IA41 - Concepts fondamentaux en Intelligence Artificielle et langages dédiés
Université de Technologie de Belfort-Montbéliard

Projet de résolution d'un tangram

Adrien Berthet, Paul Locatelli et Pierre Rognon

Printemps 2013

Sommaire

Introduction

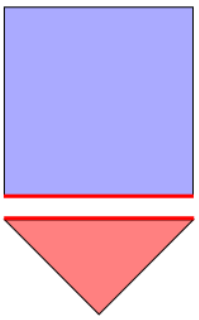
Pour ce projet, le sujet abordant le casse-tête du tangram (numéro 6) a été retenu. Le choix de ce sujet s'est fait sur l'intérêt de ce jeu. En effet, s'il apparaît assez aisé de résoudre des modèles tels que le classique "carré" que chacun connaît, d'autres modèles sont beaucoup plus complexes. De plus, c'est un test utilisé dans de nombreux tests afin de déceler d'éventuelles déficiences chez les enfants tels que le WISC (Wechsler Intelligence Scale for Children). C'est donc un problème d'Intelligence Artificielle fondamental qui sera abordé dans ce projet et dont la résolution sera tentée.
L'analyse de ce problème constitue une partie prépondérante dans la tentative de résolution du tangram. Environ 6000 configurations différentes sont connues aujourd'hui. De nombreux modèles ont donc dû être envisagés afin de couvrir l'ensemble des cas possibles. Une seconde phase qui s'avère aussi complexe est le choix des différentes représentations informatiques du problème. De nombreux traitements ont été effectués pour mettre à bien cette partie. Quelques exemples qui ont été utilisés comme support durant toute la durée du projet seront présentés afin d'indiquer leur solution. Enfin, la résolution par le logiciel crée sera abordée pour ces exemples. Les limites de ce problème et l'état d'avancement du projet seront indiqués enfin.

1 Analyse du casse-tête

Une première analyse du tangram a permis de mettre à jour plusieurs sous-problèmes. Deux problèmes d'ordre mathématique ainsi qu'un problème d'intelligence artificielle pure ont ainsi été isolés.

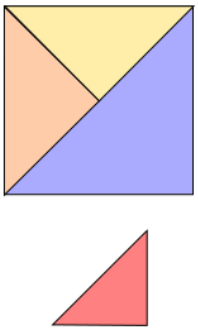
1.1 Placement de pièce dans un modèle

Le premier problème à résoudre est le placement d'une pièce dans un modèle. L'objectif de cette méthode est ainsi de renvoyer l'ensemble des positions possibles pour une pièce dans le modèle.
Pour le développement de cette méthode, l'analyse s'est faite sur la base du raisonnement humain. La première approche, la plus simple pour un humain, consiste à tester toutes les positions possibles que peut prendre une pièce.
Si cette approche parâit intéressante et rapide à effectuer par un cerveau humain, du point de vue de la programmation et donc de l'intelligence artificielle, cette méthode n'est pas très efficace, puisqu'elle implique de tester toutes les solutions.
Pour une résolution efficace, après étude de différentes possibilités, il est apparu qu'il était préférable de "coller" les pièces à une arête du modèle. Cependant, ce "matching" doit se faire d'une manière intelligente.
Le but est donc de chercher les points communs entre la pièce et le modèle afin de trouver un point ou une arête constituant la base du placement de la pièce. Le premier point commun possible entre la pièce et le modèle est la longueur d'une arête. En effet, les pièces du Tangram ayant chacune des caractéristiques différentes, si une arête du dessin coïncide avec une arête de la pièce, il y a de grandes chances pour que la pièce soit positionnée à cet endroit.



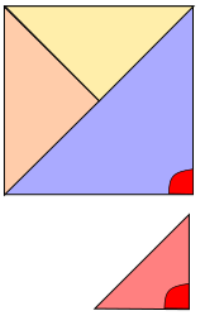
Ici, on peut voir que le triangle a une arête qui coïncide exactement avec une arête du modèle.

S'il est évident que cette méthode est efficace dans de nombreux cas, elle a ses limites : il se peut que lors de la résolution du Tangram, une arête du modèle corresponde à plusieurs pièces. Lors de ce type de configurations la première méthode est donc inefficace.



