Groupe 5

Dossier de Validation IA

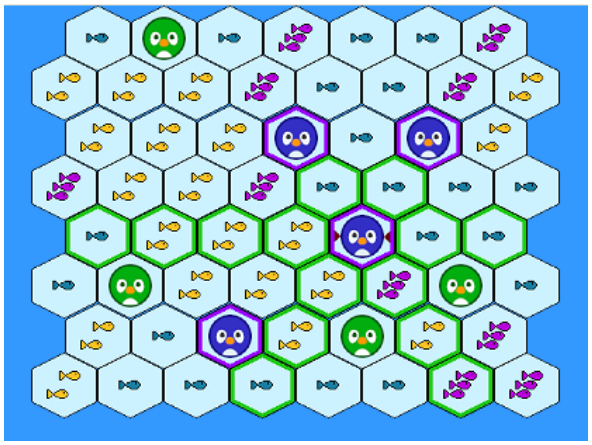
Projet S6 - 2023

Table des matières

Introduction

Description du jeu

Jeu de plateau de 2 à 4 joueurs, l'objectif est d'amasser le plus de poissons, pour la partie IA, nous nous focaliserons sur le jeu à 2 joueurs. Au début du jeu, la forme du plateau est comme sur l'image ci-dessous et les tuiles de 1 à 3 poissons sont réparties de façon aléatoire. Le plateau contient 30 tuiles de 1 poisson, 20 tuiles de 2 poissons et 10 tuiles de 3 poissons. Les joueurs placent leurs pingouins où ils veulent sur des tuiles de 1 poisson, chaque joueur doit placer 4 pingouins pour 2 joueurs, 3 pingouins pour 3 joueurs et 2 pingouins pour 4 joueurs. Un pingouin peut se déplacer dans les 6 directions de l'hexagone (le pingouin bleu sélectionné sur l'image ci-dessous peut se déplacer sur les cases vertes). Quand un pingouin se déplace sur une case, il mange le(s) poisson(s) se trouvant dessus, et la case où il se trouvait est détruite, si un pingouin ne peut plus se déplacer alors il est tué et retiré de la partie. Quand les joueurs ne peuvent plus se déplacer, le jeu est terminé et les points sont comptés pour définir le vainqueur, un joueur peut encore jouer si un de ses pingouins est encore en vie.



Analyse du jeu

En jouant plusieurs fois au jeu, on a pu analyser que le jeu peut être découpé en plusieurs étapes :

1. Positionnement des pingouins
2. Phase de bataille
3. Phase de récolte

Les différentes phases sont purement théoriques et permettent d'utiliser plusieurs algorithmes au fur et à mesure de la partie.

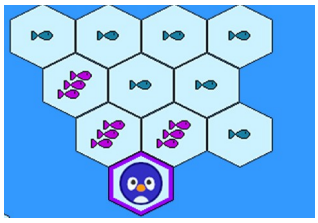
Positionnement des pingouins

Lors de cette phase, le joueur doit essayer de trouver le meilleur emplacement possible afin d'atteindre le plus grand nombre de poissons, plus il peut atteindre des poissons de haute valeur, mieux c'est, il doit faire attention à ne pas être bloqué par les futurs placements et déplacements de son adversaire. Avec un bon placement, il peut également bloquer son adversaire afin de réduire ses possibilités. Des pingouins du même coté du plateau pourrait avantager l'adversaire en lui laissant plus de poissons.

Phase de bataille

La phase de bataille, qui représente la partie principale du jeu, se déclenche après le positionnement des pingouins et avant que les pingouins se trouve sur des îlots (un îlot, dans notre jeu est lorsqu'un ou plusieurs pingouins de la même couleur se trouve sur une partie du plateau ou aucun adversaire peut arriver). Le but de cette phase est d'amasser le plus de poissons de grande valeur et de séparer le plateau en plusieurs îlots en gardant pour

lui ceux avec le plus de points. Lorsqu'un pingouin se trouve sur un îlot, nous pouvons nous concentrer sur les autres pingouins, celui-ci ne peut pas se faire voler de poissons.



Phase de récolte

Cette partie succède la phase de bataille, tous nos pingouins se trouve sur des îlots, nous pouvons désormais nous en occuper. Ici nous ne sommes pas en bataille directe contre l'adversaire, il faut essayer de récolter le plus de poissons sur l'ile.

