Rapport projet S3

Projet S3 : Génération procédurale de cartes par une approche top-bottom

Remerciements

Nous tenons à remercier toutes les personnes qui ont contribué au succès de notre projet et qui nous ont aidé pour la rédaction de ce rapport.

Premièrement, nous adressons nos remerciements à nos tuteurs de projet, LEBRETON Romain ainsi que POUPET Victor qui nous ont proposé notre sujet. De plus, il nous ont accompagnés tout au long de ce projet nous indiquant de nouvelles pistes de réflexion sur la conception de notre carte. Ils étaient également présents quand nous étions bloqués sur nos différents algorithmes. Leur écoute et leurs conseils nous ont permis de progresser et de mener à bien notre projet.

Nous voulons également remercier notre enseignante d'expression communication ainsi que de projet personnel et professionnel MESSAOUI Anita. Elle nous a permis de connaître les règles de rédaction d'un rapport. Par la suite, elle était présente pour nous corriger et nous guider pour le contenu et la mise en forme de ce dernier.

Nous tenons à remercier les membres de notre équipe : ZUMSTEIN Paulin, MARIN Léo, MARTINEZ Gabriel et ROURE Elie qui ont contribué à la réalisation de ce travail. Sans tous ces participants, ce projet n'aurait pas pu prendre cette direction. Les échanges en termes de conception nous ont permis de choisir la meilleure solution pour notre projet.

Enfin, nous tenons à remercier toutes les personnes ainsi que les instituts qui nous ont soutenus de près ou de loin à la réalisation de notre travail, l'IUT informatique de Montpellier, nos enseignants ainsi que nos familles.

Sommaire

Table des figures

Glossaire

- A
  - Approche descendante
    - Voir **Top/Bottom**
- B
  - Branche
    - Copie d'un projet git personnelle, pour pouvoir y travailler sans gêner les autres.
- D
  - Daggerfall de The Elder Scrolls II
    - Un jeu est un jeu vidéo de type action-RPG où le joueur incarne un seul personnage. La carte de ce jeu sorti en 1996 est immense pour son époque, 230 000 kilomètres carrés, la taille de la Grande-Bretagne. Ils ont réussi cet exploit par l'approche top-bottom de la programmation de celle-ci.
- G
  - Génération procédurale
    - En informatique, la **génération procédurale** (ou le *modèle procédural*) est la création de contenu numérique (niveau de jeu, modèle 3D,...) à une grande échelle (en grande quantité), de manière automatisée répondant à un ensemble de règles définies par des algorithmes. Le *modèle procédural* s'appuie sur les informations d'un algorithme pour créer.
  - Graine
    - Une **graine** (nom anglais : **seed**) est un code utilisé dans le processus de génération de monde dans notre logiciel. Chaque monde a sa propre graine qui est utilisée pour garder une certaine consistance dans le terrain généré, puisque la génération est pseudo-aléatoire. Ainsi, deux mondes ayant la même graine seront strictement identiques au moment de leur génération.
- J
  - Java
    - **Java** est un langage de programmation orienté objet créé par James Gosling et Patrick Naughton, employés de Sun Microsystems, avec le soutien de Bill Joy (cofondateur de Sun Microsystems en 1982), présenté officiellement le 23 mai 1995 au *SunWorld*.
  - JavaFX
    - **JavaFX** est un framework et une bibliothèque d'interface utilisateur issue du projet OpenJFX, qui permet aux développeurs Java de créer une interface graphique pour des

applications de bureau, des applications internet riches et des applications smartphones et tablettes tactiles

- M
  - Merge
    - Fonctionnalité github qui permet de fusionner des **branches**
- N
  - No Man's Sky
    - Jeu vidéo sorti en 2016 qui avait pour objectif de concevoir une univers immense, et tout ça de façon **procédural**. Le but du jeu est de voyager dans l'espace entièrement visitable et unique, dans l'espoir de trouver le centre de l'univers. Toutes les planètes sont uniques, et sont visitables, modifiables, les PNG sont tous différents et créés procéduralement.
- P
  - Procédural
    - Voir **génération procédurale**.
  - PNJ
    - Personnage non jouable
- T
  - Top/Bottom
    - Un algorithme qui suit une approche **top-bottom** divise l'algorithme en plusieurs parties différentes. En partant de l'ensemble on décompose en éléments toujours plus détaillés et plus petits sur plusieurs niveaux.

Introduction

Aujourd'hui beaucoup de jeux vidéo et autres applications informatiques requièrent énormément de données et nécessitent de plus en plus de performances de la part de l'ordinateur. Notre objectif est de parvenir à créer une carte d'un monde fictif de manière procédurale (aléatoirement) et d'approche top-bottom, tout en prenant le moins de d'espace possible dans la mémoire. Soit une carte du même type que Daggerfall de The Elder Scrolls II, ou plus récemment une carte qui reprend les principes de Minecraft dans sa génération, mais en deux dimensions. On veut obtenir une carte du monde cohérente, divisée en plusieurs régions, cliquables et elles-mêmes liées à d'autres sous-régions.

Néanmoins cette génération aléatoire doit répondre à une certaine logique. En effet, elle doit être cohérente entre ses différentes zones, prévoir des frontières adaptées et donner une possibilité de zoom/dézoom importante. Trois aspects que nous détaillerons dans le cahier des charges.

Notre application doit avoir plusieurs fonctionnalités clefs. Premièrement, elle doit avoir une interface graphique correcte et intuitive. Ensuite, nous devons pouvoir nous déplacer sur notre carte ainsi créée grâce aux contrôles du clavier ou de la souris. Nous devons aussi être capable de contrôler l'aléatoire de notre carte (être capable de générer plusieurs fois le même monde à la demande). Quatrièmement, les différentes régions de notre carte devront avoir des frontières cohérentes entre elles. Et pour finir, notre carte doit être sur plusieurs niveaux de zoom et proposer une option pour naviguer entre eux.

Après avoir rédigé le cahier des charges ainsi que les besoins fonctionnels et techniques, nous présenterons dans le rapport technique les choix de conception que nous avons fait pour améliorer notre algorithme et notre visuel. Il y aura aussi dans ce document un manuel d'utilisation de notre application, ainsi qu'un rapport d'activité concaténant toutes les méthodes de travail que nous avons utilisées sur toute la durée du projet.

1. Cahier des charges

Nous allons dans cette partie vous présenter notre cahier des charges en partant de l'analyse du sujet et de son contexte en passant par l'analyse des besoins fonctionnels et en terminant par l'analyse des besoins non-fonctionnels.

