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 abstraits 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 clarifier ce que l'on entend par « jeu 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 à 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. À 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 un algorithme d'intelligence artificielle en particulier pour la résolution des jeux de Hex et d'Awalé : l'algorithme MinMax avec élagage alpha-bêta.

Les principaux avantages de ces algorithmes sont les suivants :

- L'algorithme MinMax 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. Étant 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

3.1 Description des modules utilisés

Pour la réalisation de ce projet, nous avons développé un logiciel qui permet de jouer aux jeux de stratégie combinatoire abstraits du Hex et de l'Awalé. Ce logiciel est composé de plusieurs modules, dont les principaux sont les suivants :

- **Module Hex :** Ce module contient les classes et les fonctions nécessaires pour jouer au jeu de Hex. Il contient notamment la classe `HexBoard` qui représente le plateau de jeu et les joueurs du jeu de Hex. Ce module contient également les fonctions pour l'implémentation de l'algorithme MinMax avec élagage alpha-bêta pour la résolution du jeu de Hex.
- **Module Awalé :** Ce module contient les classes et les fonctions nécessaires pour jouer au jeu de l'Awalé. Il contient la classe `AwaleBoard` qui représente le plateau de jeu et les joueurs du jeu de l'Awalé. Ce module contient également les fonctions pour l'implémentation de l'algorithme de Dijsktra pour la résolution du jeu de l'Awalé.
- **Module Interface Graphique :** Ce module contient les fichiers HTML, CSS et JavaScript nécessaires pour l'implémentation de l'interface graphique des jeux de Hex et d'Awalé. Il contient notamment les fichiers `home.html`, `hex.html` et `awale.html` qui permettent à l'utilisateur de choisir le jeu auquel il veut jouer et les paramètres de la partie. Il contient également des fichiers JavaScript pour la gestion des événements et des interactions avec l'utilisateur.
- **Module Tests :** Ce module contient les fichiers de tests unitaires pour les classes et les fonctions des modules Hex et Awalé. Il contient notamment les fichiers `test_hex.py` et `test_awale.py`, qui permettent de tester les classes et les fonctions des modules Hex et Awalé. ***TODO : Ajouter les tests unitaires***

Fonctionnalités de l'interface graphique L'interface graphique de notre logiciel comprend une page d'accueil qui permet à l'utilisateur de choisir le jeu auquel il veut jouer (Hex ou Awalé), une page principale pour chaque jeu qui permet à l'utilisateur de choisir les paramètres de la partie (taille du plateau (pour Hex), mode de jeu (joueur contre joueur, joueur contre ordinateur, ordinateur contre ordinateur), niveau de difficulté (pour l'ordinateur), etc.), et une page de jeu qui permet à l'utilisateur de jouer au jeu choisi. L'interface graphique est implémentée en HTML, CSS et JavaScript, et elle communique avec les modules Hex et Awalé via des appels de fonctions JavaScript à des fonctions Python via des requêtes json et le module Flask.

Structures de données Les principales structures de données définies dans le cadre de ce projet sont les classes `HexBoard` et `AwaleBoard` qui représentent les plateaux de jeu du Hex et de l'Awalé, respectivement. Ces classes contiennent les attributs et les méthodes nécessaires pour représenter les plateaux de jeu et les joueurs, et pour effectuer les opérations de jeu (placement de pions, déplacement de pions, etc.). Les données en entrée des programmes sont sous forme de chaînes de caractères qui représentent les paramètres de la partie (taille du plateau, mode de jeu, niveau de difficulté, etc.). Les procédures de lecture et de validation des entrées sont effectuées par les fonctions des modules Hex et Awalé qui vérifient que les paramètres de la partie sont valides avant de commencer la partie.

3.2 Interaction entre fichiers

Les fichiers de l'application interagissent entre eux de la même façon lors d'une partie de Hex ou d'Awalé. Le fichier `app.py` héberge l'application flask et gère l'interaction entre le frontend et le backend.

