CAHIER D'ANALYSE DES BESOINS

Solveurs et générateurs pour SlitherLink

Auteurs :
Abderrahmane KERROUM
Ayoub LESFER
Camille AUSSIGNAC
Wassim BERRARI

Chargé de TD :
Philippe NARBEL
Client :
Emmanuel FLEURY

01 Février 2019

Sommaire

1 Description du projet

SlitherLink est un jeu de logique qui se joue sur une matrice rectangulaire formée de points. Entre ces points, on retrouve des contraintes (nombres entre 0 et 3). L'objectif est de relier horizontalement ou verticalement les points adjacents afin de créer un cycle non chevauchant. De plus, les contraintes indiquent le nombre de côtés de la ligne qui doivent être adjacents au nombre. Par exemple, si une case contient le nombre 3, trois de ses côtés doivent être touchés par le trait continu du cycle.

Le projet consiste en l'implémentation d'un programme qui résout et génère des grilles de tailles variables (M x N) du jeu Slitherlink, pour cela nous allons utiliser une approche qui se base sur la propagation de contraintes. Cette approche consiste à résoudre des sous-grilles locales et en déduire une solution de la grille totale si celle-ci existe, ceci sera décrit plus en détails dans la section description des besoins. La génération sera basée sur un algorithme qui consiste à créer un cycle sur une grille vide et choisir ensuite les contraintes suivant le cycle créé. Aussi et pour des raisons d'efficacité et d'optimisation nous allons utiliser des structures de données optimisées, ceci demande une étude des algorithmes de parallélisation SWAR. Le projet demande aussi une analyse approfondie de l'existant, en ce qui concerne d'une part les différentes approches à implémenter pour résoudre les grilles, d'autre part des heuristiques pour la génération de grilles (création de cycle, ajout de contraintes, calibration de la difficulté).

2 Analyse de l'existant

- **Slitherlink Comps Project**

  C'est un projet d'étudiants consistant a créer un solveur pour le jeu Slitherlink ainsi qu'un générateur de grilles de taille modulable, de difficultés différentes avec une solution unique. Comme la résolution d'un puzzle Slitherlink est un probleme NP-difficile, le projet se concentre sur la génération et la vitesse de résolution du puzzle.
  Ce projet nous sera utile d'abord pour comparer les résultats et les performances du solveur que nous allons implémenter, de critiquer l'approche utilisée et trouver une approche améliorée en complexité en temps et en espace.

- **Slitherlink, Minisat, and Emscripten**

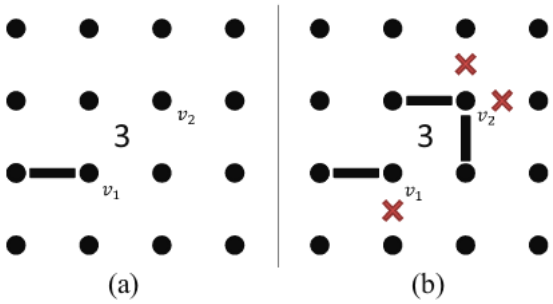
  Ce projet est une démonstration utilisant le solveur SAT "Minisat" pour résoudre les puzzles Slitherlink dans un navigateur web. Une version de code Javascript de Minisat est générée en utilisant Emscripten. Le solveur est écrit en Javascript et tourne sur le navigateur. Il n'est pas résolu sur un serveur, la performance est ainsi réduite.
  Nous allons utiliser ce projet afin de vérifier les solutions produites par le solveur ainsi que l'unicité de la solution d'une grille, puisque ce projet comporte une option pour chercher une autre solution de la grille si elle existe, et enfin de tester les grilles générées.

3 Description des besoins

3.1 Besoins fonctionnels

3.1.1 Trouver une solution de la grille si possible

Nous allons implémenter un solveur de grilles SlitherLink en C11 pour un système Linux. Le solveur utilisera une approche de propagation de contraintes qui se base sur l'application de règles simples de déduction sur chaque point jusqu'à ce qu'aucune règle ne puisse être utilisée. Une règle de déduction est par exemple de considérer les carrées de contrainte 3 (figure suivante (a)), où l'arête gauche de v_1 est dessinée. Si l'arête bas de v_1 est dessinée, alors l'arête haut et droite seront barrées puisque un sommet ne peut être que de degré 0 ou 2. Ce qui contredit la contrainte 3 du carré, donc l'arête bas est barrée (b).



