TRAVAIL 1

PRÉSENTÉ À

M ADAM JOLY

COMME EXIGENCE PARTIELLE

DU COURS

ANALYSE ET CONCEPTION D'ALGORITHMES (INF1008)

PAR

TOMA ALLARY,

ET BENOIT MATTEAU

01 AVRIL 2021

Contents

Problèmes et difficultés rencontrés

Tout d'abord, il faut savoir que l'idée du labyrinthe nous a inspiré la création d'un jeu que nous avons créé dans Unity. De ce fait, il va de soi que nous avons rencontré plusieurs difficultés dans la conception de notre programme.

Du mouvement du personnage à l'animation de la lumière des torches, en passant par la caméra principale bloquée par les murs et l'éclairage de la minimap qui ne doit pas être détecté par la caméra principale. Au final, le plus simple aura probablement été de faire l'algorithme pour générer le labyrinthe.

Les principales difficultés que nous avons eues en ce qui a trait au devis de travail sont les suivantes :

Nous avons d'abord essayé de générer le labyrinthe de façon récursive, mais Unity n'appréciait vraiment pas ça, car les nœuds se créent en temps réel, il a donc fallu recommencer la création des nœuds de façon linéaire

Une autre difficulté que nous avons eu est lors de la compilation des statistiques, à cause de la façon particulière dont Unity fonctionne, la génération du labyrinthe et l'affichage se font en même temps et sont indissociables l'un de l'autre. Nous y reviendrons plus en détail dans la section appropriée.

Instructions spéciales

Nous avons conçu notre jeu avec Unity 2019.4.18. Il vous faudra utiliser la même version pour ouvrir le jeu dans Unity. Vous pouvez aussi simplement lancer le jeu à l'aide de du build fournit avec le travail. Les fichiers de code sont situés dans le dossier
\INF1008_TP1\PrimMaze\Assets\Scripts

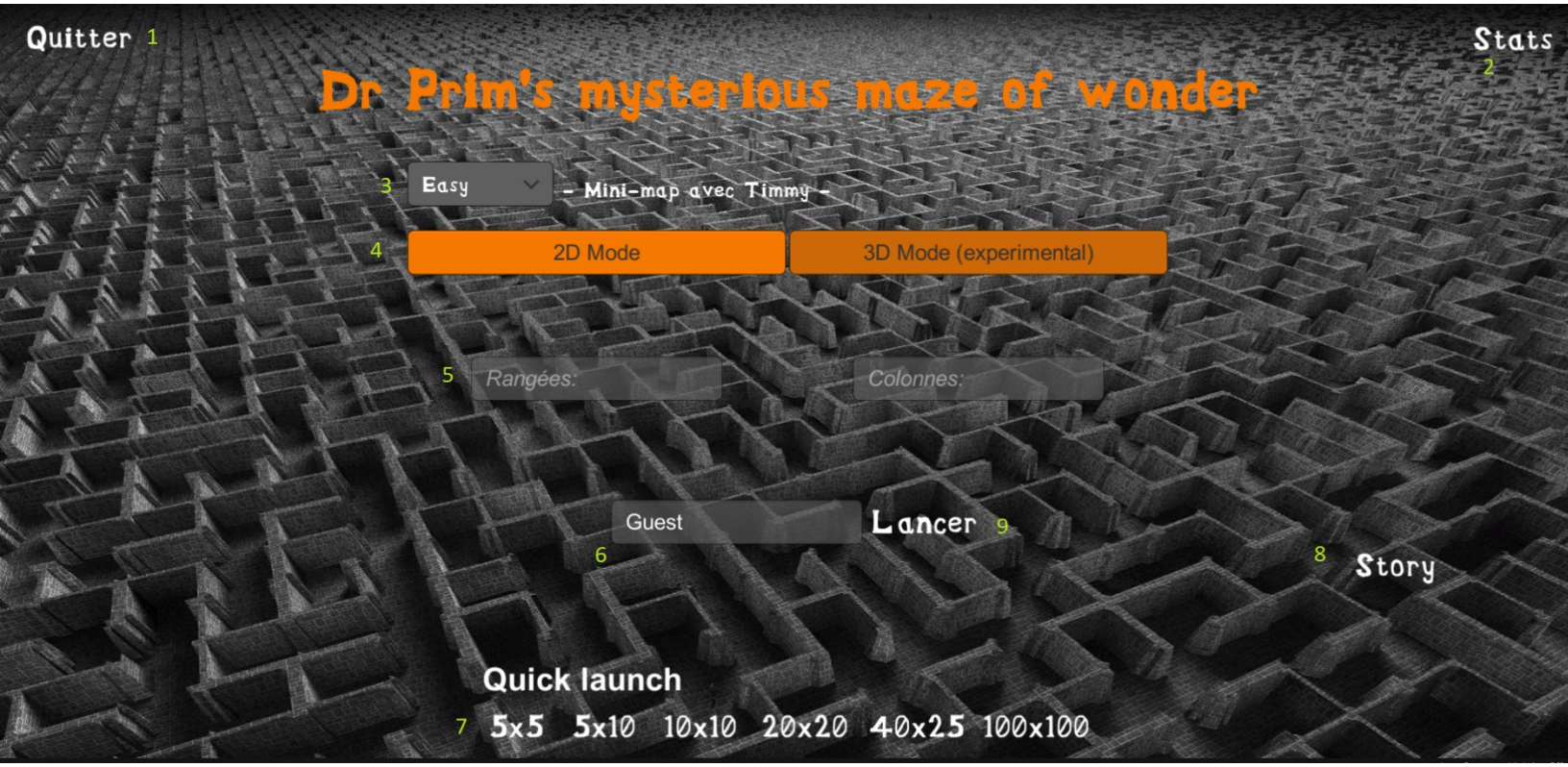
Contrôles :

