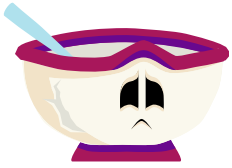
Epita

Rapport Soutenance

SeedWorld

Deuxième 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

La nouveauté implémentée pour cette soutenance : les biomes ! Léo s'est attelé à la création d'un prototype de génération de biomes. Se basant sur une suggestion d'Arnaut, il a utilisé une seconde carte générée aléatoirement (appelée carte des températures) en utilisant les même paramètres de biais et d'octave que pour la carte précédente mais en utilisant une seed aléatoire différente.
Ensuite la fonction *gen_ biomes* sera appelée en passant en paramètre les deux matrices. L'une pour les reliefs (océan,plaine, montagne) l'autre pour les "températures", des *float* entre 0 et 1. La fonction va donc parcourir ces deux matrices simultanément et croiser les informations contenues afin de générer une nouvelle matrice de *int* (représenter dans le code par un pointeur *int**). Cette matrice est donc remplie de *int* entre 1 et 9 (compris) qui représente donc le type de biome.

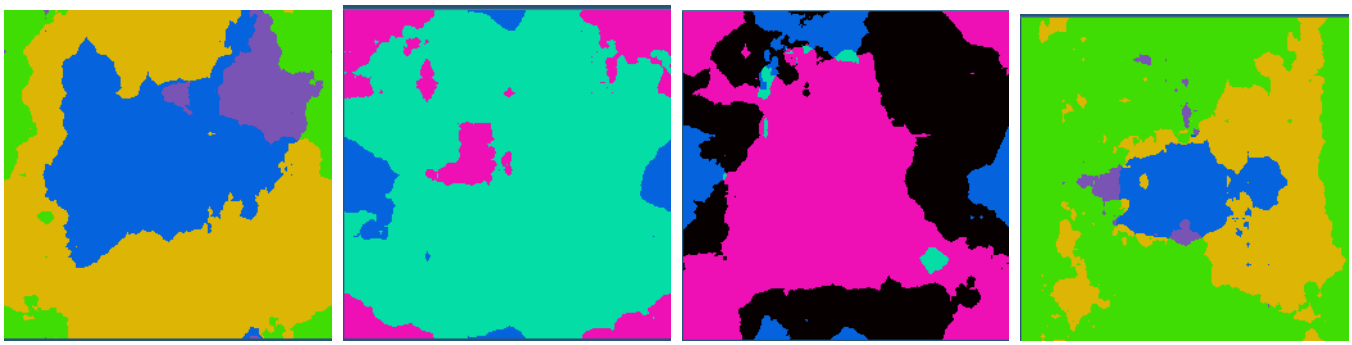


FIGURE 1 – Différentes cartes avec des biomes

Pour l'instant rien n'est fixé, les types de biomes ne sont que des entiers et ne sont pas encore associés à quoique ce soit mais cela sera ajouté pour la soutenance finale bien évidemment. Pour l'instant le code associe simplement une couleur ne correspondant à rien de particulier cela permet juste de vérifier que l'algorithme fonctionne correctement, ce qui est le cas, on observe bien des zones distinctes comme attendu.

2.1.2 Arnaut Leyre

Lors de cette soutenance la génération et la manipulation des seeds ont été ajoutées. Pour cela la seed est décidée selon la date puis grâce à la librairie *rand* les futures valeurs seront décidées en fonction de la seed et de l'ordre de leur génération. Cette solution est particulièrement efficace car tous les codes de la soutenances précédente dépendent de cette librairie.

De plus le problème d'affichage des couleurs a été réglé. Il s'agissait d'un problème dût au format de l'image utilisé pour enregistrer le résultat et donc la palette de couleur était limitée.