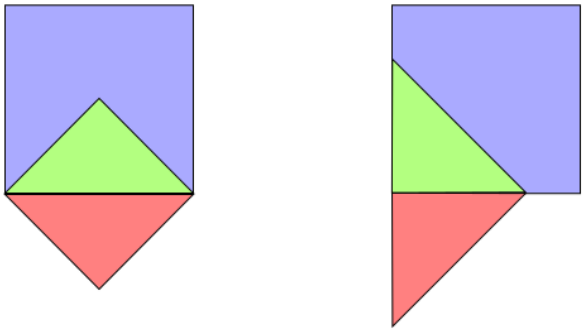
Un problème peut arriver si plusieurs pièces coïncident avec une arête.

Un second point commun, qui va pouvoir résoudre le problème précédent, a été trouvé. Cette seconde caractéristique commune, utile pour la décision de placement de la pièce, est l'utilisation des angles de chaque pièce. Cette méthode n'est utilisée que lorsque la première méthode est inefficace. Il faut alors vérifier si un angle du modèle est égal à l'un des angles de la pièce.



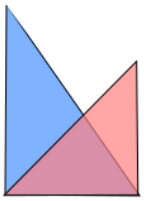
Dans ce cas, on remarque qu'aucune arête ne coïncide. La méthode des angles est donc appliquée et l'on trouve un angle égal.

Une fois la base du placement de la pièce trouvée (le point où l'angle correspond entre pièce et modèle), on obtient soit une arête, soit un angle composé de deux arêtes. Il faut ensuite positionner la pièce en fonction de l'arête (ou des arêtes) à l'aide de translation(s) et rotation(s). Cependant une fois la pièce correctement placée sur le modèle par rapport à la base, un problème important demeure. En effet, selon la rotation et le sens initiale de la pièce, celle-ci peut se trouver à l'inverse de la position voulue.



Dans les deux figures ci-dessus, la pièce doit être placée sur la surface verte. Cependant, dans certains cas, la méthode indique la surface rouge, dans le mauvais sens, donc.

La validation (ou non) de la position de la pièce est donc le problème le plus important une fois la ou les arêtes communes trouvées entre le modèle et la pièce. Après l'obtention d'une position il faut vérifier si tous les sommets de la pièce font partie de la pièce afin de s'assurer qu'elle n'est pas positionnée dans le mauvais sens ou sur arête qui ne permet pas de placer entièrement la pièce. En effet, il ne faut pas que les pièces se superposent, ni qu'elle soient à l'extérieur du modèle.



Ici, une arête coïncidente a bien été trouvée mais il n'y a pas la place pour que la pièce se place dans le modèle. Cette pièce peut alors soit être en dehors du modèle initial, soit sur une autre pièce (puisque notre méthode est récursive et que le modèle dans ce cas est une étape).

Pour résoudre ce problème, un algorithme existant et assez complexe a été utilisé. Pour chaque sommet de la figure, le nombre d'arêtes ayant un point à la même hauteur que le sommet est compté. Dans le cas où un nombre impair de points de chaque coté de l'arête est trouvé, alors le point est à l'intérieur du modèle. Sinon, il se trouve à l'extérieur.
La source de l'algorithme est disponible à l'adresse suivante : *http ://alienryderflex.com/polygon/*

Après avoir testé l'ensemble des points de la pièce positionnée et ainsi que l'ensemble des points de son symétrique par rapport à l'arête de la base, l'une des pièces est identifiée comme étant en partie à l'extérieur du modèle.

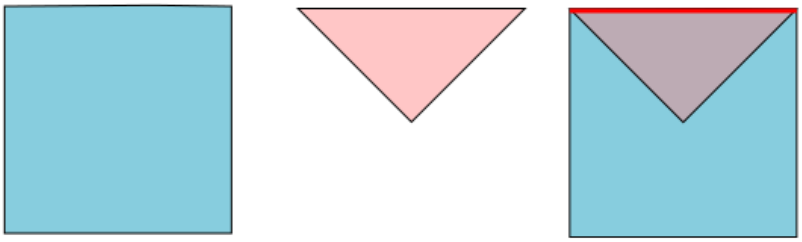
L'algorithme chargé de générer l'ensemble des placements possibles va donc pour une pièce et un modèle donnés, tenter de trouver les positions disponibles. Pour cela, la première méthode va être utilisée puis de la seconde. Dans le deux cas, la validité des résultats trouvés est vérifiée. Dans le cas où aucun placement n'est possible (c'est à dire si le modèle est trop petit pour accueillir la pièce ou si aucun angle ni aucune arête ne convient), l'algorithme échoue.

