Document Technique
Pathfinding

2024

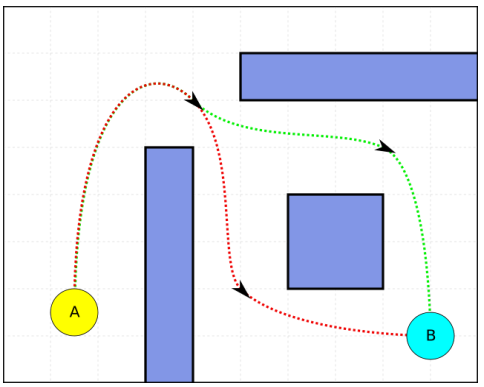
Evan CHOQUET - Hugo ROY - Melvin GUELLAFF - Yanaël CAILLOT

SOMMAIRE

LE PATHFINDING

Qu’est-ce que le pathfinding ?

Le **pathfinding**, également connu sous le nom de recherche de chemin, est **un problème fondamental** en informatique. Le pathfinding consiste **à trouver le chemin le plus court ou le plus efficace entre deux points**. Il existe une multitude d’algorithmes de pathfinding utilisés dans différents scénarios d’application.



Comment fonctionne le pathfinding et à quoi sert-il ?

Un algorithme de pathfinding commence généralement par représenter le problème sous forme **de graphe ou de grille**. Un graphe est **une collection de nœuds reliés par des arêtes**; par exemple, imaginez un diagramme de flux. Une grille est **un champ bidimensionnel de cellules**, comme un échiquier. Les nœuds ou les cellules représentent des emplacements dans l’espace du problème, tandis que les arêtes ou les cellules voisines représentent les chemins possibles entre eux.

Une fois le problème représenté sous forme de graphe ou de grille, les algorithmes de pathfinding utilisent différentes techniques pour trouver **le chemin entre deux points**. En général, les algorithmes visent **à trouver le chemin le plus court ou le moins cher tout en étant aussi efficaces que possible**.

Les différents domaines d’utilisation

Les algorithmes de pathfinding ont **de nombreuses applications en informatique**, notamment dans :

- **La robotique :** les algorithmes de pathfinding sont utilisés pour aider les robots autonomes à naviguer dans des environnements complexes. On pense par exemple aux voitures qui se conduisent toutes seules ou aux aspirateurs intelligents qui se déplacent tout seuls dans la maison.

- **Les jeux vidéo :** dans les jeux vidéo, les algorithmes de pathfinding sont utilisés pour contrôler les mouvements des personnages non joueurs (PNJ). Dans un jeu de stratégie en temps réel, si l’on envoie des unités dans la base ennemie en cliquant dessus, on utilise également des algorithmes de pathfinding.

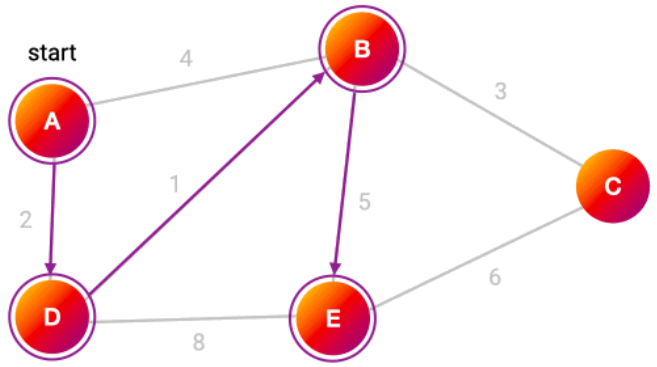
- **La logistique :** les algorithmes de pathfinding sont utilisés dans la logistique pour trouver le chemin le plus efficace pour le transport de marchandises ou de personnes.

- **La planification du trafic :** les algorithmes de pathfinding sont utilisés pour planifier les meilleurs itinéraires pour le trafic d’une ville, tout en évitant les embouteillages.

- **Le routage de réseau :** dans les réseaux informatiques, les algorithmes de pathfinding sont utilisés pour trouver le chemin le plus rapide pour la transmission des données entre les différents nœuds du réseau.