2.2 Génération procédurale des villes

2.2.1 Léa Cruciani

Pour cette deuxième soutenance Léa acorrigé les aglutinements des routes qui pouvaient poser problème pour l'insertion des maisons et aussi pour l'aspect réaliste du réseau routier. Pour ce faire différentes fonctions ont été ajoutées :

— *int not_on(int* map, int x, int cols, int vertical, int direction*
Cette fonction permet de vérifier qu'il n'y a pas une autre route autour du point de départ de notre future route subsidiaire, elle prend en paramètre la matrice d'entier représentant la carte de notre réseau routier, les coordonnées x et y de notre point de départ et enfin l'orientation de notre future route(horizontale ou verticale) et sa direction (droite/gauche, ou haut/bas). Si une case de notre matrice autour de notre point de départ vaut 1 alors on ne pourra pas construire une nouvelle route sans en empiéter sur une autre. *not_on* renverra donc 0 et 1 dans le cas contraire.

— *int is_constructible(int*map,int x, int y, int len, int cols, int rows, int vertical, int direction)*
Il s'agit de la fonction principale qui va parcourir la carte de notre réseau routier sur la longueur de notre future route pour estimer si celle-ci est assez espacée des autres pour pouvoir être construite. Elle prend en paramètre les coordonnées du point de départ de notre route, le nombre de colonnes et de lignes de notre matrice, l'orientation et la direction de notre route en construction. Dans le cas où notre route est verticale seulement x va être décrémentée(incrémentée) si la direction est en haut(bas). Dans le second cas notre route sera horizontale et y sera décrémentée(incrémentée) si la direction est vers la gauche(droite). Sur chaque point appartenant à la future route *is_constructible* fera appel à *constructible* qui va balayer les coordonnées de part et d'autre de celle-ci. À la fin du processus *is_constructible* renverra un 1 si la construction de la route est possible, un 0 dans le cas contraire.

— *int constructible(int*map, int x, int y, int fixed, int cols, int len, int i, int j, int max_pos, int vertical)*
Comme indiquer précédemment *construtible* va balayer les coordonnées autour de x et y sur une longueur qui est égale à la largeur minimale d'une maison, comme cela la possibilité d'insérer une maison sera plus élevée. Elle prend comme paramètre *int fixed* qui représente la coordonnée fixée sur la laquelle on va tester qu'il n'y ait pas de possibles routes déjà existantes de part et d'autre. Cette coordonnée fixée est importante car en fonction de l'orientation de la route celle-ci ne sera pas la même, dans le cas où elle est horizontale c'est le y qui sera fixée dans le cas contraire le x. Léa

a décidé de gérer les possibles intersections des routes. En effet, la fonction renvoie 0 si elle rencontre un 1(représentant une autre route) et cela exclut la possibilité de créer des intersections. Pour palier à ce problème Léa a inclus une condition qui stipule que si la longueur de la route qu'on est entrain de tester est supérieure à la largeur minimum d'une maison et que la coordonnée sur laquelle on est, vaut 1 et que celles de part et d'autre valent également 1 alors cela signifie que la route qu'on souhaite construire en croise une autre et sa construction est donc possible.

Les routes peuvent s'intersectionner sans créer des aglutinements. Cependant un problème reste à régler il est possible que certains réseaux routiers ne puissent pas être développés complètement dût à la génération aléatoire du point de cépart et de la longeur aléatoire d'une route. Cela peut entraîner des villes qui ne seront pas intéressantes à construire.

