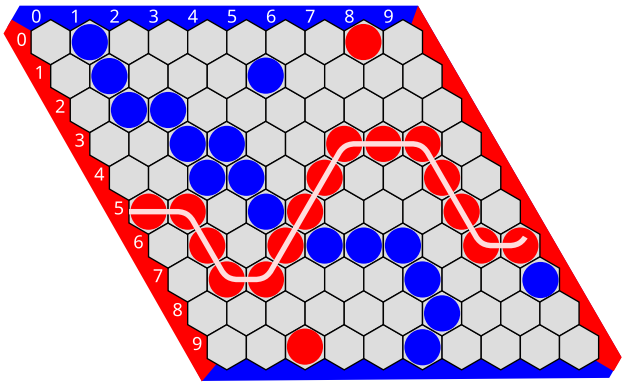
TP2 : Jeu Hex

1 Description du problème

Dans ce projet, vous êtes chargé de développer un agent capable de jouer au jeu Hex. Voici les règles du jeu (source : Wikipedia `https://fr.wikipedia.org/wiki/Hex`).
Le jeu commence avec le joueur X. Les joueurs placent à tour de rôle un pion de leur couleur sur une case vide du plateau. Une fois placé, un pion ne peut plus être déplacé ni retiré jusqu'à la fin de la partie. Un joueur gagne dès qu'il réussit à former un chemin continu de pions de sa couleur reliant ses deux côtés opposés du plateau (de gauche à droite pour X et de haut en bas pour O). Il est important de noter qu'un match nul est impossible dans Hex ; il n'existe aucune configuration de jeu où aucun des joueurs ne peut gagner.



Vous pouvez jouer contre vous-même dans l'environnement de jeu pour mieux assimiler les règles. Pour jouer un coup, cliquez simplement sur la case choisie. Si le coup est valide, votre pion sera placé sur le plateau.

2 Instructions

Votre mission consiste à implémenter une intelligence artificielle utilisant l'algorithme Monte Carlo Tree Search (MCTS). Pour l'évaluation, chaque joueur dispose de 100 seconds de temps de réflexion pour l'ensemble de la partie. Le dépassement de ce délai d'une minute entraîne la perte de la partie.

3 Langages de programmation

L'utilisation du langage C++ est **TRÈS FORTEMENT RECOMMANDÉE**, un squelette de code vous étant fourni. Vous devrez modifier le fichier `src/IA_Player.h` en redéfinissant les méthodes suivantes :

- `void otherPlayerMove(int row, int col)`, appelée après chaque coup de l'adversaire.
- `std::tuple<int, int> getMove(Hex_Environment& hex)`, qui doit déterminer et retourner le coup à jouer par votre IA.

Dans le code initial, les coups sont choisis de manière aléatoire.
Si vous optez pour un autre langage de programmation, assurez-vous que votre projet soit compatible avec l'environnement de jeu :

— Votre programme sera lancé avec comme premier argument X si votre agent joue les rouges, ou O s'il joue les bleus.
— Les échanges avec le programme se feront via l'entrée et la sortie standard, en utilisant des coordonnées sous la forme "ij", où "i" représente le numéro de la ligne (entre 0 et 9) et "j" le numéro de la colonne (également entre 0 et 9).
— Votre programme doit pouvoir être compilé et exécuté dans l'environnement de correction (Linux Ubuntu 23.10).

4 Environnement de jeu

Pour compilier l'environement :

```
$ mkdir build
$ cd build
$ cmake ..
$ make
```

Pour connaître les différentes options du jeu vous pouvez l'exécuter avec la commande "`--help`" :

