

Etat de l'art des algorithmes d'intelligence artificielle utilisés dans les jeux de stratégie

Groupe 28 - Saulcy Road

Objectif de cet état de l'art / Périmètre :

L'objectif de ce document est de nous aider à trouver quel algorithme nous allons implémenter dans notre jeu de type Crossy Road. Il faut donc que ça soit un algorithme adapté à notre niveau de programmation, relativement simple à implémenter en C et suffisamment performant pour créer un joueur virtuel dans notre jeu.

Critères de sélection :

- Pertinent : Adapté à Crossy Road (un seul joueur, jeu en temps réel, etc.)
- Performance : Assez puissant pour réussir à jouer à Crossy Road avec un niveau plutôt correct (ex : il ne faut pas que le joueur virtuel meurt directement)
- Rapidité : Le rythme de Crossy Road est assez intense, donc il faut un algorithme qui peut être exécuté assez rapidement
- Simplicité : Adapté à notre niveau de programmation, il faut qu'on puisse le comprendre assez facilement pour pouvoir l'implémenter, et qu'on puisse le faire en C.

Sources consultées pour l'étude bibliographique :

- https://fr.wikipedia.org/wiki/Algorithme_minimax
- <https://www.youtube.com/watch?v=l-hh51ncgDI>
- <https://web.stanford.edu/class/archive/cs/cs221/cs221.1192/2018/restricted/posters/nwrubin/poster.pdf>
- https://en.wikipedia.org/wiki/Alpha%E2%80%93beta_pruning
- <https://www.youtube.com/watch?v=71UbDN4csas>
- <https://towardsdatascience.com/q-learning-algorithm-from-explanation-to-implementation-cdbeda2ea187/>
- <https://fr.wikipedia.org/wiki/Q-learning>
- https://fr.wikipedia.org/wiki/Recherche_arborescente_Monte-Carlo
- https://en.wikipedia.org/wiki/Best-first_search

Analyse des algorithmes découverts dans les différentes sources consultées :

Algorithme	Fonctionnement	Pertinence	Performance / Rapidité
Minimax	Explore toutes les actions possibles en supposant un adversaire parfait, puis minimise la perte maximale.	Assez pertinent. On peut modéliser l'environnement comme "adversaire", comme l'ont fait des étudiants de Stanford.	Plus lent que Best-first search pour un jeu comme Crossy Road, nécessité de faire beaucoup d'appels récurifs. Peut-être implémenté en utilisant une profondeur assez basse.
Alpha-Beta Pruning	Optimisation du Minimax en coupant des branches non prometteuses dans l'arbre.	Pareil que Minimax.	Plus performant que le Minimax étant donné l'optimisation.
Q-Learning	Apprend quelle action est la meilleure dans chaque situation en recevant des récompenses.	Très pertinent si bien entraîné, mais très long / difficile vu la complexité / rapidité de Crossy road.	Long à entraîner mais rapide à exécuter ensuite. On pourrait peut-être lui faire atteindre un niveau passable pour des parties courtes, mais pas plus.
Recherche arborescente de Monte-Carlo (MCTS)	Simule des parties aléatoires pour chaque action possible, et choisit celle avec le meilleur score moyen.	Pas vraiment pertinent pour un jeu rapide en temps réel comme Crossy Road, il y aurait trop de situations à simuler.	Trop lent, beaucoup trop de simulations aléatoires à effectuer.
Best-first search	Exploration guidée par une fonction d'évaluation, on cherche le chemin le plus prometteur à chaque pas.	Très pertinent pour les jeux où il faut trouver un chemin comme Crossy Road tout en évitant des obstacles.	Rapide avec une bonne heuristique

Simplicité : Tous ces algorithmes sont relativement bien documentés, et implémentables en C. Certains comme le Q-Learning ou MCTS sont un peu plus difficiles à comprendre et à mettre en place, mais restent à notre portée.

Conclusion :

Les deux algorithmes les plus adaptés à Crossy Road sont Minimax (avec une optimisation possible) et Best-First Search. Ils peuvent s'appliquer aux règles du jeu, sont assez performants pour son rythme et sa complexité, et sont implémentables à l'échelle de notre projet / sont accessibles à notre niveau.

L'algorithme que nous avons choisi pour notre joueur virtuel est une version personnalisée de **Best-First Search**, avec une logique d'élagage inspirée de l'**Alpha-Beta Pruning**.

L'algorithme commence par générer les futurs états du plateau en simulant différentes directions possibles à chaque étape. À chaque niveau de profondeur, il explore tous les chemins qui n'ont pas encore été éliminés.

Les chemins qui mènent à une mort certaine (collision, sortie de la carte...) sont immédiatement retirés. Cela permet de réduire fortement le nombre d'états à explorer, ce qui rend l'algorithme plus rapide et efficace. Cette logique d'élimination ressemble à un pruning, comme dans l'algorithme Alpha-Beta cité précédemment.

Pour chaque chemin restant, une fonction d'évaluation calcule un score basé sur plusieurs critères (la progression, la sécurité, le positionnement, etc.). Si plusieurs chemins mènent au même état, nous ne gardons que celui qui a le meilleur score, ou, en cas d'égalité, celui dont la direction de départ est la plus avantageuse.

Cet algorithme nous permet d'obtenir un joueur virtuel capable de prendre des décisions rapides, d'éviter les obstacles et de maximiser sa progression dans un jeu avec un rythme aussi soutenu et dynamique que Crossy Road.