DATE : 02/02/2026

RAPPORT HEX

DUCREUX Clémence, GENTIL William, JANNAIRE Clément,
KARA-MOSTEFA Riad, VAISSE Alistair

Sommaire

Introduction

Dans le cadre de ce projet de BUT3, nous avons réalisé un jeu de plateau implémenté en Java en utilisant l'API fournie par le département. Le jeu choisi est Hex, un jeu de stratégie à information parfaite, opposant deux joueurs dont l'objectif est de relier deux côtés opposés du plateau par une chaîne continue de pions.

L'objectif principal de ce projet était de concevoir un moteur de jeu fonctionnel, accompagné de tests, ainsi que de bots capables de jouer automatiquement, dont au moins un bot utilisant une approche algorithmique avancée (alpha-bêta, Monte Carlo ou fonction d'évaluation). Le projet devait également respecter des contraintes de qualité logicielle telles que la documentation du code, l'organisation du dépôt Git et la capacité à déployer et tester facilement l'application sur une machine de l'IUT.

Le jeu Hex se joue à deux joueurs sur un plateau hexagonal. Chaque joueur dispose d'une couleur et joue à tour de rôle en plaçant un pion sur une case libre du plateau. Chaque joueur cherche à relier deux côtés opposés du plateau correspondant à sa couleur à l'aide d'un chemin continu de pions. Il n'y a pas de match nul possible dans Hex : la partie se termine obligatoirement par la victoire de l'un des deux joueurs dès qu'un chemin continu est formé.

Méthodologie

Notre groupe est composé de cinq membres. Afin de travailler efficacement et de limiter les conflits lors du développement, nous avons choisi de diviser le projet en quatre grandes tâches, dont l'une a été réalisée par deux personnes en collaboration. Cette organisation nous a permis de répartir le travail de manière équilibrée et de travailler en parallèle sur différentes parties du projet.

Pour faciliter la collaboration, nous avons utilisé Git comme outil principal de gestion de versions. Chaque tâche disposait de sa propre branche, permettant à chaque membre (ou binôme) de travailler de manière indépendante sans impacter directement le travail des autres. Les fonctionnalités étaient ensuite intégrées progressivement dans la branche principale après validation.

Le dépôt Git reflète cette organisation à travers l'utilisation de branches dédiées ainsi que la présence de tâches/tickets permettant de suivre l'avancement du projet. Les classes principales sont documentées à l'aide de Javadoc, et un README explique clairement comment lancer les démonstrations et tests du programme sur une machine de l'IUT.

Tout au long du projet, nous avons adopté une approche itérative, en privilégiant d'abord une version fonctionnelle du moteur de jeu avant d'ajouter progressivement des améliorations, notamment sur les bots et les outils de test. Cette démarche nous a permis d'identifier rapidement les problèmes de conception et de valider les choix techniques étape par étape, plutôt que de chercher à implémenter toutes les fonctionnalités dès le départ.

Cette méthodologie présente plusieurs points positifs, notamment une bonne séparation des responsabilités, une réduction des conflits lors des fusions et une meilleure lisibilité globale du projet. En revanche, certaines intégrations ont nécessité des ajustements, et certaines décisions techniques prises tôt dans le projet ont parfois limité les possibilités d'évolution sans modifications importantes du code.
Dans l'ensemble, cette organisation nous a permis de mener le projet à son terme de manière structurée, tout en acquérant une première expérience concrète de travail collaboratif sur un projet de développement logiciel.

Les tâches

Le jeu de Hex a été développé en Java en s'appuyant sur l'API fournie par le département. Le moteur de jeu est principalement implémenté dans les classes HexBoard et HexPly, qui assurent la représentation du plateau, la validité des coups, l'alternance des joueurs ainsi que la détection des conditions de fin de partie.

Un affichage textuel du plateau a été mis en place via la méthode toString() de HexBoard. Cet affichage console a servi d'outil de débogage tout au long du développement afin de visualiser l'état du jeu à chaque tour et de vérifier le bon fonctionnement du moteur ainsi que le comportement des différents bots.
Des tests fonctionnels et démonstrations ont été réalisés à travers plusieurs méthodes main, notamment dans la classe HexMain. L'application peut être exécutée en mode interactif avec un joueur humain, ou en mode automatique grâce à l'option autoplay, permettant de simuler des parties complètes sans intervention humaine. Dans ce mode, les coups sont fournis automatiquement par les bots via le moteur de jeu, ce qui a permis de valider la stabilité du système et le bon enchaînement des coups.

