Hex-Ta(c)tique
Projet de programmation 2

Auteurs

AL AYOUBI Ibrahim
BIGEY Raphaël
BONETTI Timothée
LEJEUNE Ivan

Encadrant

DA-SILVA Sébastien

12 mai 2024

Table des matières

1 Présentation du sujet

Les jeux de société ont toujours été un moyen de divertissement populaire, mais au-delà de leur aspect récréatif, ils constituent également des domaines d'étude intéressants pour les chercheurs en informatique, en mathématiques et en intelligence artificielle. Dans ce contexte, et dans le cadre de notre projet de programmation de L3 Informatique, nous avons choisi de nous intéresser à la théorie des jeux combinatoires. Nous avons ainsi mis en œuvre les jeux de stratégie combinatoire abstrait du Hex et de l'Awalé, reconnaissant ainsi leur valeur non seulement comme des défis ludiques, mais aussi comme des sujets de recherche passionnants pour explorer les interactions stratégiques et algorithmiques entre les joueurs.

Avant de poursuivre notre discussion sur notre motivation à étudier ces jeux, il est pertinent de clairifier ce que l'on entend par «jeux de stratégie combinatoire abstrait». Ce sont des jeux (généralement de société) tel que :

- opposant généralement deux joueurs ou deux équipes (ou bien un joueur humain seul contre un ordinateur « intelligent »)
- dans lequel les joueurs ou équipes jouent à tour de rôle.
- dont tous les éléments sont connus (jeu à information complète).
- où le hasard n'intervient pas pendant le déroulement du jeu.

En d'autres termes, dans les jeux de stratégie combinatoire, la victoire dépend entièrement des actions des joueurs et de leur capacité à anticiper et contrer les mouvements de l'adversaire.

L'étude de ces jeux nous motive à mieux comprendre les algorithmes de recherche et d'optimisation, et de nous familiariser avec les techniques de programmation avancée. A l'avenir, ces résultats pourront être utilisés dans un cadre plus général pour la résolution de problèmes plus complexes, par exemple pour les échecs ou le go.

Il existe plusieurs approches possibles pour la résolution de jeux de stratégie combinatoire. Parmi les approches les plus courantes, on trouve les algorithmes de recherche en profondeur, les algorithmes de recherche de chemin, les algorithmes de recherche de meilleure réponse, les algorithmes de Monte-Carlo, les algorithmes de renforcement, etc. Pour ce projet, nous avons choisi d'implémenter deux algorithmes en particulier pour la résolution des jeux de hex et d'Awalé : l'algorithme Minimax avec élagage alpha-bêta pour le jeu de hex, et l'algorithme de Dijsktra pour le jeu de l'Awalé.

Les principaux avantages de ces algorithmes sont les suivants :

- L'algorithme Minimax avec élagage alpha-bêta est un algorithme de recherche qui permet de trouver la meilleure stratégie pour un joueur dans un jeu à deux joueurs. Cet algorithme est très efficace pour les jeux de stratégie combinatoire comme le hex, car il permet de réduire le nombre de nœuds explorés lors de la recherche de la meilleure stratégie.
- L'algorithme de Dijsktra est un algorithme de recherche de chemin qui permet de trouver le chemin le plus court entre deux nœuds dans un graphe. Cet algorithme est très efficace pour le jeu de l'Awalé, car il permet de trouver la meilleure stratégie pour un joueur en minimisant le nombre de graines capturées par l'adversaire.

Le cahier des charges détaillé est disponible en annexe. A RAJOUTER : Le cahier des charges détaillé

2 Technologies utilisées

Pour la réalisation de ce projet, nous avons utilisé les langages de programmation suivants :

- Python pour l'implémentation des algorithmes de résolution des jeux et la logique du jeu,
- HTML, CSS et JavaScript pour l'implémentation de l'interface graphique des jeux,
- UML pour la modélisation des classes et des cas d'utilisation,
- Git pour la gestion du code source et le suivi des versions,
- Visual Studio Code pour l'écriture du code et le débogage,
- GitHub pour l'hébergement du code source et la collaboration,
- LaTeX pour la rédaction du rapport,
- Draw.io pour la création des diagrammes UML

Nous avons choisi Python pour l'implémentation des algorithmes de résolution des jeux et la logique du jeu, car c'est un langage de programmation très populaire et très puissant, qui offre de nombreuses bibliothèques et modules pour le développement d'applications complexes. Python est également un langage de programmation très simple et très lisible, ce qui facilite la compréhension du code et la collaboration entre les membres de l'équipe.

