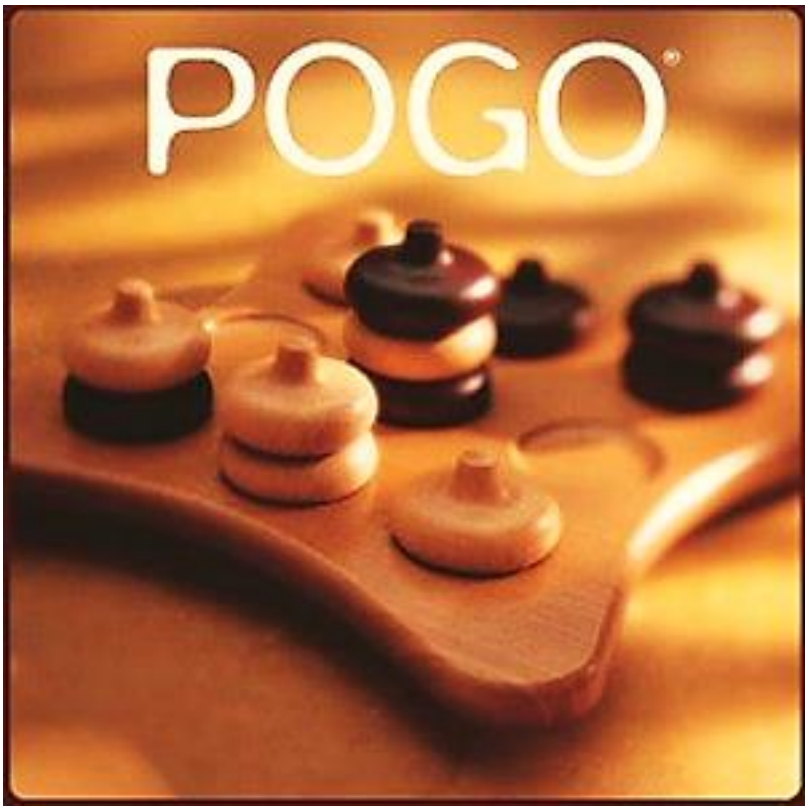
Projet IA41

Jeu du Pogo



MONTANGE Aimeric

GUILLEMAILLE Timothée

PEGON Sylvain

BENMESSAOUD Yamine

Printemps 2019

Sommaire

Introduction

Dans le cadre de l'UV IA41, nous avons à développer un projet consistant à implémenter un jeu, puis de concevoir une intelligence artificielle capable d'y jouer. Parmi les différents sujets proposés, nous avons choisi de travailler sur le sujet traitant du jeu "Pogo", un jeu de plateau où la stratégie est de mise afin de pouvoir gagner un grand nombre de parties.

Le but de ce projet est alors de développer une application en langage C++ dotée d'une intelligence artificielle en s'appuyant sur le jeu de société Pogo qui était jusqu'alors inexistant en version virtuelle.

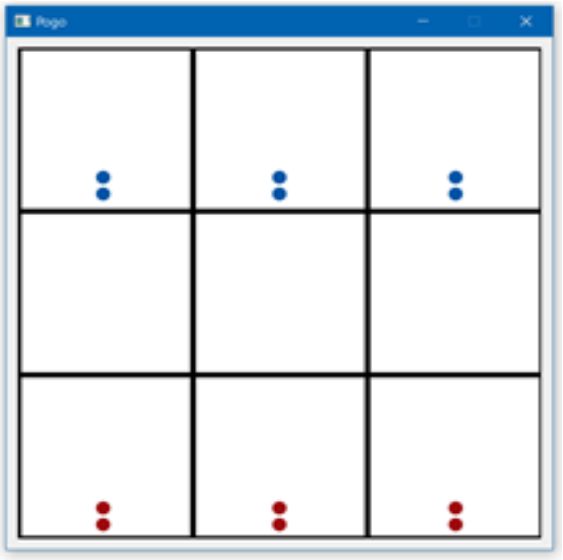
Spécification

Formulation du projet

Pour bien comprendre la conception de l'IA et des stratégies mises en place, nous rappelons rapidement les règles et les concepts du jeu.