Initialisation de la partie La sélection du jeu et des paramètres de la partie se fait dans le fichier `home_hex.html` (respectivement `home_awale.html` pour l'Awalé). Ces informations seront envoyées au fichier `app.py` lors de l'appui sur l'un des boutons de lancement de partie. Le bouton sélectionné appellera dans le fichier `app.py` la fonction adéquate renvoyant la page html correspondant aux choix fait par le(s) joueur(s). En parallèle, une instance du plateau de jeu désiré sera créée dans l'`app.py`.

Déroulement d'une partie Lorsqu'une partie est jouée, l'information des pièces placées au Hex, ou des graines déplacées à l'Awalé est récupérée par le fichier JavaScript adapté (par exemple `game_hexia.js` si la partie est une partie de Hex joueur contre l'ordinateur, ou encore `game_awale.js` si la partie est une partie d'Awalé joueur contre joueur). Cette information est gérée par le capteur d'événements hex.onclick pour le Hex et pit.onclick pour l'Awalé. Si un tel événement est entendu, le fichier JavaScript correspondant envoie une requête json au fichier `app.py` afin de savoir si le coup est valide. La validité du coup est gérée par une fonction appelée dans la classe du plateau de jeu adéquat. Si le coup est valide, alors celui-ci est joué et rajouté à l'instance du plateau de jeu, et est renvoyé au fichier JavaScript qui va s'occuper du nouvel affichage graphique (En modifiant le CSS ou le html). Si une erreur est attrapée, elle est renvoyée au fichier JavaScript qu'il va gérer en affichant une alerte sur l'écran du joueur indiquant la non-validité du coup. Lors du placement de chaque hex (déplacement de graines pour l'Awalé), la fonction *check_winner* de la classe du plateau est appelée afin de vérifier si un joueur à gagné. Si c'est le cas, alors la fonction questionnée par la requête json renvoie la variable `game_over` égale à **True**. En récupérant cette variable, le fichier JavaScript va pouvoir procéder à une animation indiquant au(x) joueur(s) que la partie est terminée.

Statistiques Voici quelques statistiques sur le logiciel développé dans le cadre de ce projet :

- Nombre de modules (sans les tests) : 39
- Nombre de classes : 11 (principalement dans les modules Hex et Awalé)
- Nombre de fonctions : 68 (en Python) et 99 (en JavaScript) pour 167 en tout
- Nombre de lignes de code : 1 405 (en Python), 2008 (en JavaScript), 553 (en HTML), 1 331 (en CSS) et 480 (en LaTeX) pour un total de 5 777 lignes de code

4 Moteur de jeux combinatoires abstraits

Implémenter un algorithme pour jouer à des jeux combinatoires contre l'ordinateur est l'un des objectifs principaux du projet Hex-Ta(c)tique mais une problématique importante apparaît : 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-bêta :

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 AlphaZero de Google qui est l'un des leaders en termes d'intelligence de jeux aux échecs. Cet algorithme peut même être implémenté dans des jeux où le hasard apparaît 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 joueurs 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.

À 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 trouverait 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. **LA PHRASE NE VEUT RIEN DIRE :** 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 qui ne sont pas des 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.

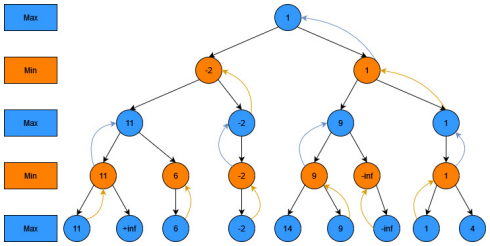

FIGURE 1 – Ici, le joueur bleu cherche à maximiser ses gains avec une profondeur 4.

