EPITA

CAHIER DES CHARGES

SeedWorld

PREMIÈRE SOUTENANCE

By M.D.W.S.

Leyre Arnaut
Cruciani Léa
Marchetti Gauthier
Tripier Léo

A rendre pour le 4 mars 2021

Table des matières

1 Introduction

Vous avez pour projet de concevoir un jeu vidéo en 2D utilisant une vue de dessus ? Ou bien vous avez besoin d'un support pour vos jeux de rôle pour être davantage immergé dans votre partie ? SeedWorld est donc un logiciel qui pourrait vous intéresser. Son but est de générer une carte du monde de votre choix, vous aurez la possibilité de choisir le type de monde que vous souhaitez : féerique, médiéval, futuriste. Et il vous sera possible de manipuler les pourcentages des différentes régions, si vous souhaitez plus d'océans, de plaines, de zone montagneuses. Vous choisirez la dimension de votre monde en fonction des choix qui s'offrent à vous et le logiciel vous enverra sous forme d'une image JPG ou PNJ la carte finale. Sur cette image finale il vous sera possible de visualiser différentes infrastructures qui seront uniques en fonction du type de monde que vous avez sélectionné et qui seront typiques de chaque région.

2 Répartitions des tâches

2.1 Génération procédurale des biomes

2.1.1 Léo Tripier

Pour cette soutenance Léo a codé un algorithme qui parcourt la matrice créée par l'algorithme d'Arnaut. Lors du parcours de cette matrice, il compare la valeur présente dans la matrice d'Arnaut afin de la placer dans l'un des trois intervalles suivant : [0,0.7[; [0.7,0.8[; [0.8,1].
Après avoir déterminé à quel intervalle appartient la valeur d'une case de la matrice il modifie la couleur du pixel correspondant sur la map. Ainsi, pour une valeur dans l'intervalle [0,0.7[le pixel correspondant sur l'image prendra une couleur bleue (représentant la mer), dans l'intervalle [0.7,0.8[le pixel prendra une couleur jaune-beige (représentant la plage), enfin dans l'intervalle [0.8,1] le pixel prendra une couleur verte (représentant l'herbe).
Pour l'instant les couleurs de l'image ne sont pas les bonnes, Léo n'ayant pas encore réussi à débugger ce problème de couleurs qui ne correspondent pas à celles assignées dans le programme, il se penchera dessus et le réglera pour la prochaine soutenance.
En même temps, l'algorithme créé une autre matrice d'entier (type *int*) qui sera envoyée par la fonction afin d'être utilisée dans la partie de Gauthier et Léa. Chaque case de cette matrice est remplie de 0, 1 ou 2. 0 pour les pixels de mer, 1 pour les pixels de plage, 2 pour les pixels d'herbe. Cette matrice permettra à Léa et Gauthier de détecter quelles zones sont "constructibles". En d'autres termes, sur quelles zones ils pourront ajouter des villes.
Il est important de préciser que la map est une image au format *BMP*. Ce n'est pas le programme qui créé l'image, il utilise une image déjà enregistrée (dans le même dossier que le code), qui sera utilisée comme base pour créer la map qui sera finalement enregistrée sous un nouveau nom. De plus, les matrices évoquées sont représentées sous forme de liste simple (un pointeur) en utilisant la représentation suivante :
*array[i][j] = array[i * nombre_de_colonnes + j]*

2.1.2 Arnaut Leyre

La génération de bruit similaire au bruit de Perlin a été parmi les priorités de cette soutenance.
Pour cela la première étape est de générer une matrice résultat nulle et une matrice *seed*. Cette dernière a plusieurs vocations. Tout d'abord elle assure une possible inclusion du concept de noyaux de génération pour la seconde soutenance. Mais elle sert aussi de base à notre prototype de bruit organique aléatoire.
À partir de cette matrice obtenue de manière aléatoire nous allons pouvoir générer notre résultat. Pour s'assurer de la cohérence et de la fluidité du résultat il faudra faire plusieurs passages de comparaison linéaire appelés *octave*. On va donc parcourir la matrice et pour chacun des points on va y appliquer plusieurs *octaves*. Pour chaque *octave* on va prendre des points-*échantillons* dont la distance est *la distance maximale* par rapport à notre point d'étude. C'est-à-dire un point situé à une distance équivalente à la taille du côté de la matrice. Si pour ce point la distance dépasse le bord de la matrice on fait le tour et on continue à compter de l'autre côté de la matrice. Cette étape est cruciale car elle est à l'origine d'une des propriétés les plus intéressante du bruit de Perlin. En effet si on regarde attentivement les bords de l'image on observe qu'ils correspondent parfaitement au bord opposé. Cette propriété permet une transition, ce qui est très utiles dans le cas d'un planisphère qui est une représentation en 2 dimensions d'une sphère.

