Compte-rendu IN104 - Checkers

Yohan BORNACHOT
Pierre HARO

May 2019

Table des matières

1 Présentation du projet

Pourquoi le jeu de Dames ? Parmis les sujets proposés, nous étions tous les deux très motivés par l'idée de découvrir le fonctionnement de base d'une intelligence artificielle. Les perspectives d'appréhender ce fonctionnement au travers d'un projet directement concret et au sein duquel on est assez libre de créer - une fois le principe compris - nous ont beaucoup attirés. En effet, l'objectif principal de ce projet est de créer à partir de zéro - mais en restant guidés - une intelligence artificielle capable de prendre des décisions pour évoluer dans la partie et l'emporter sur son adversaire.

Devant les différents sujets proposés en intelligance artificielle, nous avons préféré le jeu de Dames pour plusieurs raisons :

1. Adeptes de jeux de décision, nous voulions tenter de percer les secrets ou de déterminer les meilleures stratégies pour progresser dans le jeu
2. Les règles du jeu de Dames sont relativement basiques et les autres jeux nous sont inconnus. Comme nous voulions nous concentrer sur l'intelligence artificielle, il était inadéquat d'avoir à apprendre les règles d'un nouveau jeu en même temps.

2 Algorithmes

2.1 IA aléatoire (prise en main)

Dans un premier temps, il a fallu explorer les différents documents à notre disposition. En effet, appréhender un nouvel environnement de code avec des classes et des méthodes prédéfinies n'est pas chose aisée. C'est pourquoi la première approche a été d'en comprendre les subtilités pour automatiser le processus, et de disposer d'une IA prenant des décisions au hasard. L'efficacité n'étant nullement un élément recherché pour le moment. Toutefois, il ne s'agit pas là d'une perte de temps, bien au contraire : on a pu, par la suite, se resservir de cette IA pour tester les prochaines et vérifier si celle-ci subissait une défaite sans appel ... Ce qui fût majoritairement le cas, fort heureusement pour nous.

2.2 Création d'une IA par exploration d'arbre

Le principe de notre IA est d'explorer les différentes configurations de plateau possibles par rapport à la situation courante. Ces configurations successives sont représentées sous la forme d'arbre dont l'exploration peut se faire à une profondeur réglable via le paramètre "depth" disponible dans les fichiers définissant les différentes IA via des méthodes. C'est sur ce principe que l'on se focalisera jusqu'à la fin du projet.

Pour réaliser une telle IA, on a utilisé le célèbre algorithme récursif Minimax, consistant à donner un coefficient à chaque configuration de jeu explorée attestant de son intérêt pour le joueur, puis à choisir le coup suivant (celui avec le meilleur score trouvé), permettant d'aboutir à cette configuration. Pour chaque changement de

profondeur durant l'exploration, on cherche alternativement à minimiser ou à maximiser le score en fonction du joueur concerné. En effet, un mouvement avantageux pour un joueur sera forcément désavantageux pour l'autre. Ce score est attribué via la fonction d'évaluation, décrite plus loin.

Une première amélioration apportée visant à réduire le temps de calcul de Minimax fût de créer un dictionnaire des différents états rencontrés lors de l'exploration donnant le score de cette configuration et de stopper la descente si l'état courant correspond à une configuration de plateau déjà rencontrée. En effet, on connaît déjà le résultat alloué à cet état et donc cela ne sert à rien de continuer dans cette boucle. On a ajouté une table de transposition.

2.3 Élagage alpha-bêta

L'étape suivante, toujours vouée à réduire le temps de calcul de la fonction Minimax, a consisté à ajouter un élagage à notre descente d'arbre. Le principe est relativement simple : si, lors de la descente, on sait déjà pertinemment que le score ne constituera pas un maximum (resp. un minimum selon la phase de jeu), il ne sert à rien de continuer l'exploration de cette branche, d'où le terme "élagage". Cela a pour effet de réduire le nombre de noeuds explorés par la fonction et donc d'améliorer considérablement le temps de calcul dans le cas où se genre de situations se présentent.

