

**Projet Intelligence Artificielle**

**Snake Game**



Maata Mohammed Taha Année universitaire : Oucouc Hafid 2021/20221

# Sommaire

[**Sommaire**](https://docs.google.com/document/d/1zfDFhaJ-9uxeAwfMC1XhL3RHd3IBoumcj61qnNcI-RY/edit#heading=h.kq9eq3oywfk)

[**I)**](https://docs.google.com/document/d/1zfDFhaJ-9uxeAwfMC1XhL3RHd3IBoumcj61qnNcI-RY/edit#heading=h.aw2175rmxd3a) **Présentation du projet**

**II)Représentation du code**

-Partie jeu / interface

-Partie AI.

[**III) Description**](https://docs.google.com/document/d/1zfDFhaJ-9uxeAwfMC1XhL3RHd3IBoumcj61qnNcI-RY/edit#heading=h.y3kq84u96xc) **des méthodes de recherche utilisée**

-Methode A\*

-Methode BFS

-Methode DFS

[**IV)**](https://docs.google.com/document/d/1zfDFhaJ-9uxeAwfMC1XhL3RHd3IBoumcj61qnNcI-RY/edit#heading=h.7q8exqbx7ldm) **Comparaison des méthodes utilisées**

- Les résultats des testes.

-Comparaison entre les tests d’utilisation.

[**V)**](https://docs.google.com/document/d/1zfDFhaJ-9uxeAwfMC1XhL3RHd3IBoumcj61qnNcI-RY/edit#heading=h.7q8exqbx7ldm) **Limite**

[**VI)**](https://docs.google.com/document/d/1zfDFhaJ-9uxeAwfMC1XhL3RHd3IBoumcj61qnNcI-RY/edit#heading=h.7q8exqbx7ldm) **Conclusion**

# **I) Présentation du projet :**

Pour notre sujet de projet d’intelligence artificielle, on a choisi de travailler sur le jeu Snake. Snake est un jeu très connu car il a été implémenté dans les premiers téléphones portables, le but du jeu est de manger plusieurs pommes sans se faire tuer, sachant que chaque fois qu’on mange une pomme, la taille du serpent augmente. Généralement, Snake se joue par un utilisateur mais pour notre projet on a mis en place des intelligences artificielles se basant sur plusieurs algorithmes tels que A\*, BFS, DFS.

Après avoir implémenter les différentes méthodes de recherche, on a effectué plusieurs tests sur notre code pour voir quelle méthode de recherche est la plus efficace, en faisant des testes pour comparer le score et le temps fourni par le snake afin de manger la pomme chaque fois.

Dans notre code on a mis en place une fenêtre fermée qui fait que les mouvements du serpent sont très limités ainsi que s’il se on voulait savoir laquelle des méthodes de recherche vont aider le snake à survivre le plus longtemps possible .

On va commencer par une petite description générale du code utilisé avant de voir les autres aspects du code .

# **II) Présentation du code :**

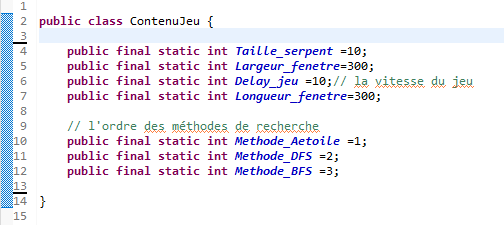
Le code utilisé dans ce projet est séparé en deux grandes parties, dans la première partie, on va faire une présentation du code qui gère le jeu, l’interface ainsi le snake. Puis dans un second temps, on s’attardera sur la partie du code dédiée à programmer les différentes méthodes de recherche.

* **Partie jeu/interface :**

On trouve dans le code 3 classes qui gèrent cette partie mais on va en voir qu’une seule, la classe ContenuJeu, le rôle de cette classe est de définir des paramètres en leur mettant les valeurs qu’on désire comme la taille du serpent,la taille de la fenêtre et la vitesse du jeu .

Le changement de ces paramètres change le mouvement du serpent et donc le score final.

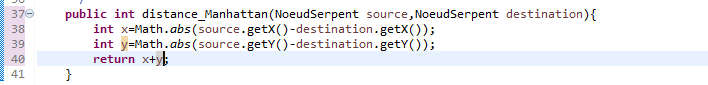
Exemple :



* **Partie intelligence artificielle :**
* **L’algorithme A\* :**

Le code utilisé pour programmer cet algorithme de recherche est séparé en plusieurs parties :

- Première partie, on va créer une méthode pour calculer la distance de manhattan entre le nœud où se trouve la tête du serpent et celui où se trouve la nourriture.



