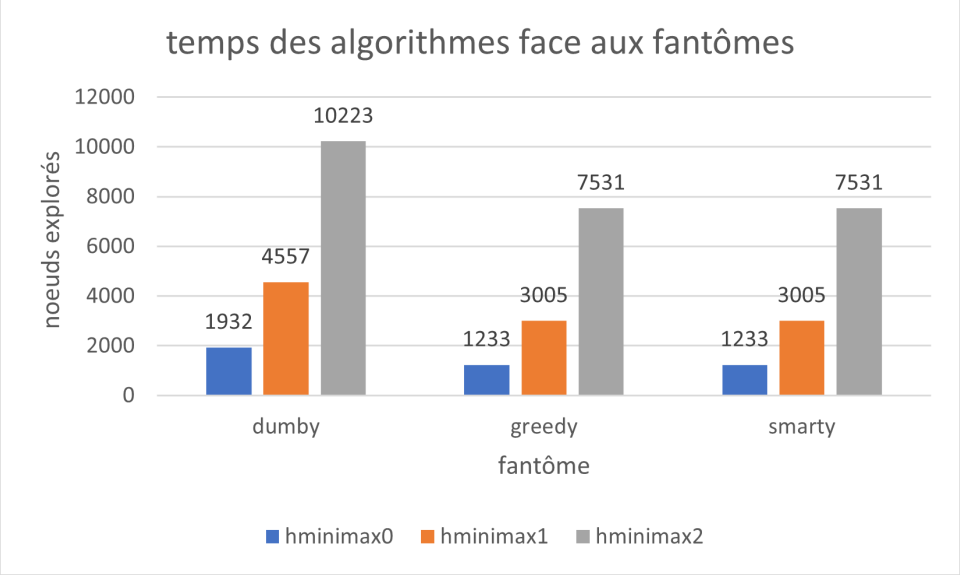

Figure 2: noeuds

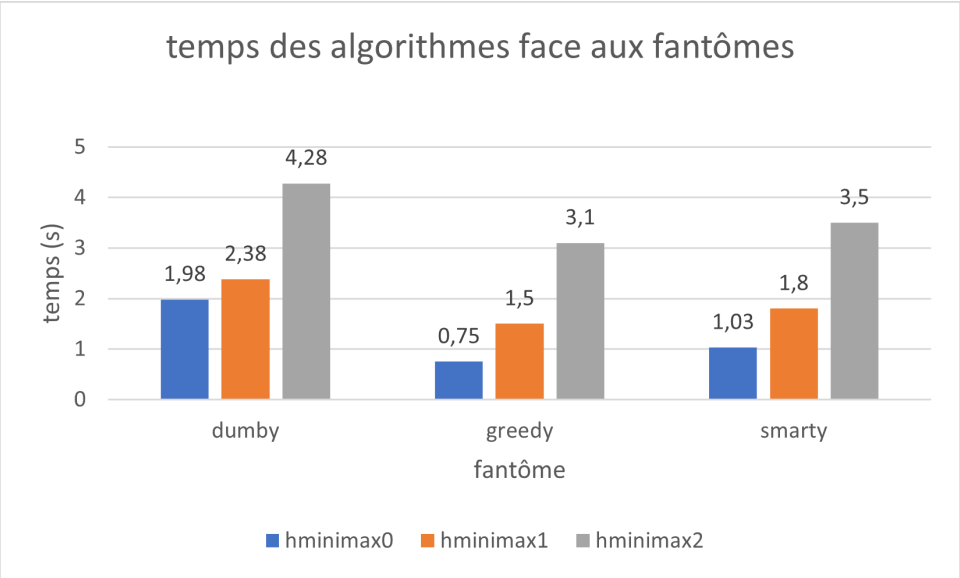

Figure 3: temps

b. Logiquement, les résultats de hminimax0 sont meilleurs que hminimax 1 qui est meilleur que hminimax2. Pour chacune des pairs 'cut-off/evaluation' on peut voir que le score par rapport au 'dumby' est inférieur à celui par rapport au 'smarty' et 'greedy'. C'est expliqué par le fait que la trajectoire du fantôme 'greedy' et 'smarty' est une trajectoire "plus intelligente" du fantôme car il cherche à nous manger et donc à minimiser la foction évaluation. Comme on prend en compte dans nos calculs que chacun (pacman et fantôme) joue son meilleur coup possible, Le recalcule à chaque étape du coup suivant suit une suite logique car le fantôme joue plus ou moins comme on l'avait prédit. En revanche, pour le 'dumby' il ne fait que tourner en rond sans essayer de nous manger, ce qui veut dire que la trajectoire qu'on envisage qu'il prenne n'est pas du tout la bonne et ainsi à chaque étape, lors du recalcul, nos résultats sont différents et on perd plus de point car on envisageait pas le bon chemin.
Les scores et nombres de noeuds visités face au 'greedy' et 'smarty' pour un algorithme sont identique car la façon de se déplacer du fantôme est chaque fois la même.
Le temps de calcul est plus élevé sont par ordre croissant hminimax0 puis hminimax1 puis hminimax2 et par rapport au fantôme, 'greedy' puis 'smarty' puis 'dumby'. En effet, vu que le 'cut-off' du hminimax2 se fait quand le compteur est à 8, hminimax1 quand il est à 6 et hminimax0 quan il est à 4, il y a de moins en moins de noeuds visités et donc un plus petit temps de calcul. Le temps face au 'smart' est légérement supérieur à celui face au 'greedy' juste à cause du temps de calcul du mouvement que va faire le fantôme. Le temps face au 'dumby' est plus grand que les deux autres car le nombre de noeuds visités est plus grand et c'est du au même fait que le fantôme ne bouge pas de façon intelligente et qu'on doit donc recalculer à chaque fois des coups qu'on avait pas prédit.

c. On peut très bien mettre plusieurs fantôme si on les fait se déplacer chacun leur tour toujours dans le même ordre avec pour chacun d'eux l'objectif de minimiser le score. On aurait ainsi une fois l'appelle de max puis n fois l'appelle de min s'il y a n fantômes. Pour les fonctions hminimax, il faudrait prendre en compte tous les fantôme dans la fonction évaluation. On peut toujours voir ça comme un 'zero-sum game' si on divise le score des chaque fantôme par le nombre de fantôme qu'il y a (e.g. avec 3 fantômes, si le score de pacman est 10, celui de chaque fantôme est -10/3).