Rapport Projet IA41

Table des matières

Marius DIGUAT-MATEUS, Saad SBAT, Albert ROYER

Introduction

Ce Projet, réalisé par Saad SBAT, Marius DIGUAT-MATEUS et Albert Royer dans le cadre de l'UV IA41 a pour objectif de mobiliser nos compétences développées le long de ce semestre d'Automne 2022 en intelligence artificielle. Nous avons choisi de traiter deux sujets : dans un premier temps nous verrons **L'IQ PUZZLER PRO**, puis nous verrons ensuite le jeu du **TEEKO**.

IQ Puzzler Pro

I. Choix utilisateur

Notre premier défi a été de programmer le jeu de L'IQ puzzler : à l'aide de PYTHON et de TKINTER, il nous a été facile de créer une interface graphique ainsi que de poser les bases du jeu, nous y avons cependant rajouté quelques fonctionnalités :

- Les pièces sont créées aléatoirement
- La taille de la grille est personnalisable *
- La grille doit être initialisée avec des pièces, puis une partie de celles-ci sont enlevées
- Les pièces doivent pouvoir être tournées

* Nous remplissons une grille complète avec des pièces créé aléatoirement. Une fois cela fait, nous enlevons certaines pièces prises aléatoirement que nous tournons et mélangeons. Ainsi le défi est de repositionner ces pièces dans la grille

II. Analyse du problème

Notre sujet, l'IQ **PUZZLER PRO**, est un problème de satisfactions de contraintes (CSP), nous avons donc défini des variables, ici les pièces du puzzle, le domaine de celles-ci (la grille), et leurs contraintes : les pièces ne peuvent pas se superposer et la grille doit être pleine à l'état final.

III. Génération

Avant de pouvoir s'attaquer à la résolution du problème nous devons d'abord le modéliser. Pour cela nous avons décidé de générer aléatoirement les pièces ainsi que leur emplacement. Pour chaque pièce, nous nous plaçons sur une case vide aléatoire de la grille, puis nous choisissons aléatoirement parmi les cases adjacentes vides une case que nous rajoutons à la pièce, prenant ensuite les cases adjacentes de la case ajoutée dans la liste des cases possibles à choisir. Nous répétons ce procédé n fois, « *n* » étant la taille de la pièce (n variant entre 3 et 6). Nous enlevons ensuite des pièces choisies aléatoirement, on les tourne et on les mélange avant de les afficher sur l'écran. Ce sont les pièces que nous devront utiliser pour compléter à nouveau la grille.

IV. Résolution

Nous avons opté pour résoudre ce problème à l'aide de l'algorithme TEST & GENERATE (Backtrack) pour résoudre ce CSP :

Nous utilisons donc un algorithme récursif qui prend en compte la grille, le numéro de la pièce et la liste de listes de coordonnées possibles pour chaque pièce.

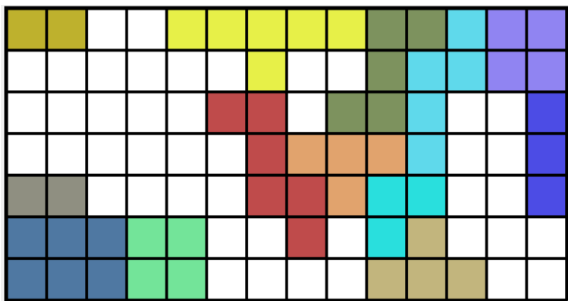
Si la grille est remplie, c'est-à-dire si elle ne contient aucun 0, alors le problème est résolu,

Sinon, on prend dans notre liste des coordonnées possibles pour chaque pièce, une coordonnée valable pour placer la pièce actuelle. On rappelle ensuite la fonction, lorsque la pièce que nous regardons n'a aucun emplacement valide, on retourne à la pièce précédente et modifions son emplacement.

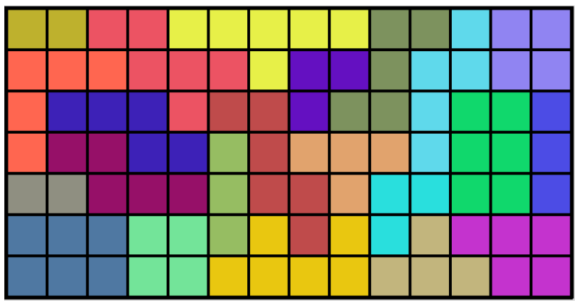
Nous avons également implémenté un algorithme GENERATE & TEST pour pouvoir comparer la différence des deux algorithmes sur le fond.

V. Résultats

Nos algorithmes nous permettent donc de résoudre le problème, bien que le *Test and Generate* soit plus performant que le *Generate and Test* que nous avons implémenté également : dans les deux cas, l'IA nous retourne une solution et l'affiche sur l'interface en complétant la grille.



