Thomas Dahmen - Oscar Bouvier - Raphaël Macquet - Jean Forissier
Tutrice : Elisabeth Brunet
9 février 2018

Projet Informatique

Étude de performance d'une IA sur le jeu d'échecs

Cahier des charges

A. Contexte et historique

Historiquement, le jeu d'échecs a été l'un des premiers défis en intelligence artificielle. Il existe même des championnats du monde d'échecs entre ordinateurs, le premier a eu lieu en 1974 et fut remporté par un programme soviétique.

La première machine a avoir battu un champion du monde dans des conditions classiques (c'est à dire, il ne s'agissait pas d'une partie rapide) est Deep Blue, développé en 1997 par IBM. À l'époque, Deep Blue fonctionnait sur une machine très puissante :

Deep Blue, développé par IBM en 1997

DeepBlue est un super ordinateur possédant 32 cores, capable d'évaluer 200 millions de positions par seconde, avec une puissance de 11,4 GFlop/s, loin des ordinateurs personnels de l'époque. Une telle puissance permet alors d'obtenir une intelligence artificielle très performante en testant un nombre important de coups sans pour autant tous les tester (technique dite « min-max »). Ceci permet de battre n'importe quel joueur sous des conditions normales, mais pour autant le fait qu'un ordinateur puisse résoudre le jeu d'échecs (avoir une stratégie permettant de gagner dans absolument tous les cas) reste un problème ouvert.

Après la victoire de la machine sur le champion du monde Kasparov, le défi informatique du jeu d'échecs s'est amoindri est s'est reporté sur le jeu de Go, réputé bien plus complexe pour un ordinateur.

Justement, en décembre 2017, AlphaZero (généralisation d'AlphaGo, intelligence artificielle développée par Google ayant battu le champion du monde de Go) a réalisé l'exploit d'apprendre à jouer en 4h, à la suite desquelles il a battu sur 100 parties la meilleure intelligence artificielle en date (Stockfish). La particularité d'AlphaZero est que l'apport humain a été très faible : on lui a seulement appris les règles basiques (quels coups sont jouables ou non). En partant d'une stratégie au hasard, avec des algorithmes d'apprentissage profond AlphaZero est devenu champion en jouant contre lui-même.

B. Description de la demande (objectifs, fonctions, critères d'acceptabilité)

Notre projet consiste à développer un jeu d'échecs muni d'une interface graphique et d'une intelligence artificielle.

L'intérêt du projet réside essentiellement dans le développement de l'intelligence artificielle, et nous souhaitons pour cela réaliser une étude comparative entre une technique éprouvée pour les jeux à « arbre vaste », l'algorithme minmax, et un réseau de neurones artificiel. En effet, grâce aux immenses bases de données de parties disponibles sur le jeu d'échecs, il pourrait être possible, moyennant une étude empirique (essai de différents nombre de

couches par exemple), d'obtenir une intelligence artificielle pour ce jeu. Il n'est cependant absolument pas dit qu'elle soit réellement « performante », d'où la nécessité de l'étude comparative.

Nous développerons ce projet en utilisant le langage Python, muni des librairies usuelles comme Numpy, ainsi que d'une librairie dédiée à l'apprentissage machine comme Theano, et de PyQt, librairie permettant d'avoir une interface graphique pour notre programme.

Nous développons en premier lieu un jeu d'échec de A à Z (sans utiliser de librairies comme PyChess) avec une interface graphique, munit d'une intelligence artificielle de type « min-max ». Les critères d'acceptabilité sont :

- règles du jeu d'échec correctement implantées
- interface graphique permettant de jouer sans la console
- IA capable de battre un joueur débutant voir moyen (comme nous)

Ensuite, nous tenterons de développer une IA à l'aide d'un réseau de neurones artificiel. Nous ne sommes pas certains de pouvoir directement utiliser notre programme, qui n'est pas aussi optimisé que ceux fournis par PyChess par exemple. Dans cet optique, nous utiliserons plutôt PyChess. Pour l'apprentissage, nous utiliserons une large base de données de parties des meilleurs joueurs humains comme Chessbase (8 millions de parties). Le critère d'acceptabilité est alors également de battre un joueur débutant voire moyen. Nous allons cependant comparer cette approche à l'algorithme min-max.