L’ALGORITHME DE DIJKSTRA

L’algorithme de **Dijkstra** est un algorithme de recherche de chemin dans un **graphe** permettant de trouver le chemin le plus **optimal** entre un **nœud initial** A et un **nœud final** B quelconque. Il trouve ce chemin en prenant en compte les **poids** entre chaque nœud, représentant de manière abstraite les **contraintes** existantes.



Vertex	Shortest distance from A	Previous vertex
A	0	
B	3	D
C	6	B
D	2	A
E	8	B

L’algorithme vérifie **chaque voisin** du nœud A et le **poids** qui leur est associé. Il vérifie ensuite chacun des **voisins** des nœuds suivants, et détermine quel est le chemin avec le **moins de poids** possible jusqu’au nœud B.

Avantages :

1. **Simplicité :** L'algorithme est relativement simple à comprendre et à implémenter.
2. **Optimal :** Garantit de trouver le chemin le plus court si tous les poids des arêtes sont non négatifs.

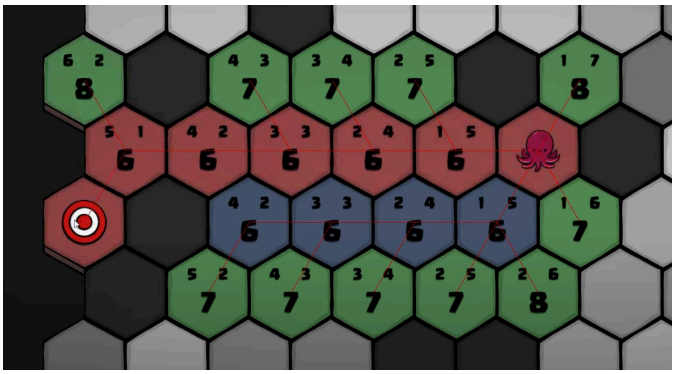
Inconvénients :

1. **Consommation mémoire** : nécessite une quantité importante de mémoire, surtout pour de grands graphes, car il doit stocker des informations sur tous les nœuds.
2. **Temps de calcul** : Dans des graphes très denses, l'algorithme peut devenir moins efficace car il doit explorer de nombreuses arêtes.
3. **Non adapté pour tous les types de graphes** : Pour certains types de graphes, comme ceux qui changent dynamiquement ou les graphes non pondérés, d'autres algorithmes comme A* ou BFS peuvent être plus appropriés.

A* (A Star)

L'algorithme **A*** est un algorithme de recherche de chemin dans un **graphe** entre un **nœud initial** A et un **nœud final** B. Il utilise une **heuristique**, donc chaque nœud se voit attribuer **trois paramètres** : **G**, **H** et **F** :

- **G** représente la distance entre le nœud A et le nœud choisi, donc la **distance parcourue**. (*ne pas oublier de la mettre à jour à chaque noeud visité*)
- **H** représente la distance **optimiste** entre le nœud choisi et le nœud B. Par exemple, si un mur se trouve entre le nœud choisi et le nœud B, le paramètre H sera inchangé. Ce paramètre représente la **distance à parcourir**.
- **F** est la **somme** des paramètres **G** et **H**. L'algorithme parcourt les nœuds en choisissant le nœud dont le coût **F** est le **moins élevé**. Si deux nœuds ont le **même coût F**, alors il choisit le nœud dont le **coût H** est le **moins élevé**, donc qui est possiblement le plus proche du nœud B.



Avantages :