Le "Pogo" est un jeu de plateau dans lequel deux joueurs s'affrontent. Le plateau est composé de neufs emplacements (nœuds) sur lesquels les joueurs placent des jetons

Initialement, chaque joueur possède 6 jetons, soit bleus soit rouges (les 6 jetons étant de la même couleur), disposés comme ceci :



C'est alors au joueur bleu de commencer la partie.

Une pile appartient au joueur qui possède le jeton le plus haut qu'importe le reste de sa composition. Un joueur peut déplacer jusqu'à trois jetons (les plus haut) en même temps d'une pile qu'il contrôle pendant un tour. La distance de déplacement est alors égale au nombre de jetons déplacés. Ainsi, il peut donc déplacer des jetons d'une autre couleur tant qu'il possède le jeton du sommet de la pile.

Pour gagner, un joueur doit contrôler toutes les piles donc avoir un jeton de sa couleur sur chaque sommet de chaque pile.

Notre première méthode pour spécifier ce jeu, était de commencer par jouer entre nous sur un plateau physique, afin de bien prendre en main les règles du jeu, mais aussi de commencer à voir quelle stratégie mettre en place pour la suite du projet, et il était clair que plusieurs stratégies étaient envisageables pour vaincre l'adversaire.
Nous devions aussi coder le jeu lui-même. Cette phase a vraiment été importante pour visualiser quelles classes abstraites permettraient la représentation informatique du jeu.

De plus nous avons vite remarqué lors des différentes parties jouées que le joueur ayant le plus grand nombre de piles de jetons avec un grand nombre de jeton les composants, était en meilleur situation de gagner que son adversaire. En effet les petites piles sont contraintes à effectuer des déplacements limités. Ainsi la possibilité de gagner de nouvelles tours est directement réduite. Nous avons également pu constater que la stratégie offensive est aussi intéressante que la stratégie défensive.
Néanmoins, notre intelligence artificielle devra étudier tous les scénarios de jeu possibles durant les n prochains tours, et sera d'autant plus dure à battre que n est grand.

Cahier de Charges

Le cahier des charges était très concret, nous devions former un groupe composé de deux à quatre personnes afin de concevoir et produire un programme de type jeu.
Ce programme doit répondre également à plusieurs attentes et consignes. Celui-ci doit être fonctionnel avec plusieurs choix pour l'utilisateur. Il pourra choisir entre plusieurs modes de jeu (User1 contre User2, User contre IA, IA1 contre IA2). De même, il pourra décider s'il souhaite jouer avec les jetons bleus ou rouges.
Enfin, il pourra également paramétrer la force de l'IA dans la partie.

