Pierre Medina Hugo Ponsin Al Akoum Abdelkader

Info 402 : Introduction a la logique
Rapport Final

Pour ce projet de logique, nous avons choisit une énigme de la cèlebre licence de jeu d'énigme Professeur Layton, et plus précisément « Professeur Layton et l'héritage des Aslantes »
Et voici une image originel de l'énigme

Rappel de l'intitulé de l'énigme 3 : Et la lumière fuse

Pour le festival, le comité chargé de l'organisation des fêtes souhaite disposer des lanternes un peu partout dans la ville, mais sans installer plus d'une lanterne par rue. En outre, la brigade du bon goût a rigoureusement interdit le croisement de plusieurs faisceaux de la même couleur.

Tout en respectant ces conditions, placez les lanternes de manière à éclairer toutes les rues de cette ville.

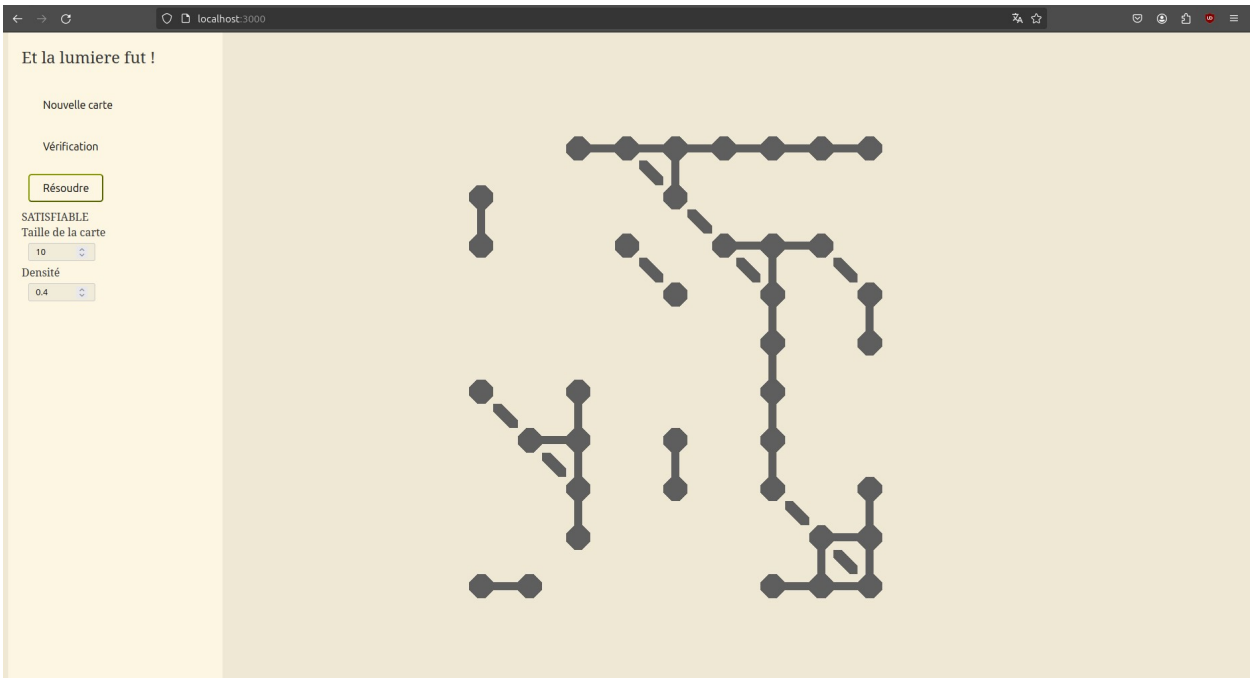
Capture d'écran du jeu Professeur Layton

Pour notre interface homme-machine, nous avons décidé de plonger dans l'univers passionnant du développement web, en utilisant Three.js.
En effet, nous avons trouvé de nombreux avantages :

Tout d'abord, nous sommes conscients que l'aspect d'une interface en mode terminal peut être intimidant pour certaines personnes peu habituées à l'informatique. Pour cette raison, nous avons opté pour une interface web graphique, avec des boutons réfléchis et une palette de couleurs douces.

