MazeML

Concours du Site du Zéro

Table des matières

C'est les vacances, autant en profiter. Comme ce concours n'est pas noté, et qu'il propose un sujet intéressant, il est assez tentant de s'y pencher... d'ailleurs c'est ce que j'ai fait.

En plus, il y a généralement peu de codes écrits en OCaml... donc cette contribution pourra servir d'exemple à ceux qui veulent s'y mettre. Elle permet d'ailleurs d'illustrer l'utilisation de LablGTK, ainsi que d'autres choses plus ou moins importantes :

- Découpage du code en modules ;
- Dessin et double buffering ;
- Et bien sûr la partie algorithmique elle-même !

Bon, assez de paroles ! Venons-en aux faits.

1 Introduction

J'ai choisi de n'implémenter qu'une partie des fonctionnalités présentées sur la page principale du concours et, en toute franchise, je n'ai pas l'intention d'en faire plus (c'est pour moi un « projet-jouet » pour m'amuser, auquel je ne peux pas consacrer *tout* le temps que je voudrais). Concrètement, voici ce que fait ce programme :

- Créer des labyrinthes parfaits de 4×4 à 240×390 cases.
- Afficher la solution des labyrinthes créés.
- Construire pas à pas un labyrinthe.
- Enregistrer et charger des labyrinthes.
- Exporter des labyrinthes sous forme d'image (PNG, JPEG ou BMP).

2 Caractéristiques du programme

2.1 Nom

Ce programme s'appelle MazeML, nom composé du mot anglais *maze*, qui signifie labyrinthe, et du suffixe ML, pour rappeler qu'il a été écrit en OCaml.

2.2 Licence

2.3 Labyrinthes

MazeML permet de dessiner des labyrinthes de 4×4 à 240×390 cases. Ces limites sont imposées par la taille réduite de la zone de dessin (fixe. . .on peut envisager de la rendre redimensionnable, et permettre ainsi de tracer de plus gros labyrinthes, mais je ne le ferai pas).

Remarque : depuis la version 1.8, l'option `-without-gui` permet de créer et d'exporter de gros labyrinthes (testé jusqu'à 700×700 cases) au format PNG. Il s'agit surtout d'une fonctionnalité *pour le fun* car l'export au format PNG est très lent et `GtkDrawable` n'est pas l'outil le plus approprié pour des dessins de cette envergure. De toutes façons, le programme échoue dès 800×800 avec un bel `Out of memory`.

2.4 Algorithme

MazeML crée des labyrinthes grâce à l'algorithme d'exploration exhaustive. Les informations de backtracking sont stockées au fur et à mesure de la progression. Il est alors possible d'écrire une fonction récursive terminale, ce qui évite les débordements de pile pour de gros labyrinthes.

Remarque : Il existe bien entendu de nombreux autres algorithmes plus ou moins efficaces pour créer des labyrinthes (avec un biais plus ou moins prononcé). Il est sans doute intéressant d'en implémenter plusieurs et de les comparer, mais je ne le ferai pas.

2.5 Résolution du labyrinthe

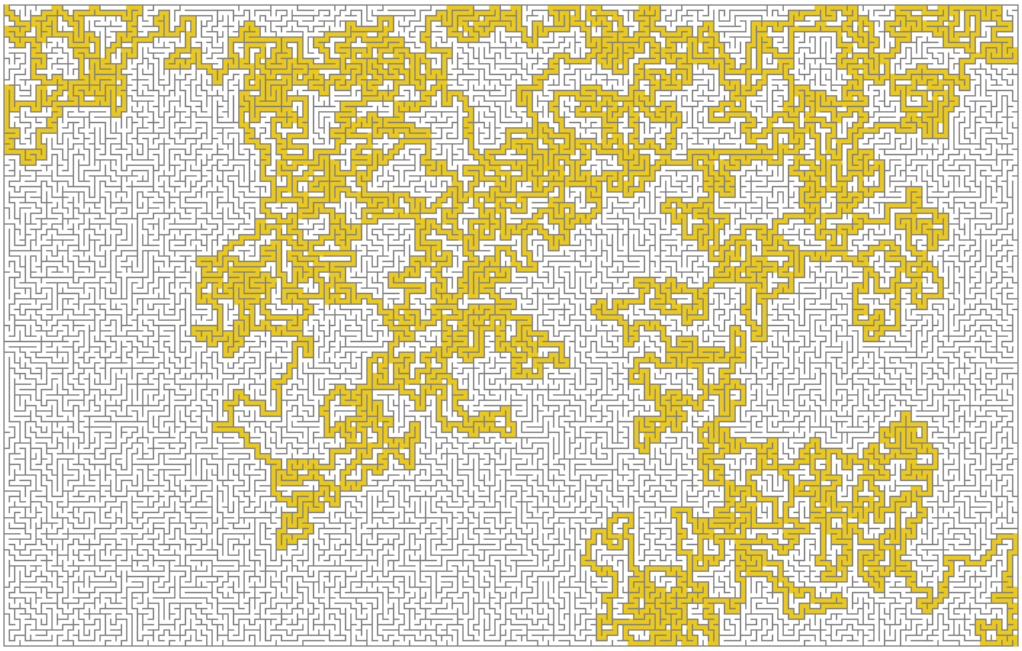
L'algorithme d'exploration exhaustive crée des labyrinthes parfaits. Il n'existe donc qu'un seul chemin pour aller de la case de départ à la case d'arrivée (où qu'elles soient placées). Par conséquent, on peut exploiter astucieusement les données de backtracking pour calculer la solution en même temps que la création du labyrinthe, sans utiliser d'algorithme dédié.