Nous avons choisi HTML, CSS et JavaScript pour l'implémentation de l'interface graphique des jeux, car ce sont des langages de programmation efficaces et simples à comprendre, qui permettent de créer des interfaces graphiques interactives et ergonomiques. HTML est un langage de balisage qui permet de structurer les pages web, CSS est un langage de style qui permet de mettre en forme les pages web, et JavaScript est un langage de programmation qui permet de rendre les pages web interactives.

Nous avons choisi UML pour la modélisation des classes et des cas d'utilisation, car c'est un langage de modélisation très complet, qui permet de représenter de façon claire et précise la structure et le comportement des systèmes informatiques.

Nous avons choisi Git et GitHub pour la gestion du code source et le suivi des versions, car ce sont des outils très puissants avec lesquels nous sommes très familiers.

LaTeX a été choisi pour la rédaction du rapport, car c'est un langage de composition de documents très puissant et très flexible, qui permet de créer des documents de grande qualité typographique. Etant donné que nous avons déjà utilisé LaTeX pour d'autres projets, nous avons préféré continuer à l'utiliser pour ce projet.

Draw.io a été choisi pour la création des diagrammes UML, car c'est un outil très simple et qui répond parfaitement à nos besoins. Il permet de créer des diagrammes UML de façon très intuitive, et de les exporter dans différents formats.

3 Développements Logiciel : Conception, Modélisation, Implémentation

4 Moteur de jeux combinatoires abstrait

Implémenter un algorithme pour jouer à des jeux combinatoires contre ordinateur est l'un des objectifs principaux du projet Hex-Ta(c)tique mais une problématique importante apparait : quels sont les algorithmes qui permettent à un ordinateur de jouer à un jeu combinatoire abstrait ? Quels sont leurs points forts et faibles ? Dans cette section nous allons présenter 2 algorithmes principaux qui aident à répondre à ces questions : la recherche arborescente *Monte-Carlo* (Monte carlo Three Search) et l'algorithme du *min-max* avec élagage alpha beta :

4.1 La recherche arborescente Monte-Carlo

Présentation La recherche arborescente Monte Carlo ou Monte Carlo tree search (MCTS) est un algorithme de recherche heuristique. Il est principalement utilisé dans le cadre de mise en place d'intelligence artificielle pour des jeux tels que le go mais pas uniquement. En effet il est aussi utilisé pour les moteurs de jeux des échecs comme le moteur Alpha Zero de Google qui est l'un des leaders en terme d'intelligence de jeux aux échecs. Cet algorithme peut même être implémenté dans des jeux où le hasard apparait par exemple au poker.

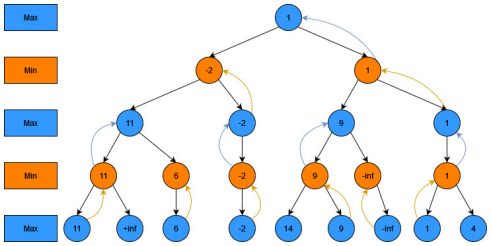
MCTS - fonctionnement succinct Monte Carlo Tree Seach ou MCTS est un algorithme qui explore l'arbre des possibles. La racine est la configuration initiale du jeu. Chaque nœud est une configuration (une situation en jeu) et ses enfants sont les configurations suivantes. MCTS conserve en mémoire un arbre qui correspond aux nœuds déjà explorés. Une feuille de cet arbre est soit une configuration finale (donc un des joueur a gagné) soit un nœud dont aucun enfant n'a encore été exploré. Dans chaque nœud, on stocke deux nombres : le nombre de simulations gagnantes, et le nombre total de simulations.