1.1. Analyse de sujet et de son contexte

La génération procédurale n'est pas une technologie inédite, elle est utilisée depuis longtemps dans divers domaines. Pour rappel, en informatique, la génération procédurale est la création de contenu numérique à une grande échelle, de manière automatisée répondant à un ensemble de règles définies par des algorithmes (cf glossaire). *Daggerfall de The Elder Scrolls II,* par exemple, est un ancien jeu vidéo de 1996 qui, pour concevoir son importante carte entièrement jouable, a utilisé un algorithme de génération procédurale. De nombreux jeux font appel à ce type de technologie, comme plus récemment *No Man's Sky*, qui a poussé la génération jusqu'au nom des créatures, des planètes, c'est d'ailleurs ce qui été beaucoup critiqué dans le jeu. Mais cela à permis de créer un univers composé de 18 trillion de planètes toutes visitables et toutes différentes.

De plus le stockage numérique est depuis toujours, et encore plus à ce jour une problématique majeure. Dans notre cas, comment manipuler et stocker une carte colossale le plus facilement possible. L'algorithme va utiliser la technique de Top-Bottom, en apportant de plus en plus de détails proportionnellement aux zooms qu'on effectuera.

1.2. Analyse des besoins fonctionnels

Dans ce projet, il est attendu que naisse une application permettant de générer des cartes. Cette application est inspirée du jeu *Daggerfall de The Elder Scrolls II* qui possède trois niveaux de "zoom" .(voir figures ci-dessous)

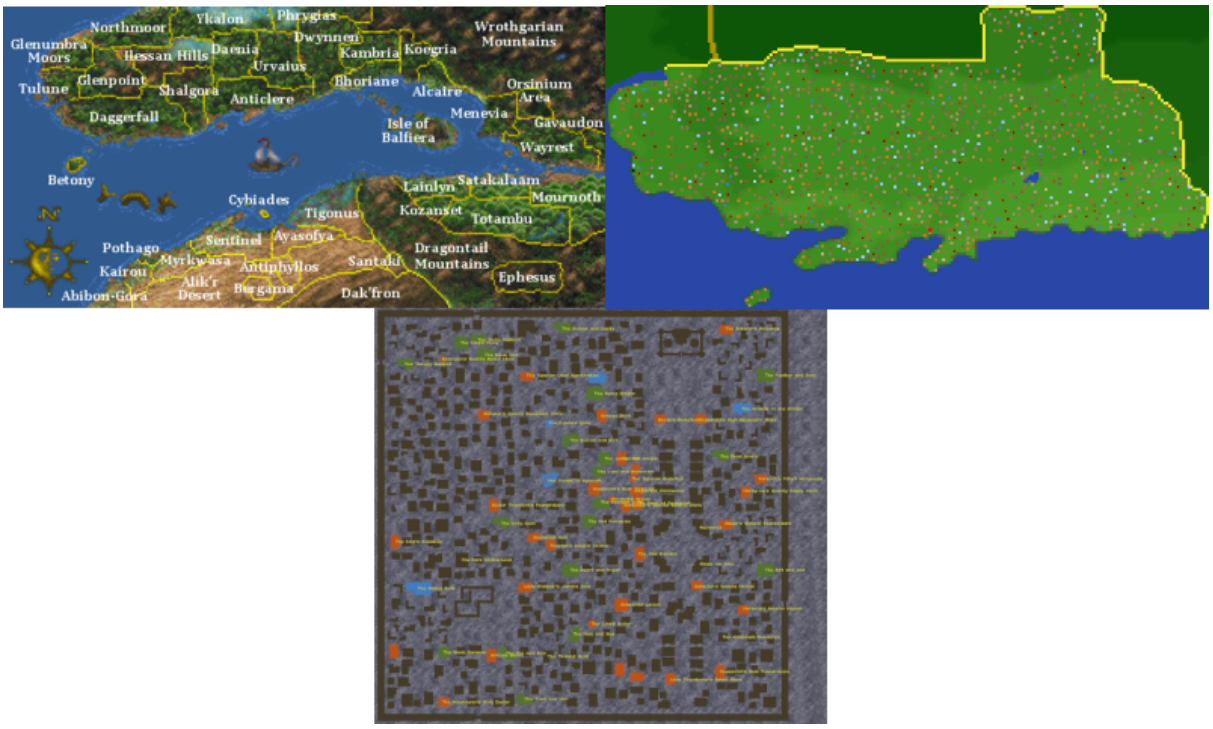

Figure 1

Images de la carte du jeu *Daggerfall de The Elder Scrolls II* avec ces différents niveaux de précision.

Le premier niveau (le plus large) est dans ce jeu une carte faite à la main et qui est stockée en mémoire. Le second niveau est un biome qui dans le jeu est généré procéduralement au même titre que les villes qui correspondent au troisième niveau. Dans notre application nous générons les trois niveaux de manière procédurale.

Néanmoins cette génération procédurale doit suivre certaines règles et garder une certaine cohérence. En effet, la transition entre chaque zone doit se faire de manière progressive. Par exemple, s'il y a une région désertique à côté d'un océan, la transition ne doit pas se faire brutalement. De plus, cette génération doit toujours être constante entre les différents zooms. Par exemple pour une région 'A' , si nous agrandissons celle-ci alors nous devons faire apparaître une nouvelle carte. L'idée ici est que si nous dézoomons de 'A' puis nous zoomons une nouvelle fois sur 'A', nous devons avoir la même carte que précédemment. Enfin, les régions et villes doivent également être cohérentes entre elles. Effectivement une rivière doit pouvoir traverser plusieurs sous-régions et ainsi être affichée dans chacune d'entre elles.

Un utilisateur peut donc faire les interactions suivantes :