2.6 Affichage des labyrinthes

MazeML affiche les labyrinthes avec les outils de base que sont `GtkDrawable` et `GtkPixmap` (pas d'OpenGL ou de bibliothèque spécialisée). Il s'agit d'une certaine façon d'utiliser des fonctions comparables à celles du module `Graphics` de la bibliothèque standard d'OCaml. Voici un exemple de tracé :

2.7 Implémentation

Les labyrinthes sont représentés par des tableaux de type `Bigarray` à une dimension, composés d'entiers non signés codés sur 8 bits. Chaque case du labyrinthe est ainsi codée par un octet. Plus précisément :



- Les 4 bits de poids fort représentent les portes N (nord), W (ouest), E (est) et S (sud). Ils stockent les informations de backtracking, c'est-à-dire la porte à franchir pour revenir en arrière.
- Les 4 bits de poids faible représentent aussi les portes N, W, E et S. Ils représentent les portes ouvertes d'une case donnée.

Remarque : Les données de backtracking sont effacées dès qu'elles ont été utilisées, sauf si elles font partie du chemin qui relie la case de départ (arbitrairement placée en haut à gauche) à la case d'arrivée (arbitrairement placée en bas à droite).

2.8 Découpage en modules

MazeML est organisé en plusieurs modules relativement indépendants les uns des autres. Ce sont :

Nom du module	Contenu
`UI`	Widgets de l'interface
`Maze`	Algorithme d'exploration exhaustive
`Drawing`	Tracés et fonctions d'export
`Dialog`	Dialogue avec l'utilisateur
`MazeML`	Callback et point d'entrée

3 Benchmark

L'implémentation proposée ici a été testée, *sans affichage*, bien au-delà des limites de MazeML, sur des labyrinthes jusqu'à 15000 cases de côté. Comme il est souvent délicat de tirer une information fiable d'un benchmark, je donne les temps de calcul moyens (10 mesures par taille de labyrinthe) obtenus sur deux machines très différentes (un PC portable assez ancien et un PC de bureau récent) pour 11 tailles de labyrinthes.

Taille	**Nombre de cases**	**Test A** (s)	**Test B** (s)
1000 × 1000	$1.0 \cdot 10^6$	0.5	0.1
2000 × 2000	$4.0 \cdot 10^6$	1.8	0.5
3000 × 3000	$9.0 \cdot 10^6$	4.0	1.2
4000 × 4000	$1.6 \cdot 10^7$	7.1	2.2
5000 × 5000	$2.5 \cdot 10^7$	11	3.5
6000 × 6000	$3.6 \cdot 10^7$	16	5.0
7000 × 7000	$4.9 \cdot 10^7$	21.8	6.8
8000 × 8000	$6.4 \cdot 10^7$	28.3	8.9
9000 × 9000	$8.1 \cdot 10^7$	37	11.4
10 000 × 10 000	$1.0 \cdot 10^8$	48.5	14
15 000 × 15 000	$2.25 \cdot 10^8$	107	31.5

(A) : Acer Aspire 2012 1.5 GHz ; **(B)** : Dell Inspiron 530 (Core 2 Duo 2.83 GHz)

Pour reproduire ces résultats sur votre machine, il suffit de compiler séparément le module `Maze`. Pour cela, ajoutez le point d'entrée suivant dans `maze.ml` :

```
let _ =
  let t0 = Unix.gettimeofday () in
  let maze = make ~rows:1000 ~cols:1000 in
  let t1 = Unix.gettimeofday () in
  Printf.printf "ROWS %d COLS %d MAZE %g TIME %.3f s\n%!"
    maze.rows maze.cols (float maze.last +. 1.0) (t1 -. t0)
```

puis compilez et exécutez en utilisant la commande suivante :

```
$ ocamlopt -nodynlink -inline 1000000 bigarray.cmxa maze.ml -o maze
$ ./maze
```

4 Manipulation en ligne de commande

MazeML est en partie configurable en ligne de commande (vous pouvez obtenir la liste complète des options reconnues grâce à `-help`). Voici les principales options reconnues :

`-rows` n : Nombre de lignes.