C. Contraintes (coûts, délais, technique)

Même si nous l'avons évoqué, il ne s'agit pas de reproduire les récents exploits d'AlphaZero. Ceci est très probablement impossible avec nos outils et connaissances (besoin de GPU très puissants et d'algorithmes extrêmement optimisés). Nous allons utiliser les GPU que nous avons sur nos ordinateurs, ou alors ceux que l'école pourrait nous prêter. Pour l'apprentissage et la construction du réseau de neurones, nous utiliserons Theano, une librairie Python open-source. Notons que nous savons que l'approche min-max fonctionne, mais celle du réseau de neurones que nous développerons a un résultat inconnu. Il est en effet illusoire de croire que cette technique peut tout faire, car au contraire elle ne fonctionne pour l'instant que dans des domaines bien balisés comme la reconnaissance de caractères. Il est alors nécessaire d'avoir une approche empirique pour aborder le problème, tout en sachant qu'il peut ne pas être résolu par un réseau de neurones.

Le produit de notre projet est constitué de deux programmes. Le premier est un jeu d'échec muni d'une interface graphique que nous avons entièrement développé ainsi que d'une intelligence artificielle « min-max ». Pour réaliser l'interface graphique, nous utilisons la bibliothèque PyQt, développé par Riverbank Computing. C'est un logiciel libre distribué sous deux licences, la licence gratuite est sous licence GNU GPL (ce qui oblige ceux qui l'utilisent à mettre leurs programmes sous la même licence, et dans ce cas à rendre code source disponible), la licence commerciale permet de revendre son programme. Une version LGPL de PyQt existe (PySide), elle est développée par Nokia, et permet alors de s'affranchir du caractère héréditaire de la licence GPL (il devient donc possible de produire du code propriétaire). Comme dans ce projet nous ne souhaitons pas produire de code propriétaire et que la documentation est plus fournie sur PyQt, nous avons choisi ce dernier. Cependant, la transition entre PyQt et PySide est aisée.

Ce programme sera développé en Python, il sera donc multi-plateforme sous réserve d'avoir un interpréteur Python installé (comme c'est le cas sur Windows, macOS ou Linux). Nous pourrons éventuellement utiliser PyInstaller pour produire un exécutable indépendant pour jouer au jeu. Le second programme sera un réseau de neurones développé en Python avec Theano,

adapté à PyChess, ou notre programme précédent si l'approche fonctionne et qu'il nous reste du temps.

L'algorithme minimax utilisé dans notre premier programme est un algorithme déjà éprouvé et efficace que l'on souhaite implémenter dans notre programme, ce dernier pouvant servir de référence en terme d'efficacité pour les tests de notre algorithme utilisant les réseaux de neurones.

Il s'applique aux jeux opposants deux adversaires et à somme nulle : c'est à dire qu'un premier joueur cherche à maximiser un certain gain tout en sachant que le second cherche à minimiser ce même gain. Il paraît ainsi naturellement adapté au jeu d'échec.

Mais afin de pouvoir caractériser la valeur d'un certain gain, il faut mettre en place une fonction d'évaluation. Dans le cas du jeu d'échec, cela consiste à faire la différence entre les pions de l'adversaire que l'on a mangé, et les pions que l'on s'est fait mangé (chaque pièce possédant une valeur plus ou moins importante, la dame ayant la plus forte et les pions la plus faible) mais aussi la position stratégique des pièces.

Ainsi, de façon théorique, si l'on veut déterminer le meilleur mouvement possible en prenant en compte les n prochains tours, l'algorithme construit un arbre de profondeur 2n de toutes les configurations possibles et détermine le gain maximal (le meilleur mouvement à effectuer) en prenant en compte que le joueur adverse cherche à le minimiser aux rangs impairs.

En pratique, faire un arbre de profondeur supérieur à 3 serait irréaliste, car incrémenter la profondeur d'un arbre correspondrait en moyenne à multiplier par 35 son nombre de feuilles.

D. Planification

Fin février : interface graphique conçue, moteur de jeu (sans intelligence artificielle) comprenant les règles du jeu

Début mars : élaboration de l'algorithme min-max.