Chaque itération de l’algorithme A\* continue de chercher un chemin vers la nourriture, la différence entre A\* et BFS et DFS c’est que A\* continue de chercher un chemin optimal vers la nourriture car chaque nœud à une valeur donc l’algorithme compare les nœuds et il cherche toujours à choisir le nœud avec la valeur la plus proche de la nourriture, tout en prenant en considération le parcours qu’il a déjà parcouru. Il faut faire attention aussi à voir si le nœud est libre sinon le serpent va se croiser avec lui-même .

**-DFS :**

Pour le code utilisé pour DFS on n’utilise pas la distance de manhattan mais ce qu’on fait c’est qu’on prend un noeud ou on commence puis on fait une boucle pour parcourir chaque noeud et ce verticalement jusqu'à ce qu’il atteigne la nourriture il faut aussi prendre en considération si le noeud est libre ou pas pour que le serpent évite de se tuer par lui-même. Le DFS prend un très long chemin car il traverse tous les nœuds dans la même ligne puis il voit s' il peut continuer sinon il change de direction .

**-BFS :**

Pour l’algorithme BFS il est pas si différent de DFS, la différence est que BFS quand il parcours les noeuds il compare les valeurs de la distance des noeuds par rapport au noeud de destination, donc il continue pas toujours dans la même ligne quand il trouve un noeud voisin qui est plus proche de la nourriture il le prend et il continue comme ça jusqu'à ce qu’il trouve le chemin vers la nourriture. Au moment o il rencontre un nœud qui n'est pas libre, il essaye de voir l’autre nœud voisin et puis il teste si c’est le bon chemin vers la nourriture .

**-Mouvement Aléatoire :**

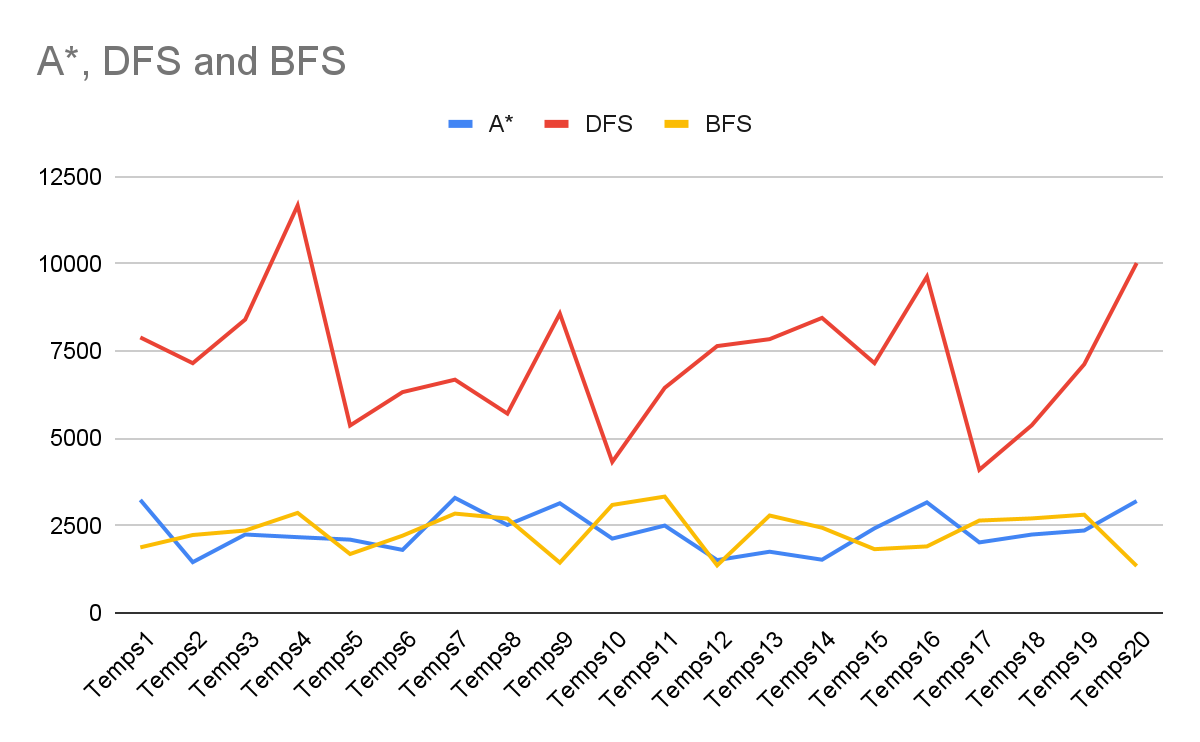
Si le snake ne trouve aucun chemin vers la nourriture ou tous les nœuds voisins sont pleins donc le snake va faire un mouvement aléatoire si il se croise avec lui même l’utilisateur va recevoir le message “partie terminée".