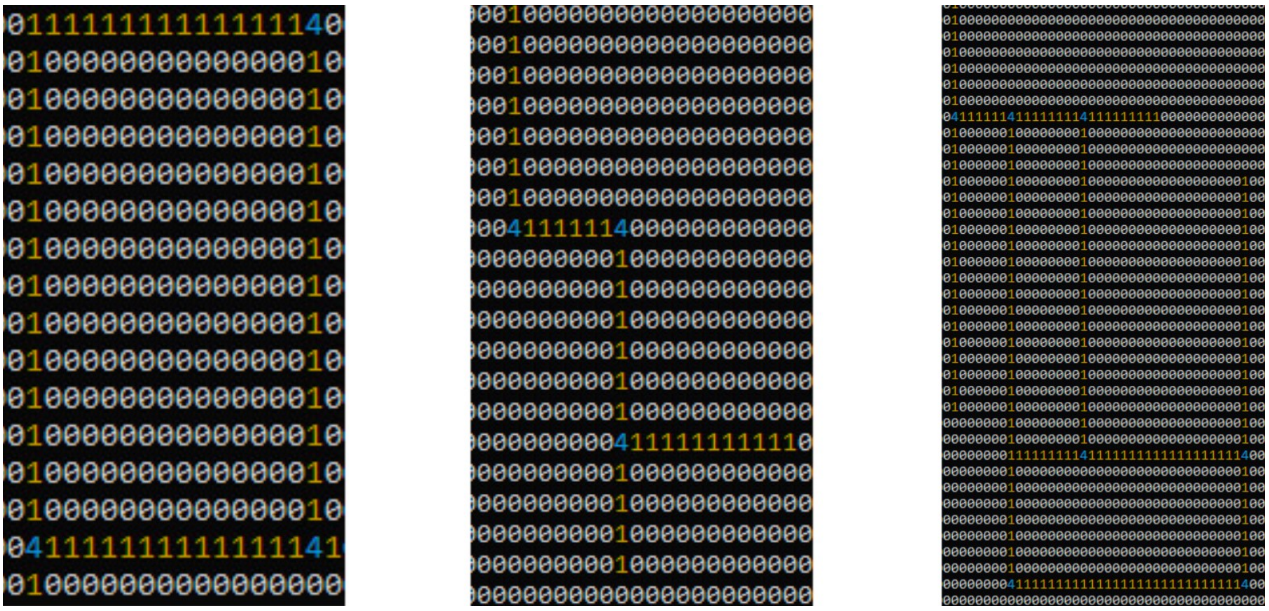


FIGURE 2 – Réseau routier peu développé et développé

D'autres fonctions ont été ajoutées pour permettre une meilleure lisibilité du code, ou pour permettre une meilleure approche de certaine partie du code.

— *define_len* permet de générer la longeur d'une route de manière aléatoire entre la longueur minimale d'une maison et sa longueur maximale qu'elle peut atteindre en partant de ses coordonnées de départ dans la matrice et non plus d'une longueur maximale définie manuellement.

— *define_begin* permet de générer aléatoirement le point de départ d'une route secondaire sur la principale.

Pour cette deuxième soutenance Léa s'est aussi chargée d'implémenter la recherche et la confirmation d'un emplacement constructible pouvant accueillir un ville. Pour ce faire il a été décider de fixer le nombre de villes qu'il est souhaité de créer, le nombre de villes différentes et enfin leurs tailles (nombre de colonnes et de lignes). Pour ce faire deux fonctions ont été créées :

— *int* find_fields(int* map, int cols, int rows)*
Cette fonction va parcourir la carte monde, si plusieurs villes sont constructibles sur un même x et y alors le choix de la ville se fera de manière aléatoire puis elle s'occupera de gérer la prochaine coordonnées sur les colonnes en ajoutant 2 fois la largeur de la ville qui a été créée sur y et le prochain x se verra ajouter 2 fois la hauteur de la ville qui a été construite. De cette manière les villes seront espacées et auront une zone d'influence ce qui empêchera les regroupements qui ne sont pas réalistes. Une fois le parcours de la carte terminée ,soit car le nombre de ville qu'on souhaite construire a été atteint soit car on arrive à la fin de la carte, la fonction renvoie un tableau de données qui contient les coordonnées x et y leurs nombres de colonnes et leurs nombre de lignes. Par exemple, si le nombre de villes que l'on souhaite construire est de 2 et que 2 villes ont été construites la tableau de sortie sera égal à

$$0, x1, y1, cols1, rows1$$

. Si seulement une ville a été construite :

$$x1, y1, cols1, rows1, -1, -1, -1, -1$$

, si aucune ville n'a pu être construite :

$$-1, -1, -1, -1, -1, -1, -1, -1$$

.