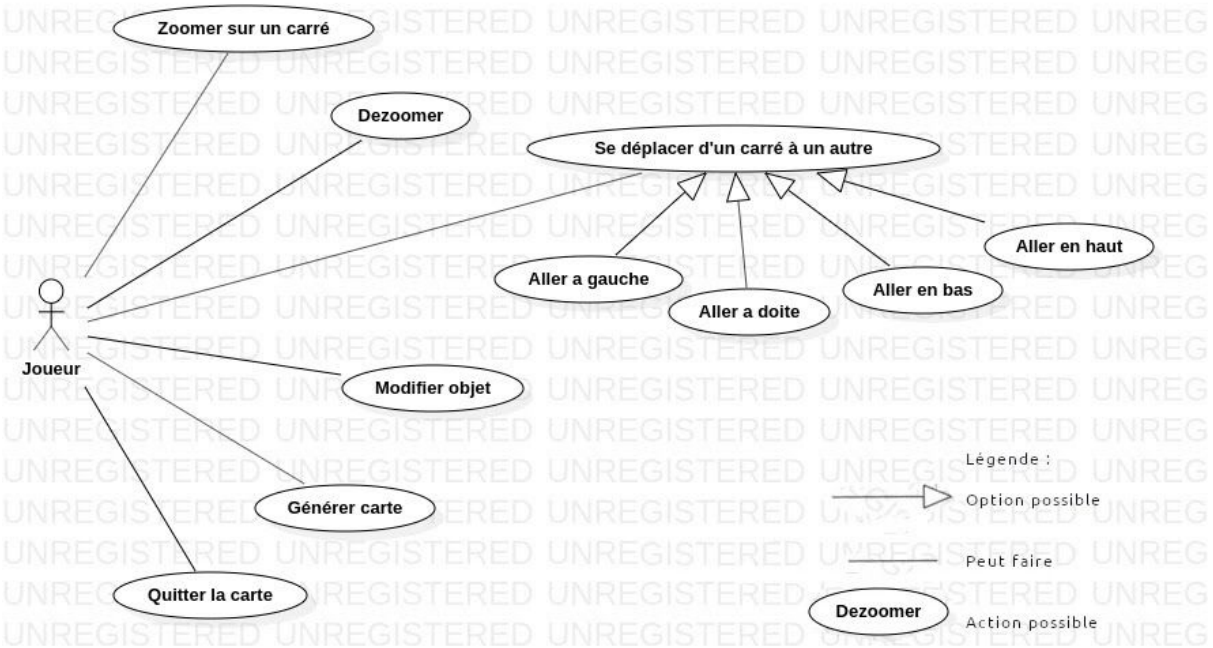

Figure 2

1.3. Analyse des besoins non-fonctionnels

1.3.1. Spécifications techniques

Premièrement, notre logiciel ne doit pas être bridé sur les dimensions et la taille de la carte. C'est-à-dire que nous devrions pouvoir générer une carte immense. Pour se faire, notre carte doit être optimisée pour ne pas être trop gourmande en ressource mémoire au risque de provoquer une erreur par manque de ressource. De plus, l'objectif de la carte est de générer une carte petit bout par petit bout car celle-ci serait trop importante pour la générer en même temps.

1.3.2. Autres contraintes : ergonomiques, juridiques

Notre logiciel doit en effet être ergonomique, prévoir donc une interface graphique adaptée et proposer des commandes intuitives. Nous devons ainsi proposer une interface graphique qui s'adapte à la taille d'écran de l'utilisateur, sa résolution ou encore du type de support sur lequel il est. Comme commandes nous devons par exemple mettre en place des contrôle par bouton, mais aussi au clavier pour améliorer le confort de l'utilisateur.

Notre logiciel ne doit pas non plus plagier tout autre application ou logiciel déjà existant.

2. Rapport technique

Pour commencer ce projet il fallait tout d'abord avoir la base informatique et graphique sur laquelle nous allions travailler et ajouter différentes spécificités et fonctionnalités. Il nous fallait une carte, du moins un objet illustrant une carte.

```
// remplissage de la fenetre (en récursif)
public void remplir(int i, int j){

    if (!(i == largeur && j == hauteur)){

        creerCarre(i,j);

        if (i == largeur){
            remplir( i: 0, j: j+1);
        }
        else {
            remplir( i: i+1,j);
        }
    }
    else {
        creerCarre(i,j);
    }
}
```

Figure 3

À l'aide de ces objets JavaFX nous pouvions les manipuler afin d'afficher des petits carrés de couleur différente pour différencier les biomes.
L'implémentation de la première méthode permettant de la remplir de manière aléatoire mais déterministe à l'aide de la fameuse graine.

2.1. Conception

2.1.1. Présentation et justification des choix technologiques

Pour la création et le développement de notre projet, nous avons utilisé différentes technologies afin de répondre au mieux à nos besoins et exigences.

Pour le langage, les tuteurs nous avaient suggéré le Python et le Java. Le Python est peu étudié durant le DUT informatique, nos connaissances et notre

aisance dans ce dernier sont assez minimes. D'où notre orientation vers des langages qui nous sont beaucoup plus familiers, le Java et JavaFX
Nous avons donc, pour les besoins fonctionnels du programme, utilisé Java et JavaFX comme langage de programmation, javaFX étant propre aux interfaces graphiques cela répondait parfaitement à nos attentes.