FIGURE 1 – Image en sortie du bruit de Perlin

La distance maximale est divisée par chaque *octave*, permettant aux *échantillons* d'être de plus en plus proches et ainsi aux formes de se créer. Ceci implique aussi qu'il y a un nombre d'*octave* maximal qui peut être efficace, au-delà de ce nombre le code compare chaque pixel à lui-même. Ainsi pour les autres cas le programme compare le point d'étude aux différents *échantillons* et détermine une valeur moyenne. Le nombres d'*échantillons* correspond à la dimension de la matrice donc dans notre cas 2.
Les valeurs moyennes sont ensuite multipliées par une *échelle* et ajoutées les unes aux autres pour enfin être sommées au résultat. L'*échelle* doit être divisée par un *biais* pour que les détails soient de plus en plus fin. Le *biais* et le nombre d'*octave* sont également les clés qui permettent de contrôler l'aspect de notre bruit de Perlin. Plus on a d'*octaves*, plus on a de détails, plus le *biais* est petit, plus le terrain généré est rugueux et aléatoire.

2.2 Génération procédurale des villes

2.2.1 Léa Cruciani

Pour la génération de ville avec des réprésentations aléatoires et uniques, Léa a pris en charge la création des routes. Pour ce faire quatre fonctions ont été créées :

- *nb_rand(int min, int max)* qui donne un chiffre dans l'intervalle min et max inclus.
- *define_dir(int x, int y, int cols, int rows, int vertical)* qui donne la direction de la route qui va être créée (si celle-ci sera vers le haut, le bas, à droite ou bien à gauche).
- *draw_road(int* map, int x, int y, int r_size, int cols, int rows, int vertical, int direction)* qui va permettre, comme son nom l'indique, de construire une route.
- *build_roads(int* map, int rows, int cols)* qui va se charger de lier les routes et de construire tout le système routier de la future ville.

Le principe de cet algorithme est de construire un réseau routier unique qui ensuite pourra permettre de placer des maisons. La fonction principale *build_roads* va prendre un tableau de 0 (*int* map*) à une dimension qui sera considéré comme la zone dans laquelle la ville devra être bâtie. L'implémentation *row-major order* qui offre la possibilité de traiter un tableau à une dimension en une matrice sera utilisée tout au long de ce projet. Pour que cela soit possible il est nécessaire que *build_roads* prenne en paramètre le nombre de colonnes (*int cols*) et le nombre lignes (*int rows*) qui vont être utilisés par notre future matrice.

La fonction *build_roads* va ensuite créer dans un premier temps la route principale : la création des variables *int x* et *int y* représenteront le début de notre première route, et *int len* sera sa longueur. Pour permettre aux routes d'avoir des dispositions uniques, ces valeurs seront générées de manière aléatoire à l'aide de la fonction *nb_rand* :

- x et y devront être choisies entre 0 et le nombre de lignes(ou colonnes), divisé par deux - un, car on veut exclure le fait que x(ou y) peut être égal aux nombres de lignes(ou colonnes).
- *len* sera quant à elle choisie entre la moitié de la longueur totale de notre tableau (*cols*rows/2*) et sa longueur totale (*cols*rows*) ;

