13128 CAFFIER Olivier

Gogetter : résolution algorithmique et génération de problèmes

Ayant découvert le jeu Go-Getter, j'ai consacré un certain temps à réussir les différents niveaux proposés. L'idée de résoudre ces problèmes via une démarche informatique m'a donc donné envie d'explorer le domaine de l'optimisation algorithmique et de découvrir une nouvelle facette de ce jeu : la création de niveaux supplémentaires.

Dans les jeux de logique, trouver un problème initial à résoudre, en évaluer la difficulté ou trouver l'unique solution parmi des milliers ne sont pas des tâches aisées. Il semble donc important de développer des algorithmes permettant de développer des jeux de sociétés comme le Go-Getter.

Positionnement thématique (ÉTAPE 1) :

- *INFORMATIQUE (Informatique pratique)*
- *INFORMATIQUE (Informatique Théorique)*

Mots-clés (ÉTAPE 1) :

Mots-clés (en français)	**Mots-clés (en anglais)**
Retour sur trace	*Backtracking*
Allocation de mémoire	*Memory management*
Heuristique	*Heuristic*
"Élagage" de l'arbre d'appels	*Pruning*
Mémoïsation	*Memoisation*

Bibliographie commentée

Dans l'optique de résoudre un problème logique, satisfaire un ensemble de contraintes à l'aide d'une machine est une approche apparue peu après la création des premiers ordinateurs, que ce soit à l'aide d'une étude précise du problème rencontré ou alors d'une généralisation de problèmes similaires (3).

La résolution algorithmique du jeu de société "Go getter" est un problème qui n'a pas encore été spécifiquement traité dans la littérature scientifique. Néanmoins, certains concepts algorithmiques tels que le *backtracking* ou la *mémoïsation* peuvent être utilisés afin de proposer une solution efficace au jeu, étant donné qu'ils possèdent une importance majeure dans la résolution de jeux logiques via la récursivité (2). Les algorithmes dits *gloutons* (1) pourront

jouer un rôle dans la recherche de stratégies utiles. Ces stratégies ont notamment été utilisées dans la résolution du jeu Rush Hour (7), un jeu de logique similaire dont l'objectif est de sortir un véhicule spécifique d'une grille en déplaçant habilement tous les autres véhicules sur cette même grille.

L'élaboration d'une telle résolution nécessite une modélisation sur machine assez précise du jeu, permettant de modifier la structure du plateau, la consulter en temps constant comme un humain afin d'utiliser la *récursivité* (1) sans perte de données, ni ralentissement qui pourraient freiner voire rendre impossible cette résolution. En effet, optimiser le coût de chaque appel récursif est primordial, notamment si l'on sait que celui-ci va mener à une solution incorrecte, d' où l'utilité d'une heuristique (4) et de mémoriser certains résultats afin de procéder à un *élagage de l'arbre d'appels* (5). Il faut donc une structure algorithmique performante afin de stocker ces résultats et établir une fonction suffisamment précise pour écarter tous les cas inutiles dans la recherche de satisfaction de contraintes.

Le stockage de résultats représente un problème à part entière dans cette étude. En effet, sur un plateau 3x3 tel que celui du Go-Getter, il y a environ 9.51x1010 possibilités de construire une solution, on pourrait alors généraliser certaines situations mais on fait toujours face à des données qui, en s'assemblant, représentent une taille importante dans la machine, il sera donc primordial de s'intéresser à la gestion de mémoire (6) et donc au langage de programmation qu' il convient d'utiliser pour que celle-ci soit optimale.

Enfin, après avoir résolu les problèmes du jeu Go-Getter, nous nous intéresserons à la création de nouveaux problèmes. L'approche la plus commune serait de générer aléatoirement des contraintes et d'appliquer notre résolution à ces dernières, le temps d'exécution indiquerait alors le niveau de difficulté du problème généré.

Problématique retenue

Gogetter est un jeu de logique à solution unique. Nous nous proposons ici de résoudre ces problèmes de manière algorithmique puis d'en générer ensuite de nouveaux en nous fondant sur les méthodes mises en place.

Objectifs du TIPE du candidat

1. Modéliser le jeu avec des types pertinents dans le langage de programmation C, permettant ainsi un accès aux données en temps constant et de manipuler les structures à partir de leurs adresses mémoires.

2. Généraliser le problème sur un plateau N x N

3. Effectuer une résolution de type *bruteforce* et l'analyser sur une structure 2x2

4. Effectuer une résolution à l'aide du *backtracking,* optimisée à l'aide d'outils tels que la mémoïsation ou une heuristique, répondant aux contraintes en un temps raisonnable.

5. Générer des problèmes de toute difficulté à l'aide de la deuxième résolution

Références bibliographiques (ÉTAPE 1)

[1] THOMAS CORMEN : Introduction à l'algorithmique *: https://www.mcours.net/cours/pdf /hasclic3/hassclic995.pdf, Chapitre 16 - Algorithmes gloutons, DUNOD, 2001*

[2] MIHALIS YANNAKAKIS : Testing, Optimization, and Games *: https://link.springer.com /chapter/10.1007/978-3-540-27836-8_ 6, Springer, 2004*

[3] HAO WANG : Computation, Logic, Philosophy *: Games, Logic and Computers, Springer, 1990*

[4] JUDEA PEARL : Heuristics : Intelligent Search Strategies for Computer Problem Solving *: Addison-Wesley, 1984*

[5] LEONARD A. BRESLOW, DAVID W. AHA : Simplifying decision trees: A survey *: https://www.cambridge.org/core/journals/knowledge-engineering-review/article/abs/simplifying-decision-trees-a-survey/CEE7A6994E66E821DB4A12DD83DC3810*

[6] MAURIZIO GABBRIELLI, SIMONE MARTINI : Programming Languages : Principles and Paradigms *: Memory Management, Springer, 2023*

[7] CLÉMENT LEGRAND-LIXON : Réalisation d'un solveur pour le problème Rush-Hour *: https://perso.eleves.ens-rennes.fr/people/clement.legrand-lixon/article_ Legrand-Lixon.pdf*