— *int buildable(int* map, int x, int y, int cols, int rows, int width, int height)*
Cette sous fonction est utilisée par *find_fields*, elle permet de tester à partir des

coordonnées et la taille de la future ville transmises par *find_fields* si celle-ci peut être construite. Elle parcourt la zone que devrait occuper la ville et si un seul point ne vaut pas 1 alors la zone n'est pas constructible et 0 sera renvoyé, dans le cas contraire 1 sera renvoyé.

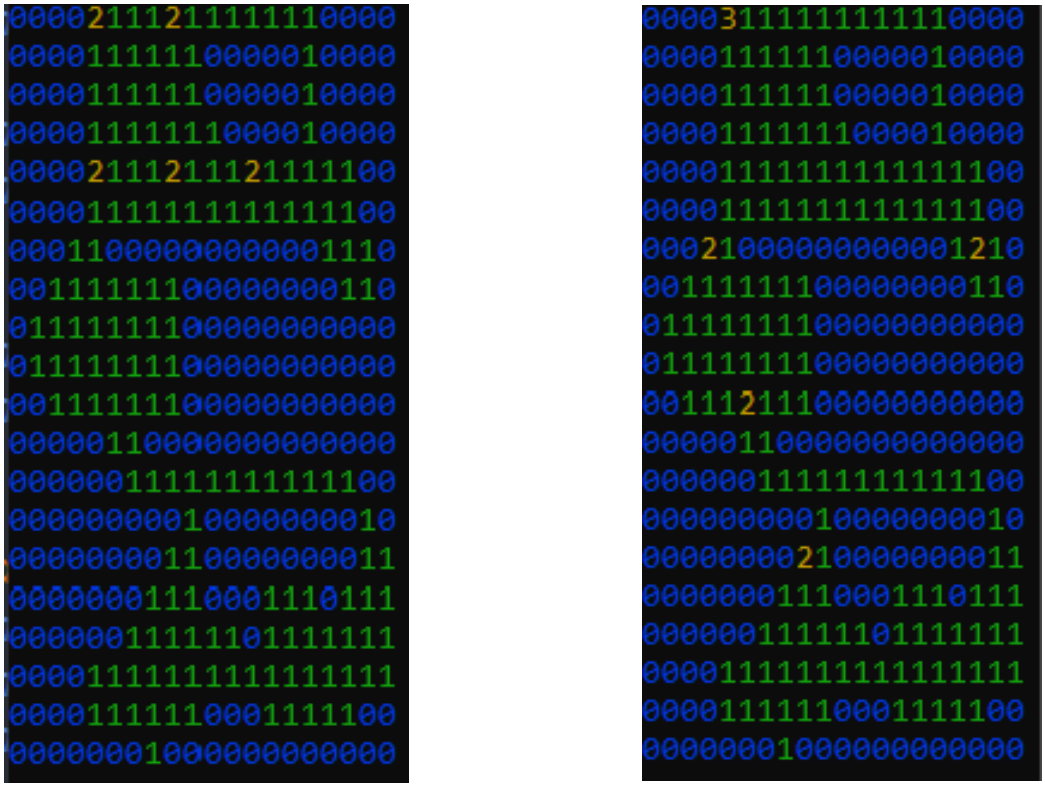


FIGURE 3 – Placement des villes 2=2*2 3=5*3 sur la même carte

2.2.2 Gauthier Marchetti

Lors de la soutenance précédente, une fois la matrice des routes créée, une procédure était appliquée qui parcourait la matrice des route afin de déterminer les zones disponibles à l'ajout de maison. Un phénomène d'agglutinement des maisons était observé. Afin de corriger ce problème, Gauthier a modifié le code de la première soutenance afin d'inclure une "zone morte" qui représente la place qu'occupe la maison sur la matrice. Ainsi, lors des futures passages de detection, les "zones mortes" ne pourront pas accueillir de nouvelles maisons.