Une fois les coordonnées de départ et la longueur de la route principale obtenues il faut désormais indiquer sa future orientation qui sera générée de manière aléatoire par *int vertical= rand()%2*, si *vertical = 1* alors la route sera verticale sinon elle sera horizontale. Cela est maintenant au tour de sa direction qui va lui être définie avec l'aide de la fonction *define_dir* qui va prendre en paramètre les coordonnées x et y de la route, le nombre de colonnes *cols*, de lignes *rows* de notre matrice et *vertical* pour savoir l'orientation de la route. Si elle est verticale, seulement notre coordonnée x sera modifiée. L'utilisation de la variable définie *MIN_LENGTH* qui sera égale au maximum entre la hauteur et la longueur d'une

maison (dans notre cas 48 pixels donc 48), sera nécessaire pour privilégier les directions des futures routes. En effet si notre route est verticale et commence aux coordonnées x =*1* et y=*0* cela ne sera pas productif qu'elle se dirige vers le haut (car elle fera 1 de long) et la deuxième route que l'on va contruire n'aura pas l'embaras du choix pour ses coordonnées de départ. Ainsi la direction de la route s'opérera pour priviligier une longeur importante.

Par exemple : notre route est orientée de manière verticale, c'est la coordonée x qui va varier. Si x+ *MIN_LENGTH* et x-*MIN_LENGTH* sont toujours définies dans la matrice alors la route peut tout à fait aller en haut ou en bas : notre direction sera générée de manière aléatoire ! *final_dir= rand()%2* si x-*MIN_LENGTH* est hors de la matrice cela signifie que notre route est trop proche du bord haut de la matrice ! Sa direction sera donc vers le bas *final_dir=0* sinon elle sera dirigée vers le haut. Il en va de même si notre route est orientée de manière horizontale sauf que cette fois-ci c'est la coordonnée y qui va varier.

Désormais en possession de la longueur de la route, ses coordonnées de départ, sa direction, et son orientation, il est possible de la dessiner. C'est à ce moment que la fonction *draw_road* participe à la tâche. Elle va prendre en paramètre le nombre de colonne et de ligne de notre matrice, les coordonnées du point de départ de notre route, son orientation, sa direction et la longeur de la route *r_size*. Si la route est verticale (horizontale) et est dirigée vers le(la) haut(gauche) $x(y)$ sera décrémentée ,dans le cas contraire incrémentée. À chaque itérations les coordonnées x et y seront testées pour savoir si elles sont toujours définies dans la matrice, ainsi si la longeur de la route désirée est trop grande la boucle sera automatiquement stoppée. Cette fonction renvoie la coordonée de la fin de la route (x si verticale, y si horizontale).
La route principale est tracée ! Il faut attaquer les routes subsidiaires qui auront pour point de départ des coordonnées entre le début et la fin de la route qui a précédé. Léa a décidé de limiter le nombre de route créer à un nombre choisi arbitrairement pour le moment, il sera modifié en fonction des besoins des prochaines soutenances et des modifications. Tant que le nombre total de route(*int nb_roads*) sera strictement supérieur à 0 il faudra continuer à en créer.

L'orientation de la route(r2) qu'on va créer va être le contraire de celle qui l'a précédée(r1). Si r1 est horizontale : r2 sera verticale. Les points de coordonnées de r2 devront obligatoirement appartenir à la route r1 ainsi il n'y aura pas de route isolée(ce qui fait sens). Si r1 est horizontale seule la coordonée y sera modifiée pour le point de départ de r2. Si r1 est dirigée vers la droite et est assez longue pour que y+*MIN_LENGTH* alors la coordonnée y sera choisie de manière aléatoire avec *nb_rand* entre y+*MIN_LENGTH* et *end* qui symbolise la fin de r1. Dans le cas contraire y sera choisie de la même manière entre y et *end*. Il s'agira

du même processus si r1 est verticale mais cette fois-ci x sera modifiée en conséquent. *define_dir* s'occupe de déterminer la direction de r2, l'orientation passée en paramètre de cette fonction devra être opposée à celle de r1 car la future orientation de r2 sera l'opposé de l'orientation actuelle de r1. Les coordonnées de départ de r2 sont récupérées, sa direction, son orientation sont définies également, il est possible de la tracer, avec bien entendu *draw_road* le retour de cette fonction sera récupéré par *end* qui sera la fin de r2. Une fois r2 tracée il faut actualiser *vertical* qui indiquera l'orientation de la prochaine route(r3) qui aura une orientation opposée à r2 : *vertical*= *!vertical* et enfin *nb_roads* sera décrémenté. Exemple d'exécution : matrice carrée de 50 par 50, nombre total de routes subsidiaires 20, les 6 indiquent le point de départ d'une route.