2.4 Prise en compte du temps alloué à l'IA pour trouver une solution

L'objectif suivant a été de construire une IA qui s'adapte au temps imparti pour prendre une décision, sans quoi, une fois dépassé ce temps, s'en suit la fin de la partie et la défaite.

Il a pour cela fallu réaménager quelque peu notre algorithme Minimax pour que ce dernier ne prenne plus en paramètre la profondeur jusqu'à laquelle il doit explorer (Depth), mais un temps limite d'exploration à ne pas dépasser : TimeLimit.

Il nous fallait donc une base de temps sur laquelle nous fixer : De combien de temps a-t-on besoin pour acquérir tout les enfants de l'état actuel ? (ie pour l'exécution de la fonction GetChildren) Pour l'obtenir, nous avons effectué une batterie de tests en jouant plus de 100000 parties et nous avons pris la borne supérieure de ces temps pour se placer dans le cas le plus défavorable et ne pas risquer de sous-estimer cet intervalle de temps, utilisé récursivement dans notre algorithme Minimax.

Ensuite, nous avons décidé de répartir le temps alloué à la prise de décisions entre les différents enfants successifs lors de l'exploration. Pour cela, on a divisé TimeLimit par (n+1) où n représente le nombre d'états accessibles dans l'élaboration de l'IA MinimaxTime. Ainsi, l'IA se sert de tout ou quasiment tout le temps qui lui est alloué pour choisir le prochain état et peut alors descendre plus ou moins profond dans l'arbre en fonction du temps qu'il lui reste. Cela a donc permis de gérer le temps et non plus de le voir comme un paramètre intouchable.

Analyse : Il est important de préciser qu'au cours de ces différentes améliorations, nous avons également procédé à de nombreux tests en faisant s'affronter nos différentes IA entre elles, en variant dans un premier temps la profondeur d'exploration lorsque l'on faisait s'affronter deux IA avec Minimax-élagage ou une IA random avec une IA avec Minimax-élagage. (Nous avons alors remarqué qu'=il suffisait d'une profondeur de 1 ou 2 pour battre une IA random, ce qui est logique et rassurant) Dans un second temps, nous avons fait s'affronter une IA avec minimax_time contre une avec élagage et nous avons pu constater que l'IA qui gérait sont temps était plus efficace dans la majeure partie des cas, puisqu'elle comporte encore l'élagage alpha-bêta en plus d'avoir une gestion efficace du temps de recherche. Cette amélioration croissante de notre IA nous a conforté que nous allions dans la bonne direction, et quand ce n'était pas le cas, c'est qu'il y avait quelque chose à corriger dans les codes ! On a notamment rencontré un frein à notre avancée :

Notre algorithme minimax ne passait les tests que dans la majeure partie des cas (au lieu d'une validation systématique), ce qui nous a beaucoup interrogé. En fin de compte (et après un bon moment de tâtonnement), nous avons compris que, comme le plateau est retourné d'un joueur à l'autre (pour donner l'illusion à chaque joueur de jouer avec les blancs), des configurations similaires pouvaient être enregistrées dans la table de transposition sans qu'on les ait réellement rencontrées deux fois, ce qui faussait les résultats de la fonction utilisée pour tester notre minimax (fournie), qui s'arrêtait prématurément. Le chargé de projet a ensuite modifié son code des tests pour que le problème n'apparaisse plus : notre programme a pu passer les tests correctement !

```
import IN104_simulateur as simu
import time
from evaluation import evaluate
from minimax_time import minimax


class MinimaxTime:

    def __init__(self, config=None, rules=None):
        self.name = 'AI' # set your AI name here

    def play(self, gameState, timeLimit):
        possibleStates = gameState.findNextStates()
        value=float("-inf")
        dico_state={}
        next_state = gameState
        for state in possibleStates:
            new_value = minimax(state,timeLimit/(len(possibleStates)+1),False,simu.GameState.findNextStates,evaluate,dico_state)
            if new_value >= value:
                value = new_value
                next_state = state
        return next_state

    def __str__(self):
        return self.name
```