Une arène de simulation a également été mise en place afin de comparer différents bots entre eux sans interaction utilisateur. Ce mode repose sur le même moteur HexBoard, mais délègue la décision des coups aux bots (RandomBot, HeuristicBot, MiniMaxBot, MonteCarloBot). La classe Simulation assure l'orchestration automatique des parties en appliquant les coups retournés par les bots sur le plateau, tandis que la classe Arena permet d'exécuter des confrontations multiples entre stratégies afin d'observer leur comportement et d'évaluer leur qualité de jeu.
Concernant les bots, plusieurs stratégies ont été implémentées.
Un bot aléatoire (RandomBot) joue des coups légaux choisis au hasard.
Un bot heuristique (HeuristicBot) sélectionne les coups selon une évaluation simple de la position.

Un bot plus avancé (MiniMaxBot) repose sur une recherche Minimax avec élagage alpha-bêta. Cette recherche explore l'arbre des coups possibles en alternant les tours des deux joueurs et sélectionne le meilleur coup à la racine. La profondeur de recherche est adaptée selon la taille de l'arbre : exhaustive en fin de partie (petit arbre) et limitée par un cut-off en début ou milieu de partie afin de maîtriser le coût de calcul. L'évaluation des positions utilise une heuristique simple basée sur la proximité au centre du plateau, ce qui permet d'obtenir un comportement cohérent tout en conservant des temps de calcul raisonnables.

L'organisation Git repose sur une répartition en tâches avec des branches dédiées, des commits réguliers et une documentation des classes principales via Javadoc. Un fichier README décrit également la procédure pour lancer les démonstrations et tests du programme.

Les bots

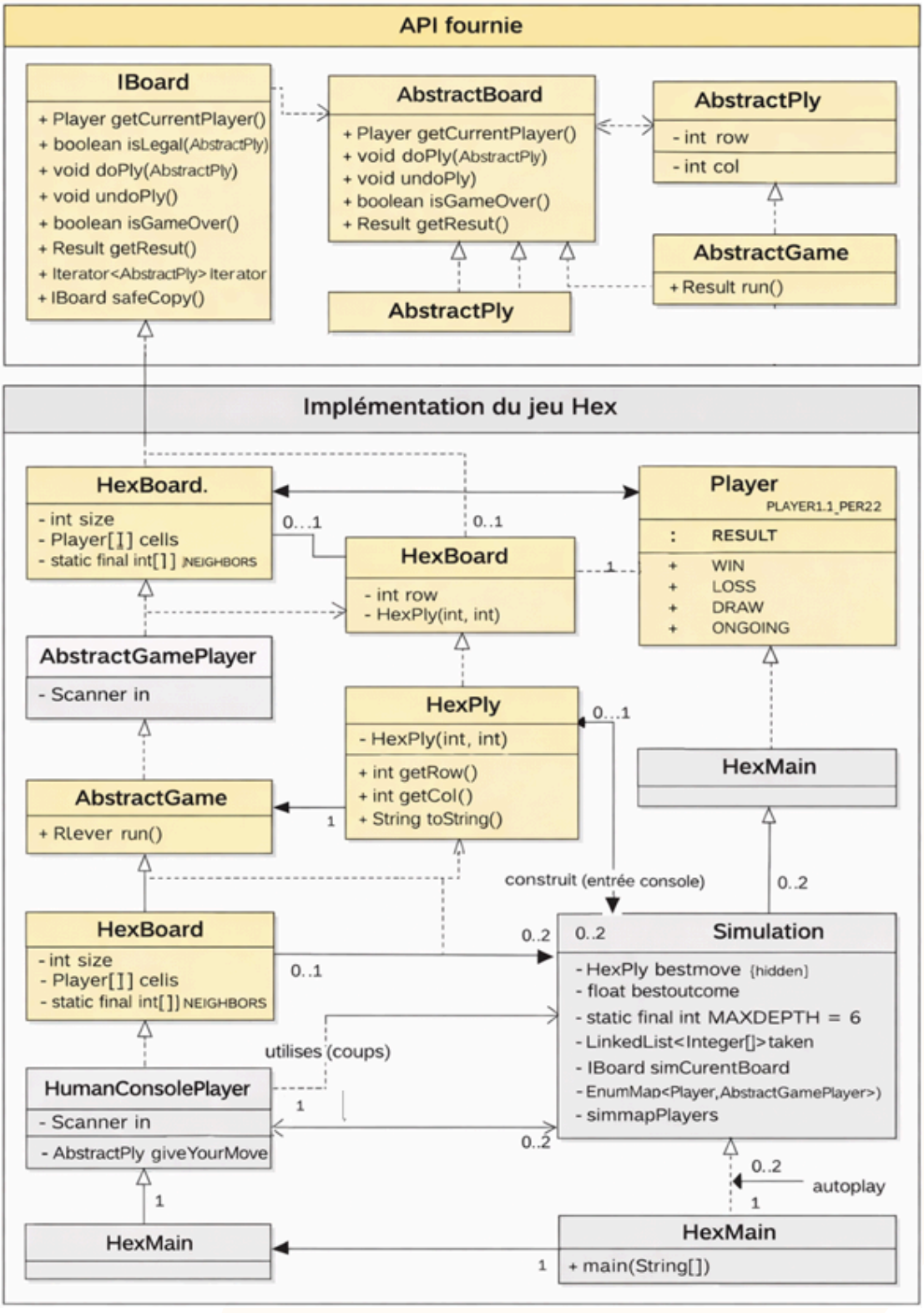
Plusieurs bots ont été développés afin de tester le moteur de jeu et de comparer différentes approches de prise de décision. Ces bots vont d'un comportement aléatoire à des algorithmes de recherche plus avancés. Parmi ces différentes implémentations, le bot basé sur une recherche Minimax est le plus abouti et constitue le meilleur bot du projet.