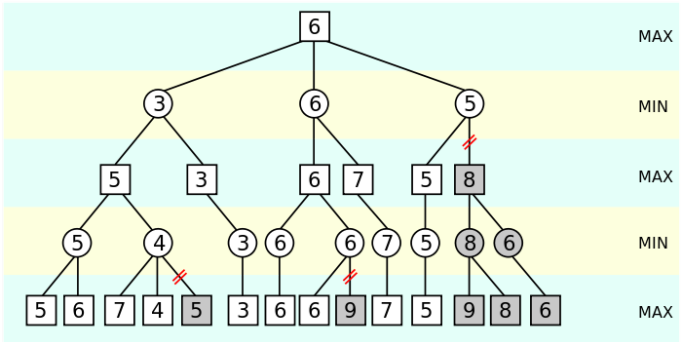
Minimax

Pour notre IA, nous avons implémenté l'algorithme minimax. L'algorithme va simuler une partie de 2 joueurs en jouant les meilleurs coups pour le joueur qui représente l'IA et son adversaire jouera les pires coups pour contrer l'IA. L'IA suppose que son adversaire jouera de la meilleure façon possible. L'algorithme va faire une énumération de tous les coups possibles sur plusieurs profondeurs, de ce fait l'algorithme peut prendre du temps si on veut l'exécuter en entier, il faut donc définir une profondeur limite ainsi qu'une fonction d'évaluation qui évaluera l'état du plateau quand la profondeur sera atteinte.

Alpha-Beta

Une amélioration possible du minimax est d'élaguer les feuilles de l'arbre minimax. Comme sur l'exemple ci-dessous, sur la partie droite de l'arbre, on peut enlever la branche de valeur 8 car son « frère » est de valeur 5 qui est plus petit que son « oncle » qui est de valeur 6. Etant donné que le père du 5 va prendre la valeur la plus petite (soit 5 au maximum dans ce cas) et que son grand-père prendre la valeur maximum (soit 6 dans ce cas), il ne sert donc à rien de calculer les autres fils de 5.

Nous avons pu implémenter l'arbre Alpha-Beta assez rapidement après le Minimax ce qui a accéléré considérablement notre algorithme, nous avons donc pu augmenter la profondeur.



Source : Wikipedia.org

MCTS

Le Monte Carlo Tree Search (MCTS) est un autre algorithme qui peut être utilisé dans des jeux à 2 joueurs, le MCTS n'aura pas une profondeur définie mais un temps donné à ne pas dépasser. Cet algorithme à pu être utilisé lorsque le nombre de configurations différentes est

beaucoup plus grande et ne peut pas être géré par un minimax (exemple : le jeu de Go). Cet algorithme à 4 différentes phases :

- Sélection

Cette phase s'occupe de sélectionner la feuille à expandre, il va fonctionner récursivement en calculant un score de chacun de ses fils en fonction de plusieurs paramètres. Généralement le calcul est : $\frac{w_i}{n_i} + c\sqrt{\frac{\ln N}{n_i}}$ où 'w' est le nombre de victoire, 'n' est le nombre de parties, 'N' est le nombre de parties de son père et 'c' est généralement $\sqrt{2}$. Le calcul peut être ajusté si nous souhaitons aller plus profondément dans l'arbre ou avoir un arbre plus équilibré.

- Expansion

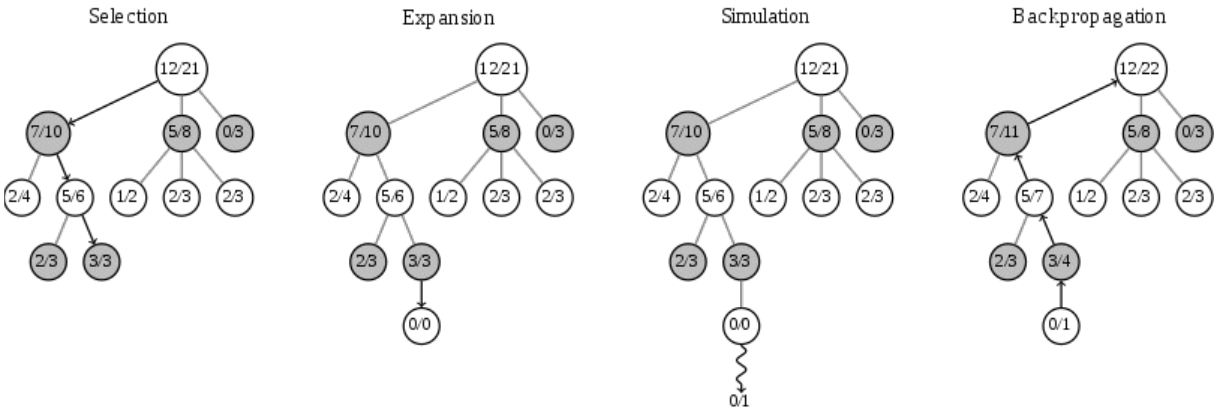
Lorsque la feuille a été sélectionnée, nous pouvons lui ajouter des fils sur lesquels nous effectuerons une simulation, c'est seulement dans cette phase que nous agrandirons l'arbre.

