|  |
| --- |
| **14 juin 2019**  **Solveur de**  **TANGRAM**  CARTERET Thomas  HILKENS Bram  Sous la tutelle de LAURI Fabrice |





Sommaire

Table des matières

Sommaire 3

I – Introduction 4

II – Analyse du problème 5

**1)** **Identification des variables en jeu** 5

**2)** **Domaine des variables** 5

**3)** **Quelles sont les contraintes liées aux variables** 5

III – Mode de représentation des connaissances 6

**1)** **Comment représenter l’évolution de la recherche de solution** 6

**2)** **Construction du graph** 6

**3)** **Recherche dans le graph** 8

**4)** **Création du Tangram** 8

**5)** **Modélisation du graph et des solutions** 9

IV – Exemples, résultats et analyse 10

**1)** **Sous-titre** 10

V – Améliorations possibles 11

**2)** **Sous-titre** 11

VI – Annexes 12

**3)** **Sous-titre** 12

I – Introduction

Pourquoi on a choisi le projet

L’interet du projet vis-à-vis de la matière

Application concrète du cours

METTRE DES IMAGES

II – Analyse du problème

1. **Identification des variables en jeu**

La première chose à établir pour la résolution du jeu Tangram est l’ensemble des variables que nous utiliserons pour modéliser le problème.

En effet, plusieurs solutions ont été envisagé mais celle retenu est l’utilisation de liste de coordonnées formant des polygones. Avec cette méthode nous pouvons aussi bien représenter le patron du jeu que les formes qui le compose. De plus, les diverses bibliothèques que nous avons pu envisager utilisaient plus ou moins ce système et l’un des principaux avantages est qu’il s’agit d’une modélisation légère et agile de formes.

1. **Domaine des variables**

Une fois le type de variable choisi, il est nécessaire d’identifier les valeurs que ces dernières peuvent prendre. Comme il s’agit de listes de coordonnées représentant des polygones, nous avons défini un axe x,y avec une échelle de 100 (1 unité = 100px) entre chaque point. Ainsi, les coordonnées peuvent donc être uniquement positive et par échelon de 100.

1. **Quelles sont les contraintes liées aux variables**

Les règles du Tangram sont assez simples, on dispose d’une série de pièces définies et on doit simplement les assembler pour reproduire un patron. Donc la seule véritable règle à respecter est que chacune des formes ne doit pas ni dépasser des contours du patron, ni se chevaucher.

III – Mode de représentation des connaissances

1. **Comment représenter l’évolution de la recherche de solution**

Pour représenter l’évolution de la recherche et donc des états des différentes variables, nous avons choisi d’utiliser un graph. En effet, il permet de stocker dans les nœuds les états de nos variables évoluant en fonction de l’avancée.

Nous avons choisi de suivre une logique dans laquelle nous stockons dans chaque nœud deux listes une correspondant aux contours du patron et la seconde la liste des formes à positionner. Le nœud dit ‘source’ contiendra donc l’état de départ c’est-à-dire le patron original ainsi que la liste de forme.