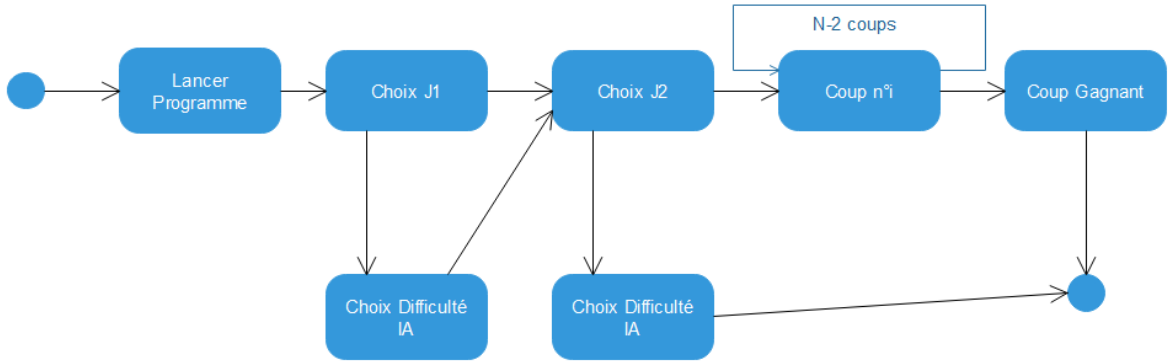
Choix du langage et de l'IDE

Afin de concevoir ce programme, nous avons décidé d'utiliser le Framework QT Creator, car celui-ci était connu du groupe grâce à différentes UV d'informatiques l'utilisant.
Cet IDE permet créer des programmes avec un code source en langage C++. Nous avons choisi ce langage de programmation étant donné que l'ensemble des membres de notre groupe a des connaissances dans ce langage.
De plus, nous avions pensé dans un premier temps à implémenter l'application en C++ et l'IA en Prolog mais la liaison entre ces deux langages s'est avérée trop complexe à mettre en place. De ce fait, nous aurions pris trop de temps à implémenter l'intelligence artificielle, ce qui n'était pas souhaitable car nous voulions plus nous consacrer à la conception de cette dernière.
Nous avons donc choisis l'IDE QT Creator puisque celui-ci nous permet de créer facilement l'interface graphique requise pour former l'interface du jeu rapidement et se concentrer par la suite sur le fonctionnement de l'IA elle-même.

Analyse du projet

Diagramme d'action

Suite à la spécification du projet et aux contraintes et exigences mises en place par le groupe, nous avons par la suite conçu un diagramme d'action qui résume les actions que l'utilisateurs doit faire s'il utilise le programme.
Pour expliquer brièvement celui-ci, l'utilisateur lancera le programme et devra ensuite configurer le jeu, c'est donc la phase où il sera libre de choisir le type des deux joueurs et les difficultés des IA (si elles sont utilisées). S'en suit la phase de jeu, c'est-à-dire que la prise en compte des règles que l'utilisateur a renseigné concernant les joueurs. La partie commence ensuite.



Conception de l'interface

À partir de ce diagramme d'action, nous avons pu concevoir la première fenêtre. Celle-ci est très sobre, afin de ne pas perdre l'utilisateur dans des informations trop complexes.
En outre, l'interface Homme/Machine n'étant pas au cœur des objectifs du projet, cette dernière est fonctionnelle mais non optimisée d'un point de vue ergonomique.