1.2 Soustraction d'une forme au modèle

Le second problème mathématique que la résolution exigeait de résoudre est la soustraction d'une forme au modèle. En effet, la solution la plus simple résidait dans le fait de placer une forme dans le modèle pour ensuite renvoyer un nouveau problème et recommencer le casse-tête la forme placée en moins avec le nouveau modèle.
Ce problème imposait donc plusieurs sous-étapes pour pouvoir retourner de façon efficace le nouveau modèle. Ces sous-étapes sont la découverte des tous les points du nouveau modèle, puis la suppression d'éventuels points qui seraient inutiles. Une précision importante à apporter est le fait qu'il faut conserver l'ordre des points du modèle afin de conserver la forme voulue. Le travail nécessite donc de passer non pas seulement par des points mais par des arêtes.

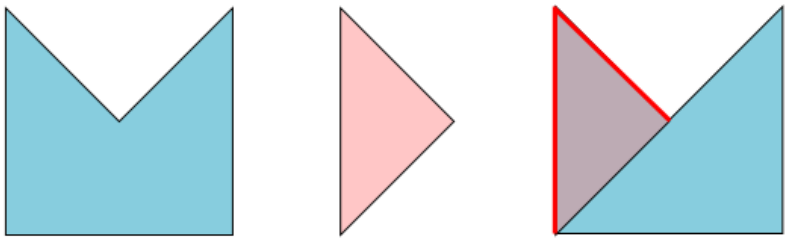
Le fait d'utiliser des arêtes a permis de réunir les deux sous-étapes puisque la suppression de points superflus se fait tout au long de la découverte des nouveaux points. Malgré tout, plusieurs étapes sont nécessaire pour la découverte du nouveau modèle.

La première étape, tout comme lors de l'étape du placement d'une pièce, est de trouver une arête commune à la pièce et au modèle. La différence étant qu'ici, l'algorithme est sûr de trouver au moins une arête commune. Dans le cas où plusieurs arêtes correspondent, l'algorithme prend simplement la première qui arrive puisque le choix de l'arête n'a pas d'importance.
Un fois cette arête commune choisie, on doit vérifier si le sens de la forme est la même que le sens du modèle. En effet, lors de la recherche d'une arête commune, on cherche sans distinction si l'arête est présentée dans le même sens ou non. Par exemple, si sur la forme l'arête a pour coordonnées $(0, 100)$ et sur le modèle les coordonnées $(100, 0)$, il faut "retourner" les arêtes afin que les coordonnées correspondent.
Une fois la pièce à insérer mise en bonne forme, ses arêtes sont ajoutées sans test dans le modèle juste après l'arête commune entre la pièce et la forme. Ainsi, les arêtes de la pièces cont ajoutées au bon endroit dans le modèle.



Ici, on peut voir que l'arête du haut du carré est commune au triangle placé. On va donc placer les arêtes du triangle juste après l'arête du modèle qui est commune à celle du triangle. Comme le triangle possède aussi cette arête, le placement est correct.

Cependant, une fois cette étape effectuée, on est en présence de doublons d'arêtes dans notre nouveau modèle. Cette nouvelle étape va donc vérifier les arêtes en double (quel que soit le sens). Ces dernières sont éliminées du modèle. Aucune occurrence de ces arêtes ne doit rester dans le nouveau modèle. En effet, en faisant apparaître des arêtes en double, l'algorithme indique que ce sont des arêtes qui étaient communes et qui n'ont donc plus lieu d'être du fait de la soustraction de la pièce.



Dans ce cas, on voit que deux arêtes sont communes. La partie de l'algorithme recherchant les doublons d'arêtes va donc supprimer les occurrences de ces arêtes, ici surlignées en rouge. Le point en haut à gauche de l'ancien carré est donc supprimé automatiquement et le nouveau modèle adopte la bonne forme.