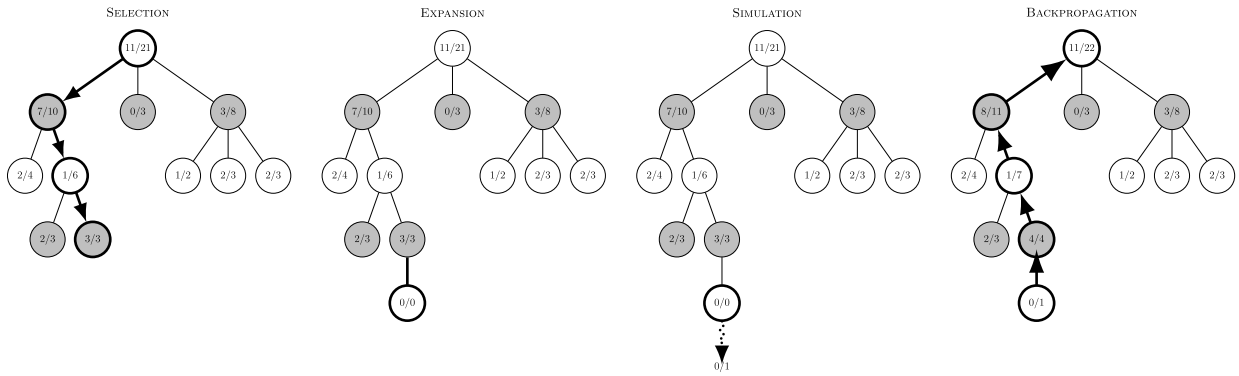
```
$ ./Hex --help
Environnement pour le jeu Hex
Usage: ./Hex [OPTIONS]

Options:
  -h,--help     Print this help message and exit
  -X TEXT       Si "-X IA" alors le joueur X (rouge) sera joué par l'IA.
                Sinon, il s'agit d'un path vers un programme externe.
  -O TEXT       Si "-O IA" alors le joueur O (bleu) sera joué par l'IA.
                Sinon, il s'agit d'un path vers un programme externe.
  --slow        Ajoutez un délai entre chaque coup de l'IA
  --noGUI       Désactiver l'affichage graphique
  --size UINT   Taille du plateau de jeu (default: 10, seulement 10 si appel
                à un programme externe)
  --seed UINT   Seed pour l'aléatoire (par défaut: 0 pour seed aléatoire)
```

5 Évaluation

Votre programme sera évalué face à un certain nombre d'IA de différents niveaux. Votre évaluation dépendra de la performance de votre IA et du respect des directives. Tous les joueurs artificiels seront également comparés lors d'un tournoi virtuel. Les 5 meilleures IA recevront un point bonus supplémentaire.

6 Algorithme MCTS

L'algorithme Monte Carlo Tree Search (MCTS) est une méthode de recherche heuristique utilisée pour la prise de décision dans certains types de jeux, tels que les jeux de plateau. L'idée principale de MCTS est d'utiliser la simulation aléatoire pour découvrir les mouvements les plus prometteurs. Cet algorithme est divisé en quatre étapes principales : Sélection, Expansion, Simulation, et Rétropropagation.



1. **Sélection** : À partir de l'état actuel du jeu, l'algorithme parcourt l'arbre de jeu existant en utilisant une stratégie spécifique, comme l'Upper Confidence Trees (UCT), pour sélectionner le prochain nœud à explorer. Ce processus se poursuit jusqu'à ce qu'un nœud non exploré soit atteint.
2. **Expansion** : Si le nœud sélectionné n'est pas un état terminal du jeu, un ou plusieurs nouveaux nœuds sont créés pour représenter les mouvements possibles à partir de cet état.
3. **Simulation** : À partir de l'état nouvellement créé, l'algorithme exécute une simulation aléatoire du jeu jusqu'à ce qu'un état terminal soit atteint. Cela donne comme résultat la victoire ou la défaite.
4. **Rétropropagation** : Le résultat de la simulation est ensuite propagé à travers l'arbre, des nœuds enfants vers les nœuds parents, en mettant à jour les statistiques de ces nœuds (le nombre de victoires et le nombre de simulations passées par le nœud).

L'algorithme répète ces étapes jusqu'à ce que le temps imparti soit écoulé ou un nombre de simulations prédéfini soit atteint. Le mouvement avec la meilleure évaluation est alors choisi comme le mouvement à effectuer dans le jeu réel.