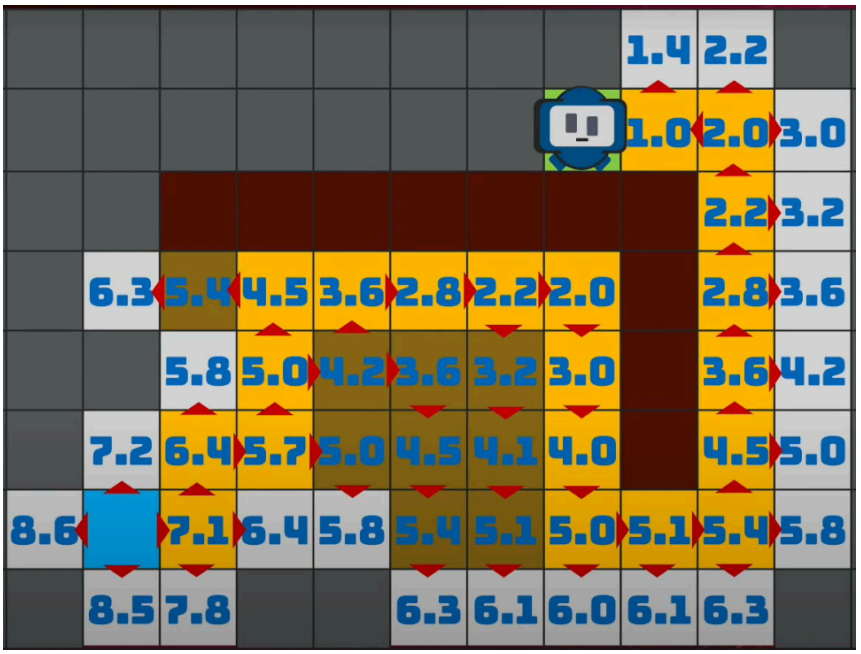
1. **Efficacité** : L'heuristique guide la recherche vers la cible, ce qui rend souvent la recherche plus courte.
2. **Optimalité** : A* garantit de trouver le chemin le plus court, si l'heuristique utilisée est admissible (qu'elle ne surestime pas le coût H).
3. **Flexible** : A* peut être adapté avec différentes heuristiques selon la nature du problème, ce qui le rend très polyvalent.

Inconvénients :

1. **Dépendance à l'heuristique** : Une mauvaise heuristique peut rendre A* inefficace.
2. **Consommation de mémoire** : A* peut nécessiter beaucoup de mémoire, surtout si l'espace de recherche est vaste, car il garde en mémoire tous les nœuds ouverts.
3. **Difficulté de mise en œuvre** : La mise en œuvre d'une bonne heuristique peut être complexe, surtout dans des environnements dynamiques ou incertains.

GREEDY BEST-FIRST SEARCH

Le **Greedy BFS** est un algorithme de pathfinding de **type BFS**, comme l'A*. Il utilise également une **heuristique** afin de déterminer le chemin entre un **nœud initial** A et un **nœud final** B. Cependant, le Greedy BFS ne prend en compte que le **paramètre H** du nœud choisi, donc la distance qui le sépare du nœud B.



Dans cet exemple, on voit que le robot prend un chemin qui n'est **pas optimal** jusqu'au **nœud final**, simplement parce que le coin du mur est proche de ce nœud.

Avantages :

1. **Rapidité** : Lorsqu'une solution approximative est suffisante, le Greedy est la meilleure option car c'est un algorithme très rapide.
2. **Simplicité** : C'est un algorithme facile à comprendre et à mettre en place

Inconvénients :

1. **Non-optimalité** : Il ne garantit pas toujours une solution optimale pour tous les problèmes.
2. **Manque de flexibilité** : Les choix effectués à chaque étape sont irréversibles, ce qui limite la capacité à corriger les erreurs si un choix initial est mauvais.

LE BFS ET LE DFS

Les algorithmes de pathfinding **BFS** (*Breadth-First Search*, ou *Algorithme de parcours en largeur*) et **DFS** (*Depth-First Search*, ou *Algorithme de parcours en profondeur*) sont des algorithmes permettant de parcourir un **graphe** ou un **arbre** afin de trouver un chemin d'un point A à un point B, ou déterminer si un graphe non orienté est connexe.
Ce document de recherche traitera uniquement de l'utilisation de ces algorithmes sur un **graphe non orienté connexe**, ou une **grille**, pouvant s'en apparenter avec chaque cellule représentant un nœud, et les voisins directs représentant les arêtes.

Principes de base du BFS et du DFS

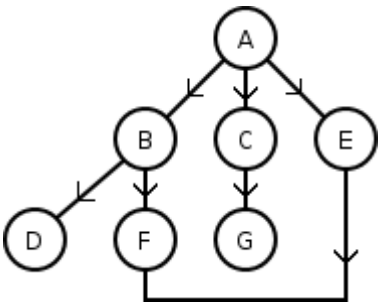
Ces deux algorithmes partagent des similarités dans leur objectif, mais fonctionnent de manière différente :

- Le **Breadth-First Search** va parcourir en priorité les nœuds **les plus anciens** : il va utiliser une "file", ou "queue", pour traiter les noeuds niveau par niveau
  La queue utilisée fonctionne sur un système **LIFO**, Last-In First-Out, qui veut simplement dire que les nouveaux nœuds voisins seront ajoutés à la fin de la liste, et que les premiers traités et retirés seront ceux au début de la liste.