Comme le travail est fait sous forme de points, l'algorithme fait repasser ensuite chaque arête en point et vérifie alors par prévention si aucun point n'apparaît en double à la suite. Si c'est le cas, on supprime une occurrence de ce point.
Enfin, comme l'algorithme a utilisé les arêtes, la première et la dernière arête ont un point en commun. Ce point va donc être présent au début et à la fin de la liste des points du nouveau modèle. Il faut donc supprimer celui qui se situe à la fin.

L'algorithme gère naturellement le cas d'arrêt de ce problème puisque lorsque le modèle passé correspond à la pièce, il va retourner un ensemble de points vides.

1.3 Implémentation de l'algorithme DFS (profondeur d'abord)

1.3.1 Analyse de la recherche

Le Tangram est un jeu qui n'est pas représentable par un système de coût suivant le placement de chaque pièce. En effet, toutes les pièces devant être placées, il n'y a pas de concurrence directe entre ces pièces. Même s'il était possible de mettre en place ce système de coût, il ne serait pas pertinent de le faire car les scores de chaque pièce seraient égaux.
Ici, le problème peut se modéliser par un arbre de recherche qui est cependant assez complexe. De très nombreuses configurations sont possibles pour chaque nœud. L'exemple du carré ci-dessous permet d'indiquer comment se construit l'arbre de recherche. Les étapes sont pour la résolution :

- Sélection de la première pièce : un gros triangle ;
- Celui-ci est positionnable sur le dessin mais à plusieurs endroits ;
- Ainsi, en plus d'avoir comme première branche le gros triangle, les branches filles représentent chaque configuration possible du gros triangle ;
- Pour chacune de ses branches, on essaye avec les pièces restantes, qui elles-même possèdent en sous-branche à nouveau leurs différentes configurations.

Ainsi, chaque profondeur est alternée par le choix d'une pièce puis d'une de ses différentes configurations. Si une pièce ne possède aucun placement disponible, elle apparaît tout de même

dans l'arbre mais n'a aucun enfant. S'il est considéré qu'un niveau est constitué de deux profondeurs, c'est à dire d'une pièce ainsi que de ses configurations, une solution est trouvée lorsque la recherche atteint le niveau 7, correspondant au nombre de pièces composant le jeu du Tangram.

1.3.2 Choix de la recherche

Dans la recherche à effectuer, il est donc nécessaire de parcourir chaque branche jusqu'à sa fin. Dans de nombreux cas, plusieurs solutions sont possibles, même si elles sont souvent proches (elles diffèrent pas rotation par exemple). Cependant, une seule solution étant nécessaire, il est plus pertinent de se diriger le plus rapidement vers la fin de chaque branche. Il faut donc exclure l'idée d'utiliser un algorithme parcourant en largeur l'arbre à construire et plutôt se focaliser sur une recherche en profondeur.
L'algorithme DFS (Depth First Search) est donc retenu. Il convient parfaitement à la situation puisqu'il étend le nœud du graphe et ses successeurs le plus longtemps possible jusqu'à atteindre le nœud but, ou jusqu'à atteindre un nœud n'ayant plus de fils. Dans ce cas, l'algorithme effectue du "backtracking" et permet d'explorer les branches suivantes.
Si la recherche atteint un nœud but (c'est à dire que toutes les pièces ont pu être placées avec succès), on arrête la recherche puisqu'on a résolu le problème. Chercher une autre solution ne ferait que surcharger le programme.

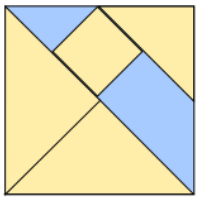
2 Représentation informatique du problème

Le passage de la phase d'analyse à la phase "informatique" du pour la résolution du tangram s'avère beaucoup plus complexe que ce qui avait été prévu. Cependant, des représentations précises ont été adoptées. Ces représentations se traduisent par des structures de données fixes qui ont été mises en place afin de mieux coordonner les différents prédicats Prolog.

2.1 Structures de données

La structure majeure qui a dû être mise en place lors de l'implémentation du projet se doit de permettre la définition des pièces du Tangram ainsi que des différentes patterns.