6.1 Stratégie Upper Confidence Trees (UCT)

La stratégie Upper Confidence Trees (UCT) permet la sélection de nœuds dans l'algorithme MCTS, équilibrant ainsi entre l'exploration de nouveaux mouvements et l'exploitation des mouvements connus pour être efficaces. La formule UCT pour un nœud spécifique S et un coup i est donnée par :

$$UCT(S,i) = \frac{S[i].win}{S[i].nb} + c\sqrt{\frac{\ln(S.nb)}{S[i].nb}}$$

Avec :

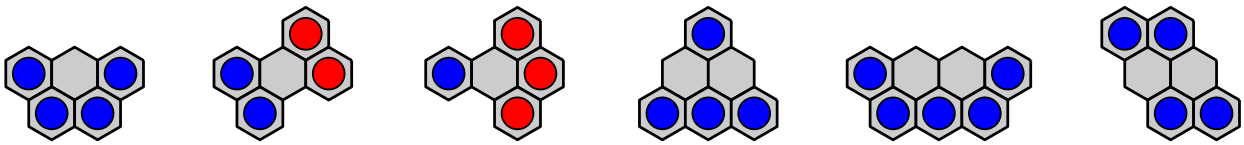
— $S[i].win$ le nombre de fois que le coup i a été gagnant à partir de S
— $S[i].nb$ le nombre de simulations du coup i à partir de S
— $S.nb$ le nombre total de simulations à partir de S
— c est une constante qui, en théorie, est égale à $\sqrt{2}$, mais en pratique, dépend du contexte. Il est donc nécessaire de la choisir expérimentalement.

7 Optimisations

Vous avez terminé d'implémenter MCTS et votre IA n'est toujours pas très forte ? C'est normal, sur une taille de 10x10, des optimisations sont nécessaire pour rendre votre IA performante.

7.1 Fillin pruning

Le "Fillin pruning" est une technique d'optimisation qui consiste à réduire l'espace de recherche en éliminant les mouvements inutiles. Dans le contexte du jeu de Hex, cela signifi ignorer des coups qui sont clairement inférieurs à d'autres options. Cette approche permet de se concentrer sur les simulations de mouvements plus pertinents, rendant l'algorithme MCTS plus efficace et rapide. Voici quelques exemples dans lesquels les cases vides peuvent être complétées par les pions de n'importe quel joueur sans changer l'issue du jeu :



7.2 RAVE (Rapid Action Value Estimation)

Avez-vous remarqué que des permutations d'une séquence de coups conduisent à une même position ? L'idée de l'algorithme RAVE est de supposer que la valeur d'un coup dépend peu de l'ordre dans lequel il est joué. Ainsi, si on n'a pas encore joué de simulation à partir d'une position S, autant se baser sur les résultats des simulations où des coups similaires ont été joués dans des positions différentes.
Pour cela, en plus de conserver la valeur moyenne d'un coup i à partir d'une position S pour calculer le score $UCT(S,i)$, on peut également y incorporer $R(S,i) = \frac{\tilde{w}}{\tilde{n}}$, avec $\tilde{w}$ et $\tilde{n}$ qui représentent le nombre de jeux gagnés contenant le coup i et le nombre de tous les jeux contenant le coup i.
Ainsi, en pondérant UCT et R en fonction de n et $\tilde{n}$ nous obtenons :

$$score(S,i) = (1-w) \times UCT(S,i) + w \times R(S,i)$$

Pour la pondération w, il existe de nombreuses formules dans la littérature, mais l'idée est que si n est petit, alors on veut utiliser majoritairement R, et si n est grand, on veut utiliser majoritairement UCT, avec par exemple : $w = \frac{1}{1+0.1\times n}$.

8 Directives

— Vous avez la possibilité de modifier ou d'ajouter des fichiers sources à votre projet, à l'exception de la classe `Player_Interface` et du nom de la classe de votre IA doit rester `IA_Player`. Votre programme doit également rester compatible avec le fichier `main.cpp`. Ce dernier sera automatiquement remplacé pour l'évaluation. Veillez donc à ne déclarer aucun élément dans ce fichier qui serait nécessaire au fonctionnement de votre IA.
— Le devoir est à rendre par Moodle avant le 11 mars 23h59 (5% de pénalité par jour de retard).
— Rendre les fichiers source seulement. Aucun fichier binaire. (supprimer le dossier build de votre archive)
— Compilable et exécutable sans modifications et via cmake pour C++.

Des points de pénalité seront appliqués en cas de non-respect des directives pour la remise.