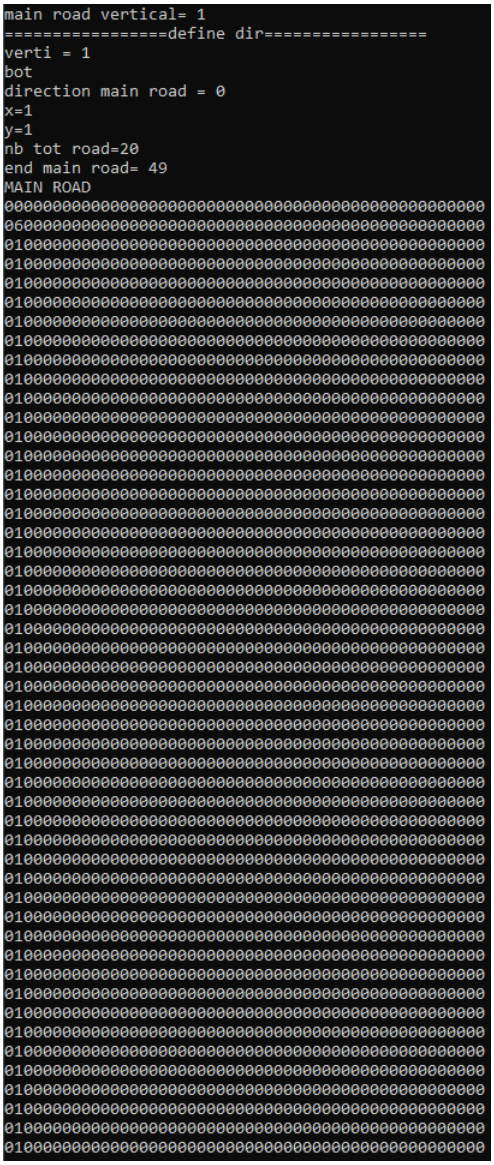


FIGURE 2 – Route principale créée

```
x = 1
y= 1
length road=23
direction of the main road = bot
waleur de x entre 7 et 49
x=30
y=1
================define dir================
right
end = 24
future road will be verti=1

==================draw the road==================
00000000000000000000000000000000000000000000000000
06000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
06111111111111111111111110000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
```

FIGURE 3 – Route subsidiaire créée

```
x = 30
y= 1
length road=6
direction of the main road = right
waleur de y entre 7 et 24
x=30
y=18
=================define dir=================
verti = 1
all possible
end = 24
future road will be verti=0

===================draw the road==================
00000000000000000000000000000000000000000000000000
06000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000010000000000000000000000000000000
01000000000000000010000000000000000000000000000000
01000000000000000010000000000000000000000000000000
01000000000000000010000000000000000000000000000000
01000000000000000010000000000000000000000000000000
06111111111111111161111100000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
01000000000000000000000000000000000000000000000000
```