- Simulation

Une fois les fils créés, nous effectuons une simulation afin d'ajouter un score à la feuille, le plus souvent en effectuant un Monte Carlo de 1, nous mettrons à jour la feuille en fonction du résultat.

- Rétropropagation

La valeur de la feuille est alors remontée à tous ses prédécesseurs afin qu'ils mettent à jour à leur score.



Source : Chess Programing Wiki

Nous avons implémenté cet algorithme mais l'IA était moins forte qu'avec le minimax pour un même temps de calcul. Nous avons donc pris la décision de garder le minimax lors de la phase de bataille. Nous utilisons un algorithme similaire lors du parcours d'ilot.

Fonction d'évaluation

La fonction d'évaluation va être utilisée pour évaluer le plateau à un instant t afin de permettre au minimax de réduire la profondeur. Plus la fonction d'évaluation est précise, plus l'IA jouera bien. Nous avons 2 fonctions d'évaluation, la mise à jour des poids est utilisée lors de l'initialisation ainsi que pour l'IA facile. Le monte Carlo est utilisé lors de la phase de bataille pour les IA moyenne et difficile.

Monte Carlo

Le monte Carlo simule x parties en choisissant des coups aléatoires, cela permet d'évaluer le plateau en fonction du nombre de victoires et de défaites. Plus x est grand, plus Monte Carlo est précis.

Mise à jour des poids

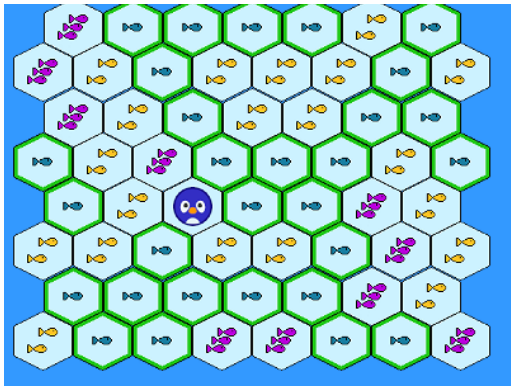
Ici nous avons utilisé différentes heuristiques, chacune d'elles est multipliée par un poids afin d'obtenir la somme totale de toutes les heuristiques :

Evaluation = H1*W1+H2*W2+H3*W3

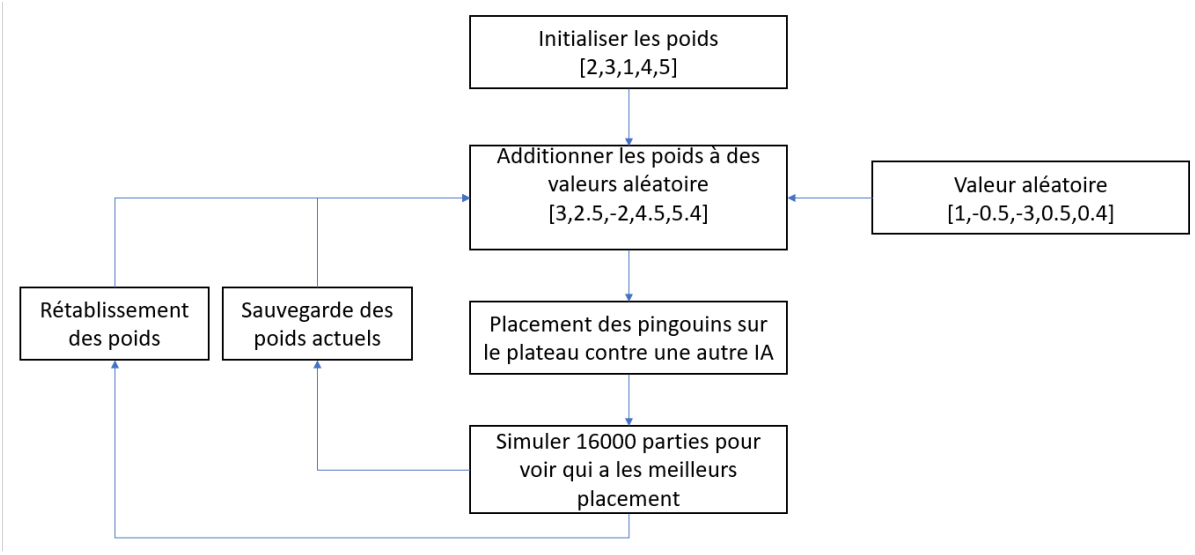
Les différentes heuristiques utilisées permettent de vérifier plusieurs parties du plateau :