Avant résolution



Après résolution

Notre programme nous permet de résoudre le jeu dans tous les cas de figures, cependant, la taille de la grille a un impact considérable sur la vitesse de résolution.

VI. Améliorations

Un des défauts du programme est qu'il peut mettre énormément de temps à résoudre le problème, en fonction de la taille de la grille, nous pourrions optimiser l'algorithme pour que celui-ci soit plus rapide.

Teeko

I. Choix utilisateur

Nous avons, pour créer un jeu du **TEEKO** opérationnel, encore une fois opté pour TKINTER pour l'affichage, mais cette fois ci, on doit avoir une grille 5x5, chaque joueur doit placer dans un premier temps 4 jetons.

On utilise un tableau de caractère a deux dimensions de taille 5x5 avec des underscores pour les cases vides, des X pour les pions noirs et des O pour les pions blancs.

Nous avons créé plusieurs modes de jeu : un contre un humain (donc deux humains), et un contre l'ordinateur, nous avons aussi pris la liberté de créer un mode de jeu « ordi contre ordi » dans lequel la machine se bat contre elle-même.

Nous voulions faire une IA capable de se battre contre nous et même de gagner, nous avions également l'envie de tester d'implémenter d'autres algorithmes, tel que l'algorithme Min-Max ainsi que l'élagage alpha-beta.

II. Analyse du problème

Notre sujet, le jeu de TEEKO, est un problème de satisfaction de contraintes (CSP), nous avons donc défini des variables, ici les jetons sur le plateau de jeu, le domaine de celles-ci (les différentes combinaisons de pions qui peuvent être placées sur chaque emplacement), et leurs contraintes : chaque joueur doit placer ses pions de manière qu'ils forment une ligne de 4 pions (ou un carré) avant l'adversaire.

Tout d'abord, les joueurs placent leur 4 pions sur le plateau à tour de rôles, puis à chaque tour, ils doivent déplacer un pion sur une case adjacente. Cela nous fait donc deux étapes différentes.

III. Mise en place

Nous avons codé le jeu de sorte que :

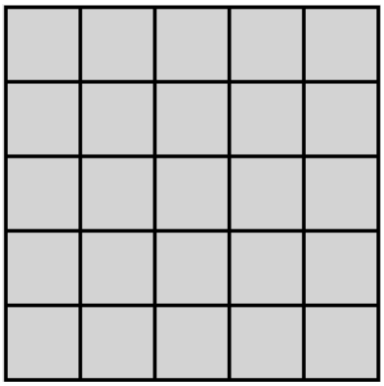
Mise à part pour le mode de jeu ordi contre ordi, le code est lancé quand on clique à l'aide de la souris sur une case jouable du plateau.

Lorsqu'on joue contre l'ordinateur, l'algorithme Min-Max qui permet de faire jouer l'ordi se lance dès que le joueur adverse a joué un coup.

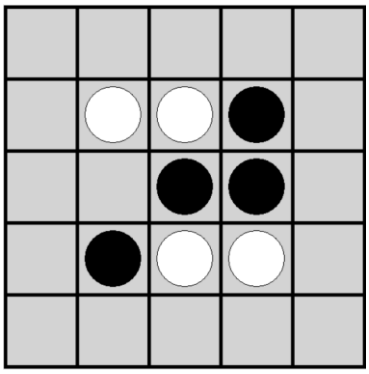
Au bout du 8^{e} tour, on ne peut plus poser de pion et on peut seulement déplacer les pions existants.

A chaque tour, on doit vérifier s'il y'a une victoire ou non : ce que nous faisons en regardant le dernier coup joué et en analysant les pièces autour de ce dernier pion déplacé.

A chaque tour, on affiche également le coup joué sur notre interface grâce à TKINTER.



Avant tour 0



Après tour 7

IV. Min-Max & élagage Alpha-Beta

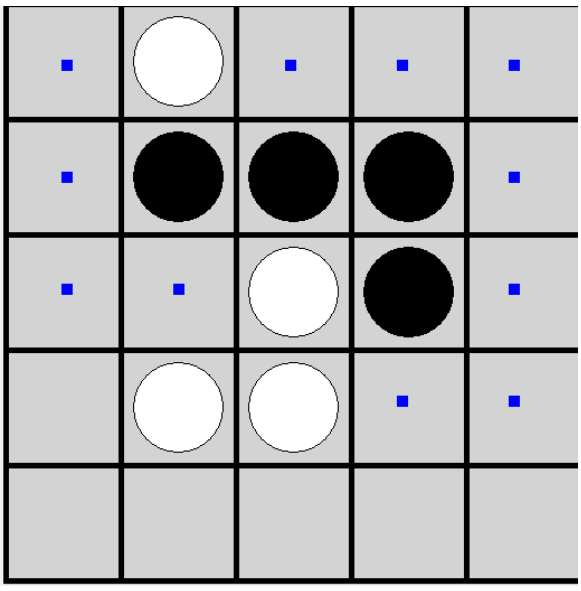
Le programme vérifie d'abord si le tour en cours est inférieur à 8.
On utilise ensuite l'algorithme Min-Max qui trouve la meilleure action à prendre dans des situations où deux joueurs s'affrontent.