Une fois que les règles de déduction ne peuvent plus être utilisées, il faut implémenter un mécanisme de supposition et backtracking, qui pour une arête vide de la grille suppose si celle-ci est barrée et continue la résolution en utilisant les règles de déduction jusqu'à soit trouver une contradiction et annuler la supposition, soit retrouver un nouvel état de la grille où les règles de déduction ne peuvent plus être utilisées. Dans ce cas, on utilise une approche basée sur une pile de suppositions jusqu'à trouver la solution de la grille si elle existe.

3.1.2 Trouver les solutions possibles pour un nombre de contraintes précis

Une option du solveur est de trouver toutes les solutions possibles d'une grille. La complexité de ce problème varie selon les paramètres de la grille (taille de la grille, nombres de contraintes). Ainsi, il faut limiter le domaine de ces paramètres pour pouvoir résoudre le problème en un temps raisonnable. L'approche dans ce cas est de considérer des suppositions contraires aux suppositions de la solution initialement trouvée, et faire dérouler le solveur afin de trouver une nouvelle solution.

3.1.3 Générer une grille SlitherLink de taille modulable

Afin de générer une grille SlitherLink, il s'agit de créer une grille vide de taille NxM, ensuite de créer un cycle initial aléatoire, et d'utiliser des heuristiques afin de le déformer (ajouter des bosses, des courbes...). Après avoir obtenu ce cycle, il faut l'utiliser pour rajouter les contraintes sur la grille. Enfin, il faut supprimer des contraintes de façon à ne pas obtenir une grille difficilement solvable.

3.1.4 Générer une grille a solution unique

Dans ce cas, il s'agit de manipuler les contraintes générées afin d'assurer l'unicité de la solution. En effet, à l'étape de suppression de contraintes, il faut utiliser un algorithme qui choisit la contrainte à supprimer et vérifie en utilisant le solveur qu'une nouvelle solution n'a pas été créée. Ceci assure donc que la grille générée a une solution unique.

3.1.5 Afficher la solution

L'affichage se fera avec des caractères ASCII sur le terminal ou dans un fichier texte selon l'option spécifiée au lancement. Une option Verbose est demandée, ce qui impose l'enregistrement du déroulement du solveur afin de pouvoir afficher chaque étape.

Voici deux affichages différents possibles.

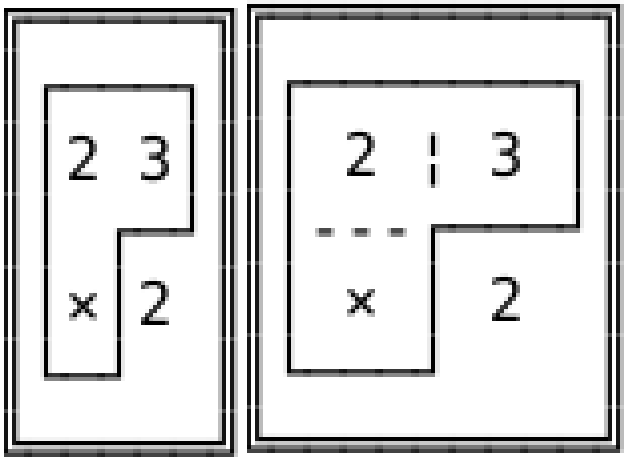


FIGURE 1 – Exemple d'affichage de grille

Les caractères utilisés ainsi que leurs codes ASCII sont fournis :

Remplissage des cases :

- × = 158
- 0 = 48
- 1 = 49
- 2 = 50
- 3 = 51

Bords de la grille :

- ╔ = 201
- ╚ = 200
- ╝ = 188
- ╗ = 187
- ╠ = 186
- ═ = 205

Bords de la courbe :

- ┌ = 218
- └ = 192
- ┘ = 217
- ┐ = 191
- │ = 179
- ─ = 196

3.2 Besoins non fonctionnels

3.2.1 Analyse d'efficacité des différentes approches implémentées

Selon les approches implémentées dans le solveur, il faut fournir une analyse de complexité en temps et en espace et une comparaison entre les différentes approches.

