Objectif du projet : Créer une application en Java offrant la possibilité de jouer à plusieurs jeux à deux joueurs.

Rappels :

Tout au long du projet, vous devrez toujours respecter une architecture MVC.

Afin de coder ce projet, nous vous autorisons à utiliser une IA d'assistance à la génération de code. Vous êtes libres du choix de cette IA.

Vous devez également utiliser des identifiants (noms des classes, noms des attributs et des méthodes) en français.

Itération 3 :

Étape 1 : Améliorer le travail rendu lors de l'itération 2 si besoin.

Étape 2 : On va maintenant coder un autre mode de jeu de l'IA que le mode naïf codé pendant la séance de TP. On demandera donc au joueur de choisir le mode de jeu de l'IA avant de commencer une nouvelle partie.

Vous pourrez réfléchir à l'utilisation d'un design pattern pour gérer les différentes façons de jouer de l'IA sachant qu'on pourrait en ajouter d'autres.

Le second mode de jeu de l'IA sera basé sur l'algorithme minimax.

Algorithme Minimax

L'algorithme minimax est un algorithme qui s'applique à la théorie des jeux pour les jeux à deux joueurs à somme nulle, c'est à dire que le gain d'un des joueurs entraîne obligatoirement la perte pour l'autre joueur. Dans un jeu à somme nulle, la somme des gains de tous les joueurs est égale à 0. Par exemple, si l'on définit le gain d'une partie d'Othello comme 1 si on gagne, 0 si la partie est nulle et -1 si on perd, Othello est un jeu à somme nulle.

Cet algorithme 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 opposant. Le meilleur choix est alors celui qui minimise les pertes du joueur tout en supposant que l'opposant cherche au contraire à les maximiser.

Le principe est basé sur la construction de l'arbre du jeu.

Arbre du jeu

Un arbre de jeu consiste à représenter sous forme d'un arbre orienté toutes les positions que l'on peut atteindre à partir de celles du coup précédent.

Selon le jeu, cet arbre devient très vite énorme. Aux échecs par exemple, les blancs ont le choix entre 20 coups (16 coups de pions et 4 coups de cavaliers) pour débuter la partie. Cela fait donc déjà 20 branches qui partent du sommet. Ensuite, les noirs ont aussi 20 coups à disposition. Cela fait 400 positions possibles après 2 coups, puisque 20 branches partent des 20 positions précédentes !

Excepté pour des jeux très simples comme le Morpion, il est impossible de dessiner l'arbre de jeu complet.

Principe de l'algorithme minimax

On visite l'arbre de jeu pour faire remonter à la racine une valeur (appelée valeur du jeu) qui est calculée récursivement.

Soit p un nœud de l'arbre de jeu et f une fonction d'évaluation de la position du jeu. Alors :

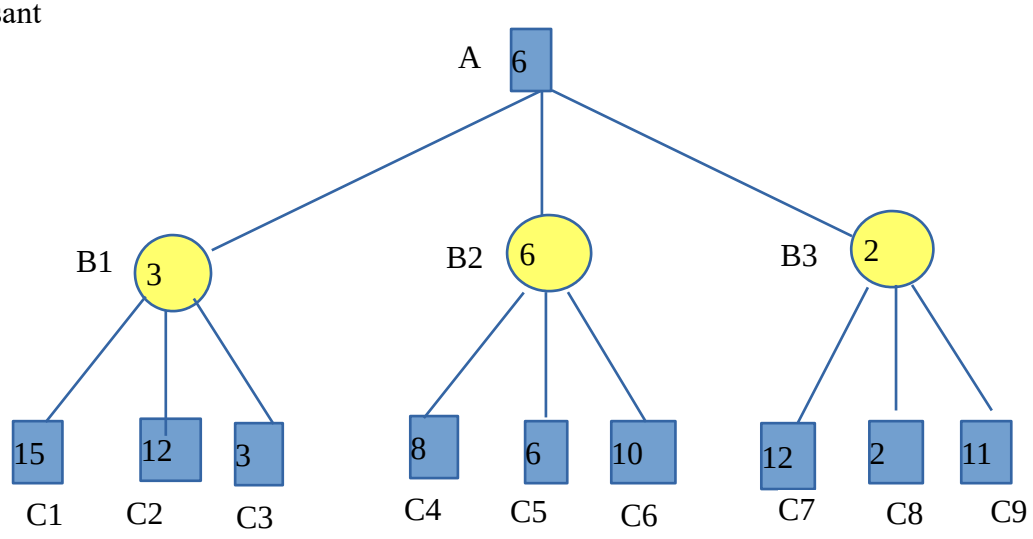
• valeur(p) = f (p) si p est une feuille de l'arbre

• valeur(p) = MAX(valeur(O1), …, valeur(On)) si p est un nœud Joueur ayant pour fils Oi

• valeur(p) = MIN(valeur(O1), …, valeur(On)) si p est un nœud Opposant ayant pour fils Oi.

Exemple :

Dans le schéma ci-dessous, les nœuds bleus représentent les nœuds Joueur et les jaunes les nœuds Opposant



Le nœud A prend donc la valeur 6. Le joueur doit donc jouer le coup l'amenant en B2. En observant l'arbre, on comprend bien que l'algorithme considère que l'opposant va jouer de manière optimale : il prend le minimum. Sans ce prédicat, on choisirait le nœud C1 qui propose le plus grand gain et le prochain coup sélectionné amènerait en B1. Mais alors on prend le risque que l'opposant joue C3 qui propose seulement un gain de 3.

En pratique, la valeur théorique de la position P ne pourra généralement pas être calculée. En conséquence, la fonction d'évaluation sera appliquée sur des positions non terminales. On considérera que plus la fonction d'évaluation est appliquée loin de la racine, meilleur est le résultat du calcul. C'est-à-dire qu'en examinant plus de coups successifs, nous supposons obtenir une meilleure approximation de la valeur théorique donc un meilleur choix de mouvement.

C'est de cette façon que l'on déterminera la force d'un programme : plus il descendra bas dans l'arbre, plus il sera redoutable (en supposant qu'il a une bonne fonction d'évaluation). La profondeur ne devra d'ailleurs pas forcement être la même pour tous les nœuds.

Fonction d'évaluation

Cette fonction a pour but d'évaluer une position : plus la valeur obtenue est grande, meilleure est la position.

Il s'agit donc de définir une fonction qui traduise les avantages ou les inconvénients des placements des jetons dans une position donnée. Par exemple, un jeton placé dans un coin rapportera beaucoup de points puisqu'on ne peut plus le retourner. Il est aussi plus intéressant d'avoir un jeton sur un bord.

Trouver une bonne fonction d'évaluation n'est pas simple.

Voici la fonction d'évaluation pour une couleur donnée que vous implémenterez :

Si la partie est terminée, on affecte une valeur de 1000 points si la couleur donnée correspond au vainqueur. Sinon, la valeur sera de -1000.

Un jeton placé dans un coin apporte 11 points, un jeton placé sur un bord apporte 6 points et un jeton placé ailleurs apporte 1 point.

Travail à rendre juste avant votre séance de TP de la semaine du 24/03 : vous déposerez sur CELENE une archive au format zip contenant le diagramme de classe correspondant au modèle de conception de votre application et le code de votre projet.

Vous ferez la soutenance du votre projet pendant la séance de TP.