Projet Technologique : Hashiwokakero

Erwan DUPLAND, Quentin DUART, Adrien JAVEL

12 avril 2017

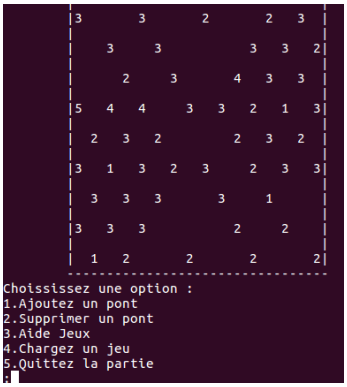
Sommaire

Première partie

Hashiwokakero en mode texte

Partie 1

Hashiwokakero V1



1.1 Les règles originales

Le jeu hashiwokakero est un jeu japonais dans lequel il faut relier des **îles** entre elles. Les îles sont numérotées de **1 à 8** et il ne peut y avoir que 2 ponts au maximum dans une même direction. A la fin, le graphe créé par les liaisons entre ces îles doit être connexe et les îles doivent avoir un nombre de ponts égal au numéro inscrit sur elles.

1.1.1 game.c

Le fichier game.c que nous avons du creer contient pratiquement toutes les fonctions nécessaires au fonctionnement du jeu : Création d'une variable de type game, calcul des degrés des nodes, suppression du jeu ainsi qu'une structure game_s. Nous avons cependant rencontrer quelques difficultés dans la mise en oeuvre de ces dernières.

```
1 struct game_s{
2   int nb_nodes;
```

```
3     node * nodes;
4     int** bridges;
5     int nb_max_bridges;
6     int nb_dir;
7   };
```

game.c

1.1.2 node.c

Le fichier node.c est quant à lui, beaucoup plus simple et compact que game.c. En effet, il contient la structure node_s en elle même :

```
1   struct node_s{
2     int x;
3     int y;
4     int degree;
5   };
```

node.c

De même que les fonctions get_x et get_y permettant de récupérer les coordonnées d'une node passée en paramètre, et la fonction get_required_degree retournant le degrée de la node souhaitée.

1.1.3 Création de libhashi.a

La librairie libhashi.a est une librairie statique créée à partir des fichiers game.c et node.c. Elle regroupera donc toutes les fonctions définis dans ces deux fichiers. Par conséquent, au moment de la compilation, on utilisera l'option **-lhashi** pour "linker" la librairie à l'exécutable ainsi généré.

La ligne de commande pour créer cette librairie est : `ar rvs libashi.a game.o node.o`

Partie 2

Hashiwokakero V2

2.1 Le caprice du client

Maintenant, le client veut que le jeu soit jouable avec de nouvelles règles. Il faut qu'il y ait 8 directions (au lieu de 4 dans les règles originales) et que l'on puisse mettre au maximum 4 ponts dans une même direction.

2.1.1 Nouvelle implémentation de game.c

Le code de game.c reste globalement le même. Il faut cependant rajouter les directions NE, NW, SE et SW et le fait de pouvoir mettre 4 ponts dans une même direction. Le code n'a donc pas beaucoup changé, il a juste été adapté.

Partie 3

Solveur du hashiwokakero

3.1 Implémentation du solveur

Le solveur permettant de résoudre une grille du hashiwokakero est composé de trois fonctions. La fonction ***play_ auto***. La fonction ***solveur_ recursif***. Et la fonction ***solveur***.

La fonction solveur va appeller la fonction play_auto et, ensuite, la fonction solveur_recursif. Normalement, si tout se passe bien, le jeu est censé être résolu.

3.1.1 Play auto

Play_auto est une fonction définit dans le fichier hashi_solve.c. Cette fonction permet de créer les ponts "obligatoires" pour pouvoir résoudre le jeu. Il y a donc moins d'opérations effectuées par la fonction solveur_recursif et on gagne ainsi en rapidité d'éxécution. Nous avons par exemple implémenter des coups obligatoires pour des nodes dont le degré est de 8 : On sait déjà qu'il y aura 2 ponts dans chaque direction, ou bien dont le degré est 7 : on ajoute un pont dans chaque direction. Nous avons également implémenter différentes techniques issues de IndigoPuzzle.

Just Enough Neighbor Technique : On test si le noeud possède uniquement 2 voisins et si les degrés de ces 2 voisins sont les mêmes que lui, auquel cas on ajoute le nombre de pont requis.

One Unsolved Neighbor Technique : On test si le noeud possède exactement un voisin et si son degré est de 1 auquel cas on ajoute immédiatement un pont dans la direction de son voisin.

3.1.2 Solveur récursif

La fonction solveur_recursif est une fonction de type bruteforce. C'est à dire qu'elle va ajouter (si c'est possible) des ponts entre deux nodes.

Étant donné que cette fonction sera appellé après play_auto, elle fera moins

d'opérations que si elle avait été apellée avant (car certains node auront déjà tous leur(s) pont(s) ou au moins quelques-uns).

3.1.3 Conclusion sur le solveur

Le solveur fonctionne correctement sur les instances ***game_ default*** et ***game_ easy***. En revanche, il ne trouve pas de solution pour ***game_ medium***.

Nous avons conclu que ce problème venait de la fonction play_auto à laquelle nous n'avons pas imposer assez de contraintes.