FIGURE 1 – L'algorithme d'IA retenu

```python
from time import clock

delta_children=1.6689300537109375e-6

def minimax(state, temps_imparti, maximize,get_children,evaluate,dico_state={}):
    time=clock()
    global delta_children
    if delta_children>temps_imparti: #PAS LE TEMPS D'EXPLORER LES ENFANTS
        return evaluate(state)

    else:
        Children=get_children(state)
        if Children==[]: #PAS D'ENFANT DONC FEUILLE
            return evaluate(state)

        if str(state) in dico_state: #CONFIGURATION DU PLATEAU DEJA RENCONTREE
            return(dico_state[str(state)])

        if maximize==True:

            j=0 #INDICE INDIQUANT QUEL ENFANT ON EST EN TRAIN D'EXPLORER
            delta=0
            value=float("-inf")

            for i in Children:
                t1=clock()
                delta= t1-time
                temps_imparti_suiv=(temps_imparti-delta)/(len(Children)-j) #TEMPS_IMPARTI_SUIV EVOLUE
                #EN FONCTION DU TEMPS QUE PRENNENT LES ENFANTS POUR RECEVOIR UN SCORE
                new_value=minimax(i,temps_imparti_suiv,False,get_children,evaluate,dico_state)
                value=max(value,new_value)
                dico_state[str(i)] = new_value
                j+=1
        else :
            time=clock()
            j=0 #INDICE INDIQUANT QUEL ENFANT ON EST EN TRAIN D'EXPLORER
            delta=0
            value=float("+inf")

            for i in Children:
                t1=clock()
                delta= t1-time
                temps_imparti_suiv=(temps_imparti-delta)/(len(Children)-j)
                new_value=minimax(i,temps_imparti_suiv,True,get_children,evaluate,dico_state)
                value=min(value,new_value)
                dico_state[str(i)] = new_value
                j+=1
        return value
```

FIGURE 2 – L'algorithme minimax_time final

3 Démarches personnelles

3.1 Reconnaissance de situations favorables

La première idée d'amélioration de l'algorithme a été la reconnaissance de situations favorables pour l'IA. De la même manière qu'aux échecs, il existe aux dames certains enchaînements de mouvements qui offrent de grandes perspectives. Les sites dédiés au jeu de dames expliquent clairement comment arriver à ces situations.

Tout d'abord, nous devions entrer une configuration de plateau dans le dictionnaire de situations déjà intégré dans l'algorithme de définition de l'intelligence artificielle. Cependant, la modélisation d'un plateau de jeu reste encore obscure pour nous. Si enregistrer une configuration durant une partie lorsque celle-ci est rencontrée est simple, créer un plateau de jeu de toute pièce nous aurait pris beaucoup de temps vis-à-vis du temps accordé au projet. De plus, il aurait fallu entrer un grand nombre de configurations différentes pour exploiter de manière intéressante cette technique.

Cependant, l'intérêt de ces configurations est qu'une fois rencontrée, l'adversaire n'a pas d'autres choix que de se faire prendre au piège. Par conséquent, lors de l'exploration de la branche associée à cette situation, l'adversaire n'a qu'un seul

coup possible à jouer. Il n'y aurait donc pas de gain de temps sur les coups de l'adversaire mais uniquement sur ceux du joueur à l'origine de la situation.

De plus, rencontrer une telle situation nécessite de sauter des étapes : s'il reste une profondeur d'exploration de n lors de la reconnaissance d'une situation impliquant (n-k) mouvements, il restera k niveaux d'exploration à effectuer. Cela ne permet donc pas de couper entièrement l'exploration de l'arbre. Devant les perspectives limitées de cet apport, nous avons remis cette étude à plus tard.

3.2 La fonction d'évaluation

Avec les méthodes d'exploration d'arbre, la fonction d'évaluation est le second paramètre explicite sur lequel nous pouvions travailler pour améliorer notre IA.

Premièrement, la fonction d'évaluation basique consistait à faire la différence entre le nombre de pions adverses et les pions de l'IA.

Nous avons tout d'abord ajouté du poids aux dames par rapport à la fonction conseillée dans le sujet initialement. Étant donné qu'elles bénéficient d'une gamme de mouvements plus élargie, il était légitime de leur confier plus d'importance dans l'évaluation d'une configuration. Cependant, avec la profondeur de récursivité possible sur les ordinateurs à notre disposition, il était fréquent que la création d'une dame apparaisse trop profondément et échappe à l'exploration.