Ensuite, nous souhaitions minimiser les problèmes de compatibilité entre les différents systèmes d'exploitation. Tester notre programme sur différentes machines est une tâche difficile à réaliser pour nous. Ainsi, seule la machine servant de serveur a besoin de l'ensemble du projet.

Enfin, l'utilisation de Node.js, le développement d'un programme séparé en client/serveur, ainsi que l'utilisation de la bibliothèque socket.io, nous permettent de porter facilement notre logiciel sur n'importe quel réseau. Cela permet à plusieurs utilisateurs d'y accéder sans avoir à télécharger et installer l'ensemble du projet.



Nous avons représenté les croisements en forme d’octogone et, comme dans l’énigme original, on ne peut poser de lanterne que sur un croisement.

Nous avons choisit de simplifier l’énigme par manque de temps en enlevant la contrainte de couleur.

Nous avons donc comme contrainte sur le placement des lanternes :
- Ne peut etre placer que sur un croisements.
- Ne doit pas éclairer une autre lanternes.
- Toute les rue doivent etre éclairer.

De ces contraintes, nous pouvons en déduire une logique plus abstraite.

En effet, nous pouvons facilement déduire que pour que chaque rue soit éclairer, une lanterne doit etre placer dans un croisement en accès directe au minimum.

De plus, nous pouvons en déduire aussi que si un croisement possède une lanterne, tout les croisements en accès directe ne doivent pas avoir de lanternes.

Nous pouvons modélise cela en logique du première ordre :

R(a) = a est rue
C(a) = a est croisements
L(a) = a a une lanterne
A(a,b)= a et b sont en accès direct

pour tout a, (R(a) => il existe b (A(a,b) et L(b) et C(b))

pour tout a, (C(a) => pour tout b (C(b) => non L(b))

Prenons un exemple concret :

Nous posons A le croisement de gauche, B la rue, et C le croisement de droite.

Nous pouvons en déduire en logique propositionnelle les clauses suivantes :

P1 : A => non B
P2 : B => non A
P3 : A ou B

Nous pouvons traduire cela en forme normal conjonctive :

P1 : non A ou non B
P2 : non B ou non A
P3 : A ou B

Nous observons dans cette exemple très simple une répétitions des propositions

Dans le cas ou un croisement a un accès directe a plusieurs croisement, nous séparons en plusieurs clause
Imaginons 3 croisements en ligne : A B et C

P1 : A => non B et non C

En forme normal conjonctive, nous pouvons écrire P1 sous forme :

P1-1 : non A ou non B
P1-2 : non A ou non C

Prenons un exemple du logiciel

Voici une carte généré avec une densité de 0,2 et une taille de 4

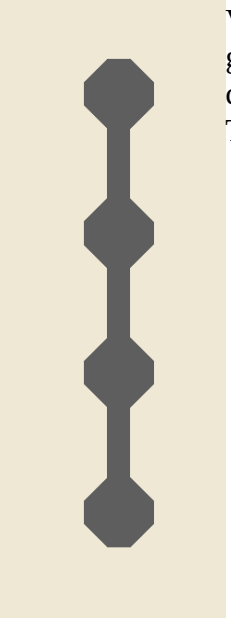
```
9
9

 #
 |
 #
 |
 #
 |
 #
```

Format de sortie du générateur de carte

Voici notre format de fichier pour une carte :

Nous avons choisit d'utiliser une représentation ASCII de la carte, avec en entête, la taille de la carte en X et Y

/!\ Nous comptabilisons dedans un contours vide autours de la carte



Version graphique de la carte par Three.JS

Le logiciel attribue a chaque croisement un nombre unique,
puis, il effectue les calcules de logiques et crée une sortie en clause.

A chaque fois que nous voyons un nombre négatifs, il s'agit de la logique d'un croisements
A chaque fois que nous voyons des nombres positifs, il s'agit de la logique d'une rue.

Comme le logiciel traite les rues et croisement dans le sens de la lecture, nous pouvons facilement déterminer le numéro de chaque croisement.

