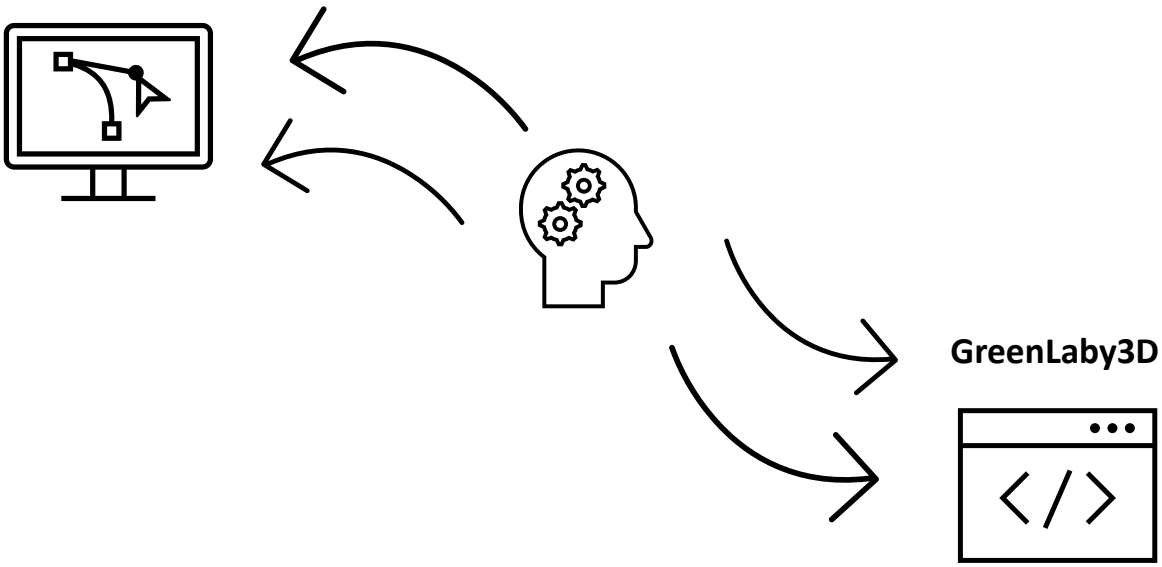
RAPPORT D'AVENCEMENT
Games On Web 2023

GreenLaby3D, de la conception à la programmation



Rédigé par Gautier BENOIT, Léna BITCH et Théo RIPOLL

Concours Games On Web 2023 – CGI – « Be green »

NOVEMBRE - JUIN 2023

Table des matières

Présentation du projet

Dans le cadre d'un concours organisé par CGI, nous avons pour but de proposer et développer un jeu web en 3D. Avant de commencer quoi que ce soit, il a fallu choisir le style de jeu à concevoir. Pour notre part, nous avons décidé de partir sur un jeu de labyrinthe.

Afin d'être en concordance avec le thème « Be green » de ce concours, nous avons ajouté quelques spécificités à notre projet. Via l'objectif final, vous comprendrez le lien entre notre jeu et le thème « Be green ».

Plus précisément, ce jeu a pour finalité de générer aléatoirement un labyrinthe comportant plusieurs éléments qui eux aussi seront positionnés de manière aléatoire sur les chemins. L'idée ici est de challenger tout en contraignant le joueur avec des positionnements qui ne sont jamais les mêmes.

Pour le moment, les éléments que nous avons retenus sont les suivants :

- Les déchets : le joueur devra en récupérer le plus possible ;
- Les poubelles de couleurs : le joueur devra mettre les déchets récolter dedans ;
- Une tour de guet : cela permettra au joueur de prendre de l'altitude et avoir un léger aperçu du labyrinthe et de ses déchets.

Afin de terminer une partie, le joueur doit ramasser la totalité des déchets présents dans le labyrinthe et les jeter dans la bonne poubelle en fonction de sa couleur. Une fois cette étape terminée, il devra trouver la sortie. De plus, comme explicité au-dessus, le joueur pourra s'aider d'une tour de guet pour avoir une vision plus importante et repérer la sortie ainsi que les déchets et les poubelles.

Tout au long d'une partie, des malus et bonus seront implémentés. Par exemple, si le joueur se trompe de poubelle, alors il obtiendra un malus caractérisé par un ralentissement. Ou à l'inverse, s'il jette correctement 5 déchets d'affilé, alors il aura un bonus représenté par une accélération. Étant donné qu'une partie se joue dans un laps de temps prédéfini, il est utile d'ajouter ces fonctionnalités. En surplus, il n'existe qu'une seule tour de guet. Elle est un atout considérable permettant au joueur d'avoir une vision périphérique des déchets qui l'entoure.