L'échéance principale de cette soutenance a été la transcription d'image matricielle vers une image de type *BMP*. Pour cela, on parcourt la carte matricielle et en fonction de la valeur matricielle on affecte une certaine couleur au pixel : gris si c'est une route (*1* ou *4*), rose si c'est une case vide (*0*). La fonction *createSpriteRGB* initialise une nouvelle surface SDL de la dimension de la matrice *map*. Lors d'un premier parcours de matrice, les cases "basiques", qui correspondent aux routes et aux cases vides. Dans un second parcours de la matrice, on cherche les valeurs correspondantes aux maisons et la fonction de placement de sprite est appelée sur une image.

Il a été choisi de mettre un pixel rose pour les cases vides car le format Windows BMP ne supporte pas la transparance dans une image.

Le second objectif de cette soutenance a été le collage d'un sprite sur une image. Pour cela, on parcourt simultanément le sprite à coller ainsi que l'image receveuse. On transcrit alors pixel par pixel les valeurs de pixel du sprite sur l'image. La fonction *place_house* récupère l'image receveuse *img* ainsi que le token correspondant au type de maison à placé. Dans la surface SDL *house* est chargé le BMP correspondant au token de type de maison. En fin de procédure on obtient l'image correspondant au collage du sprite sur l'image receveuse.

3 Interface

3.1 Léo Tripier

Depuis la dernière soutenance Léo a ajouté de nouveau bouton à l'interface. Deux boutons de type *GtkScaleButton*, qui permettent à l'utilisateur de choisir le pourcentage d'océan sur la carte ainsi que la taille des biomes. Le troisième bouton permettait à l'origine de revenir en arrière dans l'interface. Cependant Arnaut ayant repris l'interface depuis, le bouton ne sera sûrement pas garder dans la nouvelle version de l'interface. Les deux autres boutons eux seront utilisés dans la nouvelle version.
Léo a également commencé à connecter l'interface avec le reste du code. En effet la création d'une nouvelle structure *UserData* contenant les différentes variables relatives au paramètre choisi par l'utilisateur.

```
typedef struct UserData
{
  gdouble b_size;
  gdouble r_ocean;
  int mwidth;
  int mheight;
  int w_type;

} UserData;
```

FIGURE 4 – Structure UserData

Cette structure permettra donc de conserver toutes les variables qui seront ensuite utilisées lors de l'appel des fonctions principales de notre code. Parmi ces variables on trouve *(gdouble)b_size* qui permet de stocker la valeur choisie pour la taille des biomes, *(gdouble)r_ocean* qui permet de stocker le pourcentage d'océan souhaité sur la carte. Actuellement ce sont les deux variables qui sont adaptées en fonction des choix de l'utilisateur car les deux boutons ajoutés sont directement reliés à ces variables. Les trois autres : *(int)mwidth* , *(int)mheight* , *(int)w_type* n'ont pas encore leurs boutons associés donc ne sont pas réellement utilisés pour l'instant.
Lors du lancement de l'interface des valeurs sont choisies aléatoirement pour ces variables au cas où l'utilisateur souhaiterait générer une carte totalement aléatoire sans choisir de paramètres particuliers. Pour l'instant, les variables relatives à la taille de la carte (*mwidth* et *mheight*) sont fixées par défaut à 256 pour faciliter les tests.
Léo a également commencé avec l'aide d'Arnaut à tenter d'utiliser le paramètre de proportion d'océan sur la carte. Cependant les tests ne furent pas concluant, il faudra retravailler

sur cela pour la soutenance finale. Le problème semble venir de la formule utilisée, Arnaut et Léo se documententeront pour trouver quelle formule utiliser afin de manipuler cette proportion d'océans.
D'après les informations issues des premières recherches il semblerait qu'un calcul de l'écart-type des valeurs de la matrice représentant la carte afin de la combiner avec le pourcentage souhaité soit une bonne base. Il faudra tout de même approfondir les recherches et les tests effectués.