**-Methode nodeProcess :**

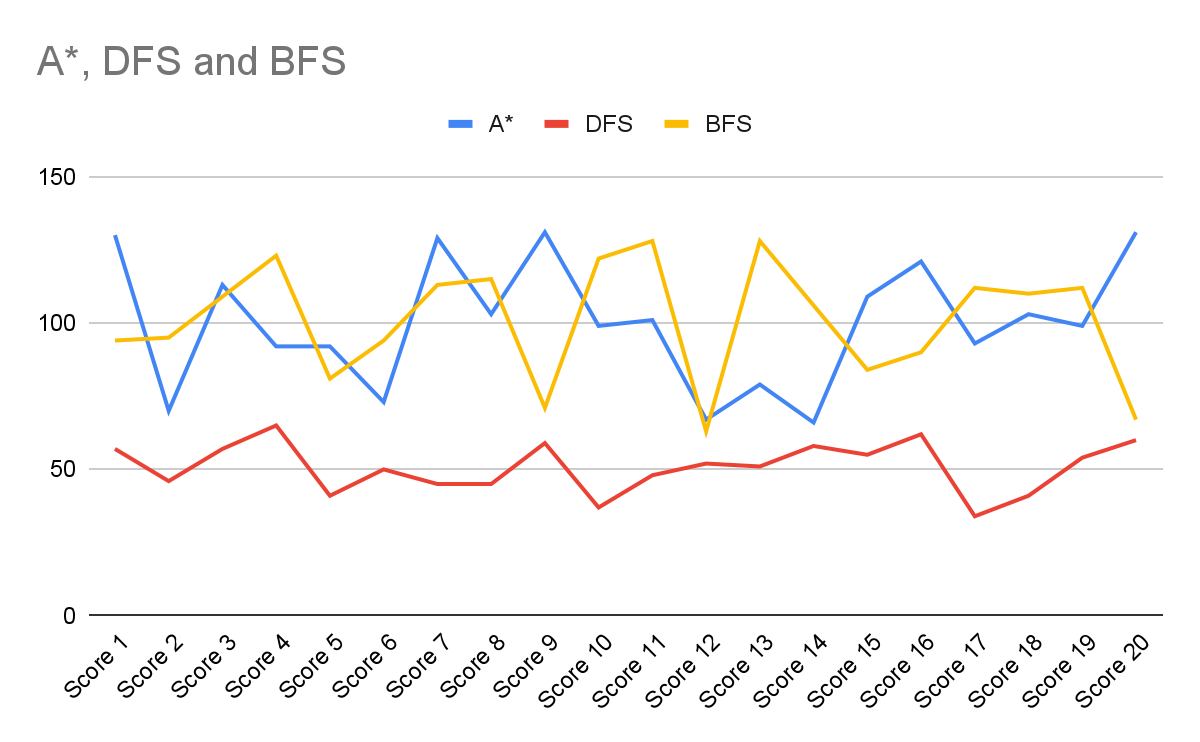
Pour voir si le nœud existe ou pas dans la fenêtre, on a utilisé la méthode nodeProcess où on fait une importante condition, celle que le serpent ne peut jamais sortir de la fenêtre.

[**IV)**](https://docs.google.com/document/d/1zfDFhaJ-9uxeAwfMC1XhL3RHd3IBoumcj61qnNcI-RY/edit#heading=h.7q8exqbx7ldm) **Comparaison des méthodes utilisées :**

On a effectué pour chacune des différentes méthodes 20 simulations, on a choisi d’en faire 20 afin de donner plus de crédibilité à notre analyse qui va s’en suivre.  
Les résultats des simulations sont donnés dans les 2 graphiques ci-dessous :



Ce graphique montre le temps qu’a pris chacune des simulations utilisant les différents algorithmes (le temps entre le début de la simulation et la mort du serpent)