Voici un aperçu simplifié du labyrinthe sans ajout des éléments :

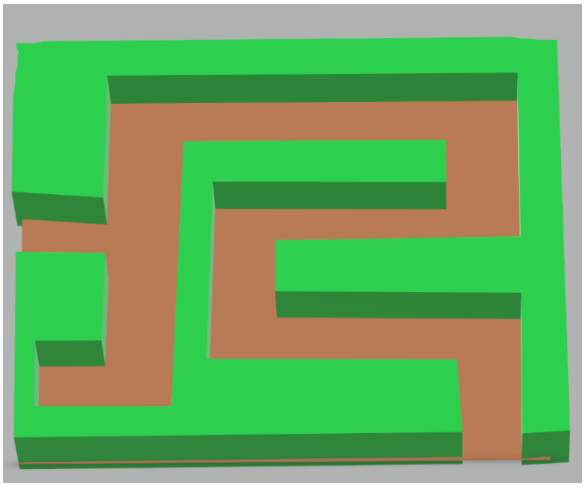


Figure 1 : Labyrinthe simplifié

Petite précision, seul le chemin de réussite sera généré aléatoirement, l'entrée et la sortie seront toujours positionnées aux mêmes endroits.

Pour renforcer le thème du « Be green », nous avons décidé de faire un labyrinthe naturel avec pour mur des arbres et pour sol de l'herbe.

Description de l'état actuel du projet

Afin de mener à bien ce projet, nous avons décidé de le découper en plusieurs phases.

La première consiste à rendre le jeu jouable avec le strict minimum. C'est-à-dire en utilisant des cartes de labyrinthe prédéfinies en amont afin de pouvoir implémenter toutes les bases du jeu.

Nous implémentons ces cartes en définissant dans le code la position de chaque objet, que ce soient les déchets, les murs ou encore les arbres, via leurs coordonnées x et y.

Tout d'abord, nous avons commencé par créer dans un canva une scène contenant un « ground » de couleur verte pour le sol. Puis, l'élément « tree » en définissant la structure et le design de l'arbre. Pour cela, nous avons utilisé une extension communautaire de Babylon.js qui s'intitule « Tree Generators ». Plus précisément, c'est une fonction qui nous permet de définir directement le nombre d'étages de l'arbre, sa hauteur, le matériel pour le feuillage ainsi que le tronc et la scène où il va être.

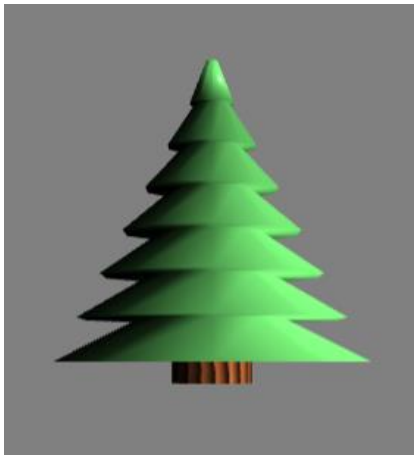


Figure 2 : Rendu d'un arbre par défaut

Concernant le matériel utilisé, nous avons opté pour l'utilisation de texture déjà proposée par des assets de babylon.js. Voici, un aperçu de notre première implémentation d'arbre, qui était à la base un rectangle :

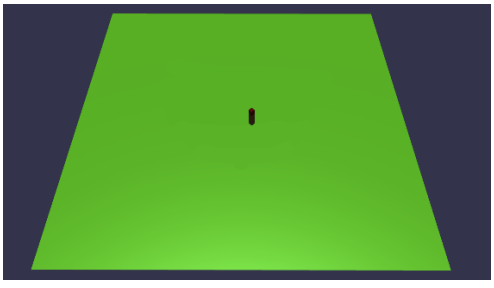


Figure 3 : Premier arbre sous forme de rectangle implémenté

Par la suite, nous avons positionné nos arbres, étant des rectangles, afin de simuler un parcours :

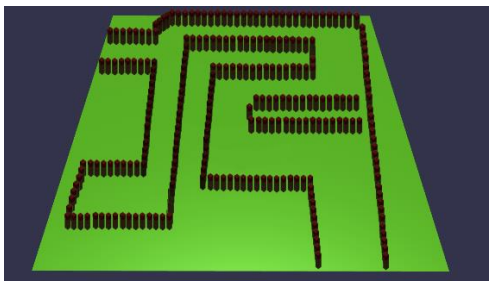


Figure 4 : Circuit d'arbres sous forme de rectangle implémenté manuellement

Finalement, nous aimerions remplacer ces rectangles par l'élément « tree » généré par la fonction citée précédemment, cette étape est toujours en cours de développement.