16 mars : remise du prototype

Fin mars : bouclement de min-max, début de la création de l'algorithme utilisant les réseaux de neurones artificiels

Mi-mai: logiciel terminé, tests effectués, rapport écrit

Regroupement modulaire

A. Premier programme

Le premier programme contient les modules suivants :

- moteur de jeu (sans IA ni interface graphique)
- interface graphique (gui.py, fichier Python et ressources graphiques)
- intelligence artificielle

Ces différents modules sont dépendants pour une expérience complète du jeu d'échecs (nous verrons leurs liens dans les flux). Cependant, il est possible de jouer au jeu sans interface graphique ni intelligence artificielle, mais aussi sans interface graphique avec intelligence artificielle, ou encore avec interface graphique avec/sans intelligence artificielle.

B. Deuxième programme

Le second programme contient les modules suivants:

- jeu d'échecs (PyChess)
- base de données (Chessbase)
- algorithme d'apprentissage (Theano)
- réseau de neurones (Theano)

Notons qu'une fois le réseau de neurones entraîné, nous n'avons plus besoin de l'algorithme d'apprentissage ni de la base de données (ce qui constitue d'ailleurs un des avantages des réseaux de neurones artificiels).

C. Répartition des modules

Nous essayons le plus possible de répartir les différents modules entre les quatre membres du groupe. Ceci est difficile car les modules interagissent entre eux, à certains moments il y a donc plusieurs personnes travaillant sur un module. Ou alors, une personne est désignée « intégrateur », et a pour charge d'intégrer les modules au programme final, ce qui limite les conflits entre des versions produites par différents membres.

Pour faciliter la collaboration et la gestion de versions, nous utilisons déjà le serveur Git fourni par l'école.

Flux des données entre modules

A. Premier programme

Voici le flux des données entre modules du premier programme pour une « expérience complète » (partie avec interface graphique et intelligence artificielle).

Le moteur de jeu permet de connaître la position des pièces, les pièces prises par chaque joueurs, les coups possibles pour un joueur donné lors de son tour, et est capable de détecter les situations d'échec, d'échec et mat et de partie nulle (comme le pat par exemple). C'est lui qui contient le « véritable » plateau de jeu auquel se référencent les autres modules. Le module d'intelligence artificielle interagit directement avec le moteur de jeu, en récupérant les positions des pièces et lui donnant les coups à jouer. Ensuite, le moteur de jeu répercute les déplacements sur le module de l'interface graphique.

Lorsqu'un humain joue, il interagit avec l'interface graphique qui questionne le moteur pour indiquer au joueur quels sont les mouvements

légaux. Lorsque le mouvement est effectué depuis l'interface graphique, il est répercuté sur le véritable plateau de jeu détenu par le moteur, et l'intelligence artificielle peut ensuite le récupérer. En résumé, il y deux fonctions permettant de déplacer des pièces : une première accessible depuis le moteur de jeu qui modifie le plateau dans le moteur puis dans l'interface graphique, et une seconde accessible depuis l'interface graphique qui répercute le déplacement dans le moteur de jeu.

Conception préliminaire

A. Moteur de jeu

Nous avons commencé par développer un moteur de jeu sans intelligence artificielle, avec plusieurs fonctions comme « move » qui permet de déplacer des pièces sur le plateau en vérifiant que le déplacement est légal, ou encore les parties « chess » et « chessmate » qui vérifient si un roi est en échec ou échec et mat, et dans le cas d'un échec obligent le joueur à faire un mouvement qui le libère.

Voici la représentation du plateau que nous avons choisie :

	0	1	2	3	4	5	6	7	8	9	10	11
0	−15	−15	−15	−15	−15	−15	−15	−15	−15	−15	−15	−15
1	−15	−15	−15	−15	−15	−15	−15	−15	−15	−15	−15	−15
2	−15	−15	−4	−2	−3	−5	−6	−3	−2	−4	−15	−15
3	−15	−15	−1	−1	−1	−1	−1	−1	−1	−1	−15	−15
4	−15	−15	0	0	0	0	0	0	0	0	−15	−15
5	−15	−15	0	0	0	0	0	0	0	0	−15	−15
6	−15	−15	0	0	0	0	0	0	0	0	−15	−15
7	−15	−15	0	0	0	0	0	0	0	0	−15	−15
8	−15	−15	1	1	1	1	1	1	1	1	−15	−15
9	−15	−15	4	2	3	5	6	3	2	4	−15	−15
10	−15	−15	−15	−15	−15	−15	−15	−15	−15	−15	−15	−15
11	−15	−15	−15	−15	−15	−15	−15	−15	−15	−15	−15	−15