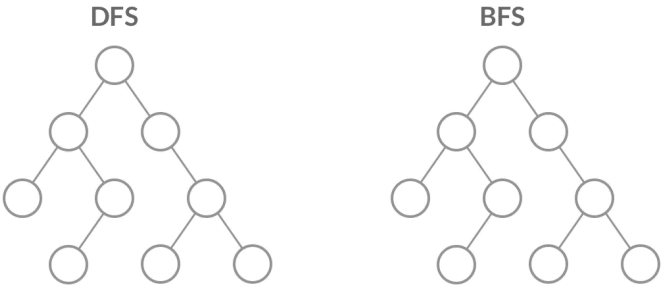
- Le **Depth-First Search** va parcourir en priorité les nœuds **les plus récents** : il va utiliser une "stack" ou une récursion favorisant l'exploration en profondeur avant de revenir en arrière.
  La stack utilisée fonctionne quand à elle sur un système **FIFO**, First-In First-Out, ce qui signifie que les nouveaux noeuds voisins seront ajoutés à la fin de la liste, mais que c'est également le dernier élément de la liste qui sera traité et retiré en priorité.

Tandis que le DFS parcours **branche par branche**, ou chemin par chemin, jusqu'à ce qu'un nœud n'ait plus de voisins non explorés, le BFS parcours le graphe en explorant **en priorité tous les voisins d'un nœud**.

Exemples imagés

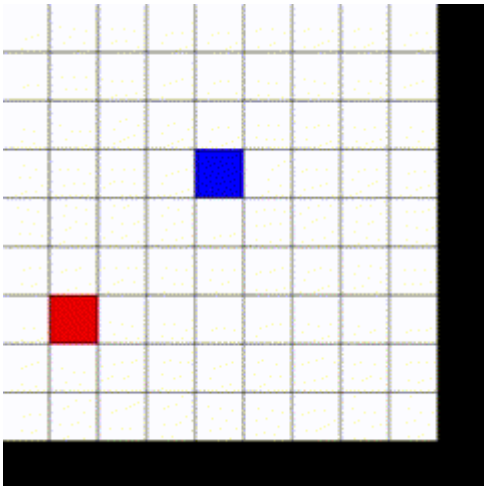


Le BFS parcourra ce graphe de la manière suivante : A > B > C > E > D > F > G
Le DFS parcourra ce graphe de la manière suivante : A > B > D > F > C > G > E



Comparaison du parcours d'un arbre par le BFS et le DFS.

Puisque le BFS parcours chaque nœud au plus proche du nœud de départ, le chemin retourné sera forcément le parcours le plus court, alors que le DFS parcours un graphe au plus profond en priorité, et peut faire des détours inutiles.



Utilisation de l'algorithme DFS sur une grille | Utilisation de l'algorithme BFS sur une grille

Avantages et Inconvénients des algorithmes de Pathfinding de type DFS/BFS

BFS :