3.2 Arnaut Leyre

Pour ce qui est de l'interface il y a eu beaucoup de changement dans la mise en page mais le squelette du code reste inchangé. Toutes les différentes options de l'application ont été rassemblées sur une seule fenêtre qui est divisée en trois parties :

— Le header sur la partie haute de l'écran qui s'adapte à la taille de la fenêtre avec le titre de l'application et les boutons qui gèrent la fermeture de l'application, la réduction de la fenêtre et la mise en plein écran seulement si le système d'exploitation le support.

— La liste des options à gauche qui permet d'accéder à toutes les fonctionnalités déjà implémentées. Pour cela divers objets sont utilisés tel que des zones de texte, des boutons et des scales.

— Enfin tout le reste permet d'afficher l'image résultat.

Par-dessus cela nous avons réalisé beaucoup de test différent pour ajouter des contrôles plus intuitifs. Nous avons donc travaillé sur l'implémentation d'une barre de texte permettant de récupérer une chaine de caractère qui permet de récupérer une seed ainsi que d'afficher celle générait. Parallèlement nous avons commencé à travailler sur les radios Buttons. C'est boutons connecter permette la sélection d'une option parmi une palette de chois.

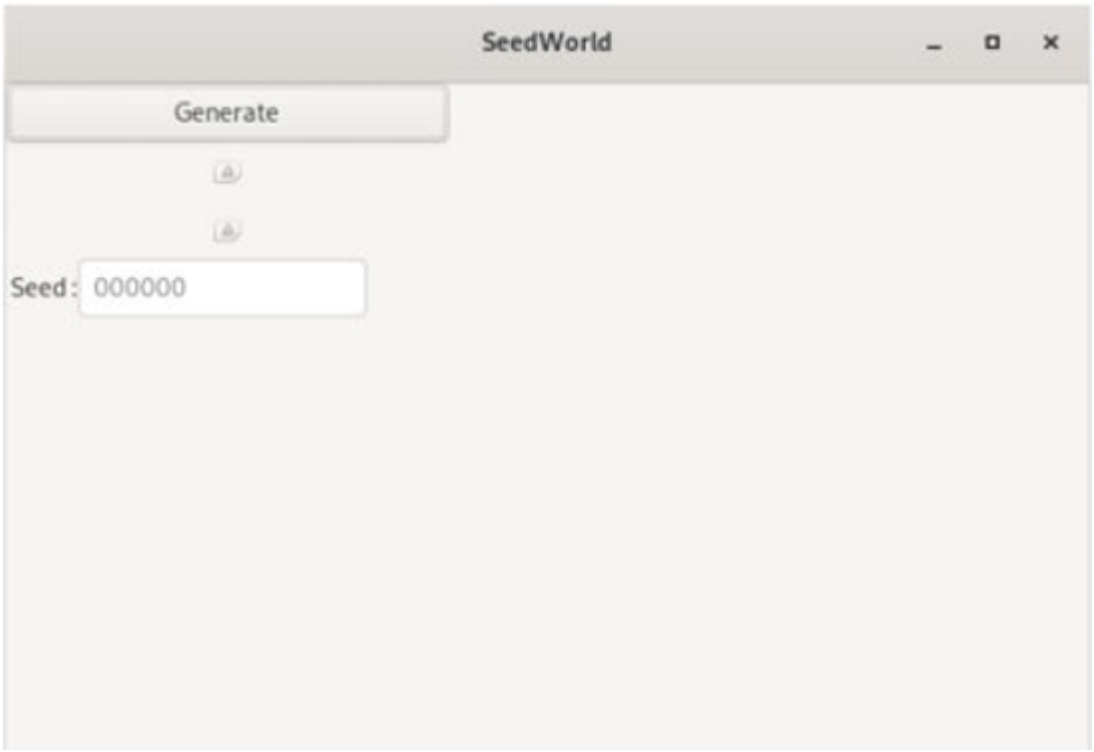


FIGURE 5 – inter

4 Site web

4.1 Léa Cruciani

Concernant le site web, l'onglet progression a été actualisé avec l'affichage des pourcentages avec une barre de progression circulaire pour chaque différente tâche que doit accomplir l'équipe M.D.W.S. Cela permettra à l'internaute de mieux visualiser où en est la réalisation du logiciel. L'onglet soutenance a également été modifié pour pouvoir rendre possible le téléchargement du rapport de soutenance 2 en fichier *.pdf*. Enfin le site a été hébergé sur gitHub Pages (`https://myochi.github.io/seedworld/`).

5 Graphismes

FIGURE 6 – Auteure Léa

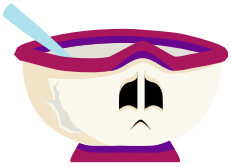


FIGURE 7 – Auteure Léa

6 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	40%	X	X	40%
Interface	X	60%	60%	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

7 Bilan

7.1 Arnaut Leyre

Lors de cette soutenance le plus gros problème rencontré est dû à la dispersion des efforts dans trop de petites taches spécialisées, ce qui à empêcher la finalisation des tâches. Cependant tous ces avancements permettront d'être plus efficace pour la dèrnière soutenance.

7.2 Gauthier Marchetti