Élagage alpha-bêta L'algorithme minimax effectue une exploration complète de l'arbre de recherche jusqu'à un niveau donné. L'élagage alpha-bêta permet d'optimiser l'algorithme minimax sans en modifier le résultat. Pour cela, il ne réalise qu'une exploration partielle de l'arbre. En effet, on observe qu'il n'est pas utile d'explorer les sous-arbres qui conduisent à des valeurs qui ne participeront pas au calcul du gain de la racine de l'arbre. Dit autrement, l'élagage alpha-bêta n'évalue pas des nœuds dont on peut penser, si la fonction d'évaluation est à peu près correcte, que leur qualité sera inférieure à celle d'un nœud déjà évalué.

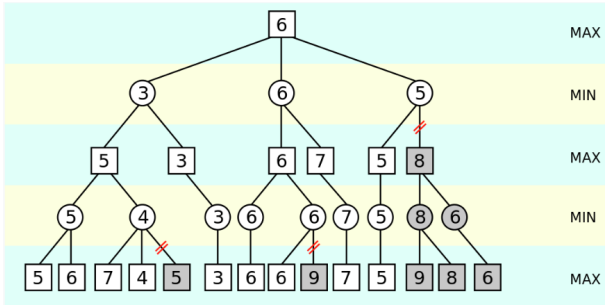

FIGURE 2 – Minmax avec élagage α-β.

Plusieurs coupures ont pu être réalisées. De gauche à droite :

1. Le nœud MIN vient de mettre à jour sa valeur courante à 4. Celle-ci, qui ne peut que baisser, est déjà inférieure à α=5, la valeur actuelle du nœud MAX précédent. Celui-ci cherchant la plus grande valeur possible, ne la choisira donc de toute façon pas.
2. Le nœud MIN vient de mettre à jour sa valeur courante à 6. Celle-ci, qui ne peut que baisser, est déjà égale à α=6, la valeur actuelle du nœud MAX précédent. Celui-ci cherchant une valeur supérieure, il ne mettra de toute façon pas à jour sa valeur que ce nœud vaille 6 ou moins.
3. Le nœud MIN vient de mettre à jour sa valeur courante à 5. Celle-ci, qui ne peut que baisser, est déjà inférieure à α=6, la valeur actuelle du nœud MAX précédent. Celui-ci cherchant la plus grande valeur possible, ne la choisira donc de toute façon pas.

5 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 l'Awale ? Nous observons que les performances du MinMax sont très différentes d'un jeu a l'autre, étudions pourquoi :

5.1 Hexgame : performances décevantes

L'implémentation de l'algorithme MinMax sur le jeu de Hex n'arrive quasiment jamais à 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 coups possibles à chaque tour : en effet, par exemple sur un plateau classique de 13×13 le premier joueur a 169 coups possibles et au coup suivant il y en a 168 etc...Cela rend les calculs lents. S'il joue en premier et que la profondeur demandée est de 6, alors l'ordinateur doit calculer $2.1298467e+13$ (ou $169 \times 168 \times 167 \times 166 \times 165 \times 164$) fois la fonction d'évaluation. Cela n'est pas réalisable en temps réaliste. Dans notre implémentation, la taille des côtés du plateau varie entre une longueur de 5 cases et de 17, nous avons remarqué que pour un plateau de dimension 11×11 l'algorithme mettait déjà plusieurs secondes à calculer le meilleur coup. Notons que ces calculs ne prennent pas en compte l'élagage alpha-bêta qui optimise significativement le MinMax. Mais même si une portion des nœuds n'est pas évaluée, le calcul reste trop lent. On peut aussi noter que la performance de l'élagage alpha-bêta dépend de la qualité de la fonction d'évaluation, qui, comme nous allons le voir, n'est pas assurée.