Méthode proposée

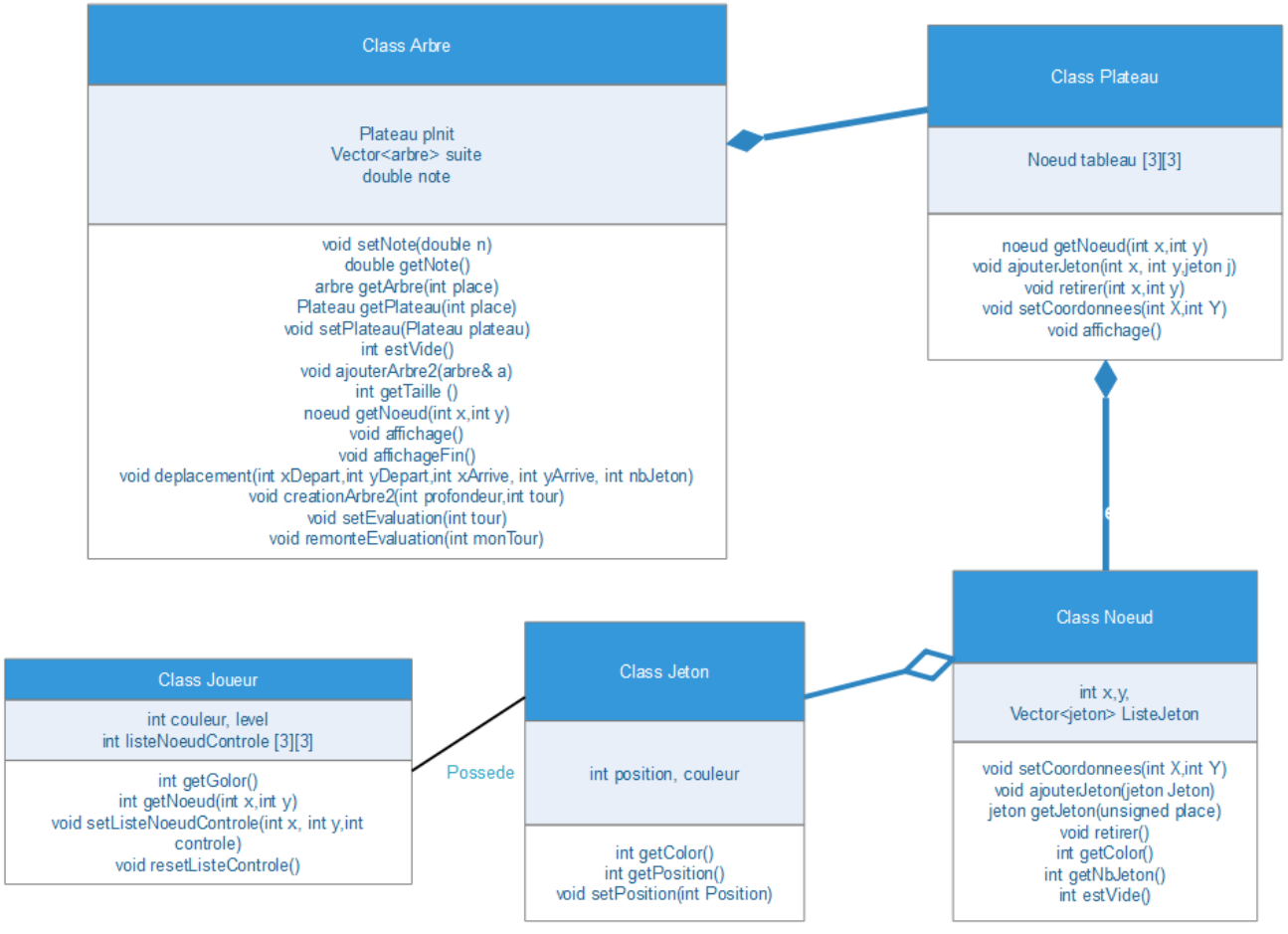
Diagramme de classe

À partir des choix que nous avons énuméré précédemment, nous avons réfléchis sur les classes abstraites permettant de représenter virtuellement le jeu. Nous avons donc construit un diagramme de classe UML représentant les bases de notre programme.

Tout d'abord, nous avons réfléchi au jeu lui-même. Pour les cases, nous avons choisi de les représenter comme des nœuds, caractérisés par leurs positions sur le plateau de jeu et par leurs compositions, c'est à dire des jetons. Les jetons sont quant à eux représentés par une couleur pour savoir à quel joueur ils appartiennent ainsi que par leurs positions dans la pile si une pile est présente sur ce nœud. Enfin, les plateaux de jeu sont composés de nœuds, changeant de caractéristiques pendant la partie.

Pour l'IA que nous expliquerons plus précisément par la suite, elle sera caractérisée par la construction d'arbres comprenant des plateaux de jeu avec une valeur de jeu qui correspondra à la qualité du plateau suivant le joueur concerné.

Ainsi, le diagramme ci-dessous permet également d'avoir une vision plus claire de ce que nous proposons pour la réalisation de ce projet.



Choix algorithme IA

Afin de choisir le type d'algorithme, nous nous sommes intéressés au type de jeu qu'était le "Pogo". Nous avons décidé de la stratégie à adopter pour l'intelligence artificielle en se basant sur les préceptes de l'algorithme "Min-Max". En effet le "Pogo" est un jeu à somme nulle qui se joue à deux joueurs. Ainsi, il est parfaitement adapté à l'algorithme Min-Max.

Principe de fonctionnement de l'IA

Le principe de notre IA est donc basé sur celui d'un algorithme "Min-Max". Ainsi, après qu'un adversaire ait joué, l'IA en déduit tous les scénarios de jeu possibles sur plusieurs tours (en fonction de la puissance de l'IA). Elle va ensuite donner une valeur de jeu à chacun des scénarios pour ensuite choisir quel scénario lui est le plus favorable afin de faire perdre l'adversaire ou de lui-même gagner la partie.

De ce fait l'IA génère tous les plateaux possibles en fonction des n prochains coups (ordre de grandeur de 20^n plateaux).

Il applique ensuite une fonction d'évaluation sur les plateaux de l'arbre à la profondeur n auquel il va attribuer une valeur de jeu, c'est-à-dire une note correspondant à la situation de jeu (est-elle favorable ou non pour l'IA active). La fonction évaluation analyse trois critères :

- Le nombre de piles contrôlées
- Le nombre de piles contrôlées par l'adversaire
- Le nombre de pions contrôlé.