FIGURE 4 – Route subsidiaire créée

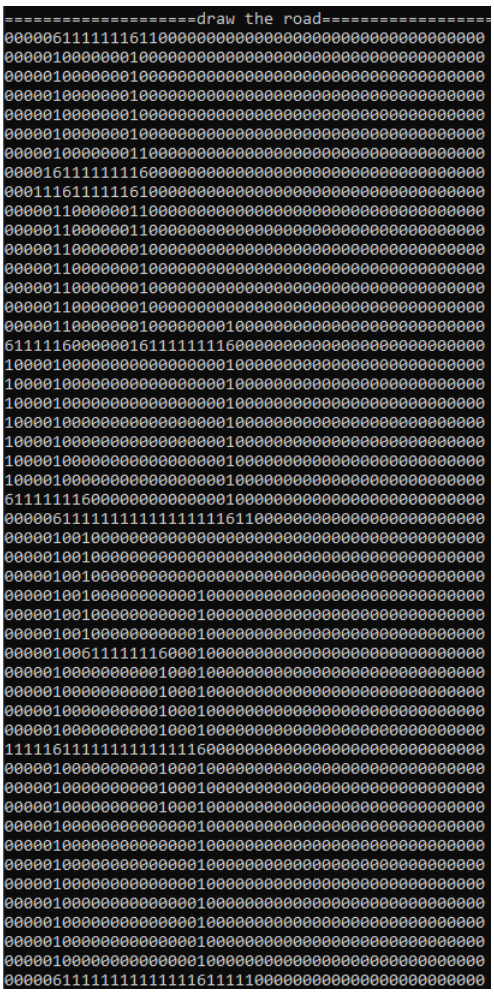


FIGURE 5 – Fin du processus

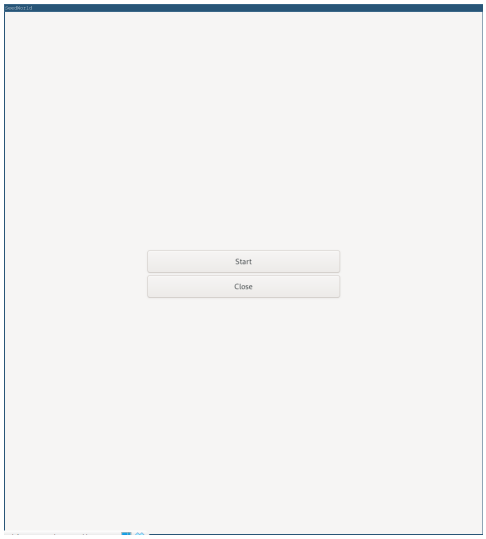
2.2.2 Gauthier Marchetti

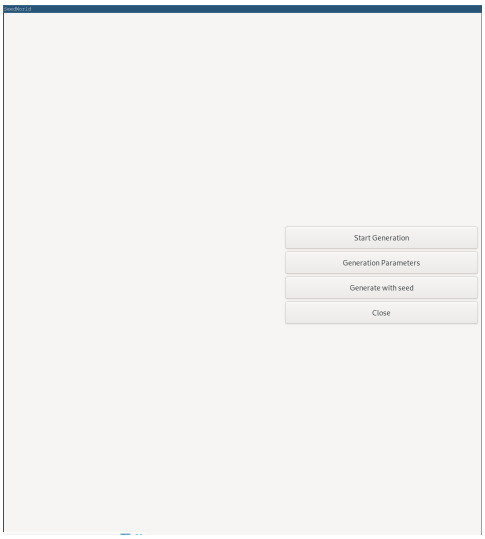
Une fois la carte des routes de la ville créée, il faut placer les maison là où l'espace est suffisant. Pour ce faire, Gauthier a récupéré la matrice établie par Léa. Il s'est attelé à scanner les zones autour des intersection produites par les differentes routes. La fonction *find_inter* parcoure l'entièreté de la matrice à la recherche du caractère correspondant à une intersection.
Une fois l'emplacement de l'intersection déterminé, la fonction *draw_house* est appelée. Cette fonction est le cœur de la procédure de placement des maison. Elle se sépare en quatre sous-parties, correspondantes au quatre coins de l'intersection (supérieur gauche, supérieur droit, inférieur gauche, inférieur droit). Pour chaque sous-zone, la fonction scanne, selon la taille prédéfinie des maison, la surface nécéssaire au placement d'une maison. Si la surface est non conforme ou trop petite : traversée par une route, en bord de matrice, déjà occupé par une autre maison ; la procédure recommence avec une maison de taille inférieure, et cela jusqu'à ce que toute les tailles de maison ai été testées.
Pour ce faire, on intitialise deux paramètres *width* et *length* avec la largeur et la longueur de la plus grande maison, ainsi qu'un bouléen *valid* qui servira à vérifier si la surface traitée est conforme. Pour éviter des tours de boucle inutile, la surface minimale requise pour la plus petite des maisons est testée. C'est à dire que sans même regarder si la zone est obstruée on vérifie que l'espace brut est suffisant. Par la suite, on vérifie si la zone choisie n'est pas déjà occupé par une autre route ou une autre maison. Si c'est le cas, *valid* devient faux et arrête immédiatement les itérations inutiles, la procédure recommence avec la taille de maison suivante. Si les itération arrivent à leur terme et que *valid* est toujours vrai, alors s'enclenche le placement en soi des maisons, de manière matricielle.
Pour le placement des maisons de manière matricelle, nous avons établit notre propre convention : un caractère spécial, ici *4*, *5* ou *6*, sera placé dans le coin supérieur gauche de la surface alloué à la maison.