Fonction d'évaluation : Un vrai casse-tête Le MinMax n'arrivant pratiquement jamais à une feuille terminale, nous avons compris que la fonction d'évaluation du Hex devait être performante et peu coûteuse en temps. Cependant, faire comprendre à l'ordinateur si une position est gagnante ou non s'est révélé difficile. En effet, le Hex est un jeu possédant de nombreuses stratégies, et relier toutes ces stratégies en une seule fonction d'évaluation est quelque chose de très compliqué. Ainsi, nous avons rapidement cherché à faire comprendre à l'ordinateur quelle stratégie adapter au fur et à mesure de la partie ? Parmi elles, la stratégie de faire des ponts (voir la figure 3) nous a posé beaucoup de problèmes. Un pont permet au joueur de s'assurer de pouvoir connecter 2 pièces lors d'un prochain coup. En effet, si le joueur adversaire cherche à bloquer l'une des deux cases, nous avons juste à ajouter notre jeton afin de créer un chemin entre ces deux pièces. Nous pouvons noter que lorsque la profondeur de réflexion du MinMax est grande, il arrive parfois à remarquer que créer un pont le meilleur coup. Mais nous revenons au problème n°1 : la complexité. **PRECISER CETTE PHRASE (un peu confuse) :** Dans notre projet, la fonction d'évaluation finale va permettre à l'ordinateur de créer les groupes de jetons les plus grands en largeur ou en hauteur en fonction du joueur. Elle est cependant capable de battre des humains sur de petits plateaux. Mais en général, lorsque le plateau est de grande dimension, la fonction d'évaluation prend de nombreuses secondes avant de jouer.

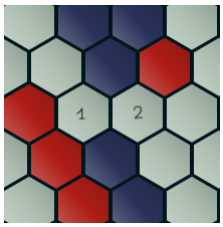

FIGURE 3 – Ici bleu a créé un pont [1].

1. Si rouge joue sur la case 1, bleu joue sur la case 2, et relie ainsi les deux jetons bleus (respectivement si rouge joue sur la case 2)

5.2 Awale : Bonnes performances

Une fonction d'évaluation efficace Le jeu de l'Awalé possède un avantage significatif par rapport au Hex, le nombre maximum de coups possible pour un joueur est de 6. Ainsi, l'arbre de recherche est bien plus petit que celui du Hex. L'algorithme MinMax est donc efficace en termes de temps et de performances avec une grande profondeur (la profondeur initiale est de 6).

Le choix de notre fonction d'évaluation est arrivé naturellement pendant nos recherches. Nous avons implémenté la simple fonction cherchant à maximiser les points de l'ordinateur tout en minimisant les points de son adversaire. Cette fonction combinée à une profondeur de taille 6 en fait un adversaire redoutable que peu d'humains ont réussi à battre.

Conclusion L'algorithme MinMax est donc capable d'être efficace lorsque le nombre de nœuds est petit à chaque itérations. On peut alors lui mettre une profondeur relativement grande afin d'explorer toutes les branches de cet arbre. Cependant, lorsque le nombre de nœuds devient gigantesque, l'algorithme ne devient plus très bon. Il est en effet très lent et peu performant.

6 Gestion du Projet

Pour la gestion du projet, nous avons suivi un cycle de développement similaire à celui de la méthode Scrum. Nous avons donc défini des sprints courts de quelques jours chaque semaine, avec une réunion hebdomadaire le mercredi pour faire le point sur l'avancement du projet. Ces réunions étaient l'occasion de discuter des tâches effectuées, de celles à venir, et de résoudre les problèmes rencontrés. Nous avons également utilisé le logiciel Jira pour gérer les tâches et les sprints, et pour suivre l'avancement du projet.

6.1 Diagramme de Gantt

Officiellement, nous n'avons pas utilisé de diagramme de Gantt pour la gestion du projet, mais nous avons tout de même réalisé un planning prévisionnel des tâches à effectuer. Ce planning a été réalisé en début de projet, et a été mis à jour régulièrement pour refléter l'avancement du projet. Il a été utilisé pour définir les tâches à effectuer pour chaque sprint, et pour suivre l'avancement du projet. Un exemple de diagramme de Gantt est présenté en annexe **??**.