Notre meilleur bot : recherche Minimax à profondeur limitée

Le bot Minimax principal est implémenté dans la classe MiniMaxBot. Il repose sur une recherche Minimax avec élagage alpha-bêta explorant l'arbre des coups possibles en alternant les tours des deux joueurs. Les coups sont générés à partir des coups légaux du plateau Hex, puis appliqués et annulés lors de la recherche afin d'explorer les différentes branches de l'arbre.
La profondeur de recherche est adaptée dynamiquement selon la taille de l'arbre de jeu. Lorsque le nombre de coups restants est faible (fin de partie), la recherche est exhaustive jusqu'aux positions terminales. Lorsque l'arbre est plus grand (milieu ou début de partie), une profondeur maximale fixe (cut-off) est utilisée afin d'éviter l'explosion combinatoire du nombre de positions explorées.
Le meilleur coup trouvé à la racine est sélectionné puis joué sur le plateau.

Fonction d'évaluation :
Les positions terminales sont évaluées par des scores extrêmes correspondant à la victoire ou à la défaite du joueur Minimax.
Lorsque la profondeur limite est atteinte, une heuristique simple basée sur la proximité des pions au centre du plateau est utilisée afin d'estimer la qualité de la position.
Cette heuristique favorise le contrôle du centre, généralement stratégique dans le jeu de Hex, mais ne prend pas encore en compte des caractéristiques plus spécifiques comme la connectivité ou les structures de ponts.
L'utilisation de l'élagage alpha-bêta permet de réduire significativement le nombre de positions explorées tout en conservant la qualité de la décision.
L'adaptation de la profondeur entre recherche exhaustive en petit arbre et recherche avec cut-off en grand arbre améliore le compromis entre temps de calcul et qualité des coups.