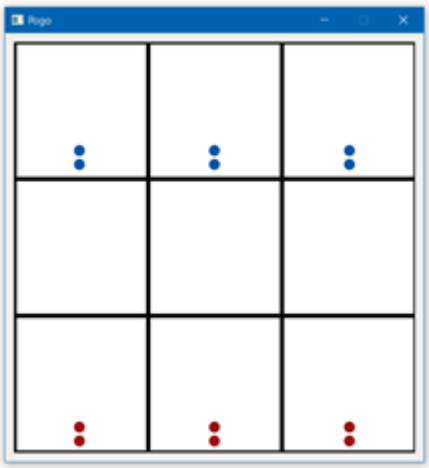
Une valeur de jeux est déduite de ces trois paramètres, et c'est sur cette valeur de jeu que l'IA va se baser pour juger si un plateau est perdant, gagnant ou plus ou moins bien qu'un autre plateau.

Puis, on applique un algorithme Min-Max qui va remonter la valeur de jeu aux plateaux de profondeurs 1. Grace à cette remonté des valeurs de jeu des plateaux, l'IA choisira ensuite le coup avec la meilleure valeur de jeu dans les différents plateaux de profondeur 1, afin d'être en bonne position pour gagner la partie lors de prochains coups ou le même principe sera opéré.

Plus l'IA voie loin (c'est-à-dire en profondeur dans les arbres, elle réfléchira donc sur un plus grand nombre de coup à la suite) plus elle sera à même de choisir le coup qui lui permettra de se mettre dans une bonne situation future. Ce ne sera pas forcément le meilleur coup sur le moment, mais le meilleur pour gagner la partie car ça vision de jeu est réfléchit sur plusieurs coup en avance.

Implémentation

Lorsque l'on lance le jeu, le plateau est initialisé comme en début de partie.

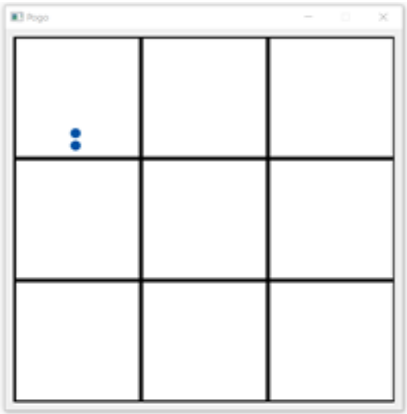


Pour réaliser le jeu, la classe **Plateau** nous permet de modéliser le plateau sur lequel on va venir, au lancement du jeu en ajoutant 12 jetons :

```
plateau.ajouterJeton(0,0,(jeton(1,1)));
plateau.ajouterJeton(0,0,(jeton(2,1)));
```

Les deux premiers chiffres correspondent aux coordonnées du jetons, les deux suivants correspondent à la hauteur (les pions se mettent les uns sur les autres ainsi dès l'initialisation on indique aux pions qu'il est le premier ou le nième jeton) ainsi qu'à la couleur du jeton (1=bleu, 2= rouge).

Par exemple, les deux lignes ci-dessus correspondent ainsi à la configuration suivante :



Ensuite il existe deux possibilités de jeu :

C'est le tour d'un joueur

Le programme attend un premier clic sur une case de la couleur du joueur puis sur une autre case. Une fonction va vérifier si le déplacement est réalisable sous les conditions suivantes :