2.1.2. Conception orienté objet générale

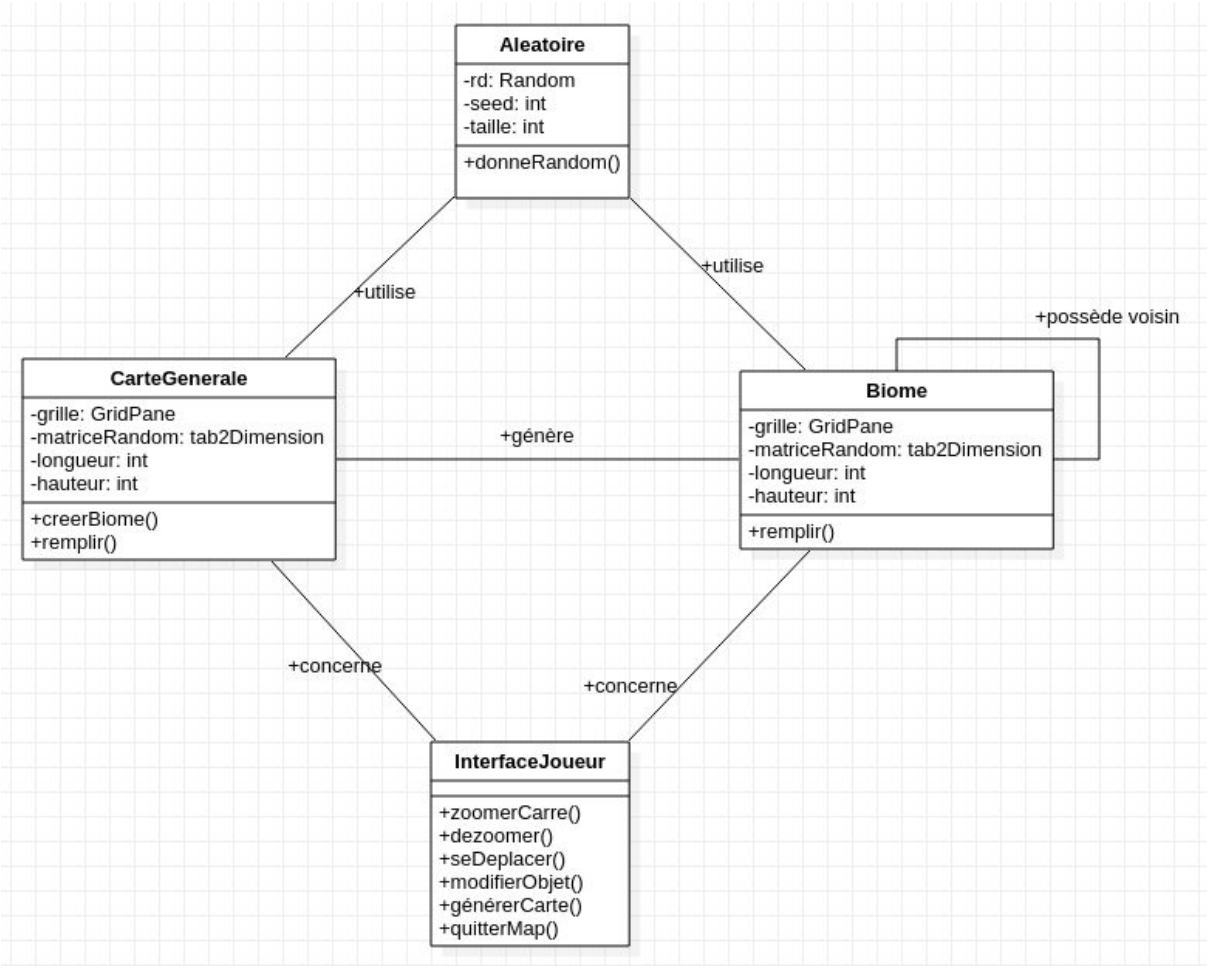

Figure 4

Dans le diagramme de classe ci-dessus qui décrit le fonctionnement actuel de notre application, il est possible d'observer des rôles différents selon les classes. On peut remarquer que par exemple, une classe est consacrée aux interactions entre l'utilisateur et le programme. Ces interactions se font à l'aide de la technologie JavaFX et sont gérées par la classe **InterfaceJoueur**. L'utilisation du JavaFX se fait exclusivement dans cette classe ce qui permet d'alléger les classes **MapProcedural** et **Biome** qui sont consacrées à la logique de notre application. Les classes

MapProcedural et **Biome** sont donc réservées à l'algorithmique. Enfin la classe **Aleatoire** encapsule un objet random pour faciliter son utilisation.

Cette conception nous a permis de remplir les objectifs que nous nous étions fixés. Nous avons pu créer une application qui génère une carte et des biomes en respectant les contraintes évoquées dans l'introduction. Mais nous sommes conscients que cette conception à de nombreuses limites. En effet, par exemple, il est impossible de faire plus de deux niveaux de zoom. Une fois le premier zoom effectué pour agrandir un biome, il est impossible de zoomer une nouvelle fois. Pour répondre à ce problème nous avons réfléchi à une conception plus complète qui sera expliquée dans la partie *"Perspective de développement futur"*.

Dans l'application un objet **MapProcedural** est vu par l'utilisateur de la façon suivante :

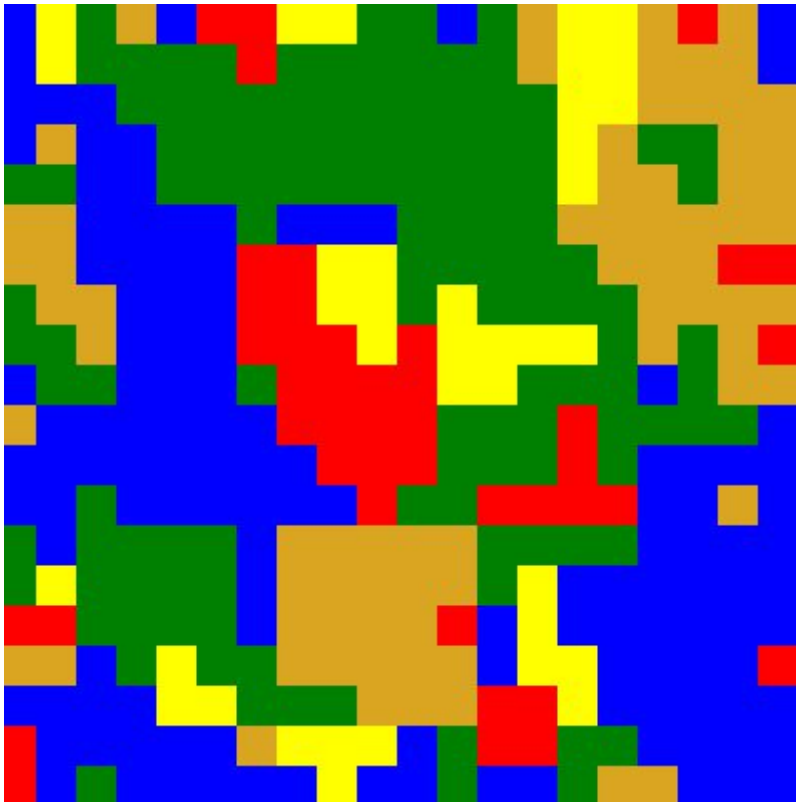

Figure 5

Sur cette **MapProcedural** on peut voir de nombreux carrés qui correspondent aux différents biomes qu'il est possible de générer et d'afficher pour simuler un zoom. Un **Biome** et une **MapProcedural** ne sont pas très différents. En effet, les deux sont représentés en interne par une matrice sur laquelle nous allons travailler. Une fois le travail sur cette matrice fini, cette dernière va être interprétée pour afficher un rendu similaire à l'image ci-dessus. Plus de précisions seront apportées à ce sujet dans la partie *"Description des algorithmes principaux"*.

2.2. Réalisation

2.2.1. Description précise de la réalisation de MapProcedural et de Biome

Premièrement, la **MapProcedural** est la map complète. Notre idée est de faire en sorte qu'une **MapProcedural** est composée de différents **Biome**. Ces différents biomes sont matérialisés par des carrés de différentes couleurs. Afin de réaliser notre idée, nous avons créé une classe **MapProcedural** (cf Figure 5 et 6) de telle sorte qu'un objet **MapProcedural** est instancié avec une *hauteur*, une *largeur* et une graine. La *hauteur* ainsi que la *largeur* permettent de donner les dimensions de la matrice d'entiers *matricerandom*. Cette matrice est remplie par la suite par des entiers aléatoires avec la fonction *remplirNbAleatoire()*. Celle-ci utilise l'attribut *aleatoire* qui est lui de type **Aleatoire** et est instancié grâce à la *seed* de la **MapProcedural**. Après avoir attribué un entier a chaque indice de la matrice, nous allons afficher les couleurs correspondantes grâce à la méthode *remplirDeCarre().* Celle-ci va faire appel à une autre fonction qui va creer un carré de couleur en JavaFx pour que l'utilisateur puisse voir le resultat. La couleur est choisie selon la valeur de la case grâce à la méthode *choisirCouleur()*

```
////////////////////////////////////////////////////////// attributs : ////////////////////////////////////////////////////////////////////

public int[][] matricerandom;
private int largeur;
private int hauteur;
public static int seed;
private Aleatoire aleatoire;

// pas définitif :
public boolean zoom;
private boolean destructible;

// pas encore utile :
private int l2;         // nb de case dans une map
private int h2;         // nb de case dans une map
private int nbniveau;   // nb de niveau de la map (nb de niveau de précision possible)

private Color[] couleurs;
public ArrayList<Biome> biomesAffiche;

////////////////////////////////////////////////////////// constructeur : ////////////////////////////////////////////////////////////////////


public MapProcedurale(int largeur, int hauteur, int seed) {

    Color[] c1 = {Color.GOLDENROD,Color.RED,Color.GREEN,Color.BLUE,Color.YELLOW};
    couleurs=c1;

    this.largeur = largeur-1;
    this.hauteur = hauteur-1;
    MapProcedurale.seed = seed;
    aleatoire = new Aleatoire(seed,  taille: 5);
    matricerandom = new int[largeur][hauteur];
    zoom = true;
    destructible = false;
    biomesAffiche = new ArrayList<>();
    remplirNbAleatoire();   // remplie matriceRandom
    remplirDeCarre();       // remplie la fenetre de carré
}
```