L'algorithme commence en évaluant l'état actuel du jeu, c'est-à-dire la disposition des pions sur le plateau. Ensuite, il génère toutes les actions possibles pour le joueur actuel, c'est-à-dire tous les endroits où il peut placer son prochain pion.
Pour chaque action du joueur actuel, l'algorithme calcule la valeur minimale des actions possibles pour l'adversaire en utilisant une fonction d'évaluation.
Pour chaque action de l'adversaire, l'algorithme calcule la valeur maximale des actions possibles pour le joueur actuel en utilisant la même fonction d'évaluation. L'algorithme continue de répéter ces étapes jusqu'à ce qu'il atteigne une "profondeur" prédéfinie dans l'arbre de recherche, c'est-à-dire jusqu'à ce qu'il ait évalué toutes les actions possibles jusqu'à une certaine limite.
Enfin, l'algorithme retourne l'action qui a la valeur maximale pour le joueur actuel, car cette action est considérée comme la meilleure pour lui étant donné les actions possibles de l'adversaire.

Nous avons amélioré Min-Max à l'aide d'un élagage alpha beta qui permet de réduire drastiquement le temps de calcul :
L'algorithme Min-Max utilise deux valeurs de référence appelées "alpha" et "beta" pour déterminer si cette action mérite d'être explorée plus en détail.
Si la valeur de l'action est inférieure à alpha (pour un joueur maximisant) ou supérieure à beta (pour un joueur minimisant), alors l'algorithme peut ignorer cette action et passer directement à la suivante, car elle ne peut pas être la meilleure action. Cela nous permet ainsi d'augmenter la profondeur de l'arbre de recherche.

V. Résultats & Améliorations

Notre algorithme est très performant pour ne pas perdre, cependant il aura beaucoup plus de mal de trouver des stratégies qui l'emmènent vers la victoire car il essaye d'attaquer seulement lorsqu'il sait qu'il va gagner, c'est-à-dire si la victoire est en ligne de vue dans la profondeur.



L'ordinateur (ici les pions noirs) sait qu'il va jouer la case à droite, en prolongement de la ligne noire, les points bleus sont les possibilités de coup de l'ordinateur.

Pour améliorer le programme, on pourrait déterminer le meilleur coup parmi tous les coups qui ont la même valeur, pour ça on pourrait chercher une heuristique, toutefois, notre algorithme a battu celui d'un autre groupe (Victor, Osman, Gilles et Hakan).

Conclusion

L'**IQ PUZZLER PRO**, est donc capable de résoudre une grille dans un laps de temps raisonnable et est facilement utilisable.

D'un autre côté, le jeu du **TEEKO** permet à un joueur de s'entrainer contre un adversaire qu'il ne pourra normalement jamais battre : notre IA. Faire un algorithme Min-Max sur un jeu simple comme celui du Teeko nous permet de comprendre le jeu du Teeko et ses stratégies.

Nous avons apprécié travailler sur ces projets ainsi que d'utiliser *TKINTER* comme interface graphique.

Pour conclure, ces deux sujets nous auront apportés l'expérience d'une découverte de l'IA, ce qui nous permet d'avoir une compréhension plus profonde du monde de l'IA en général, et cela nous servira donc de base pour notre cursus.

Annexe

Fonction 1 :

```
def resolution_1(grid, pos_list, count):

    if is_full(grid):

        print("grille remplie")
        return True

    else:

        # On place les pièces une par une

        val, choice_list = pos_list[count]

        # choice_list = toutes les possibilités de la pièce "count"

        for coord_list in choice_list:

            # coord_list = 1 possibilité de la pièce

            if can_place_w_list(grid, coord_list):

                place_w_list(grid, coord_list, val)
                count += 1

                if resolution_1(grid, pos_list, count):
                    return True

                else:
                    count-=1
                    remove_w_list(grid, coord_list)

    return False
```

Fonction 2 :

Les projets sur GitHub :
https://github.com/SkateZen/IQ_Puzzler

https://github.com/SkateZen/Teeko