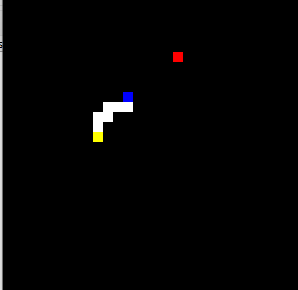
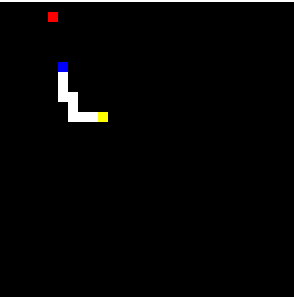


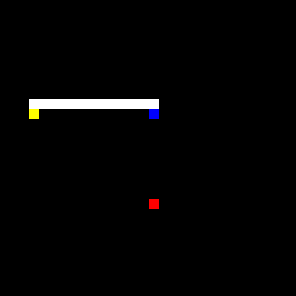
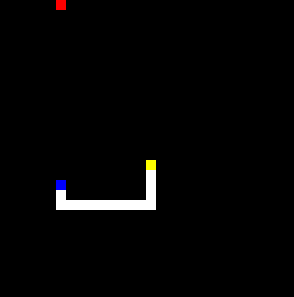
Ce graphique nous montre le score obtenu dans chacune des simulation utilisant les différentes heuristiques (le nombre de pommes mangées)

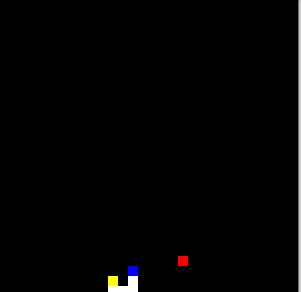
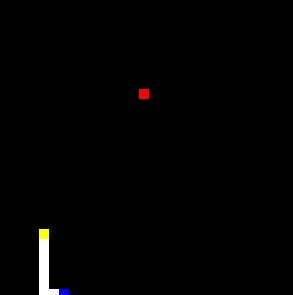
Suite aux différentes simulations qu’on a effectué, on a regroupé tous les résultats afin d’avoir une vision plus claire pour mieux comparer l’efficacité des méthodes de recherche utilisées :

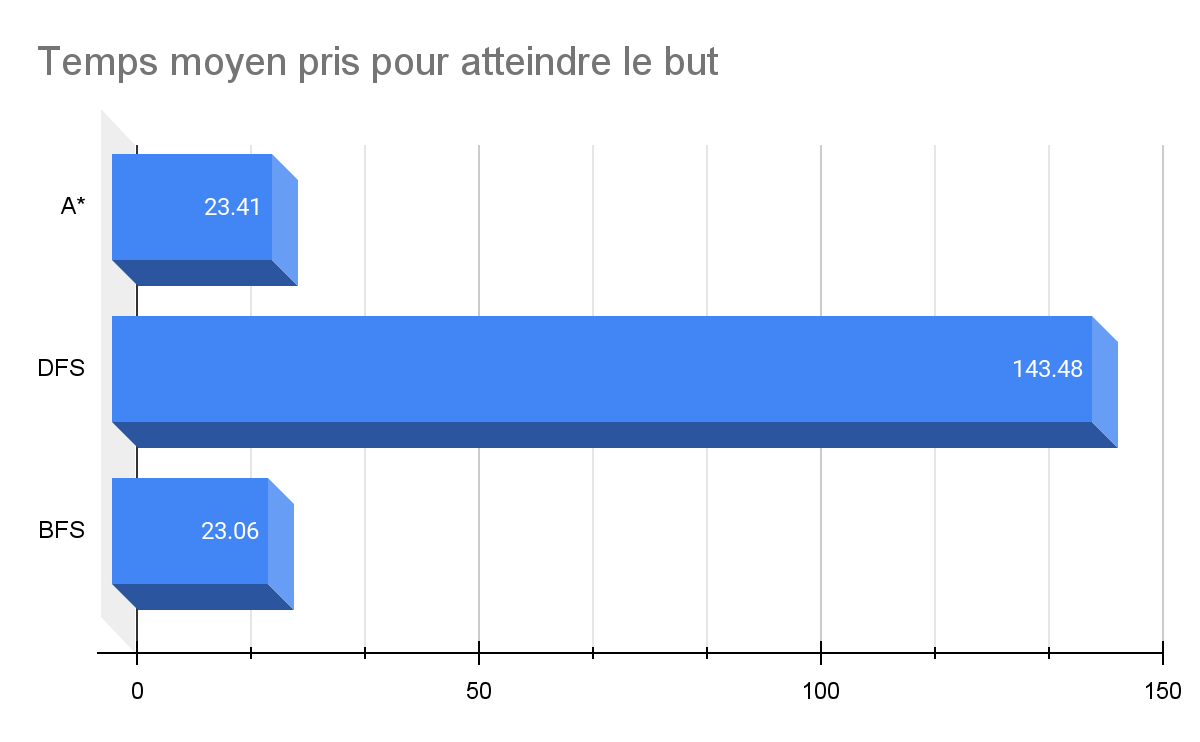
| **Algorithme** | **Temps moyen** | **Score moyen** |
| --- | --- | --- |
| **A\*** | 2342.25 | 100.05 |
| **DFS** | 7296 | 50.85 |
| **BFS** | 2326.05 | 100.85 |