A chaque itération MCTS va chercher la feuille la plus prometteuse, comprendre la suite de coups qui possède la meilleure heuristique, et ensuite depuis cette feuille créer, grâce aux règles du jeu, une nouvelle feuille au hasard dans le but d'atteindre une configuration finale. Dans le cas où on trouve une configuration gagnante pour un joueur, on va incrémenter le nombre de simulations gagnantes, pour les nœuds correspondant au joueur, dans les parents de la feuille que l'on vient de créer. L'algorithme répète cette opération un certain nombre de fois avant de choisir un coup, ie : choisir le coup qui a le plus de simulations gagnantes et le moins de simulations totales.

4.2 MinMax

Présentation L'algorithme MinMax, comme MCTS, est un algorithme décisionnel mais celui-ci s'applique sur des jeux à somme nulle : c'est-à-dire un jeu où la somme des gains et des pertes de tous les joueurs est égale à 0. Cela signifie donc que le gain de l'un constitue obligatoirement une perte pour l'autre. L'algorithme va utiliser cette propriété dans sa recherche pour le meilleur coup possible. En effet il amène l'ordinateur à passer en revue toutes les possibilités pour un nombre limité de coups et à leur assigner une valeur qui prend en compte les bénéfices pour le joueur et pour son adversaire. Le meilleur choix est alors celui qui minimise les pertes du joueur tout en supposant que l'adversaire cherche au contraire à les maximiser (d'où le nom MinMax).

MinMax - fonctionnement succinct Comme le MCTS le MinMax va explorer l'arbre des possibles. L'algorithme va explorer toutes les possibilités et mettre une valeur positive ou négative à chaque feuille de l'arbre qui sont des nœuds terminaux ou des nœuds à la profondeur de recherche maximale. Une valeur est positive si la position favorise le joueur que joue l'ordinateur (le joueur maximisant) et négative dans le cas contraire. Les nœuds non feuilles héritent de leur valeur a l'aide de leurs enfants. Ainsi, les nœuds conduisant à un résultat favorable, comme une victoire, pour le joueur maximisant ont des scores plus élevés que les nœuds plus favorables pour le joueur minimisant. Les valeurs heuristiques pour les nœuds feuilles terminaux (fin du jeu) sont des scores correspondant à la victoire, à la défaite ou à l'égalité pour le joueur maximisant. Pour les nœuds feuilles non terminaux à la profondeur maximale de recherche, une fonction d'évaluation estime une valeur heuristique pour le nœud. La qualité de cette estimation et la profondeur de recherche déterminent la qualité et la précision du résultat final de minimax.


Max
Min
Max
Min
Max
1
-2
1
11
-2
9
1
11
6
-2
9
-inf
1
11
+inf
6
-2
14
9
-inf
1
4

5 Algo MinMax observations

6 Algo MinMax observations

une fois l'implémentation de l'algorithme MinMax faite sur nos deux jeux combinatoires nous pouvons nous demander est ce que cet algorithme est adapté pour le jeu de hex et le awale ? Nous observons que les performances du MinMax sont très différentes d'un jeu a l'autre.Étudions pourquoi :

6.1 Hexgame : Mauvaise performance

l'implémentation de l'algo MinMax sur le jeu de Hex n'arrive quasiment jamais a battre un humain et cela pour 2 raisons principales : La faible profondeur et la difficulté de trouver une fonction d'évaluation satisfaisante.

Trop grande complexité Le premier Problème provient du nombre de coup possible a chaque tour : en effet par exemple sur un plateau classique de 13 par 13 le premier joueur a 169 coups possibles et au coup suivant il y en a 168 etc. . .cela rend les calculs lents car par exemple : si il joue en premier et que la profondeur demandée est de 6 alors l'ordinateur doit calculer 2.1298467e+13 (ou 169*168*167*166*165*164) fois la fonction d'évaluation ce qui n'est pas réalisable en temps réaliste. Avec une profondeur seulement de 4 comme nous l'avons implémente sur un plateau de 11*11 ou plus l'ordinateur prend déjà plusieurs secondes a calculer le meilleur coup.