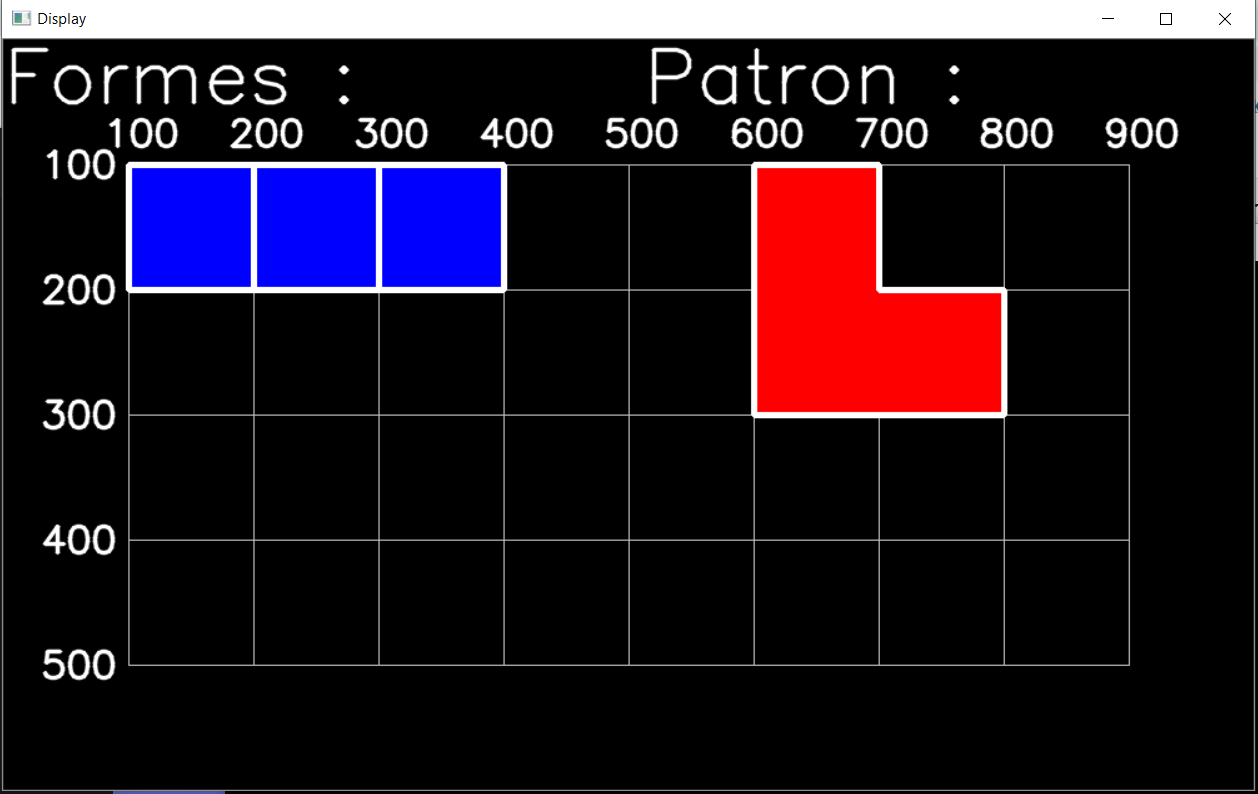
Pour représenter ce graph deux possibilités s’offraient à nous, le construire nous même à partir des classes ‘Graph’, ’Node’ et ‘Edge’, ou bien utiliser une bibliothèque à cet effet. Nous avons choisi cette seconde option car en Graph Theory au semestre précédent, nous avons déjà pu réaliser un projet autour des graphs abordant ces aspects, donc par économie de temps et surtout pour avoir une interface graphique permettant l’affichage de graph, nous avons choisi d’utiliser ‘networkx’.

1. **Construction du graph**

Une fois cette étape préliminaire de réflexion achevée, nous sommes rentrés dans le cœur du sujet, à savoir l’établissement des prédicats dans le but de construire notre graph et trouver la meilleure solution.

Premièrement, nous avons dû générer un premier exemple simpliste pour établir les prédicats. Nous avons donc choisi d’utiliser ‘cv2’ qui est une bibliothèque de reconnaissance d’image permettant notamment de gérer des formes, ce choix a aussi été motivé par le fait que nous envisagions de faire de la reconnaissance d’image sur une image de tangram en tant qu’entrée. Nous n’avons malheureusement pas eu le temps de réaliser cette fonctionnalité mais cela explique le choix d’une si grosse bibliothèque.

Voici donc notre set de données qui a permis l’établissement des règles :



A partir de là, nous avons pu établir notre premier prédicat qui est à la fois le plus important et le plus subtile à réaliser. Il s’agit de ‘shapeFits’ joint en annexe. Son but est avec une forme et un patron de trouver l’ensemble des emplacements où cette pièce peut être positionnée. Pour cela nous utilisons ‘matplotlib’ qui nous permet notamment de savoir si une pièce est à l’intérieur d’une autre. Afin de trouver un maximum de solution nous positionnons cette pièce grâce à ‘offsetShape’ à chaque coordonnée du patron, avec cette méthode, nous trouvons casi toutes les solutions pour des patrons assez petit. A défaut, au fur et à mesure de la réduction du patron l’algorithme trouvera quand même ces solutions à l’origine cachée.