Figure 6

[1]

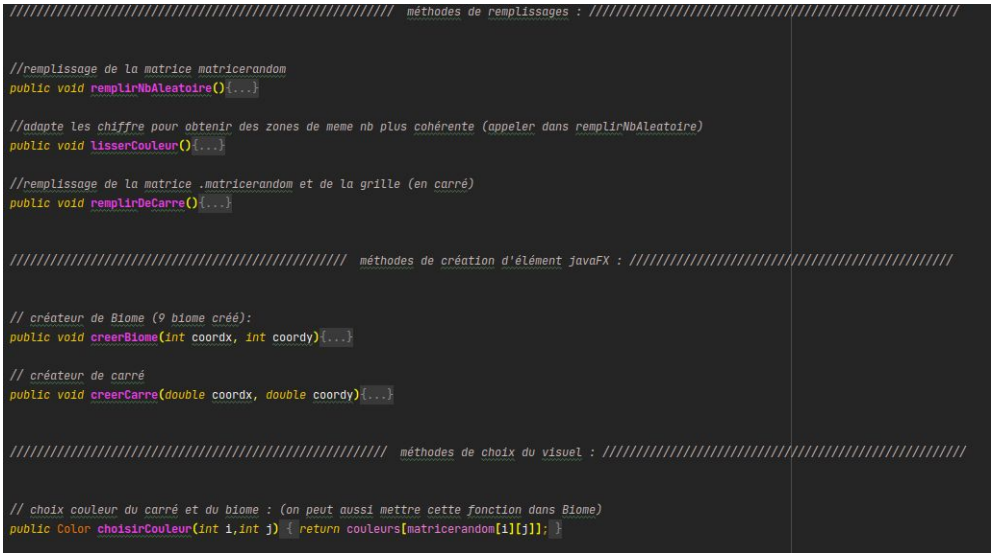

Figure 7

Deuxièmement, la majeure partie des comportements de notre carte procédurale sont située dans la classe **Biome**. Un **Biome** reprend plus ou moins le même principe qu'une **MapProcedural** (cf figure 5 et 7), il est construit avec une matrice d'entiers, une *hauteur*, et une *largeur* cependant ces deux dernières sont définies par défaut à 20 afin d'avoir un **Biome** de 20 par 20 (cf figure 8). La grande particularité des **Biome** par rapport à une **MapProcedural** est le fait qu'ils possèdent des voisins puisqu'ils composent la carte. Ces voisins sont stockés grâce à leur entier qui les définit dans l'attribut *matriceVoisin.* Lors de la génération interne d'un **Biome** il faut donc prendre en compte les 8 voisins du biome courant pour l'attribution des entiers de *matricerandom.* Pour cela, la méthode *remplirNbAleatoire(nbAleatoire),* prend en paramètre l'entier qui le définit puis génère les nouveau entier propre à *matricerandom* de **Biome,** *remplirNbAleatoire(nbAleatoire)* identifie la position du biome courant dans le but de générer les frontières entre les différents biomes. Celle-ci appelle une autre fonction : *remplirComplet()* qui elle va attribuer réellement les entiers suivant les voisins du biome courant. Par exemple, pour la partie de droite il va prendre en compte le voisin de droite et générer la frontière ensemble. Pour l'angle en haut a droite ca prend le voisin du haut, de droite et également celui en haut à droite, les quatres biomes réalisé ensemble ce bout de frontière. D'autres méthodes comme *frontiereDroite()* ou *frontiereBas()* sont présentes pour faciliter la génération commune entre deux ou quatre biomes.

```
////////////////////////////////////////////////////////////  attributs : ////////////////////////////////////////////////////////////


// attributs en commun a tous les Biome zoomer:
private int l;                    // longueur biome
private int h;                    // hauteur biome
private int l2;                    // longueur grille dans Biome
private int h2;                    // hauteur grille dans Biome


// attributs lié au Biome courant :
private Aleatoire aleatoire;
private int nbAleatoire;              // nombre aléatoire du Biome
private int[][] matricerandom;        // matrice du biome (20*20)
private int coordx;                  // coord x du nb du Biome dans MatriceMap
private int coordy;                  // coord x du nb du Biome dans MatriceMap

// les voisins:
private int[][] matriceVoisin;

// pas définitif :
private Color couleur;              // couleur du Biome
private Color variationColor = Color.BLACK;
public boolean dezoom;
private boolean destructible;

private MapProcedurale mapProcedurale;


private Color[] couleurs;
private boolean estAuCentre;
```

Figure 8

```
////////////////////////////////////////////////////////////  constructeur : ////////////////////////////////////////////////////////////

public Biome(int l2, int h2, int coordx, int coordy, Color couleur, int nbAleatoire, int[][] matriceVoisin, int[] place, MapProcedurale mapProcedurale, boolean estAuCentre) {

    Color[] c1 = {Color.GOLDENROD,Color.RED,Color.GREEN,Color.BLUE,Color.YELLOW};
    couleurs=c1;

    l = 500;
    h = 500;
    this.coordx = coordx;
    this.coordy = coordy;
    this.l2 = l2;
    this.h2 = h2;
    this.couleur = couleur;
    this.nbAleatoire = nbAleatoire;
    this.mapProcedurale=mapProcedurale;

    matricerandom = new int[l2][h2];
    this.matriceVoisin = matriceVoisin;
    this.estAuCentre=estAuCentre;

    dezoom = true;
    destructible = false;

    aleatoire = new Aleatoire(Integer.parseInt( s: "" + nbAleatoire + coordx + coordy),  taille: 100);

    remplirNbaleatoire(nbAleatoire);
    remplirBiome(place[0],place[1]);
}
```