Fonction d'evaluation : casse tête Le MinMax n'arrivant quasiment jamais a une feuille terminale la fonction d'évaluation du jeu de Hex devait être performante et de faible cout en temps. Or évaluer si une position est gagnante ou non s'est révélé très difficile et couteux. Et cela en grande partie a cause du fait que le hex est un jeu qui possède de nombreuses stratégies, par exemple au début de la partie il est préférable de jouer au millieu du plateau pour se laisser le maximum de possibilités mais plus tard dans la partie jouer au centre n'est pas forcement une bonne idée. Un autre aspect fondamental du jeu compliqué a faire comprendre a un ordinateur ce sont les ponts (cf exemple) dans ces cas la le joueur est assurer de pouvoir connecter 2 pièces a la condition que l'autre joueur ne joue pas a cet endroit. Il faut donc ne pas jouer a cet endroit pour jouer autre part mais si l'adversaire joue a cet endroit il faut absolument compléter le pont. On peut noter que quand la profondeur est plus grande le MinMax arrive souvent a voir les ponts mais on en revient au problème n°1 : la complexité. Notre fonction d'évaluation finale essaye de créer les groupes de hex les plus grands en largeur ou en hauteur en fonction du joueur elle bat parfois des humains (surtout sur des petits plateaux) mais prend du temps et des que le taille du plateau est trop grande le problème de complexité devient ingérable

6.2 Awale : Bonne performance

tout le contraire du hex : En effet au Awale le nombre maximum de coup possible est toujours de 6 cela réduit grandement la complexité nous pouvons mettre un profondeur de 10 par exemple sans craindre un trop long temps de calcul cela rend l'algo bien meilleur dans l'ensemble. De plus une fonction d'évaluation naturelle apparait avec les règles : il suffit que le joueur maximisant maximise ses points et minimise ceux de sont adversaire pour gagner. Chose que fait très bien l'algorithme, En effet avec une profondeur de 8 et cette fonction simple et très rapide nous n'arrivons pas a battre l'ordinateur.

Conclusion En conclusion nous pouvons dire que l'algo MinMax fonctionne très bien quand la profondeur maximale est grande et que le jeu n'est pas trop complexe a comprendre pour un ordinateur. Notons que dans le cas du Hex le principal problème est celui du temps de calcul si nous pouvions mettre une grande profondeur maximale même avec une fonction d'évaluation relativement simple l'algorithme jouerai bien.

7 Gestion du Projet

8 Bilan et Conclusions

9 Bibliographie

10 Annexes

10.1 Annexe I : Cahier des charges

Objectifs du projet :

Pour répondre aux attentes de notre projet, nous avons décidé de développer une application web sur laquelle nous pourrions jouer au jeu du Hex et de l'Awalé.

Pour chaque jeu nous devons implémenter les règles officielles afin d'assurer une cohérence entre tous les joueurs. De plus, nous voulons fournir une interface graphique agréable pour l'utilisateur. Cela comprend donc une page d'accueil intuitive ainsi que des plateaux de jeu faciles d'utilisation.

Description des jeux :

- Hex :

  Le Hex est un jeu de stratégie à deux joueurs. Il se compose d'un plateau, de pions bleus et de pions rouges. Le plateau du jeu de Hex est composé de cases hexagonales formant un losange. La taille du plateau peut varier, mais est généralement de 11×11. Deux côtés opposés du losange sont bleus, les deux autres sont rouges.

  Le joueur bleu commence. Les joueurs jouent chacun à leur tour. A chaque tour, un joueur place un pion de sa couleur sur une case libre du plateau. Le premier joueur qui réussit à relier ses deux bords par un chemin de pions contigus de sa couleur a gagné. Il ne peut y avoir qu'un pion par case. Les pions posés le sont définitivement, ils ne peuvent être ni retirés, ni déplacés.