Après avoir envisagé plusieurs solutions il a été décidé de représenter toutes les figures dans un repère orthonormé où chaque point d'une figure est représenté par deux coordonnées X et Y.
En ce qui concerne Prolog, un point est donc représenté par une liste de deux éléments où le premier est la coordonnée en abscisse (X) et le second celle en ordonnée (Y).
Comme en géométrie où une figure est définie par un ensemble de points, en Prolog, la représentation des figures se structure selon une liste de points, ces derniers étant ordonnés de façon à ce que deux points l'un à coté de l'autre dans la liste forment une arête sur la figure. Plus simplement, les points sont placés dans l'ordre. Bien entendu, le couple formé par le premier et le dernier point représente également une arête.
Les pièces sont donc représentées par des listes de points, chacune de ces pièces ayant un point qui est positionné à l'origine du repère. Grâce à cette modélisation, lors du placement des pièces sur un modèle, il suffira de transformer la figure par rotation ou par translation vers sa position.
Les modèles sont un peu plus complexes à modéliser car ils peuvent être divisés en plusieurs parties pour certaines configurations. Aussi, comme il n'est pas possible de prévoir la position des pièces à l'avance, il se peut que le positionnement d'une pièce dans le modèle forme deux nouveaux sous modèles. Il a donc été choisi de modéliser un modèle suivant une liste de figures.



Dans le cas ci-dessus, on remarque que le placement du carré dans le modèle fait passer celui-ci d'une partie à deux en sortie de l'algorithme de soustraction d'une pièce à un modèle.

2.1.1 Espace d'états

Les états sont composés de plusieurs éléments :
- les pièces restantes à placer ;
- les coordonnées représentant le modèle restant à combler ;
- les pièces déjà placées, ainsi que leurs coordonnées.

Il est donc possible de coder ces états comme suit :
$(Pieces, Modele, PiecesPlacees)$

Pieces est défini dans l'ensemble des pièces disponibles pour résoudre le Tangram (soit 7 pièces, donc 7 valeurs différentes).
PiecesPlacees est défini de manière semblable, sauf que les coordonnées sont définies dans le plan défini par le modèle.
Modele est défini également dans le plan, ses valeurs ne pouvant aussi pas être plus petite ou plus grande que les différentes coordonnées composant le modèle de base. Il est important de noter que toutes les coordonnées sont positives (définies dans la partie Nord-Est du repère).

Exemple pour les coordonnées pour le modèle du carré
Les coordonnées sont définies telles que $0 \leq X \leq 100$ et $0 \leq Y \leq 100$.

Exemple de différents états
État initial pour le modèle du carré :

```
(
(petit_triangle, gros_triangle, moyen_triangle, carre,
parallelogramme, petit_triangle, gros_triangle),
((0, 0), (100, 0), (100, 100), (0, 100)),
()
)
```

État intermédiaire :

```
(
(petit_triangle, moyen_triangle, carre,
parallelogramme, petit_triangle),
((100, 0), (100, 100), (0, 100)),
(
(gros_triangle,
((0, 0), (0, 100), (50, 50))
),
(gros_triangle,
```

```
((0, 0), (100, 0), (50, 50))
)
)
)
```

État final :

```
(
(),
(),
(
(gros_triangle,
((0, 0), (0, 100), (50, 50))
),
(gros_triangle,
((0, 0), (100, 0), (50, 50))
),
...,
(moyen_triangle,
((100, 50), (100, 100), (50, 100))
)
)
)
```

2.1.2 Système de production

Les règles du système de production traitent des contraintes que les pièces placées doivent respecter, mais ne rentrent pas dans le déroulement de la recherche de placement des pièces. Il est très difficile pour un problème de ce type de définir un ensemble de règles précises, puisque celles-ci diffèrent en fonction du modèle à résoudre.
Les règles concernant les ensembles de définition des coordonnées sont les premières à être affectées par ce problème. Ainsi il convient d'adapter à chaque cas ces règles. De plus, le modèle évoluant constamment, il n'est pas possible de définir ces règles, car elles vont évoluer en même temps que le modèle.
Une règle simple ($R0$) pour ceci peut être que chaque point d'une pièce dans $PiecesPlacees$ à un état N doit être dans la surface de $Modele$ à un état $N-1$.