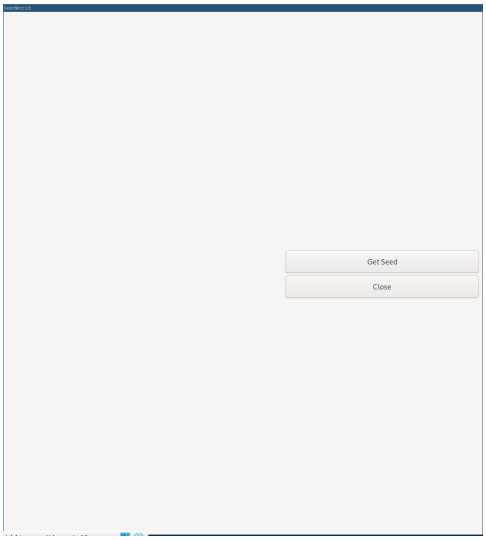
3 Interface

3.1 Léo Tripier

C'est essentiellement sur cette partie que Léo a travaillé pour cette soutenance. Il a codé le squelette de notre interface. La structure principale repose sur un fichier xml en *.glade* qui sert de base pour créer l'interface.
Pour écrire ce fichier il s'est inspiré de celui qu'il avait déjà fait pour le projet OCR du S3. Il a cependant réécrit pratiquement entièrement le fichier car il souhaitait que la mise en forme de cet interface soit différente. Pour l'interface de ce projet, il utilise des objets appellés "GtkGrid" qui lui permettent de "stocker" des boutons et des les disposer comme il le souhaite dans la fenêtre de l'interface. Pour l'instant il a créé 3 GtkGrid qui seront les 3 principales. Chacunes contenant des boutons différents. Le gros avantage d'utiliser les GtkGrid c'est que cela minimise grandement les lignes de codes dans le programme principal. En effet grâce à ces outils il suffit (dans le programme principal) de "connecter" l'un de ces objets à la fenêtre pour qu'il s'affiche, ou de le "déconnecter" pour qu'il ne s'affiche plus. Cela ne prend donc que 2 ligne de code pour modifier l'affichage de la fenêtre (une pour déconnecter l'ancienne Grid, une autre pour connecter la nouvelle).
Cependant cela nécéssite de bien écrire le fichier en *.glade* qui sert de base afin que tout soit propre. Léo a préféré faire ce choix car cela rend le code beaucoup plus lisible étant donné qu'il ne faut que quelques lignes pour changer la GtkGrid qui sera affichée. Ce qui lui pris le plus de temps est donc bien entendu l'écriture du fichier *"interface.glade"* mais surtout la documentation sur la manière dont s'écrit un fichier de ce type. Il est en effet nécéssaire de connaitre les différents objets mis à disposition par GTK ainsi que les différentes propriétés qui leurs sont propres.
Il lui a également fallut tester beaucoup de fois comment cela rendait car certaines propriétés sont prioritaires par rapport a d'autres, bien que cela ne soit pas toujours explicite dans la documentation GTK. De plus il faut faire attention à ce qu'une nouvelle propriété ne vienne pas annuler une ancienne. Une fois l'équilibre trouvé et la mise en forme satisfaisante il ne reste plus qu'a ajouter les quelques lignes de code évoquées précédemment. Voici quelques images du squelette de l'interface :







Pour l'explication des différents boutons que vous avez pu observer sur ces images :

— Close : ferme la fenêtre
— Start Generation : lance la génération et affiche la grid suivante
— Generation Parameters : affiche les réglages de la génération (à implémenter)
— Generate with seed : permet de donner une seed au générateur afin qu'il génère une map précise (à implémenter).
— Get Seed : affiche la seed de la map qui a été générée (à implémenter)

Il reste donc encore beaucoup de chose à faire. L'objectif pour la prochaine soutenance sera d'implémenter la page des réglages de la génération ainsi que d'embellir un peu cette interface qui est encore très terne.