On remarque donc que quand le serpent recherche la nourriture en utilisant la méthode DFS, certes il reste en vie largement plus longtemps mais il obtient tout de même un score très bas(la moitié du score des autres méthodes), et puisque l’intérêt du jeu est de manger le maximum de nourritures et non de rester en vie, donc la méthode DFS est la moins efficace pour le jeu Snake.  
Quant au deux autres algorithmes de recherche A\* et BFS, on trouve qu’ils se rapprochent énormément, ceci revient au fait que malgré que les chemins pris par les serpents qui utilisent les deux méthodes de recherche sont différents (A\* se dirige vers la nourriture en diagonale alors que BFS en fait beaucoup moins de changement de direction)

 Mouvement du serpent qu’on on utilise la méthode A\*

 Mouvement du serpent qu’on utilise la méthode BFS

Mouvement du serpent qu’on utilise la méthode DFS



Ce graphique montre le temps moyen que prend chaque méthode de recherche pour manger une nourriture

On déduit de ce graphique toujours que la méthode de recherche DFS est la moins efficace puisque le temps que prend le serpent pour arriver à la pomme est énorme par rapport aux autres méthodes de recherche.

On remarque aussi qu’il y a une toute petite meilleure performance qu’on on emploie BFS que A\* (23.06 contre 23.41) mais la différence n’est pas significative et peut n’être que le fruit du hasard.

**V) Limites :**

* Dans le but de bien comparer les différentes méthodes de recherche, on avait essayé de mettre en place le programme pour qu’il soit capable de faire jouer deux serpents utilisant différentes méthodes de recherche en même temps, et de comparer qui sera le meilleur (lequel mangera plus de pommes), sauf qu’on s’est rendu compte que les deux serpents s'entremêlent rapidement ce qui fait que la partie se termine beaucoup trop rapidement.
* De même, mettre en place un serpent contrôlé par un utilisateur n’avait pas trop d'intérêt pour les mêmes raisons, ainsi que l’IA qui utilise A\* ou BFS est dotée d’une vitesse de calcul très performante qui la rend presque imbattable face à un utilisateur lambda.

* Lorsque le serpent recherche le chemin optimal vers la pomme, il ne prend pas en considération le fait que sa queue bouge aussi et pourrait donc lui offrir des possibilités plus rapides qui lui éviteraient des fois de mourir plus tôt.
* Lorsque le serpent ne trouve pas un chemin vers la pomme (car le chemin est barré par son propre corps), il fait des mouvements aléatoires, et des fois faire un mouvement aléatoire (par exemple tourner à droite) pourrait être fatale pour le serpent dans la mesure où il sera coincé quelques cases après peu importe les mouvements qu’il fait.

**VI) Conclusion :**

Le travail sur ce projet a été très intéressant dans le sens où on a su mettre en pratique les notions qu’on avait vu en cours dans une situation réelle et ce en employant les différentes méthodes de recherche. L’analyse des résultats a aussi été intéressante car au début on croyait que l’algorithme A\* allait devancer de loin les deux autres méthodes, mais il s’est avéré que la méthode se reposant sur BFS est aussi bonne voire légèrement meilleure.

Finalement, on est convaincu que le code n’est peut être pas écrit de la manière la plus optimale possible puisque le serpent fait des fois des mauvais choix, en plus c’est vers la fin de la deadline qu’on a pensé à mettre les 2 serpents en même temps mais qu’ils peuvent se superposer l’un sur l’autre sans causer de dégâts mais c’était un peu trop tard.