Pendant la partie, le joueur peut contrôler Timmy en utilisant les touches WASD du clavier. La caméra se contrôle avec la souris, et il est possible de mettre le jeu sur pause en appuyant sur la touche escape. Le but du jeu est de se rendre au bout du labyrinthe le plus rapidement possible. La cible est marquée sur la minimap par un carré vert.

Précisions sur les fichiers de script :

Comme nous avons fait un jeu complet dans Unity, nous avons besoin de beaucoup de fichiers de scripts. Le code qui intéressera le correcteur se trouve au niveau des fichiers architect.cs qui se charge de controller le labyrinthe au complet, noeudScript.cs, qui gère l'objet nœud, lienScript qui gère l'objet lien, et globalScript qui contient des variables globales, notamment les entiers incrémentés lors de l'observation des complexités.

Menu d’intro :



1- Bouton quitter : Permet de quitter le jeu (fonctionne uniquement dans le build)

2- Bouton stats : Permet d’afficher l’écran avec les statistiques des opérations de l’algorithme

3- Sélecteur de difficulté : Permet de choisir entre 3 difficultés (la différence est au niveau de la minimap).

4- Sélecteur de modes : le mode 2d est le mode normal de notre projet, dans lequel Timmy doit trouver son chemin dans un labyrinthe. Le mode 3d est un projet personnel de Toma dans lequel Timmy doit trouver son chemin dans un labyrinthe en 3 dimensions. Notez que le labyrinthe 3d n’est pas construit avec l’algorithme de Prim. Il est créé à partir d’une algorithme de recursive backtracking qui assure une solution unique au labyrinthe.

5- Permet de sélectionner le nombre de rangées et colonnes que le labyrinthe construit aura.

6- Permet au jouer d’inscrire son nom

7- Quick launch : Permet de générer le labyrinthe avec des dimensions prédéfinies. C’est ces dimensions qui seront prises en compte pour le calcul des opérations.

8- Permet d’entendre l’histoire du labyrinthe du Dr prim

9- Permet de lancer la partie selon les réglages choisis.

Questions :

Démontrez que votre implémentation pour la partie de l'application qui exécute l'algorithme de Prim est bien de complexité algorithmique Θ(n2)

La façon dont nous avons choisi de procéder pour répondre à cette question est d'incrémenter une variable chaque fois que le programme passe dans une boucle liée à l'algorithme.

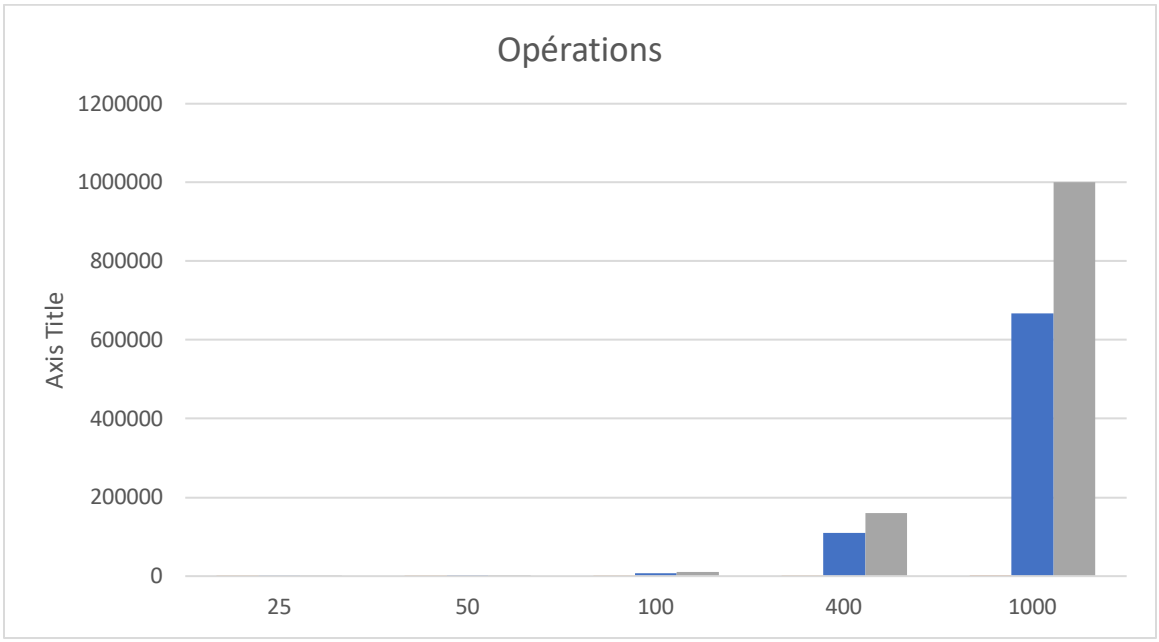
Les valeurs obtenues sont conservés dans des fichiers textes pour les labyrinthes dont 'n' a la valeur de 25, 50, 100, 400 et 1000. Chaque valeur de 'n' a son propre fichier.