3.2.2 Fournir un corpus de grilles

Il faut fournir un ensemble de grilles de test qui comporte :

— Des grilles dans des cas critiques
— Des grilles faciles à résoudre
— Des grilles difficiles à résoudre
— Des grilles non-solvables par le logiciel (au moins une heure de calcul sans résultat)

3.2.3 Fournir des tests

Il faut fournir plusieurs types de tests :

- Tests unitaires.
- Tests d'intégration.
- Tests fonctionnels

3.2.4 Compatibilité

Il faut assurer la compatibilité avec plusieurs distributions Linux (Debian, Ubuntu, Fedora).

4 Diagrammes

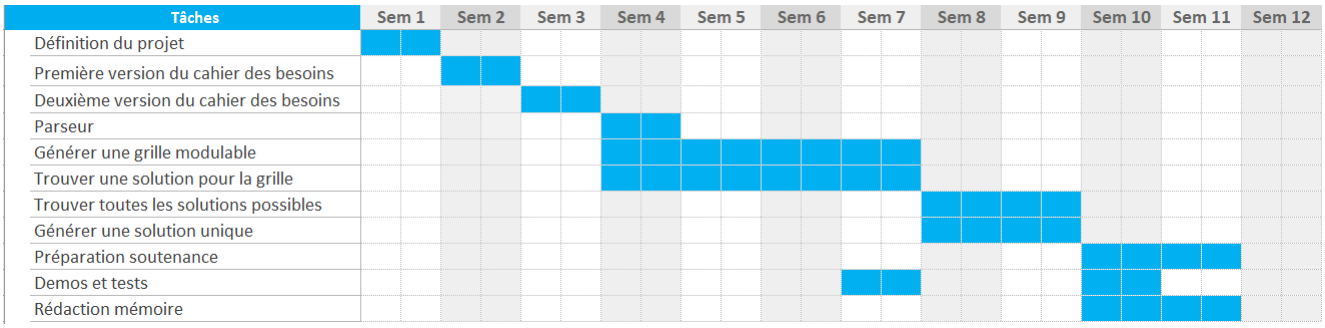


FIGURE 2 – Diagramme de Gantt

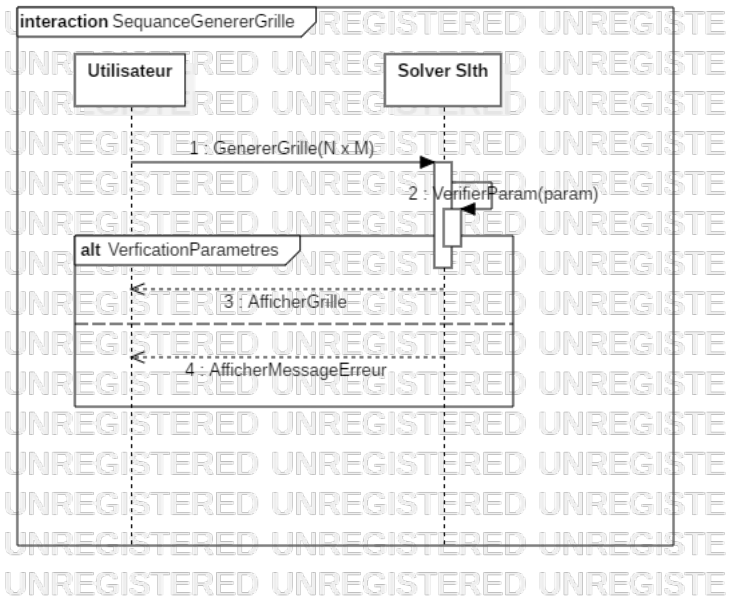


FIGURE 3 – Diagramme de séquence : Générer Grille

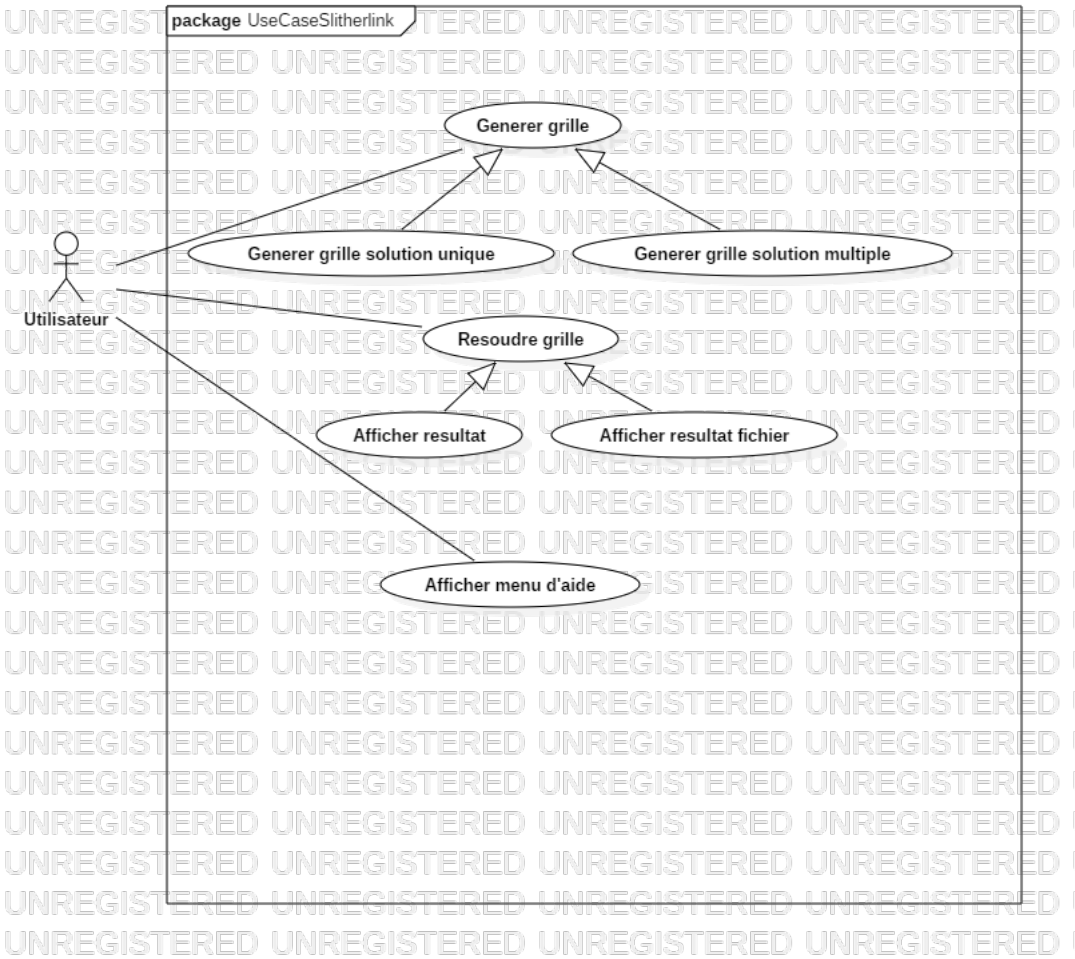


FIGURE 4 – Diagramme de cas d'utilisation

5 Bibliographie

Références

[1] J. DeMaiffe D. W. Brown. “Slitherlink, Minisat, and Emscripten”. In : `http://www.dougandjean.com/slither/index.html`.

[2] E.Kwiatkowski D.Anderson D.Hamalainen. “Slitherlink Comps Project”. In : `http://cs.carleton.edu/cs_comps/1516/slither/final-results/index.html`. 2016.

[3] Stefan Herting. “A Rule-Based Approach to the Puzzle of Slither Link”. Thèse de doct. University of Kent.

[4] Jun Kawahara Ryo Yoshinaka Toshiki Saitoh. “Finding All Solutions and Instances of Numberlink and Slitherlink by ZDDs”. Thèse de doct. Kyoto University, 2012.

[5] Takahiro SETA Takayuki YATO. “COMPLEXITY AND COMPLETENESS OF FINDING ANOTHER SOLUTION AND ITS APPLICATION TO PUZZLES”. Thèse de doct. Tokyo University.

[6] Liu Tung-Ying, Wu I-Chen et Sun DER-Johng. “Solving the Slitherlink Problem”. Thèse de doct. Hsinchu University, Taiwan, 2012.