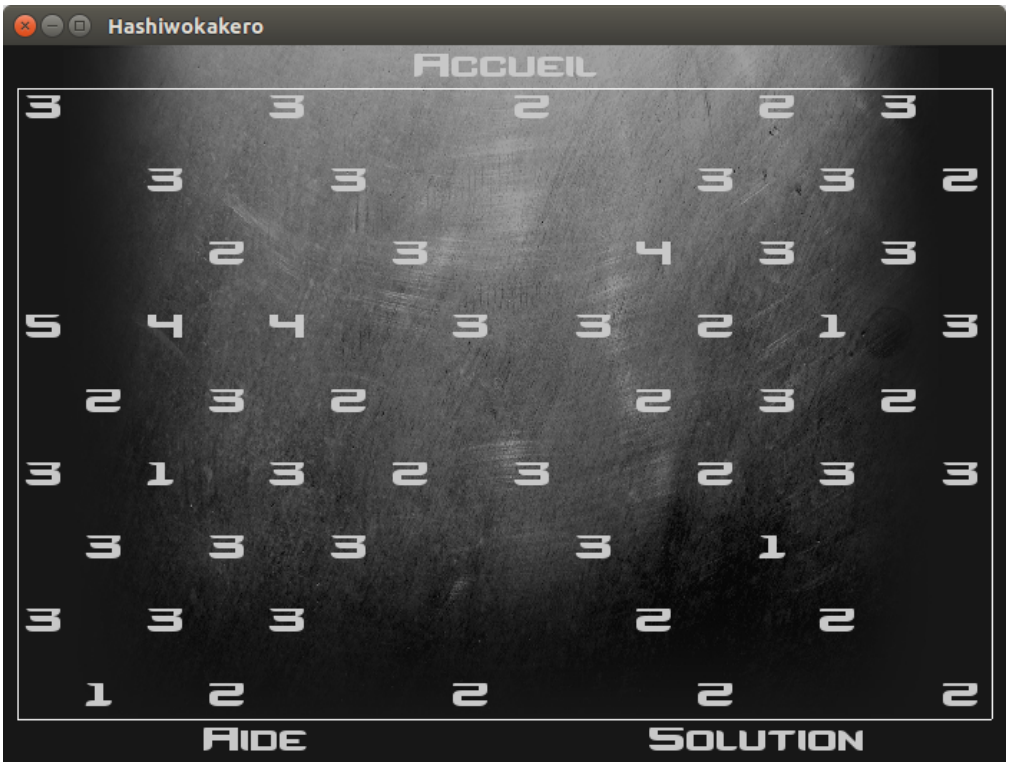
Deuxième partie

Hashiwokakero en interface graphique

Partie 4

Hashiwokakero SDL2

4.1 Le jeu en interface graphique avec la SDL2



4.1.1 Implémentation de l'interface graphique

Pour implémenter l'interface graphique, nous nous sommes aidés d'une interface graphique de base. Nous avons donc utilisé trois fonctions *principales*

qui utilisent des fonctions *auxiliaires* : **init**, **render** et **process**.

La fonction *init* permet d'**initialiser** tous les **paramètres** stockés dans la structure *env*, la fonction *render* permet de **placer** dans l'interface graphique les différents **paramètres** de la structure *env* et la fonction *process* permet d'**intéragir** avec l'interface avec la *souris* ou avec le *clavier*.

Nous avons donc créer un **menu** avec le titre *HASHIWOKAKERO* et les rubriques *Nouvelle partie*, *Règles* et *Quitter*. Si on clique sur la rubrique **Nouvelle partie**, nous avons un nouveau menu qui arrive avec les rubriques *Facile*, *Moyen* et *Difficile* qui permet de choisir le niveau du jeu. Dès que l'on clique sur un des trois niveau, on arrive sur un nouvel écran avec un *cadre*, la notation *Aide*, la notation *Solution* et la notation *Accueil*.

La rubrique **Règles** possède un écran avec, comme son nom l'indique, les *règles* écrites dessus.

La rubrique **Quitter** permet de *fermer* l'interface graphique.

Si nous pouvons améliorer l'interface graphique, nous aurions pu être plus précis dans les zones cliquables en calculant les coordonnées avec précision.

4.1.2 Implémentation du jeu

Pour voir la documentation sur notre interface graphique cliquez ici

L'implémentation du jeu au sein de la SDL n'est pas chose facile : Il a d'abord fallu *afficher chaque noeud* dans un cadre à des coordonnées bien précises. Il a également fallu rendre chaque *noeud cliquable*. Pour cela, dans la fonction process, nous avons vérifié que nous cliquions bien entre les coordonnées de chaque noeud. Une fois que deux noeuds sont sélectionnés, si c'est possible, un pont est tracé. En revanche lorsque, entre 2 noeuds, le degré maximal est atteint, les ponts sont supprimés.

C'est dans la fonction render que nous rendions visibles noeuds et ponts.

Troisième partie

Compilation

Partie 5

Compilation

5.1 Makefile

Dans un premier temps, nous avons du réaliser un fichier ***Makefile***, qui est un fichier particulier permettant de compiler rapidement en utilisant la commande : `make`

Un Makefile est simplement un fichier dans lequel on définit un ensemble de règles nécessaire à la compilation. Par exemple pour compiler un programme s'appellant programme.c et ayant besoin de fichiers include : include1.h et include2.h, on fera :

```
1    programme: programme.c include1.o include2.o
2        gcc $^ -o $@
```

$^ correspond à tous les fichiers après les " :" et $@ correspond au nom du fichier qui devra être créé, ici : programme.

5.2 cmake

Ici, nous avons du créer un fichier ***CMakeLists.txt*** qui est un fichier dans lequel on définit des règles pour créer un Makefile. La syntaxe est un peu plus facile à assimiler et il y a moins de lignes à écrire. Pour créer la cible programme à partir de programme.c et des includes include1.h et include2.h, nous ferons ici :

```
1    add_executable(programme programme.c include1.c
         include2.c)
```

Pour générer le Makefile il faut alors utiliser la commande `cmake <répertoire contenant le CMakeLists.txt>`
Nous n'avons ensuite plus qu'à utiliser la commande `make` pour compiler le programme.