**Rapport Projet IA41**

Table des matières

[Introduction 2](#_Toc122081219)

[IQ Puzzler Pro 3](#_Toc122081220)

[I. Choix utilisateur 3](#_Toc122081221)

[II. Analyse du problème 3](#_Toc122081222)

[III. Génération 3](#_Toc122081223)

[IV. Résolution 3](#_Toc122081224)

[V. Résultats 4](#_Toc122081225)

[VI. Améliorations 4](#_Toc122081226)s

[Teeko 5](#_Toc122081227)

[I. Choix utilisateur 5](#_Toc122081228)

[II. Analyse du problème 5](#_Toc122081229)

[III. Génération 5](#_Toc122081230)

[IV. Résolution 5](#_Toc122081231)

[V. Résultats 5](#_Toc122081232)

[VI. Améliorations 5](#_Toc122081233)

[Conclusion 5](#_Toc122081234)

# 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

## 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

## 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.

## Génération

Pour générer le jeu,

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).

## 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 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.

Une image contenant texte

Description générée automatiquement

## 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.

## 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

## Choix utilisateur

## Analyse du problème

## Génération

## Résolution

## Résultats

## Améliorations

# Conclusion

L’IQ PUZZLER PRO, est donc capable de résoudre une grille dans un laps de temps raisonnable et est facilement utilisable grâce à l’interface graphique de TKINTER.

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, et est utilisable facilement grâce à TKINTER.

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.