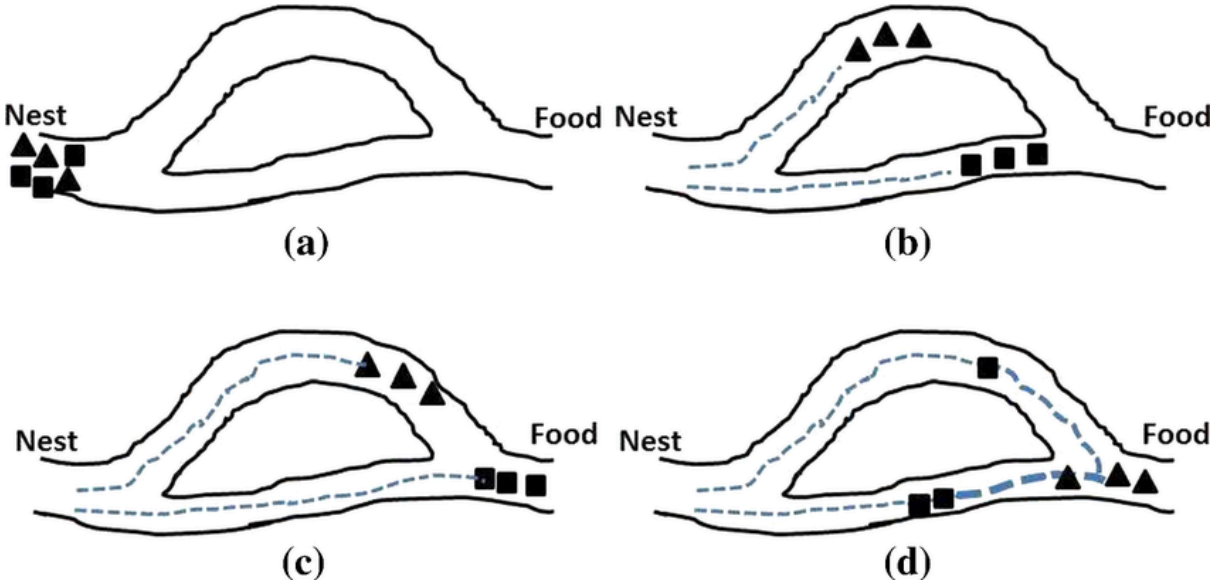
- Avantage majeur : garantit le chemin **le plus court** si les arêtes ont toutes la même valeur (non pondérées).
- Inconvénient : sur des graphes très vastes, sa consommation mémoire peut devenir **problématique**, car il stocke les nœuds à explorer pour chaque niveau.

DFS :

- Avantage : fonctionne bien pour des graphes très vastes ou des labyrinthes, car il consomme **peu de mémoire** si bien implémenté (notamment en récursif).
- Inconvénient : peut explorer des chemins **beaucoup plus longs** que nécessaire avant de trouver la solution optimale, surtout dans des graphes denses.

LE SWARM

Le **pathfinding de type Swarm** (souvent appelé **Swarm Behavior** ou **Swarming AI**) est un ensemble de techniques d'intelligence artificielle inspirées du comportement collectif d'animaux tels que les essaims d'insectes, les bancs de poissons ou les troupeaux d'oiseaux. Il s'agit d'une approche décentralisée où chaque agent individuel suit des règles simples, mais dont le comportement collectif peut aboutir à des mouvements et des décisions complexes et coordonnées.



En prenant comme exemple l'image, au-dessus on peut constater que le Swarm reprend le comportement des fourmis qui déplacent de la nourriture vers le nid et qui finissent par s'accorder pour prendre le chemin le plus court., c'est grâce à plusieurs itérations que le groupe finit par avoir le meilleur chemin

Principes de base du Swarming

Le Swarming repose généralement sur trois règles comportementales principales :

1. éviter les collisions avec les autres agents et les obstacles proches.
2. s'aligner sur la direction générale du groupe d'agents voisins.
3. se déplacer vers le centre du groupe d'agents voisins pour rester avec le groupe.

Ces règles locales, appliquées par chaque agent indépendamment, permettent de simuler des mouvements organisés sans avoir besoin de centraliser la coordination ou de calculer des chemins complexes pour chaque entité. Les comportements d'essaim permettent également de réagir rapidement aux changements de l'environnement, ce qui les rend adaptés à des environnements dynamiques et imprévisibles.

Voici un exemple vidéo de ce pathfinding

Avantages du Pathfinding de type Swarm

Scalabilité : Le Swarming est très efficace pour des groupes de grande taille, car les agents n'ont besoin de connaître que leurs voisins immédiats, évitant ainsi des calculs coûteux et des complexités de gestion pour de grands nombres d'agents.

Réactivité : Le système est naturellement réactif aux changements dans l'environnement. Si un obstacle apparaît, chaque agent peut ajuster son mouvement localement, et l'ensemble de l'essaim s'ajuster rapidement.

Simplicité des règles : Les règles sont simples à implémenter, ce qui permet de créer des comportements émergents complexes à partir de calculs locaux simples.