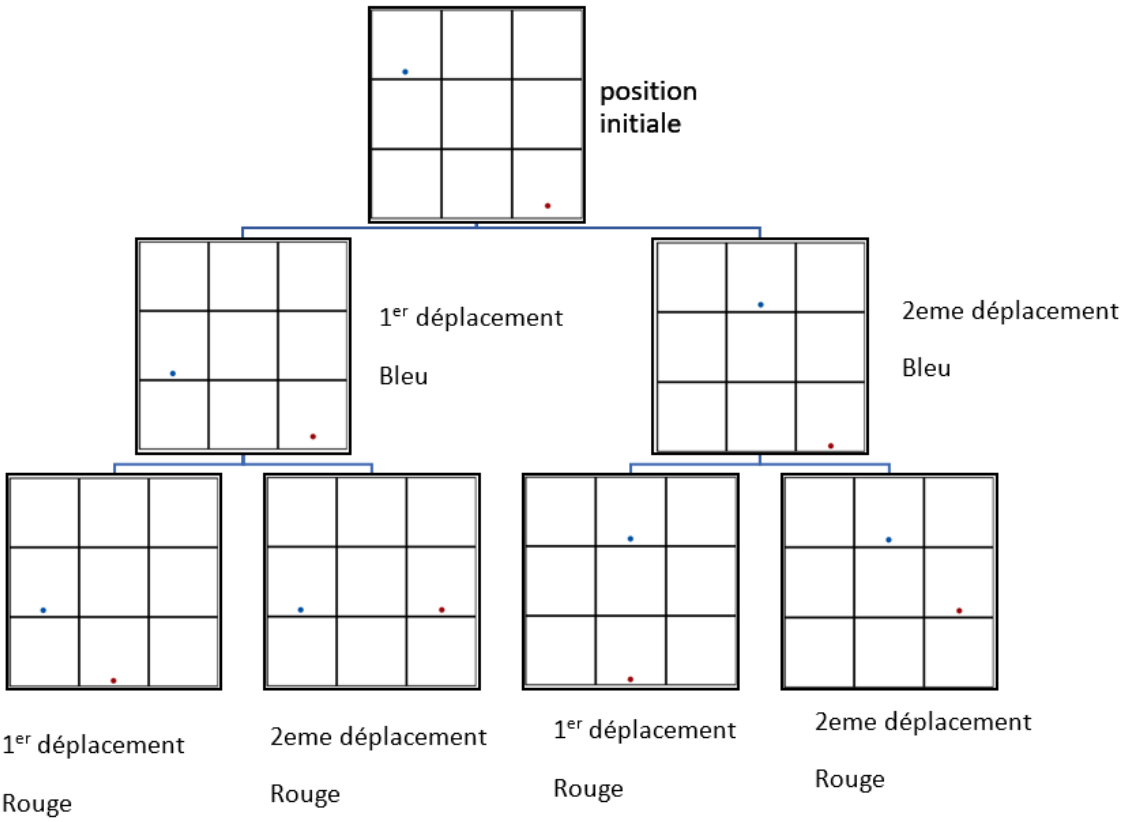
- La distance entre les cases est comprise entre 1 et 3
- Il y a un nombre suffisant de jetons sur la case (supérieur ou égale à la distance)

Si ces deux conditions sont remplies, le programme va venir modifier le plateau de jeu en retirant n fois le premier jeton de la pile du nœud de départ et en ajoutant n jetons sur le nœud d'arrivée.

Une fois le déplacement des pions est effectué, le programme appelle la méthode finTour().

C'est le tour d'une IA

L'IA doit pouvoir analyser les différentes possibilités de jeu pour choisir quel coup est le meilleur pour elle. Pour cela, elle devra créer un arbre partant de la position initiale, relatant tous les plateaux possibles (comme le montre le graphe suivant sur une profondeur de 2 avec 2 pions pour simplifier) :



C'est la méthode **creationArbre**, définie récursivement, qui va créer cet arbre. Pour ce faire, la méthode prend en paramètre la profondeur de l'arbre à créer ainsi que deux entiers qui représentent une couleur (2 pour bleu et 3 pour rouge).
La variable **monTour** définit la couleur du joueur en train de jouer alors que la variable **tour** définit la couleur du joueur qui va jouer à cette profondeur précise.
La valeur de **tour** changera donc à chaque appel plus profond de la méthode, alors que la variable **monTour** restera le même.

La profondeur de l'arbre initiale est définie par le niveau de l'IA : une IA de niveau 1 créera un arbre d'une profondeur, alors qu'une IA de niveau 3 créera un arbre de profondeur 3.

```
void creationArbre(int profondeur,int tour,int monTour) //instatie tout l'arbre avec une profondeur définis {...
```

Cette méthode va déterminer tous les plateaux qu'il est possible de créer à partir du plateau initial en faisant un mouvement d'un, deux ou trois jetons. Pour cela, la méthode va parcourir tout le plateau case par case. Une fois pour la case de départ d'un déplacement de coordonnées (i,j) et une fois pour la case d'arrivée de coordonnée(I,J).

```
for (int i=0;i<3;i++)//parcours du plateau
{
    for (int j = 0; j < 3; ++j)
    {
        if (this->pInit.getNoeud(i,j).getColor()==tour)//si le jeton au dessus de la pile est de la couleur du joueur
        {
            for (int I=0;I<3;I++)//alors on va reparcourir tout le plateau
            {
                for (int J=0;J<3;J++)
                {
                    if (J!=j || I!=i)//si on changé de case
                    {
```