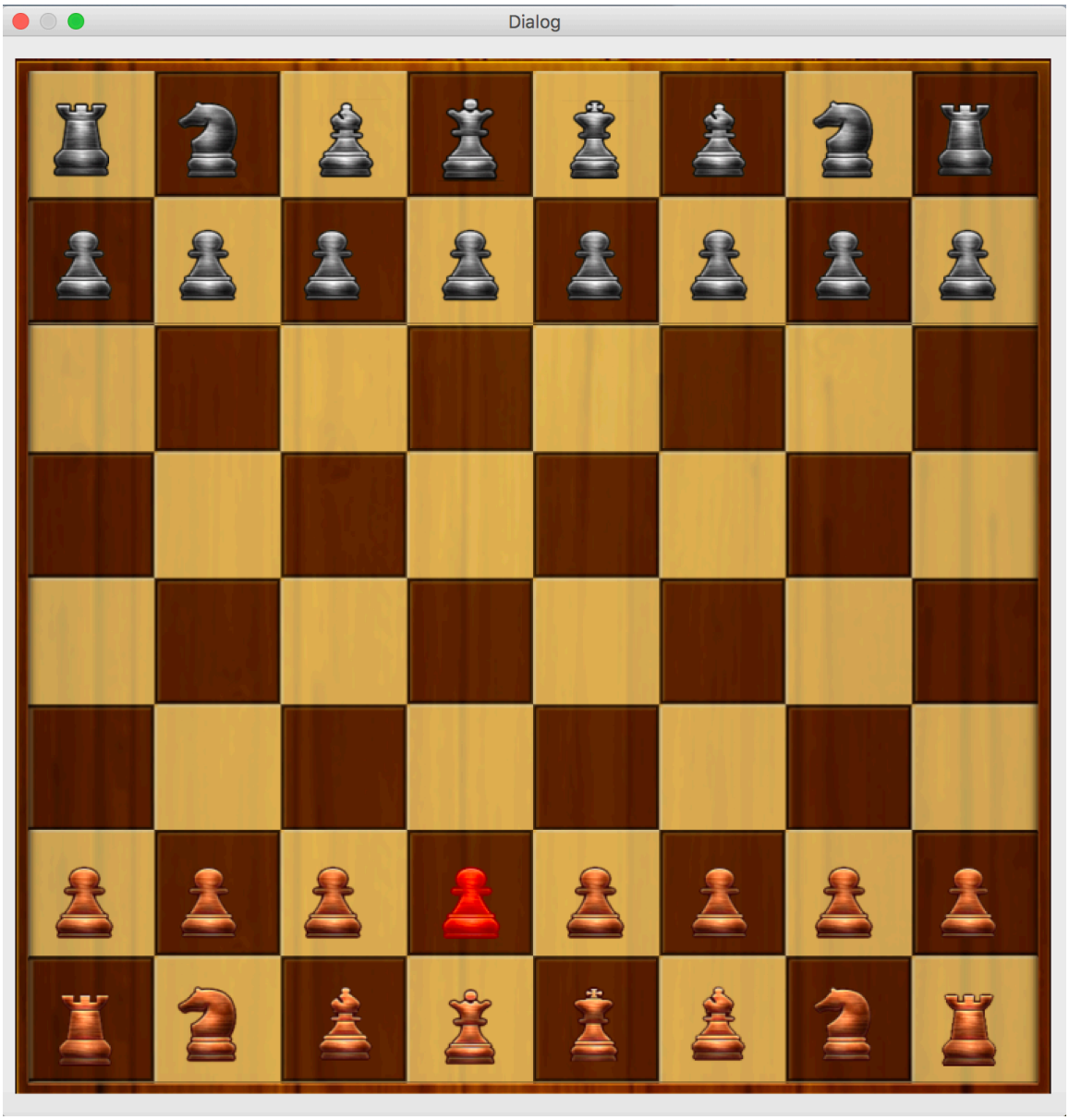
Chaque pièce est représentée par une valeur dans un tableau carré de 12 cases, les pièces blanches correspondant aux valeurs strictement positives (par