Exemples d'utilisation

Le pathfinding de type Swarm est souvent utilisé pour des unités de type "horde" ou "troupe" où chaque entité suit des règles simples pour se déplacer ensemble, créant ainsi une illusion de coordination . Il est également utilisé dans la robotique, cette technique est utilisée pour des systèmes où de nombreux robots doivent accomplir des tâches collectives, comme la surveillance ou la recherche dans des environnements vastes.

Boids

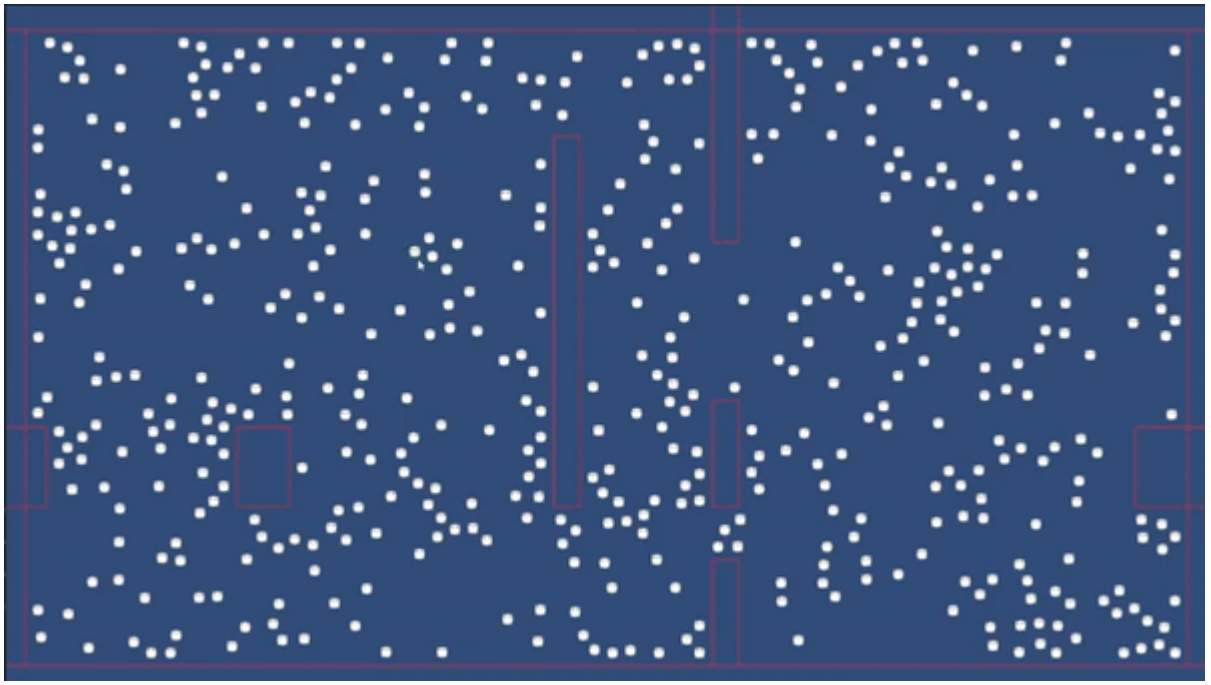
Le **pathfinding de type Boids** est une technique d'algorithme inspirée du comportement collectif d'animaux comme les bancs de poissons ou les volées d'oiseaux. Elle repose sur des règles simples qui, combinées, produisent un comportement collectif complexe et cohérent. Ce modèle a été introduit en 1987 par **Craig Reynolds** dans son papier sur les "Boids" (bird-oids, ou "objets ressemblant à des oiseaux").

Principes fondamentaux des Boids

Chaque boid (individu) suit trois règles de base :

1. Éviter les collisions avec les boids voisins en maintenant une certaine distance entre eux.
2. Aligner sa direction avec celle des boids proches pour bouger de manière cohérente.
3. Se diriger vers le centre de masse local de ses voisins pour rester dans le groupe.

Ces trois règles permettent aux boids d'afficher des mouvements réalistes et de simuler une forme d'intelligence collective sans qu'un boid spécifique dirige le groupe.



Pathfinding dans le contexte des Boids

En termes de **pathfinding**, les boids ne sont pas conçus pour suivre un chemin prédéfini comme le ferait un personnage dans un jeu vidéo avec un algorithme de type A*. Au lieu de cela, ils réagissent dynamiquement à leur environnement et à leurs voisins immédiats, ce qui produit un mouvement plus fluide et organique.