`-cols` n : Nombre de colonnes.

`-wall RRGGBB` : Couleur des murs du labyrinthe.

`-cell RRGGBB` : Couleur des cases de la solution.

5 Édition de labyrinthes

C'est un point qui m'intéresse parce qu'il permet d'illustrer l'efficacité des fonctions de type `map_file`. MazeML les exploite bien entendu. Il enregistre et charge des labyrinthes stockés dans un simple fichier texte (jeu de caractères ISO-8859-15). Celui-ci se présente sous la forme :

```
ROWS 0000 COLS 0000\n
(contenu du tableau)
(contenu du tableau)
(contenu du tableau)
...
(contenu du tableau)
```

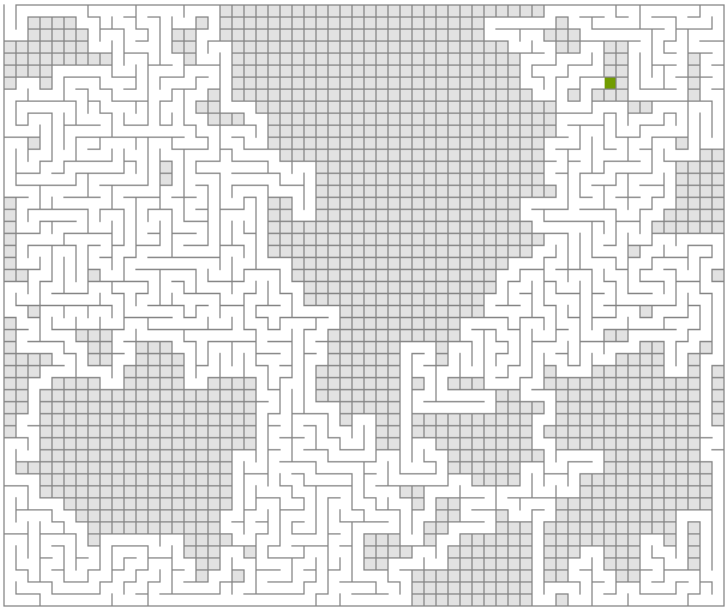
De façon très générale, pour un labyrinthe de n lignes et c colonnes, la taille du fichier est de $n \times c + 20$ octets (ces 20 octets servent pour `ROWS` et `COLS` comme indiqué plus haut).

6 Export

MazeML propose d'exporter les labyrinthes au format PNG (avec transparence) ou JPEG. L'export s'effectue avec ou sans la solution du labyrinthe, selon ce qui est affiché à l'écran. La couleur rendue transparente peut être modifiée en ligne de commande.

7 Construction pas à pas

La version 1.7 de MazeML introduit une nouvelle fonctionnalité : la construction pas à pas des labyrinthes. Pour tester cette fonction, vous devez lancer le logiciel avec l'option `-interactive`. Vous pouvez régler la vitesse des animations avec l'option -rate n où n désigne le nombre de millisecondes qui sépare deux tracés. Voici une illustration de cette fonctionnalité :



Remarque : Cette fonctionnalité n'est pas disponible pour les labyrinthes chargés à partir d'un fichier, car les informations de backtracking nécessaires à l'animation sont perdues lors de l'enregistrement.

8 Historique des versions

Version 1.0 Première version proposée sur le SdZ.

Version 1.1 Optimisation selon les résultats du profiling (gprof).

Version 1.2 Simplification de la mise en mémoire de la solution.

Version 1.3 Meilleur affichage du chemin solution (tracé continu).

Version 1.4 Abandon des modifications de 1.3, nouvelle implémentation avec `Bigarray`

Version 1.5 Export au format PNG (avec transparence) ou JPEG.

Version 1.6 Tracé continu du chemin solution (restauration de 1.3).

Version 1.7 MazeML peut désormais construire pas à pas un labyrinthe.

Version 1.8 Export de gros labyrinthes avec `-without-gui`.

Version 1.9 *(à venir)* Version finale.