Le 10 est attribué au croisement du haut et 64 est attribué au croisement du bas.

```
Dimac created and outputing
[ 10, 28, 46, 64 ]
Debut Output DIMACS
[
   [ -10, -28 ], [ -10, -46 ],
   [ -10, -64 ], [ 28, 46, 64, 10 ],
   [ -28, -46 ], [ -28, -64 ],
   [ -28, -10 ], [ 46, 64, 28, 10 ],
   [ -46, -64 ], [ -46, -28 ],
   [ -46, -10 ], [ 64, 46, 28, 10 ],
   [ -64, -46 ], [ -64, -28 ],
   [ -64, -10 ]
]
```

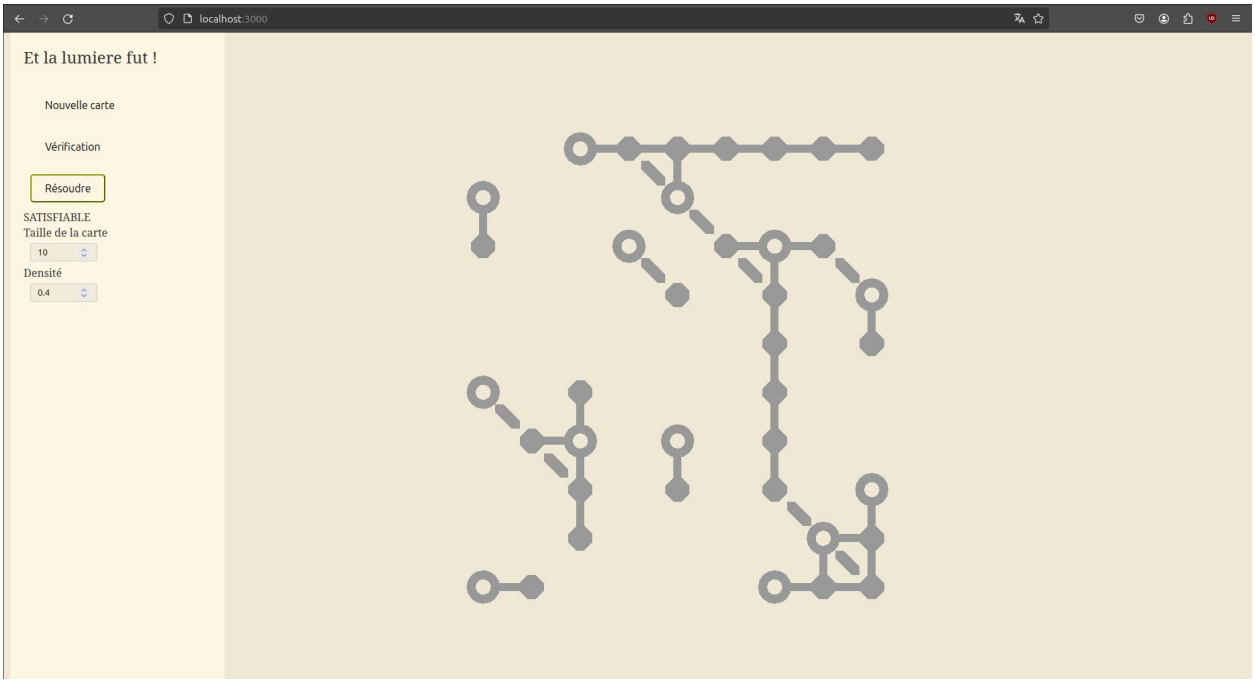
Ensuite, nous appliquons un algorithme qui nous permet de transformer une suite d'entier disjoint en une suite d'entier continue en gardant le même ordre.

```
-1 -2 0
-1 -3 0
-1 -4 0
2 3 4 1 0
-2 -3 0
-2 -4 0
-2 -1 0
3 4 2 1 0
-3 -4 0
-3 -2 0
-3 -1 0
4 3 2 1 0
-4 -3 0
-4 -2 0
-4 -1 0
```

Pour finir, nous appelons le SAT solver pour nous aider a résoudre se systeme

```
Interogation du sat solver
Le fichier ./RuiQmNYHnlsiPMGRAAAB.cnf a été créé
Data READED
s SATISFIABLE
1 -2 -3 -4 0
```

/!\ Il est à noté que nous avons pas fait de systeme anti DDOS. L'appele en grand nombre de carte ayant une importante quantité de croisement peut rapidement causer d'important ralentissement



Voici le résultat final d’une carte ayant un modèle.