6.2 Changements Majeurs

Comme mentionné dans la section **??**, nous avons effectué des changements majeurs en cours de projet. Ces changements ont été discutés en réunion, et ont été validés par l'ensemble de l'équipe. Ils ont été intégrés au planning prévisionnel, et ont été pris en compte dans la gestion du projet. Les changements majeurs effectués en cours de projet sont les suivants :

- Changement de la hiérarchie des classes : nous avons changé la hiérarchie des classes pour une homogénéisation du code et une meilleure compréhension.
- Changement de l'organisation des fichiers : nous avons changé l'organisation des fichiers pour mieux organiser le code et faciliter la maintenance. Cela a initialement été fait pour le backend, puis pour le frontend. Beaucoup de problèmes ont été rencontrés lors de ces changements, mais ils ont permis d'améliorer la qualité du code.
- Changement de la gestion des erreurs : nous avons changé la gestion des erreurs pour une meilleure gestion des exceptions et une meilleure gestion des erreurs. Cela a permis d'améliorer la robustesse du code. Cela a également renforcé la sécurité du code sans nuire à la performance ou au debuggage.

7 Bilan et Conclusions

Nous avons implémenté la plupart des fonctionnalités prévues dans le cahier des charges, à l'exception de quelques fonctionnalités mineures. Nous avons également ajouté des fonctionnalités supplémentaires qui n'étaient pas prévues dans le cahier des charges, mais qui ont été jugées nécessaires pour améliorer la qualité du projet.

En bref, nous avons implémenté les fonctionnalités suivantes :

- Possibilité pour les utilisateurs de jouer en JcJ au *Hex* et à l'*Awale*
- Possibilité pour les utilisateurs de jouer au JcIA au *Hex* et à l'*Awale*
- Possibilité pour les utilisateurs de regarder deux IA jouer au *Hex* et à l'*Awale*
- Possibilité pour les utilisateurs de changer de thème de couleur pour le jeu du *Hex*
- Possibilité pour les utilisateurs de défaire un coup dans tous les modes de jeu (sauf IAvIA), pour le *Hex* et l'*Awale*
- Possibilité pour les utilisateurs de rejouer une partie dans tous les modes de jeu (sauf IAvIA), pour le *Hex* et l'*Awale*
- Possibilité pour les utilisateurs de choisir la taille du plateau de jeu pour le *Hex*
- Possibilité pour les utilisateurs de choisir son camp (en JcIA) pour le *Hex* et l'*Awale*
- Possibilité pour les développeurs de tester, déployer et maintenir facilement l'application
- Possibilité pour les développeurs de rajouter facilement de nouvelles fonctionnalités
- Possibilité pour les développeurs de rajouter facilement de nouveaux jeux

8 Bibliographie

9 Annexes

9.1 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 jeux 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. À 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.

