Auteurs : PERRETTE Emilien, LE GAL LA SALLE Marc, HUCHOT Pierre

Document explicatif mini projet labyrinthes:

I. Cahier des charges

Notre programme a pour but de créer des labyrinthes aléatoirement et de les résoudre. L'utilisateur peut choisir la taille du labyrinthe et plusieurs méthodes de générations et de résolutions.

L'intérêt du programme repose sur deux fonctions principales :

- Afficher étape par étape la construction et la résolution du labyrinthe. Cette fonction permet de comprendre les différentes méthodes utilisées, qui s'avèrent souvent des versions inspirées d'algorithme de parcours de graphe (en largeur, en profondeur, …).

- Mesurer le temps que prend chaque algorithme (génération ou résolution) pour se compléter, on peut ainsi observer dans quel combinaison génération/résolution fournit les meilleurs/moins bon résultats.

De plus, chaque algorithme de génération ayant ses spécificités, on peut observer des motifs propres à chacun.

Dans un souci de simplicité pour la résolution, on se limite à la génération de labyrinthes parfaits, sauf une option particulière qui permet de mettre en lumière l'inefficacité de certains algorithmes de résolutions.

II. Description du problème posé

Le problème posé est le suivant : comment générer et résoudre des labyrinthes parfaits ?

Un labyrinthe parfait. Cela veut dire que dans un labyrinthe parfait il n'y a pas d'îlots isolés du reste du labyrinthe, toutes les cases sont accessibles et il y a un unique chemin qui relie deux cases différentes.

Notre problème se décompose naturellement en deux sous-problèmes : la génération et la résolution.

La difficulté de la génération consiste à créer un labyrinthe qui ne contient pas d'îlots. Il existe des travaux sur les labyrinthes décrivant des méthodes pour les créer et dont nous avons pu nous inspirer.

Pour la résolution nous nous sommes largement inspiré des algorithmes bien connus de parcours de graphes et leurs variantes, ainsi que d'algorithmes spécifiques aux labyrinthes.

III. Principe des algorithmes

III.1. Génération

L'algorithme doit créer et résoudre des labyrinthes. Pour les trois méthodes de créations de labyrinthes parfaits, les algorithmes commencent par générer des grilles des cases séparées par des murs. Les cases sont carrées donc on a quatre murs générés par case. Ensuite on va parcourir le tableau de cases de différentes manières pour « casser » des murs entre les cases et créer des couloirs. On continue avec ce procédé jusqu'à ce que l'on obtienne des labyrinthes parfaits.

L'algorithme ***unicitéChemin*** associe à chaque case un numéro de zone. Chaque case a donc initialement un numéro différent. L'algorithme choisit ensuite une case et une case adjacente aléatoirement et si la case avec laquelle elle va fusionner a un numéro de zone différent alors les cases peuvent fusionner et les numéros de zone sont recopiés, afin de former qu'une seule zone au lieu de deux. On recommence jusqu'à ce que l'on ait plus qu'une seule zone. Le labyrinthe créé est donc parfait puisque lors de sa création, il est divisé en zones qui sont réunies de manière unique.

L'algorithme ***récursivitéAléatoire*** choisit une case aléatoirement et créé un couloir en serpentin en cassant des murs. Un même serpentin ne peut pas boucler sur lui-même car l'algorithme enregistre les cases déjà visitées. Lorsque le couloir atteint une extrémité du labyrinthe ou lune case lui appartenant, le couloir cesse d'avancer. Un autre couloir en serpentin est donc commence don à un endroit qui n'a pas été visité par l'algorithme. Un couloir en serpentin ne pourra être communiquant qu'avec un seul autre couloir. L'algorithme continue jusqu'à ce que toutes les cases aient étés visitées. Les différents couloirs ne communiquant que par une seule case, le labyrinthe est parfait.