/!\ Notre système de génération est tellement performant que l'utilisation de la seed time provoque une génération identique en cas d'appel trop rapide (plusieurs par seconde)

Logique pour la résolution avec la contrainte des couleurs :

Par manque de temps, nous n’avons pas implémenté la contrainte de couleurs dans notre logiciels, mais nous avons fait la logique :

Pour se faire, nous gardons exactement la même logique qu’avant

Nous avons la contrainte supplémentaire suivante ;
« Tout croisements ne peut avoir deux lanternes de couleurs identique les éclairants »

Puisqu'une lanterne ne peut éclairer une autre, aucun croisement avec une lanterne ne peut contredire cette règle, sans violer la clause précédente.

On peut donc interprété la contrainte d’une manière plus simple :

Dans chaque intersection sans lanterne, toutes les lanternes visibles depuis cette intersection doivent avoir des couleurs différentes.

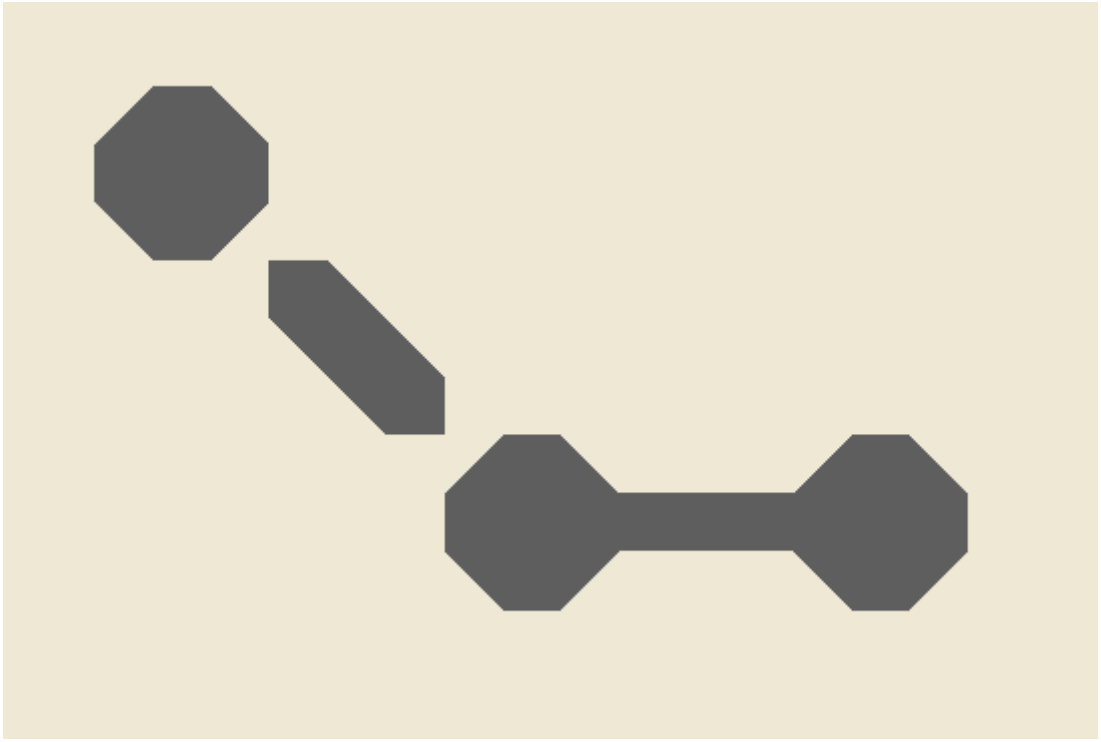
Nous pouvons traduire cela en logique tu première ordre par :