Lors de la correction du code de la soutenance précédente, quelques nouveaux bugs on été créés. L'utilisation de la librairie SDL fut quelque peut chaotique mais la finalité s'est avérée concluente. Pour la prochaine et la dernière soutenance, Gauthier va se charger du placement des sprites de ville sur la carte du monde ainsi que de la création des noms des villes.

7.3 Léa Cruciani

Léa a pu corriger les aglutinements des routes, et a implémenté la fonction qui permet de trouver un emplacement constructible pour une ville en fonction de la taille de celle-ci. Pour la prochaine soutenance elle implémentera le réseau de neuronnes permettant de générer les noms des villes, finira l'onglet téléchargement du site, et codera une fonction qui trouvera un autre chemin si une route secondaire n'est pas constructible (changer la longueur, la direction, ou encore son point de départ). Enfin elle participera à la réalisation des différents sprites des structures pour les mondes.

7.4 Léo Tripier

Léo a rencontré quelques difficultés lors de la modification des fichiers *Perlin.c* et *gen.c*. D'abord lors de la compilation, des problèmes liés au *Makefile* des références à des librairies manquaient ainsi que des références à certains fichiers, le *Makefile* à donc été modifié en conséquence. Le second problème fut rencontré lors de la tentative d'implémentation de la proportion d'océan dans la génération, le problème étant que la proportion n'était absolument pas respectée dans la génération. De plus, certaines générations ont totalement buggé avec des proportions d'océan dépassant les 98% (avec un ratio théorique de 33%).
L'origine de ce problème n'as pas été clairement identifiée mais il semble cependant que l'implémentation des biomes ait réglé ce problème. Cela reste à confirmer par des tests plus approfondis mais il est donc possible que ce dysfonctionnement soit déjà réparé. Pour cette soutenance, Léo a mis son code en commun avec Arnaut. Ceci fut assez facile car les deux

codes sont bien organisés, clairs et lisibles. Léo a ainsi pu comprendre le fonctionnement du bruit de perlin implémenté par Arnaut et le modifier pour répondre au besoin de la création de biomes. Le début de fusion des codes s'étant bien passée, c'est encourageant pour la suite. Il reste tout de même quelques problèmes à régler dans le code (comme le problème des proportions d'océan) mais les résultats de la création de biomes sont satisfaisants.

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`