1. Parmi les 6 cases autour du pingouin, nous calculons le total de poissons autour, cela permet au pingouin de favoriser un emplacement avec des poissons de grande taille autour de lui et d'éviter les bords du plateau.
2. Les poissons de valeur 1 accessible en 1 coup.
3. Les poissons de valeur 2 accessible en 1 coup.
4. Les poissons de valeur 3 accessible en 1 coup.
   Les trois dernières heuristiques permettent de faire des choix afin de se trouver a proximité de certains types de poisson.
5. Parmi les 6 cases autour du pingouin, nous calculons le nombre de pingouins de la même couleur que lui.
6. Parmi les 6 cases autour du pingouin, nous calculons le nombre de pingouins de différentes couleurs.

En multipliant les différents poids par rapport aux heuristiques, cela permet de plus prendre en compte les heuristiques les plus importants.



Afin de trouver les bons poids pour ses différentes heuristiques, nous avons au départ initialisé les différents poids entre 0 et 5 sur des valeurs qui nous semblait les meilleures. Ensuite nous avons fait jouer l'IA avec ses heuristiques contre le Monte Carlo pour la phase de placement. A la fin du placement, nous avons effectué 16000 simulations de parties pour vérifier qui avait le meilleur placement entre nos heuristiques et le Monte Carlo. Avant chaque phase de placement, nous mettions à jour les poids de manière aléatoire, on additionnait nos poids à des valeurs aléatoires afin d'avoir des nouveaux poids. Si jamais le Monte Carlo avait plus de victoires, alors on rétablissait les poids et on recommençait le processus. Quand notre algorithme battait le Monte Carlo, alors on le faisait jouer contre l'ancienne version de lui-même pour essayer de l'améliorer encore.



Parcours d'ilot

Quand l'IA est sur un îlot, nous n'avons plus besoin de faire le minimax, nous devons faire un algorithme de parcours. Pour cela, nous avons effectué un algorithme qui repère les îlots du joueur, cet algorithme va parcourir toutes les cases adjacentes sans revisiter la même case jusqu'à ce qu'il trouve un autre pingouin (dans ce cas, ce n'est pas un îlot). Si jamais il n'en trouve pas et qu'il a exploré toute l'ile, alors nous avons un îlot. Pour l'exploration de cet îlot par le pingouin, nous avons mis en place un algorithme similaire à MCTS afin de manger tous ou le plus de poissons possible sur cet îlot. La sélection de la feuille va prioriser la profondeur qui a été la moins explorée.

Conclusion

Nous sommes arrivés à mettre en place l'algorithme de minimax avec une profondeur de 3 au maximum pour le jeu comme le nôtre, car le jeu de pingouins à une possibilité de configuration trop grande. Pour cela on a envisagé l'algorithme MCTS qui est beaucoup utilisé pour des jeux avec des possibilités de configuration élevés. Nous avons implémenté cet algorithme (MCTS), mais l'IA ne joue pas de manière efficace et donc pour notre 3 difficultés de L'IA nous avons utilisé le minimax avec des profondeurs différentes, deux fonctions d'évaluation utilisant le Monte Carlo et une avec des heuristiques associées à des poids. Sur les 3 différentes étapes expliquées plus haut, le fonctionnement de placement et de la dernière phase sont identiques quel que soit la difficulté de l'IA, seule la seconde étape change. Pour l'IA facile, elle n'utilise pas le Monte Carlo mais les poids qui ont été entrainés pour le placement, ce qui rends l'IA moins efficace qu'avec Monte Carlo mais sans jouer des coups négatifs pour elle, ce qui est parfait pour une IA facile, le fait de la non-utilisation du Monte Carlo permet d'avoir une meilleure profondeur, nous l'avons fixé à 5. Les IA de difficulté normale et difficile utilisent une profondeur respective de 2 et 3 ainsi qu'un Monte Carlo avec respectivement 250 et 5 simulations. Les 2 IA ont une façon de jouer différente, l'IA avec plus de profondeur va essayer de plus bloquer l'adversaire et jouer de façon plus « agressive ».