Si les deux cases sont différentes, le programme va tester si un déplacement est possible avec un, deux ou trois jetons.

```
for(int k=0; k<min(this->getNoeud(i,j).getNbJeton(),3)+1;k++)//on teste si il y a un deplacement possible sur les 3 premiers jetons
{
    if (deplacementPossible(i,j,I,J,k,this->pInit,tour)==1)//si le deplacement est possible
    {
```

Si un déplacement est possible, alors le programme va créer un arbre auquel il va attribuer le plateau initial avec le déplacement de jeton en question. On ajoute le nouvel arbre créé à notre arbre initial.
Puis, si on ne parvient pas à la profondeur souhaitée, la méthode **creationArbre** est appliquée à l'arbre en question (en prenant soin de changer la couleur du tour, mais pas celle de **monTour**)

```
arbre temp;//crée un arbre

temp.setPlateau(this->pInit);//on crée un arbre

temp.deplacement(i,j,I,J,k);//definis le nouveaux plateau comme le plateau actuel auquel on effectue un deplacement
temp.setEvaluation(monTour);//on evalue le plateau

this->ajouterArbre(temp);//on ajoute le plateau à l'arbre

if(profondeur!=1)//puis on crée les arbres recursivement
{
    this->suite.at(unsigned(nb)).creationArbre(profondeur-1,3-tour,monTour); //on crée l'arbre du pateau que l'on vient d'ajouter
}
```

De plus, pour chaque arbre créé, on instancie sa note. La note d'un arbre correspond à sa valeur de jeu. Pour se faire, la méthode **setEvaluation** va analyser le plateau et attribuer la valeur de jeu au plateau. Cette méthode prend en paramètre la couleur du joueur en train de jouer.

La fonction **setEvaluation** analyse ensuite le plateau en fonction des deux critères qui sont le nombre de jetons contrôlés et du nombre de piles contrôlées. La méthode va donc parcourir tout le plateau puis incrémenter trois variables (nombre de jetons contrôlés, le nombre de piles que l'on contrôle et le nombre de piles que l'adversaire contrôle) en fonction des jetons présent sur la case. Puis elle va instancier une variable **« eval »** qui va être égale à ces deux critères.

```
eval=((nbTourControle/(2*nbTourAdversaire))/(nbTourControle+nbTourAdversaire)+(nbJetonControle/12));
```

Une fois tout l'arbre créé, on applique un algorithme « Min-Max » pour pouvoir faire remonter les valeurs de jeux des plateaux inferieurs aux plateaux supérieurs.
Pour cela on appelle la méthode récursive **remonteEval** qui va parcourir tout l'arbre. Chaque arbre va alors recevoir la valeur min ou max des valeurs de jeu de ses arbres enfants.

```
if (monTour)//si on regarde un coup jouer par nous on prends la valeur max
{
    note=max(note,suite.at(k).note);
}
else {//sinon on prend la min
    note=min(note,suite.at(k).note);
}
```

Une fois que toutes les évaluations sont remontées selon l'algorithme « Min-Max », la fonction choix plateau va choisir le plateau avec la note la plus élevée. Ce plateau est renvoyé à la fenêtre d'affichage et la méthode **finTour()** est appelée.

La méthode **finTour()** va remettre à zéro le tour, actualiser l'affichage, et va regarder si un des joueurs contrôle toutes les tours. Si c'est le cas la victoire est annoncée et le programme se termine. Sinon, on redémarre un autre tour.

Tests

Dans un premier temps nous avons effectué des tests sur le jeu et l'affichage lui-même, en jouant les uns contre les autres sans aucune IA. Une fois le jeu opérationnel nous avons testé les différents niveaux de difficultés de l'IA en jouant contre lui.