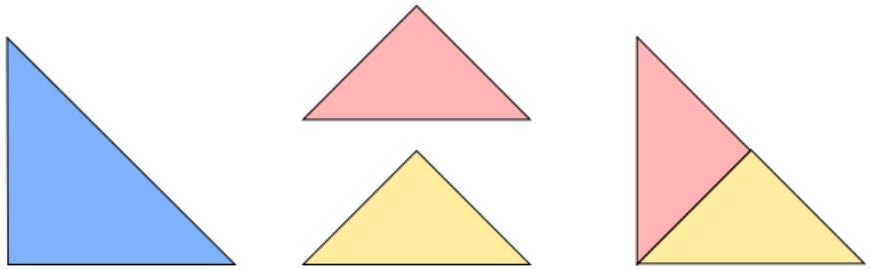
Les autres règles, plus simples à définir, sont :
- ($R1$) Le nombre de pièces dans $Pieces$ est compris entre 0 et 7 ;
- ($R2$) Le nombre de pièces dans $PiecesPlacees$ est compris entre 0 et 7 ;
- ($R3$) L'aire du $Modele$ à un état N doit être égal à l'aire du $Modele$ à l'état $N-1$ moins l'aire de la dernière pièce dans $PiecesPlacees$;
  Remarque : cette règle n'a pas été développée mais pourrait l'être à l'avenir.
- ($R4$) $Modele$ est vide si $Pieces$ est vide et que le nombre de pièces dans $PiecesPlacees$ est égal à 7 ;
- ($R5$) Le nombre de pièces est égal à 7 moins le nombre de pièces dans $PiecesPlacees$;
- ($R6$) Le nombre de pièces dans $PiecesPlacees$ est égal à 7 moins le nombre de pièces dans $Pieces$.

3 Résultats obtenus et exemples

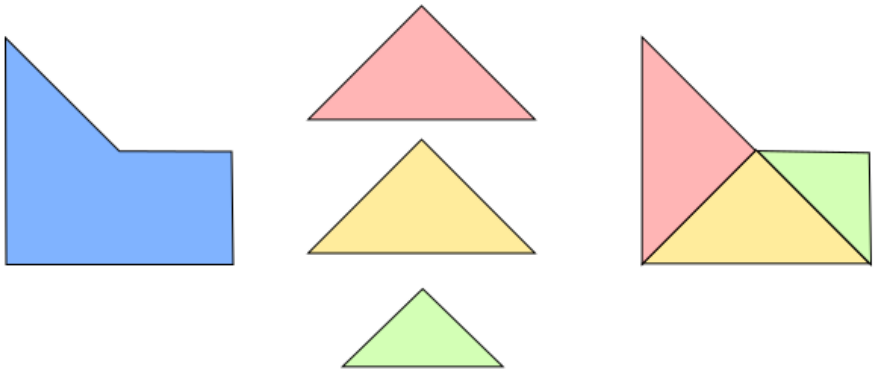
De nombreux problèmes ayant été rencontrés durant le projet, les résultats obtenus n'indiquent pas une résolution totale de différents modèles. Ceci est dû au fait des différentes difficultés apparues sur les algorithmes mathématiques touchant à la géométrie.
Malgré tout, sur des modèles simples, l'ensemble des prédicats fonctionnent ensemble et parviennent à s'articuler pour fournir des résultats. Le logiciel et donc fonctionnel dans son ensemble.

Les deux Tangrams suivants sont résolus sans problèmes par le logiciel et sont issus du modèle carré classique connu de tous :

– Cas du demi carré avec deux triangles à place :



– Cas plus complexe avec trois triangles à placer :



4 Problèmes rencontrés

Comme déjà abordé tout au long de ce rapport, de nombreux problèmes ont été rencontrés lors du projet. La première des difficultés était que le problème n'a que très peu été traité jusqu'à aujourd'hui et qu'il fallait donc partir de zéro au niveau des idées de conception de l'IA. Des sous-problèmes ont donc rapidement été isolés, cependant, c'est pendant la résolution de ces différents problèmes que les plus grosses difficultés ont été rencontrées, notamment pour les parties mathématiques.
Ces difficultés ont été assez rapidement surmontées algorithmiquement, mais lors du passage en Prolog, de nouvelles difficultés ont été rencontrées. En effet, la formalisation de problèmes géométriques dans ce langage ne nous a pas apparu aisé. Il reste donc à ce jour certains cas où les prédicats liés à la géométrie (particulièrement pour le placement) qui ne fonctionnent pas. Ceci est dû au fait que de nombreux différents cas étaient à gérer dont de nombreux qui étaient isolés. Ces cas là n'ont donc pas tous été traités, soit parce qu'ils n'ont pas été découverts, soit parce qu'aucune solution n'a été trouvée.
Du fait de ces difficultés, une interface graphique n'a pas pu être développée par manque de temps. Les résultats sont donc retournés en ligne de commande ce qui n'empêche pas le bon fonctionnement du programme.