R(a) = a est rue
C(a) = a est croisements
L(a) = a a une lanterne
A(a,b)= a et b sont en accès direct

f(a) = couleur de a

pour tout x, C(x) et non L(x) => pour tout a, pour tout b, (a!=b) et C(a) et L(a) et C(b) et L(b) => f(a)!=f(b)

Pour plus de facilité, partons du principe que chaque couleurs est une dimension
pour cette exemple, nous allons prendre 3 couleurs, nous avons donc 3 dimensions



De gauche a droite, nous avons les croisements A, B et C, ainsi que les rus i et j

Notre logique de non éclairage de lanterne pour le croisement A :

A => non B devient A1 => non B1 et non B2 et non B3
A2 => non B1 et non B2 et non B3
A3 => non B1 et non B2 et non B3

En effet, si la lanterne A est allumer dans une des dimensions, alors les lanternes en accès directe doivent être éteinte dans tout les dimensions.

Notre logique de propagation de la lumière pour la rue i :

A1 ou A2 ou A3 ou B1 ou B2 ou B3

La logique des rue ne change pas beaucoup, il faut qu'une lanterne en accès directe soit allumer dans une des dimension.

Pour notre nouvelle contrainte de non croisement de même couleurs, nous avons sur B :

non B => non (A1 et C1) et non (A2 et C2) et non (A3 et C3)

ce qui se traduit par :

B ou non (A1) ou non (C1)
B ou non (A2) ou non (C2)
B ou non (A3) ou non (C3)

Nous pouvons augmenté le nombre de couleurs en jeu en augmentant le nombre de clause, ce qui a pour conséquence d'augmenté considérablement le nombre de clause pour une carte identique.

Architecture logiciel :

Notre projet est divisé en plusieurs partie distincte :

Dans le dossier ./Pierro_generator, nous avons un programme C responsable de la génération d'une carte de jeu.

Dans le dossier ./sat-solver, nous avons le SAT Solveur pris
ici :https://gitlab.aliens-lyon.fr/ablot/sat-solver.git

Dans le dossier ./public, nous avons toute la gestion coté client (interface utilisateur)
Dans le dossier ./server, nous avons :
- Traducteur en clause logique DIMAC
- Gestion des connections clients
- Fonctionnalité des programmes externes

Le client et le serveur communique grâce a deux version du script socketServices.JS

Pour le guide d'installation et d'utilisation du logiciel, veuillez consulter le README dans le dossier racine.

LICENSE : Veuillez consulter le fichier LICENSE dans le dossier racine.

Pour conclure, notre projet d'introduction à la logique s'est avéré très intéressant et passionnant. Notre choix de reprendre une énigme de la série de jeux "Professeur Layton" nous a permis d'obtenir une énigme à la fois visuelle et stimulante.

L'utilisation de technologies web nous a permis d'obtenir un rendu agréable et accessible à tous, sans pour autant négliger l'aspect essentiel de la fonctionnalité. Toutefois, il est important de souligner que sur le plan du développement, cela a représenté un défi supplémentaire.

Ce projet a également été l'occasion de tenir des réunions, car l'organisation est une clé essentielle lors du développement d'un projet aussi conséquent impliquant plusieurs personnes. Nous avons également défini des normes de nomenclature, de format de fichier et de méthodes de communication entre les différents modules logiciels que nous avons essayé de respecter tout le long du développement. Cela nous a permis de limiter la création de programmes supplémentaires chargés de traduire entre ces différents modules.

Notre approche modulaire nous a permis de réaliser le développement du projet en parallèle, permettent une réalisation plus rapide. De plus, l'utilisation d'outils de développement externe tels que Git-hub nous a permit une gestion très précises des versions.

Vous pouvez aussi retrouver le dépos du projet en ligne :

https://github.com/raybac38/Et_la_lumiere_fut

Source :

https://www.gamekult.com/jeux/professeur-layton-et-l-heritage-des-aslantes-98777/guide/page/4.html