Ces fichiers se trouvent dans : \INF1008_TP1\PrimMaze\Assets pour la version projet, ou

\PrimMaze\PrimMaze_Data dans la version build.

Les moyennes de ces nombres sont recalculées à chaque lancement de l'écran de statistiques, et sont affichés sous « moyenne instanciation ».

Le graphique des moyennes ainsi obtenu à la fois pour l'algorithme de Prim et l'affichage du labyrinthe est le suivant :



À des fins de comparaisons, nous avons ajoutés la courbe n^2 en gris pour mieux illustrer notre propos. On voit ici clairement que la courbe Prim en bleu suit une courbe exponentielle se rapprochant de $0.7n^{2.}$

De plus, nous avons comptabilisé les valeurs maximales et minimales obtenues avec chaque valeur dans le tableau suivant :

N	25	50	100	400	1000
min	476	1,731	6,755	107,454	638,397
max	594	2,042	8,019	114,695	700,844

On peut en conclure que les valeurs obtenues sont toujours entre 0.6 n^2 dans le meilleur des cas, et 0.8n^2 dans le pire des cas. Nous avons donc bien à faire à un algorithme de complexité $\Theta(n^2)$.

Quelle est votre complexité algorithmique pour la lecture/le balayage de votre labyrinthe

Il faut premièrement savoir que, dans notre cas, l'affichage du labyrinthe et sa création se font en même temps et sont donc absolument indissociables. En effet, nous commençons par instancier des objets nœuds sous forme de cases au sol, et des objets liens, sous forme de murs, dans l'espace jeu, en les plaçant sous forme de quadrillé avec les dimensions demandées par le joueur. Les objets ainsi créés sont gardés dans des tableaux dans le script Architect. De plus, les nœuds contiennent des références sur les liens adjacents et vice et versa.

Nous avons donc procédé de la même manière que à la question précédente en incrémentant une variable à toutes les boucles qui touchent à la génération / affichage du labyrinthe, et de comptabiliser les résultats pour 'n' = 25, 50, 100, 400 et 1000. Ici, nous obtenons toujours la même valeur tant que le nombre de rangées et de colonnes est identiques. Par exemple, pour 'n' = 400, dans le cas ou on génère un labyrinthe de 20 rangées par 20 colonnes, nous obtenons un total de 1160 opérations. La raison pour laquelle le nombre est toujours le même, est que nous n'avons plus de composante aléatoire. Le labyrinthe, dans la mesure ou on génère tous les nœuds et tous les liens à chaque fois, se crée toujours de la même façon.

Bien sûr, si on change les dimensions pour une même valeur de 'n', on un nombre d'opérations différent, mais la différence reste négligeable. Par exemple, toujours pour 'n' = 400, dans le pire des cas, on obtient 1160 opérations, pour 20 rangées par 20 colonnes, et dans le meilleur des cas, on obtient 799 opérations pour un labyrinthe de 400 rangées par 1 colonne, ou 1 rangée par 400 colonnes, ce qui fait, en passant, un labyrinthe particulièrement ennuyeux.

Le tableau suivant montre les variations avec toutes les valeurs que nous enregistrons.

N	25	50	100	400	1000
min	49	99	199	799	1,999
max	65	135	280	1,060	2,935

On peut donc aisément constater que, pour toutes les valeurs, le nombre d'opération ici oscille entre 2n-1 et environ 2.7n à 2.9n.

Il s'agit donc d'un algorithme de complexité $\Theta(n)$.

Quelle est la complexité générale de tout le processus de génération du labyrinthe (incluant l'initialisation, la génération aléatoire des poids pour les arêtes, l'exécution de la matrice de Prim et toute autre instruction reliée)?

La complexité générale de la génération du labyrinthe peut se calculer par l'addition des algorithmes de génération et d'instanciation du labyrinthe. Selon les réponses déjà fournies, nous pouvons donc en conclure que la complexité générale se situe environ entre $0.6n^2 + 2n - 1$ dans le meilleur des cas, et $0.8n^2 + 2.9n$ dans le meilleur des cas, ce qu'on peut, une fois de plus, simplifier à $\Theta(n^2)$.

Lien vers le Git

https://github.com/TomaAllary/INF1008_TP1.git

Crédits

Tout le code est fait par Toma Allary et Benoit Matteau.

L'asset et les animations de Timmy sont la propriété et sont utilisés avec la permission de Malicious Goats.

Toutes les musiques sont tirées du jeu vidéo Final Fantasy 7

Toutes les images et / ou autres assets sont libres de droits ou ont été fait par nous pour le jeu.