Utilisation dans les jeux vidéo

Dans les jeux vidéo, le pathfinding de type Boids est utilisé pour gérer les mouvements de grands groupes d'entités, comme des foules ou des groupes d'ennemis, en réduisant la complexité de calcul associée au contrôle individuel. Cela est particulièrement utile pour :

- **Simuler des comportements de foule** : Les boids peuvent être utilisés pour représenter une foule se déplaçant de manière fluide et réaliste à travers un environnement.
- **Mouvements de groupes d'ennemis ou d'alliés** : Les jeux de stratégie en temps réel (RTS) peuvent s'en servir pour contrôler les mouvements de grandes armées sans que chaque soldat ait besoin de calculer son propre chemin.

SWARM VS BOIDS

Le **pathfinding Swarm** et le **pathfinding Boids** sont tous deux des approches pour simuler des comportements collectifs et dynamiques, mais ils diffèrent dans leur fonctionnement, leurs objectifs et leurs règles sous-jacentes.

1. Origine et Contexte

- **Boids** : Créé par **Craig Reynolds** en 1987 pour simuler des mouvements réalistes de groupes d'oiseaux et de poissons. Il est principalement utilisé pour des comportements naturels et organiques, comme la formation de bancs de poissons, de volées d'oiseaux ou de foules.
- **Swarm** : Inspiré des comportements d'essaims d'insectes (comme les abeilles, les fourmis ou les essaims de drones), il est souvent utilisé dans le cadre d'algorithmes de résolution de problèmes, comme la **Swarm Intelligence** (par exemple, Particle Swarm Optimization ou Ant Colony Optimization).

2. Objectif Principal

- **Boids** : Son objectif principal est de simuler un mouvement naturel et fluide au sein d'un groupe. Les boids sont des entités autonomes qui réagissent à leur environnement immédiat (voisins) pour maintenir une cohésion de groupe.
- **Swarm** : Ici, l'objectif est souvent l'optimisation ou la recherche d'une solution collective à un problème, où chaque individu dans l'essaim (ou swarm) contribue à atteindre un objectif global, souvent basé sur l'exploration et l'exploitation de l'environnement (recherche de nourriture, optimisation d'une fonction, etc.).

3. Règles et Comportements

- **Boids** :
    - Fonctionne principalement avec trois règles : séparation, alignement et cohésion, pour maintenir la fluidité du mouvement du groupe.
    - Le comportement des boids est local et dynamique, ils réagissent essentiellement à leurs voisins immédiats et aux obstacles environnants.
    - L'accent est mis sur l'apparence visuelle et naturelle du mouvement de groupe, plutôt que sur l'atteinte d'un objectif spécifique.
- **Swarm** :
    - Le swarm, en revanche, suit des règles plus orientées vers la **collaboration** pour résoudre un problème ou atteindre un objectif global. Par exemple, les algorithmes comme **Particle Swarm Optimization (PSO)** ajustent les trajectoires individuelles pour améliorer une solution collective à un problème.
    - Chaque membre de l'essaim peut ajuster sa position en fonction de son expérience personnelle, de celle des autres membres de l'essaim, et parfois d'une solution optimale perçue par le groupe.
    - Ce n’est pas nécessairement visuel, mais plutôt orienté vers la performance de la tâche (comme la recherche de solutions optimales).

4. Interaction avec l'environnement

- **Boids** :
    - L'interaction avec l'environnement est locale et concerne principalement les voisins immédiats. Les boids évitent les collisions, maintiennent la cohésion et bougent de manière collective en fonction de la proximité des autres boids.
    - Ils n'ont pas nécessairement un objectif global ou une destination spécifique, sauf si un mécanisme additionnel de pathfinding est couplé.
- **Swarm** :
    - Dans un modèle Swarm, les individus peuvent interagir à un niveau plus global. Par exemple, dans les fourmis simulées dans l'**Ant Colony Optimization**, chaque fourmi laisse une **trace de phéromone** pour guider les autres vers une solution optimale.
    - Les membres du swarm peuvent explorer l'environnement globalement et échanger des informations indirectes pour converger vers une solution.