Le jeu contre une IA était possible et fonctionnait. L'IA de plus haut niveau paraissait plus dure à battre que ceux de plus bas niveaux. Cependant lors du test IA contre IA nous nous sommes aperçus que lorsque l'IA1 était de niveau 1 et que l'IA2 de niveau 3 (plus forte que le l'IA1), l'IA1 gagnait ce qui n'est normalement pas possible car celle-ci ne réfléchis pas assez loin dans le jeu pour pouvoir avoir une chance contre une IA de niveau supérieur.

Tout d'abord, nous avons cru que la/les cause(s) de cet échec pouvaient venir d'un mauvais choix de critères d'évaluations et de leurs notations, ou bien d'un problème au niveau du choix du plateau lors de l'exploration des arbres.
Afin de bien comprendre d'où venait le problème, nous avons de nouveau joué contre l'IA et nous nous sommes rendu compte qu'effectivement les IA de plus haut niveau voyaient plus loin que ceux de bas niveaux, mais choisissaient la première solution gagnante qu'elles rencontraient dans l'arbre.
Nous avions donc identifié le problème qui était que l'exploration de l'arbre se faisait de bas en haut. Donc, si l'IA trouvait une situation gagnante trois coups plus tard, elle choisissait de jouer en fonction, alors qu'un coup gagnant était possible ce tour ci ou le tour prochain plus tôt que le coup gagnant dans 3 tours.
Depuis que ce problème fût corrigé, les IA de plus hauts niveaux ont toujours gagné contre les IA de niveau inférieur.

Une dernière information que nous avons tiré de nos tests est qu'il n'y a qu'un seul scénario de jeu lorsque deux IA s'affronte. En effet si l'IA a le choix entre deux coups différents, mais qui sont les deux aussi efficaces, elle va choisir le premier sur lequel elle va tomber lors de l'exploration des arbres.

Difficultés rencontrées

Nous avons rencontré différentes difficultés lors de la conception de l'IA, notamment avec la fonction d'évaluation, car cette fonction permet de noter la valeur d'un coup ou d'une situation de jeu. N'étant pas des experts dans ce jeu, nous ne savons pas quel est le critère stratégique le plus fort pour pouvoir gagner une partie. C'est pourquoi nous avons décidé de former notre IA avec les critères suivants : le nombres de tours contrôlées par le joueur 1 par rapport au joueur 2 et le nombre de ses jetons dans ses propres tours.

Un autre problème rencontré lors de la réalisation de ce programme était la manipulation informatique des arbres de l'IA.

De plus, une fois le jeu et l'IA codés, la communication entre le jeu et l'IA nous a paru difficile car la maîtrise de certaines manipulations de boucles, d'arbres et de matrice s'est avéré complexe.

Améliorations possibles

Comme expliqué précédemment, la fonction d'évaluation de l'IA peut être modifiée ou optimisée en changeant certains critères et leur niveau d'importance.
Il serait également possible de mettre en place plusieurs stratégies différentes possédant chacune leurs propres critères afin d'obtenir plus de diversité.
Par ailleurs, on pourrait améliorer la fonction de choix du prochain coup pour éviter que les scénarios de parties IA contre IA soient toujours les mêmes.

De plus les interfaces du programme sont vraiment minimales et simplistes, car nous avons voulu se focaliser sur la conception du programme. C'est pourquoi une interface Homme/Machine plus ergonomique et agréable à utiliser serait une idée d'amélioration. En respectant au mieux les règles de la méthode GOMS et avec des analyses keystrokes afin de justifier chaque choix d'implémentation des interfaces graphiques.

Conclusion

Ce projet nous a permis d'une part d'appliquer notre savoir-faire en programmation orientée objet et en création d'interface graphique. D'autre part, ce fut l'occasion d'appliquer nos connaissances liées à l'IA vues en cours tout au long du semestre.

Les notions d'arbre et de récursivité ont été primordiales pour notre projet en particulier pour la fonction d'évaluation de l'algorithme Min-Max.

Travailler en groupe fut très important puisque nous avons eu de nombreux débats sur le choix des critères de la fonction d'évaluation, et nous avons aussi chacun pu montrer nos différentes manières de réfléchir face à une situation en jeu.

Cela nous a fait comprendre que pour créer une fonction d'évaluation qui permet de rendre l'IA très forte au jeu, il faut en amont connaître chaque élément qui avantage plus ou moins un joueur, ainsi que les différentes stratégies de jeu.