5 Améliorations possibles

Comme le programme n'est pas en mesure de gérer tous les cas, de nombreuses améliorations pourraient encore être apportées, en particulier pour la gestion des nombreux cas particuliers. Aussi, comme il en a été question plus tôt dans ce rapport, la vérification par l'aire, c'est-à-dire en comparant l'aire du modèle avec la somme des aires des figures restant à être placées, nous paraissait une méthode intéressante à implémenter. Ceci éviterait de descendre dans des branches vouées à l'échec lors de l'exécution de l'algorithme de recherche d'une solution.

Conclusion

Table des matières

Annexes

A Prédicat soustraction

```
% file : soustraction.pl
soustraction([Patterns], Piece, [NewPatterns]):-

tools:liste_all_couple_aretes(Patterns,CoupleAretePatterns),
tools:liste_all_couple_aretes(Piece,CoupleAretePiece),

search_commmon_arete(CoupleAretePatterns,CoupleAretePiece,Arete),
search_sens_forme(CoupleAretePiece,Arete,Sens),
retourne_forme(CoupleAretePiece,Sens,CoupleAretePiece1),

mix_pattern_forme(CoupleAretePatterns,CoupleAretePiece1,Arete,Mixed),
clean_double_arete(Mixed,CleanedMixed),

arete_to_point(CleanedMixed,NewListPoint),
clean_double_point(NewListPoint,NewPatterns1),
clean_first_last_double_point(NewPatterns1,NewPatterns).
```

- **Patterns** : variable représentant le modèle actuel à réaliser ;
- **Piece** : variable représentant la pièce positionné en fonction du modèle ;
- **NewPatterns** : variable représentant le nouveau modèle.

B Prédicat placePiece

```
% file : placePiece.pl
placePiece(Piece,Patterns,ReturnPiece):-
    test_placement_patterns_exact_match(Piece,Patterns,Placement),
    positionne(Piece,Placement,NewPiece),
    test_all_placement(NewPiece,Patterns,Placement,ReturnPiece).

placePiece(Piece,Patterns,ReturnPiece):-
    test_placement_patterns_angle_match(Piece,Patterns,Placement),
    positionne(Piece,Placement,[Axe1,Axe2],NewPiece),
    test_placement(NewPiece,Patterns,[Axe1,Axe2],ReturnPiece).
```

- **Piece** : variable représentant la pièce à placer ;
- **Patterns** : variable représentant le modèle où placer la pièce ;
- **ReturnPiece** : variable représentant la pièce correctement placée.

C Prédicat essai_piece

```
% file : main.pl
essai_piece(_,[[]],_,_):-!.
essai_piece([], _, _, _) :- !.
essai_piece(_, [], Solution, _) :- !,
    affiche_resultat(Solution).

essai_piece(_, _, _, 0) :-
    write_ln('\tBout de la branche atteint'), !, fail.

essai_piece([Piece|PiecesRestantes], Pattern, [[Piece, Placements]|PiecesRetenues], _) :-
    points_piece(Piece, PointsPiece),
    write('Essai de la pièce '),
    write(Piece),
    placePiece(PointsPiece, Pattern, Placements),
    retirer_piece(PiecesRestantes, Pattern, Placements, PiecesRetenues).

essai_piece([PieceNonUtilisee|PiecesRestantes], Pattern, PiecesRetenues, N) :-
    append(PiecesRestantes, [PieceNonUtilisee], Pieces), N1 is N-1,
    write_ln(' -> la pièce est mise de côté pour le moment'),
    essai_piece(Pieces, Pattern, PiecesRetenues, N1).
```

- **ListePiece** : variable représentant la liste des pièces à placer ;
- **Pattern** : variable représentant le modèle à réaliser ;
- **PiecesRetenues** : variable représentant la liste des pièces placées ;
- **N** : variable représentant la profondeur maximale de l'arbre de réalisation.