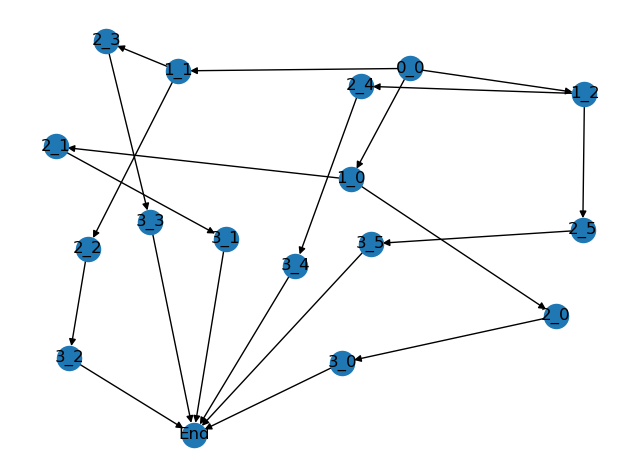
Un second prédicat très important est ‘reshapePatron’ qui permet de soustraire une pièce au patron. En effet, si on sait qu’une certaine forme peut être positionnée à un certain endroit du patron, on lui soustrait cette forme réduisant la taille de patron et on l’enlève ensuite de la liste des formes disponibles. Cette méthode est très utile pour la construction du graph et donc de l’évolution des états.

Le dernier prédicat directement lié à la construction du graph est ‘graphBuilderBFS’ qui permet de construire un le graph complet du Tangram suivant les prédicats cités ci-dessus avec la méthode d’un BFS (voir annexe logigramme BFS). Ce choix regretté par la suite était motivé par le fait que nous voulions construire un graph complet pour ensuite trouver la meilleure solution de manière optimisée. Cependant, comme le montre la partie exemple et résultats, les graphs deviennent très rapidement immenses il n’est donc pas stratégique de trouver toutes les solutions.

1. **Recherche dans le graph**

Dans un premier temps, nous nous sommes contentés du BFS pour trouver les solutions ce qui était fonctionnel mais non optimisé. Ensuite, conformément au sujet nous sommes partis sur un algorithme A\* pour rechercher de manière optimisée la meilleure solution. Cependant, nous avons rapidement rencontré un problème dans la méthode ‘heuristicCalculator’, en suivant la logique de fonctionnement de l’algorithme A\*, nous avons réalisé que l’information que donnait l’heuristique était innutile. En effet dans la logique, l’heuristique est censée estimer le cout restant or quel que soit la branche le cout sera le même puisque dans notre cas le cout est le nombre de pièces restantes. Nous avons donc choisi de modifier son calcul en pondérant le cout en fonction de la taille de la pièce. Ainsi, les grosses pièces qui dans la logique doivent être positionnées en premier auront un cout plus faible et à l’inverse les plus petites censées être mises à la fin pour boucher les trous auront un cout plus élevé. Avec cette méthode transgressant le fonctionnement de l’algorithme A\*, nous obtenons un algorithme efficace pour le Tangram. Il aurait donc fallu construire le graph en suivant aussi cette logique ce qui aurait permis de résoudre des Tangram plus complexes.

1. **Création du Tangram**
2. **Modélisation du graph et des solutions**

Comme expliqué précédemment, nous avons utilisé une bibliothèque pour les graphs permettant des affichages très utile pour la vérification surtout au début de projet.

Ensuite, grâce à l’interface initiale, il est possible d’afficher les différentes étapes de résolution du chemin déterminé avec A\*. On visite ainsi l’ensemble des nœuds dis gagnants. (voir Exemple et resulats)

IV – Exemples, résultats et analyse

1. **Exemple du carré**

V – Améliorations possibles

1. **Sous-titre**

VI – Annexes

1. **Sous-titre**