Figure 9

2.2.2. Description des algorithmes principaux

Comme expliqué dans la partie “Conception Orienté Objet" les cartes sont représentées et manipulées sous forme de matrice *matricerandom[][]* dans notre code). La matrice d’une carte contient des entiers qui correspondent à des couleurs. Ainsi on distingue deux étapes bien distinctes. La génération de la matrice et l’interprétation de cette dernière pour l’afficher. La génération de la matrice est la partie sur laquelle nous allons nous attarder en effet c’est dans celle ci que réside la vrai difficulté de la génération procédurale (aléatoirement) en approche top-bottom. C’est pourquoi nous allons expliquer la méthode *remplirNbAleatoire()* de la classe **MapProcedural** puis *remplirNbAleatoire()* de la classe **Biome**. Ces deux méthodes ont chacune pour rôle la génération de la matrice de la carte en question. En revanche, elles ne le font pas de la même manière.

Classe : **MapProcedural** - Fonction : *remplirNbAleatoire()*

L’objectif de cette fonction est de remplir *matricerandom[][]* avec des chiffres qui pourront être interprétés comme des couleurs lors de l’affichage. Pour cela la fonction parcourt la matrice de gauche à droite et de bas en haut. La méthode regarde si les voisines de gauche et du haut de chaque cases existent et elle s'adapte en conséquence. Par exemple, si la case courante de la matrice est à la ligne 0 (*matricerandom[0]*) alors son voisin du haut n’existe pas. La méthode en tiendra compte. Une fois cette vérification faite, la méthode va remplir la case en question avec un chiffre tiré aléatoirement mais en donnant une plus grosse probabilité de tomber sur la ou les couleur(s) des voisines de gauche et du haut si elles existent. Cette enchaînement d’actions est effectuée pour chacune des cases lors de parcours de la matrice. La méthode ne tient compte que de la couleur des voisines de gauche et du haut puisque cette dernière remplit la matrice en même temps qu’elle la parcourt. Ainsi les voisines du bas et de droite ne sont donc pas encore initialisées.

Classe: **Biome** - Fonction: *remplirNbaleatoire()*

Comme expliqué dans la sous-partie *“Conception Orienté Objet”* les Biomes sont séparés visuellement par des frontières. Ces frontières composent le côté commun des deux biomes qui ont seulement certains pixels de la couleur du voisin afin de créer un dégradé.

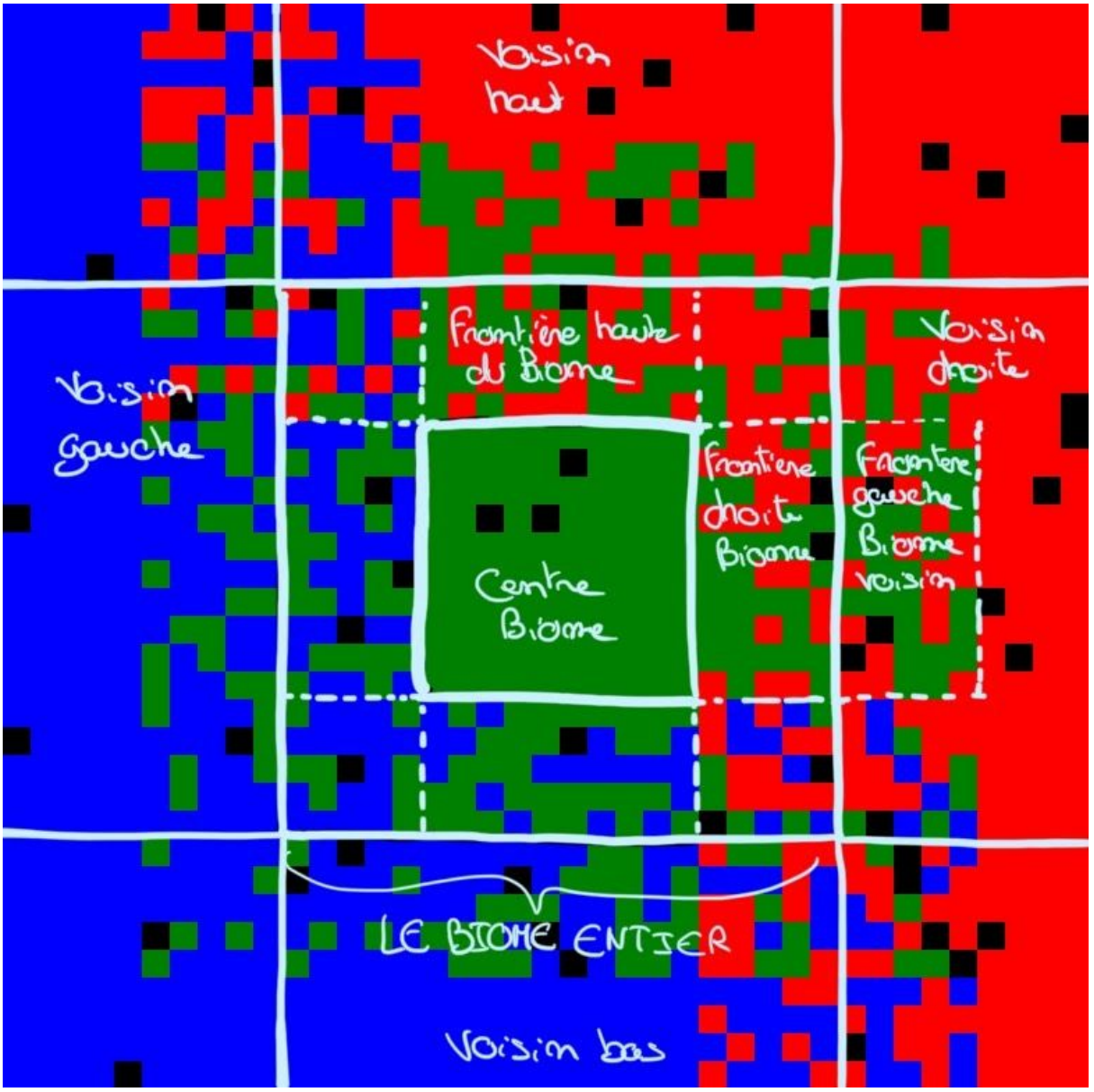

Figure 10

Sur l'image nous pouvons voir que chaque **Biome** est en réalité divisé en neuf sous parties. Le centre, qui n'est pas influencé par les voisins, et les huits frontières qui l'entourent, qui varient selon les voisins. Le grand carré central (*"LE BIOME ENTIER"* sur l'image ci-dessus) correspond dans le code à la matrice du **Biome**. C'est pourquoi chaque **Biome** remplit uniquement sa propre zone. Le déroulement de création de la zone du **Biome** est le suivant. Les coins sont générés en premier puis c'est au tour des côtés et enfin le centre. Cet ordre de génération permet de pouvoir faire dépendre les zones moins complexes des zones plus complexes, ou du moins de s'en laisser la possibilité. Les coins sont les premiers à être construits, puis les côtés se créent idéalement en fonction des coins et enfin le centre dépend des 4 côtés. Néanmoins nous n'avons pas utilisé cette fonctionnalité.