Nous avons en parallèle, commencé à créer nos propres textures pour les éléments ajoutés, comme la poubelle ci-dessous suivant :

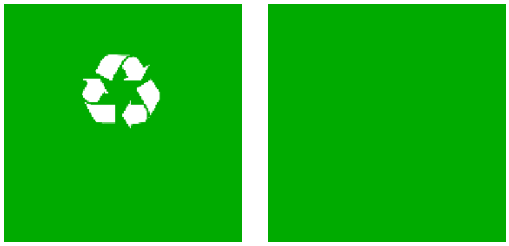


Figure 5 : Texture d'un poubelle avant et arrière

L'implémentation du joueur devrait être fait lors de cette phase aussi. De plus, nous comptons faire une vue à la première ou troisième personne. Le personnage pourra être mis en mouvement via les flèches directionnelles.

Pour ce qui est de la seconde phase, elle portera sur la génération de cartes aléatoires. Elles devront respecter certaines contraintes citées ci-dessous :

- La carte doit avoir uniquement une entrée et une sortie ;
- Il doit y avoir au moins deux déchets sur la carte ;
- Les déchets doivent se trouver sur une zone accessible au joueur donc pas sur un mur ;
- La carte doit avoir au moins une impasse.

La carte sera générée grâce à un fichier texte comme ceci :

```
999999999999999
990000000000099
990999999999099
990900000000099
000909999999999
990900000000099
900999999999099
999999999999099
```

Les nombres correspondent à la hauteur d'un mur, 0 pour le sol et 9 pour une zone infranchissable soit un mur. Les déchets seront caractérisés sur le fichier par des X :

```
999999999999999
99X000000000099
990999999999X99
990900000000099
0009X9999999999
990900000000099
9X0999999999099
999999999999099
```

La zone de départ sera signalée par un D et la zone d'arrivée par un A :

```
999999999999999
99X000000000099
990999999999X99
990900000000099
D009X9999999999
990900000000099
9X0999999999099
999999999999A99
```

Afin d'implémenter ces spécificités, un algorithme sera utilisé. Deux possibilités s'offrent à nous, premièrement nous pouvons élaborer nous-même de A à Z un algorithme qui lira le fichier et qui positionnera les éléments aux bons emplacements. Ou sinon, nous pouvons nous appuyer et nous inspirer d'un algorithme générateur de labyrinthe compatible avec Babylon.js.

Une des idées serait de créer une fonction qui génère des arbres de taille aléatoire et qui les implémentent dans la scène. Puis, elle irait placer aléatoirement les arbres sur les cases de la grille.

Voici à quoi doit ressembler en théorie le labyrinthe si l'on utilise le fichier représenté ci-dessus :

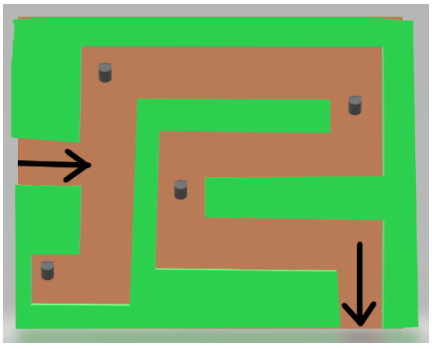


Figure 6 : Labyrinthe généré grâce à un fichier

Une fois ces étapes effectuées, nous pourront implémenter la tour de guet.

Plan de la suite du projet

Pour ce qui est de la continuité de notre projet, nous avons envisagé plusieurs éléments à réaliser. Tout d'abord, nous avons décidé d'implémenter le système de génération du labyrinthe dans le but d'obtenir un parcours réalisable, qu'il soit généré via un fichier ou de manière aléatoire. En parallèle, nous implémenterons aussi la totalité du mouvement du personnage afin qu'il puisse se déplacer au travers des routes qui lui seront proposées.

Une fois cette phase fonctionnelle, l'ajout des éléments tels que les déchets et les poubelles sera effectué. Grâce à ces éléments, nous pourrons par la suite ajouter les bonus et malus afin de rendre ce jeu toujours plus complet. Une minuterie sera aussi présente pour challenger le joueur et augmenter la difficulté de l'objectif.

Afin d'intégrer la gravité, l'ajout de la tour de guet sera effectué. Le joueur pourra monter sur la tour afin d'avoir un panorama de la zone qui l'entoure mais aussi descendre en lâchant toutes les touches.

Enfin, nous implémenterons différents modes de jeu comme l'ajout et l'obligation de trier les déchets mais aussi l'obligation de ramasser tous les déchets sans quoi il ne pourra pas quitter le labyrinthe ou encore le fait de faire passer le joueur par des « checkpoint » à des endroits bien précis.