Pour encourager ce déplacement vers l'avant et donc la création de dame, nous avons décidé d'attribuer comme valeur aux pions le numéro de la ligne de leur position. Ainsi, même sans manger de pièce adverse, le joueur gagne des points en avançant. Il devient alors plus important de protéger un de ses pions bien avancé sur le plateau et de défendre ses lignes de fonds. Nous avons également placé une valeur variable sur le poids d'une dame que nous avons fait varier lors de différentes parties contre des IA. Il en est ressorti que la valeur donnant les meilleurs résultats était celle correspond au poids du meilleur pion plus trois, soit 11 pour un plateau standard. En effet, cette fonction s'adapte à la taille du plateau. Il est possible d'envisager un plateau plus ou moins grand que le plateau traditionnel.

Cet algorithme est d'une efficacité supérieure à celui affectant le dmême poids à tous les pions indépendamment de leur position. C'est sur cette version finale que s'est arrêté notre choix, la fonction d'évaluation doit en effet rester suffisamment simple pour limiter le temps d'exécution.

Néanmoins, faire affronter deux IA avec deux fonctions d'évaluations différentes était difficile. Nous ne trouvions pas comment faire car le code des méthodes nous était caché. Clément Masson, chargé du projet, nous a aidé en nous proposant la semaine suivante une structure alternative des fonctions permettant le passage en paramètre des fonctions d'évaluation et résoudre le problème.

4 Evolutions et améliorations possibles

Comme nous l'avons évoqué plus haut, une possibilité serait la prise en compte de la reconnaissance de situations classiques et avantageuses.

Pour aller plus loin, si notre intelligence artificielle exécute un nombre suffisant de partie et conserve la connaissance des parties précédentes, elle disposerait d'un

```python
import IN104_simulateur as simu

def evaluate(state):
    Cell = simu.Cell
    boardState = state.boardState
    score = 0
    cell = boardState.cells
    nRow = boardState.nRows
    boardSize = len(cell)
    for k in range(0,boardSize):
        [r,c] = boardState.indexToRC(k)
        if cell[k] is Cell.w:
            score += r
        elif cell[k] is Cell.b:
            score -= r
        elif cell[k] is Cell.B:
            score -= nRow+3
        elif cell[k] is Cell.W:
            score += nRow+3
    return score
```

FIGURE 3 – La fonction evaluate retenue

avantage certain. Notre version de Minimax conserve les configurations rencontrées au cours d'une partie mais les efface à la fin, c'est une grande quantité de données intéressantes qui est perdue. Les conserver s'apparente à l'idée que nous nous faisons du machine learning mais nous n'avons pas encore les connaissances requises pour maîtriser ce domaine.

Au niveau du travail de groupe, nous aurions plus exploité plus intensément les possibilités de GitHub. Il nous a fallu un certain pour comprendre suffisamment le fonctionnement de cet outil afin de s'en servir efficacement. Nous pensons notamment aux commentaires des commit et aux fréquences de pull. Des choses basiques mais qui peuvent entraver l'avancée du projet. Il y a sûrement de nombreux points que nous ne maîtrisons pas encore. par exemple la fusion de branche, qui nous aurait permis de travailler avec plus d'efficacité.

5 Conclusion

Les méthodes qui ont été vues et acquises peuvent s'appliquer à plusieurs jeux de décisions comme les échecs où le go, mais il s'agit d'un parti-pris de commencer sur le jeu de Dames, dont les règles sont simples et plus facilement implémentables sur machine.

Ce cours nous a ainsi permis d'en apprendre plus sur la décision automatique d'une IA lors de parcours d'arbre de possibilités de jeu, notamment en affectant un coefficient à chaque situation de jeu, plus ou moins intéressantes à atteindre selon le déroulement de la partie.

De manière plus transversale, nous avons aperçu les techniques qui permettent de travailler à plusieurs sur un projet informatique. Jusqu'alors, l'informatique a été pour nous quelque chose d'individuel.