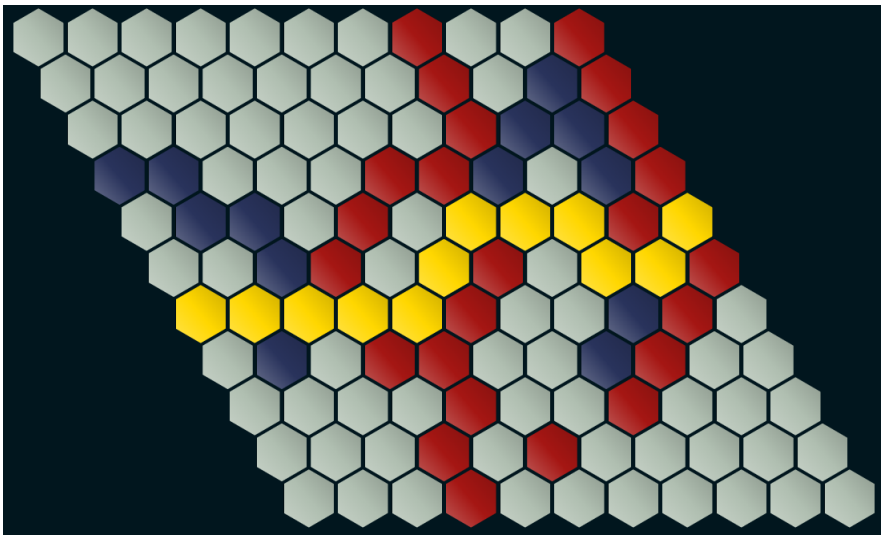


FIGURE 1 – Ici bleu a gagné, en reliant ses deux bords.

- Awalé :

  L'Awalé est un jeu de stratégie à deux joueurs. Il se compose d'un plateau et de graines. Au début de la partie, 4 graines se situent dans chaque cases du plateau. Les joueurs jouent à tour de rôle. A chaque tour, le joueur prend toutes les graines d'un des trous de son camp puis il les égrène une par une dans toutes les cases qui suivent sur sa rangée puis sur celle de son adversaire suivant le sens de rotation (une graine dans chaque trou après celui où il a récupéré les graines).

  Le joueur « capture » des graines lorsque la dernière case où il pose une graine est une case du camp adversaire et si contient 2 ou 3 graines en comptant la nouvelle (elle contenait 1 ou 2 graines avant). Le joueur prend alors les graines de cette case (2 ou 3), puis il prend également les graines

de la case précédente si celle-ci répond aux mêmes conditions : être une case du camp adverse et contenir 2 ou 3 graines. Il continue ainsi à prendre les graines des cases antérieures tant que celles-ci répondent aux conditions.

La première façon qu'une s'achève est lorsqu'un joueur n'a plus de graines dans son camp alors que c'est à lui de jouer et que son adversaire n'est plus en mesure de lui en apporter une selon la règle de « l'obligation de nourrir l'adversaire ». Dans ce cas, son adversaire gagne toutes les graines restantes. C'est la fin par famine.

La deuxième façon qu'une partie s'achève est lorsqu'il reste trop peu de graines pour qu'aucune prise ne soit désormais possible (en pratique 2 ou 3). Chaque joueur récupère la ou les graines restantes de son camp. C'est la fin par indétermination.

Fonctionnalités de l'application :

- Possibilité de jouer en joueur contre joueur sur les deux jeux.
- Possibilité de jouer en joueur contre IA avec la couleur de notre choix sur les deux jeux.
- Possibilité d'observer une partie entre deux IA sur les deux jeux.
- Possibilité d'annuler son dernier coup sur les deux jeux.
- Choisir la taille du plateau du Hex.
- Divers boutons pour naviguer à travers l'application.
- Possibilité de réinitialiser le plateau des deux jeux lorsqu'on le souhaite.

Interface utilisateur :

L'application s'ouvrira sur une page d'accueil sur laquelle nous pourrons choisir le jeu auquel nous souhaitons jouer. Pour chaque jeu nous pourrons trouver une page home nous permettant de selectionner le mode de jeu que nous voulons choisir. Nous retrouverons alors pour les deux jeux les modes joueur contre joueur, joueur contre IA et IA contre IA.

De nombreux boutons de couleur bleus seront mis en place afin de naviguer entre toutes les pages de l'application. D'autres boutons plus petit seront disponibles. Ils permettront pendant les diverses parties de réinitialiser le plateau de jeu, ou de défaire notre dernier coup dans les modes de jeu joueur contre joueur et joueur contre IA.