exemple : le cavalier est de valeur absolue 2 et le roi de valeur absolue 6) et inversement pour les pièces noires, les valeurs nulles du tableau correspondent aux cases vides du plateau. Enfin, nous avons arbitrairement choisi la valeur -15 pour caractériser les cases inaccessibles afin de pouvoir plus facilement déterminer les cases accessibles par une pièce. Nous utilisons deux couches de telles cases pour pouvoir empêcher tout problème de débordement dans le traitement du mouvement du cavalier pouvant bouger de manière horizontale ou verticale de deux cases.

De plus nous répertorions la position de chaque pièce individuellement (et non plus par valeur comme cavalier ou pion) dans une liste propre à sa couleur : sa position dans cette liste caractérise alors son individualité. Cela permet de connaître instantanément la position et la nature de chaque pièce et d'éviter ains de nombreuses confusions dans le traitement individuel de chaque pièce. Cependant, modifier les valeurs d'une liste sans modifier son ordre peut être très compliqué, c'est pourquoi nous utilisons également un dictionnaire dont les clés seraient les positions des pièces et ses valeurs la position de cette pièce dans la liste des positions.

B. Interface graphique

Pour réaliser l'interface graphique, nous utilisons la bibliothèque PyQt. Voici un premier rendu de ce que nous avons développé :



Nous avons récupéré les images des pièces et du plateau en détourant avec Photoshop une capture d'écran d'un jeu iOS que nous avons trouvé particulièrement esthétique (développé par Optime Software). Pour le rendre public il faudrait donc demander l'accord du développeur, ou utiliser des éléments graphiques libres de droit.

Il est possible dans cette interface de sélectionner des pièces et de les mettre en surbrillance (pion blanc sur la capture d'écran). Voici un extrait du code de l'interface graphique, il s'agit de la méthode appelée lorsque l'on clique sur une pièce (qui est un bouton du point de vue de PyQt):

```
# Fonction d'action lorsqu'une pièce est sélectionnée

def chk(self, button): # on passe en argument le sender
    global pChecked # pChecked est la pièce sélectionnée
    name = button.objectName()
    if pChecked == "": # Aucune pièce sélectionnée
        print(name + " sélectionné")
        button.setIcon(QtGui.QIcon(name + "sel.png")) # on affiche la pièce rouge
        pChecked = button
    else: # Une pièce est déjà sélectionnée
        if button == pChecked: # On a sélectionné la pièce déjà sélectionnée
            print(name + " dé-sélectionné")
            button.setIcon(QtGui.QIcon(name + ".png")) # on affiche la pièce normale
            pChecked = ""
        else: # On a sélectionné une autre pièce
            print(pChecked.objectName() + " dé-sélectionné")
            print(name + " sélectionné")
            pChecked.setIcon(QtGui.QIcon(pChecked.objectName() + ".png"))
            pChecked = button
            namePChecked = pChecked.objectName()
            pChecked.setIcon(QtGui.QIcon(namePChecked + "sel.png"))
```

On utilise pour le nom des pièces une convention très pratique. Par exemple, la tour blanche qui est à gauche au départ est appelée « tour1B », ce qui donne « tour2B » pour la noire. Ce nom est passé en argument (appelé « button »). Le nom du fichier de l'image associé est obtenu en rajoutant l'extension « .png » au nom de la pièce. Enfin, pour obtenir le nom du fichier

image lorsque la pièce est sélectionnée, il suffit de rajouter l'extension « sel.png ».

La prochaine étape consiste à relier l'interface graphique au moteur de jeu. Par exemple, lorsqu'une pièce est sélectionnée, nous souhaitons mettre en rouge les pièces accessibles (qui sont déjà stockées dans une liste par le moteur de jeu), et une fois le mouvement validé, l'effectuer graphiquement. Pour cela, il suffit de changer les images des boutons associés. En définitive, les boutons de l'interface graphique sont fixes (ils ne correspondent réellement à des pièces qu'au départ), et les pièces sont représentées par les images de ces boutons, que l'on change en fonction des déplacements demandés.