À ce principe s'ajoute l'idée que chaque **Biome** ne génère en réalité uniquement sa frontière du bas, de droite et d'en bas à droite. En revanche, un

Biome génère l'intégralité de la frontière sous forme de matrice. Si on prend l'exemple de la frontière droite du biome vert sur l'image (cf figure 7) alors le biome vert va générer sa propre frontière droite mais également la frontière gauche de son voisin de droite. Même principe pour les frontières du bas et d'en bas à droite. Dans l'exemple précédent, même si c'est le biome vert qui génère la totalité de la frontière, ce dernier ne se charge de remplir dans sa matrice uniquement la partie qui le concerne (sa propre frontière de droite et non la frontière gauche de son voisin de droite) car comme précédemment indiqué chaque **Biome** se charge de remplir sa propre zone. Ceci en récupérant la partie de la matrice de la frontière qui l'intéresse pour l'ajouter à sa matrice. Ainsi pour que le biome rouge voisin de droite puisse afficher sa frontière de gauche il va falloir qu'il demande au biome du centre vert de générer pour lui la frontière pour récupérer sa partie et l'ajouter à sa propre matrice.

Pour résumer en quelques lignes, chaque **Biome** possède une matrice et va remplir cette dernière en récupérant les parties des frontières qui l'intéressent. Ces frontières sont soit générées par le biome courant (bas, droite et en bas à droite) soit par un voisin. Mais ce qui est important c'est qu'une frontière est générée par un seul **Biome** peu importe où elle est affichée.

3. Résultats et perspectives

Nous voyons maintenant le à quoi ressemble le résultat final, si nous avons respecté les critères, les attentes, si l'objectif établi initialement a été rempli.

3.1. description des résultats obtenus

Dans le produit final de notre création de carte procédurale nous avons pu obtenir l'essentiel de ce qui était demandé initialement. C'est-à-dire que notre programme génère une carte constituée de biomes, le tout généré de façon procédurale à l'aide d'une graine choisie par l'utilisateur.

Pour cela lors du lancement du programme l'utilisateur se retrouve face à une interface lui demandant de personnaliser sa carte en choisissant les dimensions et la graine de départ orientant la procédure de création. Dès lors qu'il décide de générer la carte une nouvelle fenêtre s'ouvre prend place affichant la carte et les boutons de commande permettant les déplacements.

La carte se découpe en 2 niveaux de zoom. Le premier affiche l'intégralité de la carte, on peut apercevoir les biomes représentés par des petits carrés de différentes couleurs. 5 couleurs : bleu, jaune, rouge, goldenrod (verge d'or), vert choisi aléatoirement en fonction des couleurs voisines fromant des régions/continents. Plus il y de biomes bleu autour d'un biome plus ses chances d'être bleu augmente jusqu'à atteindre 100% s'il est complètement entouré de bleu.

Chaque biome peut être zoomé accédant ainsi à une version plus détaillée de ce dernier, c'est qu'on a appelé le deuxième niveau de zoom. Désormais le biome est affiché avec une partie de ses voisins, les frontières. Ces dernières sont construites de sorte à ce qu'elles soient identiques de chaque côté. Les frontières sont des nuages de carrés mélangés et de couleurs correspondant aux biomes concernés. Pour une frontière latérale 2 biomes sont impliqués, pour les coins 4 biomes sont impliqués. De plus, des carrés noirs apparaissent rarement, ce serait la représentation des villes par exemple.

Ensuite, l'un des critères primordiaux est de rendre la création de la carte déterministe, ce qu'on a respecté, c'est-à-dire que si l'on retourne sur un même biome il se reconstruit à l'identique, les villes sont au même endroit, les frontières sont exactement les mêmes.

A partir du moment où nous zoomons sur un biome nous pouvons utiliser les commandes de déplacement afin de visiter la carte à travers les biomes.

On remplit donc les critères primordiaux et les fonctionnalités essentielles sont implémentées :

→ Procédural
→ Déterminisme
→ Ordre de création Coins-Côtés-Centre d'un biome
→ Plusieurs niveaux de zooms avec différentes précisions