5. Pathfinding et Optimisation

- **Boids** :
    - Le pathfinding de Boids ne cherche pas nécessairement à atteindre un objectif spécifique. Il est conçu pour créer des mouvements naturels et collectifs dans un espace donné, sans préoccupation d'optimisation.
    - Toutefois, des systèmes de pathfinding supplémentaires (comme A*) peuvent être ajoutés pour aider le groupe à naviguer dans des environnements complexes.
- **Swarm** :
    - Le pathfinding Swarm a souvent un **objectif d'optimisation**. Par exemple, dans **PSO**, les particules se déplacent pour minimiser (ou maximiser) une fonction donnée. Dans l'ACO, les fourmis cherchent à trouver le chemin le plus court vers une source de nourriture.
    - Le but est souvent de trouver la meilleure solution ou le chemin optimal en s'appuyant sur les interactions collectives.

6. Applications

- **Boids** :
    - Utilisé principalement dans l'animation et les jeux vidéo pour simuler des mouvements réalistes de groupes d'entités (ex. bancs de poissons, foules).
    - Moins adapté pour résoudre des problèmes complexes d'optimisation.
- **Swarm** :
    - Utilisé dans des algorithmes d'optimisation comme le **PSO** ou l'**ACO**, mais également pour des tâches comme le **pathfinding collaboratif** dans des environnements complexes (robots en essaims, drones autonomes).
    - Souvent utilisé pour résoudre des problèmes complexes (planification de routes, optimisation de réseaux).

Synthèse :

- **Boids** est un modèle plus visuel et centré sur la dynamique de groupe et le réalisme du mouvement.
- **Swarm** est davantage axé sur la résolution de problèmes collectifs, l'optimisation et l'exploration de l'environnement global avec un objectif précis.

Ces deux modèles peuvent parfois être combinés ou ajustés pour des applications spécifiques, mais leurs finalités et méthodes de fonctionnement restent distinctes.

CONCLUSION

L'algorithme de **Dijkstra**, le **A***, le **Greedy BFS**, le **BFS** et le **DFS** sont des **algorithmes de graphe** qui fonctionnent avec des nœuds.
En comparaison, Les algorithmes **Swarm** et Boids sont des algorithmes qui permettent le pathfinding d'un **groupe d'individus**.

BFS / DFS

Le **BFS** et **DFS** fonctionnent sur des **graphes non pondérés**, donc ils ne fonctionnent pas lorsqu'il y a des contraintes sur la map choisie.
Le **BFS** donne toujours **le chemin le plus optimisé** entre un point A et un point B. Il est notamment utilisé dans les GPS, mais toujours couplé avec d'autres algorithmes pour le rendre plus performant.
Le **DFS** est optimal dans les cas où l'on veut explorer **tous les chemins** d'un graphe/**toutes les possibilités** d'un arbre de probabilités.

Dijkstra / A* / Greedy BFS

Le **Dijkstra** donne toujours **le chemin le plus optimisé** entre un point A et un point B, fonctionne sur des **graphes pondérés**, mais est très lent car il ne prend **pas** en compte la distance qui le sépare du point B.
En revanche, le **A*** et le **Greedy BFS** le prennent en compte.
Le **A*** prend en compte la distance qui le **sépare du point A** et la distance qui le **sépare du point B**. Il trouve toujours **le chemin le plus court**, mais n'est **pas optimisé** pour des grands graphes parce qu'il garde en mémoire tous les nœuds visités.
Le **Greedy BFS** ne prend en compte **que** la distance qui le sépare du point B, permettant un algorithme rapide mais qui **ne trouve pas** de chemin non optimisé.

Swarm / Boids

Le **Swarm** et le **Boids** sont tous deux des algorithmes n'ayant **pas** pour but de trouver et suivre un chemin prédéfini entre un point A et un point B.
En effet, ces deux algorithmes servent pour la **gestion de groupe d'individus**.
Ce style d'algorithme est utile afin de simuler **des foules** ou **des déplacements de masse**.
Afin de simuler un **déplacement de groupe**, ces algorithmes sont souvent **associés** à des algorithmes de pathfinding plus classique, avec ces derniers responsables du **chemin général**, et le swarm et le boids servent à garder le groupe **ensemble** et **cohérent**.