3.2 Arnault Leyre

Pour cette soutenance l'interface n'a pas été une priorité car elle est vouée à s'adaptée au reste du projet. Ainsi seules les fondations ont été poser. Pour les soutenances futures nous prévoyons d'y mettre plus d'effort et de fonctionnalité. Pour cela nous allons notamment permettre à l'utilisateur d'influencer sur la génération notamment au travers du choix de la seed de génération et de diverses tailles qui serons disponible.

4 Site web

4.1 Léa Cruciani

Le site web a été réalisé avec html et mis en page avec css. Léa a décidé de réaliser une barre de navigation fixe positionnée à gauche de la page web pour permettre à l'internautre de passer facilement d'un onglet à un autre sans avoir à retourner au début du site. La barre de navigation est composée de plusieurs onglets :

— Accueil
— Soutenances
— Graphismes
— Progression

L'accueil sera la première page rencontrée par l'internaute une fois sur le site. Il faut qu'elle le renseigne sur l'équipe M.D.W.S, ainsi que bien évidemment sur le logiciel *SeedWorld*. Pour ce faire, plusieurs balises ont été écrites dans la page html *nav* qui sera la barre de navigation à gauche, *article* qui est au centre de la page, il est constitué d'une description du projet *SeedWolrd*, d'une définition de la génération procédurale. À la droite se trouve une brève présentation de l'équipe M.D.W.S, avec le logo(à venir) qui est associé.
L'onglet *Soutenances* permet d'accéder aux téléchargements du rapport pour chaque soutenances, les liens de téléchargements seront actualisés en fonction des soutenances à venir.
L'onglet téléchargement permettra de télécharger *SeedWorld*.
L'onglet progression servira à indiquer l'avancement des tâches de chacun des membres de l'équipe en pourcentage.
L'onglet graphismes montrera un aperçu des bâtiments, des routes, et des biomes dans le logiciel. Il sera actualisé en fonction des dessins réalisés.

5 Organisation des tâches pour la prochaine soutenance

Soutenance 1	Léa	Arnaut	Léo	Gauthier
Génération des régions	X	30%	30%	X
Génération/Placement des villes/Sprites	30%	X	X	30%
Graphismes	30%	X	X	30%
Interface	X	30%	30%	X
Site web	50%	X	X	X

TABLE 1 – Répartition des tâches S1

Soutenance 2	Léa	Arnaut	Léo	Gauthier
Génération des régions	X	80%	80%	X
Génération/Placement des villes/Sprites	80%	X	X	80%
Graphismes	70%	X	X	70%
Interface	X	50%	50%	X
Site web	80%	X	X	X

TABLE 2 – Répartition des tâches S2

Soutenance finale	Léa	Arnaut	Léo	Gauthier
Génération des régions	X	100%	100%	X
Génération/Placement des villes/Sprites	100%	X	X	100%
Graphismes	100%	X	X	100%
Interface	X	100%	100%	X
Site web	100%	X	X	X

TABLE 3 – Répartition des tâches S3

6 Graphismes

FIGURE 6 – Maison médiévale n°1, auteur : Gauthier

FIGURE 7 – Maison médiévale n°2, auteur : Gauthier

FIGURE 8 – Maison médiévale n°3, auteur : Gauthier

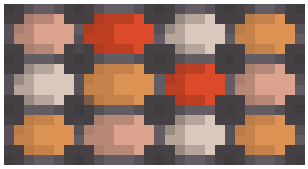


FIGURE 9 – Route médiévale horizontale, auteure : Léa

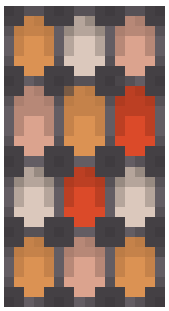


FIGURE 10 – Route médiévale verticale, auteure : Léa

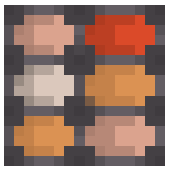


FIGURE 11 – Route médiévale intersection, auteure : Léa

7 Bilan

7.1 Arnaut Leyre

Pour la seconde soutenance, mes objectifs seront d'amener le système de seed dans le programme ainsi que de me concentrer sur l'interface. En implémentant notamment des options pour influencer la génération.