L'algorithme ***ArbreBinaire*** choisit un biais de construction (en haut ou en bas, à gauche ou à droite) et parcours toutes les cases du labyrinthe en supprimant un mur de chaque case de manière aléatoire suivant le biais choisi au départ. Cet algorithme est de loin le plus simple présenté ici pour la génération mais en contrepartie un peut voir clairement le biais choisi se refléter dans la disposition du labyrinthe (de plus sur deux côtés du labyrinthes deux longs couloirs sont présents et sont aussi dus aux biais choisi) .

L'algorithme de ***labyrinthe à Ilot*** créé des couloirs carrés concentriques avec des portes entre les couloirs. Le labyrinthe n'est pas parfait mais il est fait pour mettre en évidence l'incapacité de certains algorithmes de résolution à le résoudre. En effet l'algorithme RésolutionDroite est incapable de résoudre le labyrinthe à ilots.

III.2. Résolution

Plusieurs algorithmes de résolutions sont proposés. On peut relever l'importance de RésolutionAléatoire qui met en valeur l'importance de ne pas parcourir un labyrinthe à l'aveuglette.

L'algorithme ***RechercheDroite*** est une évolution de RechercheAléatoire. Cette algorithme permet de parcourir le labyrinthe en suivant systématiquement le mur de droite jusqu'à passer par la case arrivée. Une fois l'arrivée trouvée, toutes les impasses sont supprimés afin de garder le chemin le plus rapide.

L'algorithme ***RechercheLargeur*** implémente le parcours de graphe en largeur. Il considère chaque case comme un sommet du graphe. On le lance sur la case de départ et il ajoute tous les voisins de cette

case à une liste de case qui attendent d'être traitées. A chaque tour de boucle la prochaine case traitée est celle en première position dans la liste. On utilise donc une queue pour stocker l'ordre des cases à parcourir.

L'algorithme ***RechercheProfondeur*** implémente le parcours de graphe en profondeur. Le principe de base de ce labyrinthe est le même que RechercheLargeur, mais dans celui-ci on utilise une pile pour stocker l'ordre des cases à parcourir. Cela a pour effet que l'algorithme va chercher dans un chemin jusqu'à être bloqué et ensuite remonter à la dernière intersection visitée.

IV. Bibliographie

Liste non exhaustive des ressources utilisées pour la réalisation de ce programme.

https://ilay.org/yann/articles/maze/

https://fr.wikipedia.org/wiki/Algorithme_de_parcours_en_profondeur

https://fr.wikipedia.org/wiki/Algorithme_de_parcours_en_largeur

http://www.astrolog.org/labyrnth/algrithm.htm

http://weblog.jamisbuck.org/2011/2/1/maze-generation-binary-tree-algorithm

https://perso.liris.cnrs.fr/christine.solnon/supportAlgoGraphes.pdf

V. Carnet de route

Séance 1 (25/02) : Documentation sur les algorithmes existants. Réalisation d'un diagramme UML provisoire. Choix des algorithmes de génération.

Séance 2 (04/03) : Développement de la classe RécursivitéAléatoire et ArbreBinaire : Création d'une première version de la classe affichage afin de réaliser les premiers tests.

Séance 3 (11/03) : Finalisation de RécursivitéAléatoire et ArbreBinaire. Développement de la classe UnicitéChemin. Choix des algorithmes de résolution.

Séance 4 (18/03) : Finalisation UnicitéChemin. Développement de RésolutionDroite, RechercheLargeur et RechercheProfondeur. Optimisation de l'affichage dynamique.

Séance 5 (25/03) : Finalisation de l'affichage et des algorithmes de résolution. Implantation de chronomètre au sein des algorithmes de génération et de résolution.

Séance 6 (01/04) : Correction des dernières erreurs rencontrées lors de la résolution. Mise au propre des algorithmes.

VI. Amélioration possible

Voici différents éléments qui peuvent être amélioré dans ce programme :

- Suppression des artefacts d'affichages du labyrinthe
- Amélioration de la fonction chronomètre qui est actuellement perturbé par l'affichage