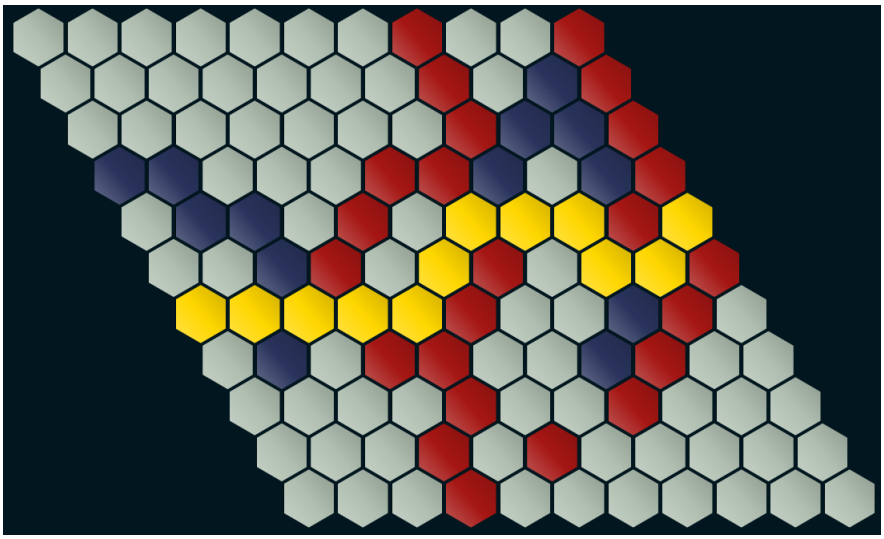

FIGURE 4 – Ici bleu a gagné, ses deux bords sont reliés.

- 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 case du plateau. Les joueurs jouent à tour de rôle. À 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 sélectionner 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 petits 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.

9.2 HexGame et stratégie gagnante

Au Hex, pour toutes les tailles de plateaux, il existe une stratégie gagnante théorique pour le joueur qui commence. Celle-ci n'est pas connue pour la plupart des tailles de plateaux, en effet elle demande une connaissance totale de toutes les parties possible ce qui n'est pas calculable en temps réaliste.

9.2.1 Stratégies gagnantes et arbre du jeu

Modélisons le jeu de Hex à l'aide d'un arbre représentant toutes les parties possibles. L'arbre commence avec la position initiale, un plateau vide. De cette racine partent autant de branches qu'il y a de premiers coups. Ainsi, pour un plateau de taille 2×2, on va retrouver 3 nœuds fils. Nous réitérons l'action à partir de chaque nœuds fils en leur rajoutant des branches vers de nouveaux nœuds correspondant aux autres coups disponibles. Pour le plateau précédent ; chaque fils possède alors 2 nouveaux fils.

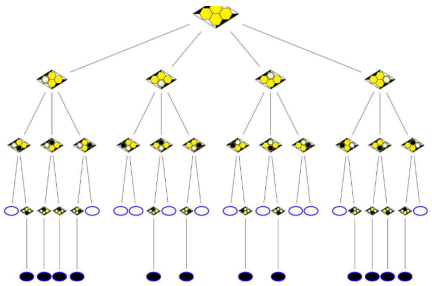

FIGURE 5 – Arbre complet pour un plateau 2×2

L'arbre va nous aider à nous convaincre qu'il y a bien une stratégie gagnante pour l'un des deux joueurs. Le principe consiste à colorier tous les nœuds de l'arbre en blanc ou en noir. Chaque nœud colorié correspond à une position à partir de laquelle le joueur de la couleur correspondante possède une stratégie gagnante. Nous commençons par colorier en blanc (respectivement en noir) les feuilles représentant une victoire par les blancs (respectivement par les noirs). Nous souhaitons maintenant attribuer une couleur à un nœud qui n'en a pas encore. On remarque dans un premier temps que toutes les branches descendantes mènent à des nœuds déjà coloriés. Supposons que ce nœud représente une position où c'est aux blancs de jouer. Si au moins l'une des branches issues du nœud mène à un nœud blanc, alors on colorie le nœud en blanc. Sinon, si toutes les branches mènent à des nœuds noirs, on le colorie en noir. On procède de la même façon si c'est aux noirs de jouer. De cette manière, nous sommes assurés de remplir l'arbre en entier de couleurs. La racine se voit donc attribuer une couleur. Ainsi, le premier joueur possède une stratégie gagnante.

Remarque : Pour un plateau de dimension 2×2, l'arbre possède un total de 24 feuilles et 52 branches. il est calculé que pour le Hex a 11*11 la Game-tree complexity ou le nombre de feuilles de l'arbre est approximativement 10^{98} et le nombre de positions totales que peut prendre le jeu est de $2.4 * 10^{56}$ contre $4.2 * 10^{56}$