7.2 Gauthier Marchetti

Pour la seconde soutenance, il est prévu de transformé la carte matricelle des villes produites en véritable sprite, afin de pouvoir les placer sur la carte du monde. Pour ce faire, il faudra, à une large échelle, scanner l'entiereté de la carte afin de définir l'emplacement idéal de la ville. En outre, les recherches et le début d'implémentation du résaux de neurone pour la determination des noms des villes sera envisagé. Enfin, le dernier objectif est d'étoffer notre collection de sprite de maisons.

7.3 Léa Cruciani

Concernant la seconde soutenance les tâches à prévoir sont les suivantes :
— Site web : onglet progression et téléchargement à faire.
— Construction des routes : trouver un moyen d'éviter les aglutinements. Des routes qui sont toutes côte à côte.

— Contribuer à l'algorithme qui permettra de trouver un emplacement constructible pour une ville.
— Faire des recherches sur les réseaux de neuronne pour créer des noms de villes plausibles.

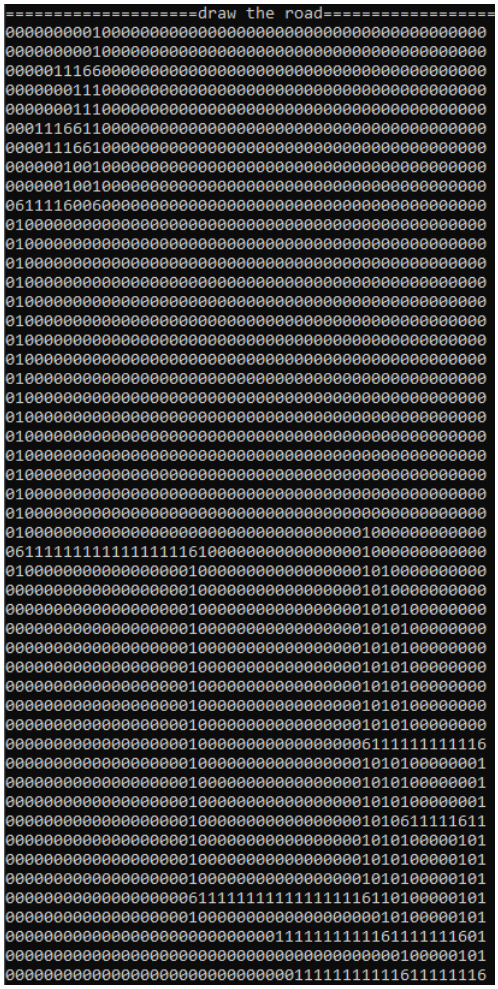


FIGURE 12 – Exemple d'aglutinements

7.4 Léo Tripier

Pour cette soutenance, les bases du projet ont été établies. Nous avons les bases nécéssaires à la réalisation d'un bon projet. Pour la prochaine soutenance Léo se mettra à jour sur le bruit de Perlin codé par Arnaut. Ainsi il continuera de développer cet algorithme et ainsi améliorer la génération des biomes. Il faudrat également qu'il complète l'interface en rajoutant les dernières GTKGrid.

8 Annexes

Références

[1] Page wikipédia du thème du projet :
`https://fr.wikipedia.org/wiki/Génération_procédurale`, consulté 1 mars 2021

[2] Page Wikipédia du bruit de Perlin :
`https://fr.wikipedia.org/wiki/Bruit_de_Perlin`, consulté 3 mars 2021

[3] Page de renseignement sur le bruit de Perlin :
`https://web.archive.org/web/20080724063449/http://freespace.virgin.net/hugo.elias/models/m_perlin.htm`, consulté 3 mars 2021

[4] Document étudiant sur le sujet :
`https://constellation.uqac.ca/4559/1/Prin_uqac_0862N_10451.pdf`, consulté 3 mars 2021

[5] Archives ouvertes sur le sujet :
`https://tel.archives-ouvertes.fr/tel-00841373/document`, consulté consulté 3 mars 2021

[6] Documentation GTK :
`https://developer.gnome.org/gtk3/stable/index.html`