Diagrammes de classes



Diagrammes de classes

Le projet s'appuie sur l'API fournie (GameAPI) qui impose une structure classique : un plateau (IBoard / AbstractBoard), des coups (AbstractPly), des joueurs (AbstractGamePlayer) et une boucle de jeu (AbstractGame). Le jeu de Hex est ensuite implémenté en spécialisant ces abstractions avec des classes concrètes adaptées au jeu.
La classe centrale côté modèle est HexBoard, qui représente l'état du plateau (taille size et grille cells). Elle implémente les opérations essentielles du moteur : vérification de la légalité d'un coup (isLegal), application et annulation d'un coup (doPly, undoPly), génération des coups possibles via l'itérateur (iterator) et détection de la fin de partie (isGameOver, getResult). L'affichage de débogage repose sur la méthode toString(), permettant d'inspecter facilement une position en console.
Les coups sont représentés par HexPly, une classe simple qui stocke les coordonnées (row, col). Elle sert d'objet de transfert entre les joueurs (humains ou bots) et le moteur, et elle est utilisée partout où l'API attend un AbstractPly.

Pour les joueurs humains, HumanConsolePlayer hérite de AbstractGamePlayer et fournit une implémentation concrète de giveYourMove(IBoard). Ce joueur lit une entrée console, vérifie sa validité et retourne un HexPly. Cette classe a été particulièrement utile pendant le développement pour tester le moteur manuellement.
Les joueurs automatiques (bots) héritent également de AbstractGamePlayer. Le bot principal MiniMaxBot implémente une recherche Minimax avec élagage alpha-bêta afin de sélectionner le meilleur coup à jouer. D'autres stratégies ont également été implémentées, notamment RandomBot, HeuristicBot et MonteCarloBot, permettant de comparer différents niveaux de jeu dans l'arène.
La classe Simulation hérite de AbstractGame et redéfinit la méthode run() afin d'exécuter automatiquement une partie entre deux joueurs (humains ou bots) sans interaction utilisateur. Elle se contente d'orchestrer la partie en appelant giveYourMove sur les joueurs fournis et en appliquant les coups sur le plateau HexBoard. Contrairement aux bots, elle ne calcule pas elle-même la stratégie de jeu mais délègue la décision aux joueurs.

La classe Arena permet d'exécuter des confrontations automatiques entre plusieurs bots afin de comparer leurs performances. Elle instancie différentes stratégies, lance des simulations répétées et collecte les résultats des parties.
Enfin, la classe HexMain constitue le point d'entrée du programme : elle instancie un HexBoard, construit la map de joueurs (EnumMap<Player, AbstractGamePlayer>) et lance soit une partie interactive avec AbstractGame, soit une partie automatique via Simulation lorsque l'argument autoplay est fourni.

Précision : il y a eu un changement mineur à l'API au niveau de AbstractGamePLayer, l'ajout d'une fonction jeSuisMinimax, qui permet de corriger certains bugs pour l'arène sans quoi les bots jouaient des coups illégaux.

Conclusion individuelle

Clémence : Ce projet m'a permis de mieux comprendre la mise en place d'un moteur de jeu et l'implémentation d'un bot. Il m'a également aidé à progresser sur l'organisation du code et le travail collaboratif avec Git.

William :

Clément : Ce projet m'a permis de comprendre les aspects théoriques liés à la représentation d'un jeu comme Hex, notamment la gestion du plateau et des règles. Il m'a aussi montré l'importance d'une conception rigoureuse pour garantir la cohérence du jeu et faciliter le travail des autres parties du projet.

Riad :

Alistair : J'ai trouvé le projet assez intéressant, en particulier lorsqu'il fallait se pencher sur les méthodes d'évaluation d'une position. Voir l'algorithme de ABminmax en action est assez fascinant, et je trouve que c'est une bonne application des notions qu'on a travaillé en cours. Certaines contraintes de temps dans la dernière semaine m'ont empêcher de faire tout ce que je voulais, mais dans l'ensemble, je suis plutôt satisfait du travail effectué.

Conclusion collective

Ce projet a permis à notre groupe de mettre en pratique les notions vues en cours, aussi bien sur le plan algorithmique que sur le plan méthodologique. Malgré certaines difficultés liées à la coordination et à l'intégration des différentes parties, nous avons réussi à produire un projet fonctionnel, documenté et conforme aux attentes.

La répartition des tâches et l'utilisation de branches Git dédiées ont été des éléments clés dans la réussite du projet. Ce travail nous a permis de mieux appréhender les enjeux d'un développement collaboratif sur un projet de taille moyenne.