3.2. Les perspectives de développement futur

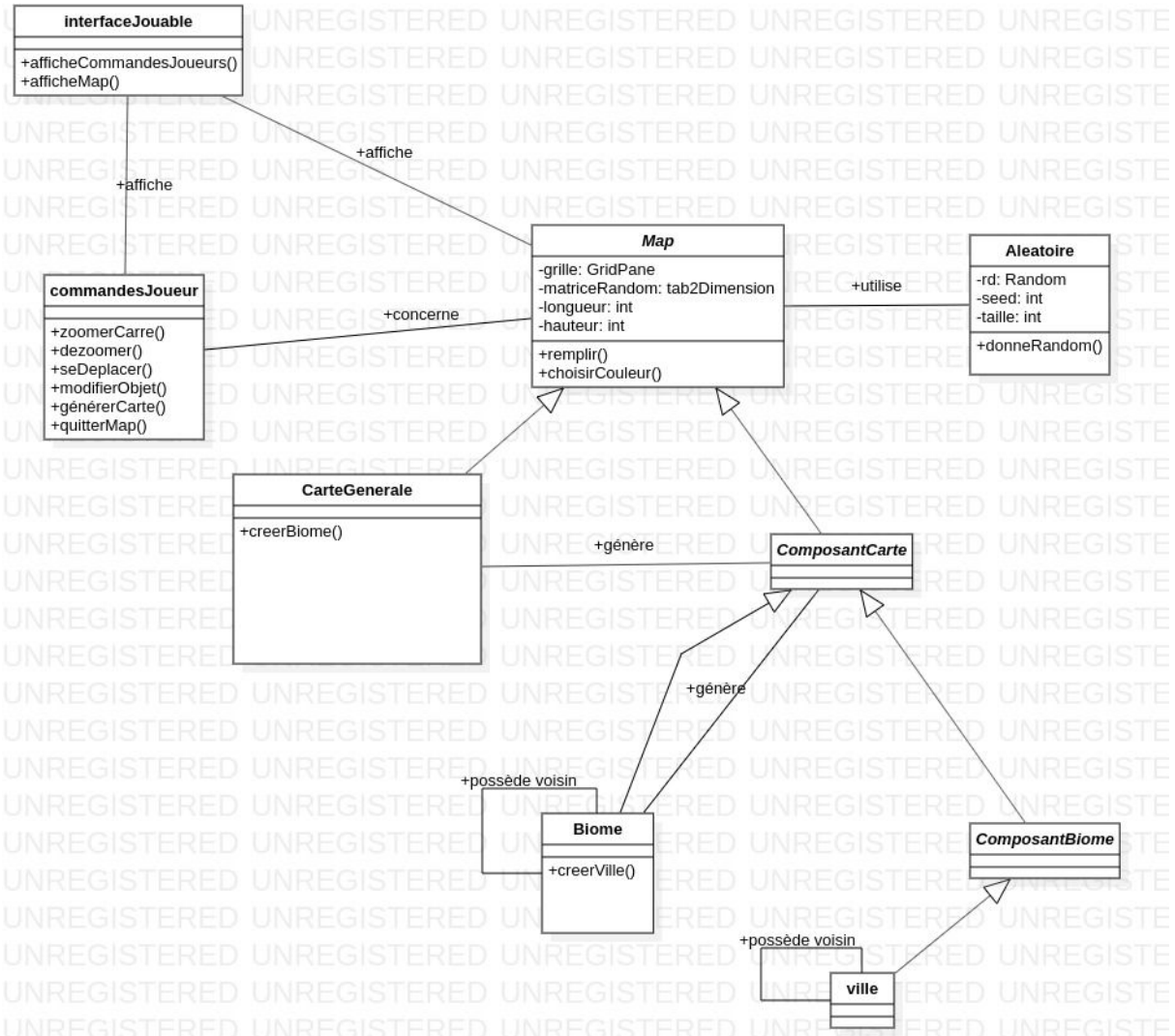

Figure 11

D'un point de vue conception nous avons réfléchi à une option plus complète que celle qui a été réalisée dans l'application. Mettre en œuvre le diagramme ci-dessus permettrait de pouvoir générer potentiellement une infinité de niveaux. En effet puisqu'il serait possible qu'un biome en génère un autre, puis un autre et ainsi de suite. De plus, l'utilisation de cette solution permettrait de factoriser le code afin de mieux structurer le projet. L'utilisation des différentes interfaces permettrait d'ouvrir de nombreuses possibilités d'amélioration pour l'application puisque leur utilisation faciliterait la création de nouvelles classes. Ceci grâce aux "contrats" qu'imposent les interfaces.

D'un point de vue plus algorithmique, il y a bien entendu des points sur lesquels nous pourrions travailler pour améliorer notre programme. En effet

concernant la fonction *remplirNbAleatoire()* de la classe **MapProcedural** décrite plus tôt il y a des perspectives d'amélioration. Dans cette fonction les seuls voisins que l'on utilise pour jouer avec les probabilités sont celui du haut et celui de gauche. Pour rappel la méthode fait en sorte qu'il y ait une plus grosse probabilité qu'une case ait la même couleur que son voisin de gauche et du haut. Car elle parcourt une seule fois la matrice dans le sens de la lecture. Le problème de cette fonction est que les formes créées sont très souvent étirées en diagonale (celle qui part de en haut à gauche et qui finit en bas à droite).

Pour y remédier et créer une carte plus cohérente, une solution serait de parcourir une première fois la matrice pour l'initialiser en bleu (océan), tout en mettant une faible probabilité d'initialiser un carré en marron (terre) par exemple. Puis une fois cette initialisation faite, parcourir plusieurs fois la matrice pour développer et agrandir ce ou ces points marrons (terre).

La fonction *remplirNbAleatoire()* de la classe **Biome** peut être également améliorée. En réalité c'est surtout les fonctions *frontiereDroite()*, *frontierBas()* et *frontiereBasDroite()* de la même classe sur lesquelles il faudrait se pencher pour améliorer *remplirNbAleatoire()*. La méthode *remplirNbAleatoire()* faisant appel aux méthodes précédemment citées est dépendante au bon fonctionnement de celle-ci. Les trois sous fonctions sont utilisées pour créer les matrices correspondantes aux frontières dont nous avions parlé dans la partie *"Description des algorithmes principaux"*. Par manque de temps, les trois sous fonctions créent une matrice sans aucun dégradé de couleurs, seulement une matrice qui mélange les couleurs de chaque voisin. Alors si nous améliorons les sous-fonctions utilisées par *remplirNbAleatoire()* en leur faisant créer des matrices avec des dégradés, le résultat visuel serait totalement changé.

3.3. Manuel d'utilisation

→ Pour démarrer le programme il suffit d'exécuter le **Main**
→ Ensuite inscrivez les dimensions souhaitées pour la carte
→ Après, indiquez la graine (seed) qui va orienter la procédure de fabrication de la carte
→ Maintenant, cliquez sur générer afin de créer la carte
→ Si vous désirer zoomer sur une biome, il suffit de cliquer sur le carré coloré en question
→ À présent vous pouvez vous déplacer entre les zones à l'aide des touches directionnelles adaptées :
- Z haut
- Q gauche
- D droite
- S bas

→ Pour dézoomer, la touche A rempli cette fonction
→ Vous avez aussi la possibilité de passer la fenêtre en plein écran en cliquant sur le bouton à cet effet
→ La touche echap pour quitter le mode plein écran

4. Rapport d’activité

Dans quelle mesure nous avons réalisé et géré le projet autant sur le plan technique, que sur plan organisationnel.

4.1. Méthode de développement et outils

Pour coder et développer nos algorithmes nous avons eu besoin de l’aide de deux logiciel propre aux programmeurs travaillant en symbiose : IntelliJ IDEA et Github. Tandis que pour ce qui est de la communication au sein de l’équipe, nous avons utilisé des outils moins spécialisés comme les réseaux discord, snapchat ou encore les mail. Nous avons enfin utilisé différents logiciels collaboratifs comme docs ou jambonard pour effectuer des tâches précises comme l’élaboration des rapport de projet et de réunion mais aussi la conception de schéma explicatif en collaboration avec nos tuteurs.

4.2. Planification des tâches

Nous avons opté, pour la gestion de projet, pour une méthode itérative un peu différente des méthodes que nous avions apprises lors du 2nd semestre en gestion de projet, mais toujours dans le même principe. Chaque semaine nous devons faire une réunion avec les tuteurs pour leur présenter notre travail fait durant celle-ci, ainsi que pour discuter des futures problématiques de notre projet et des moyens de les résoudre. Ensuite, à chaque fin de réunion, nous devions faire un rapport de réunion à rendre pour la semaine suivante. Ainsi le projet

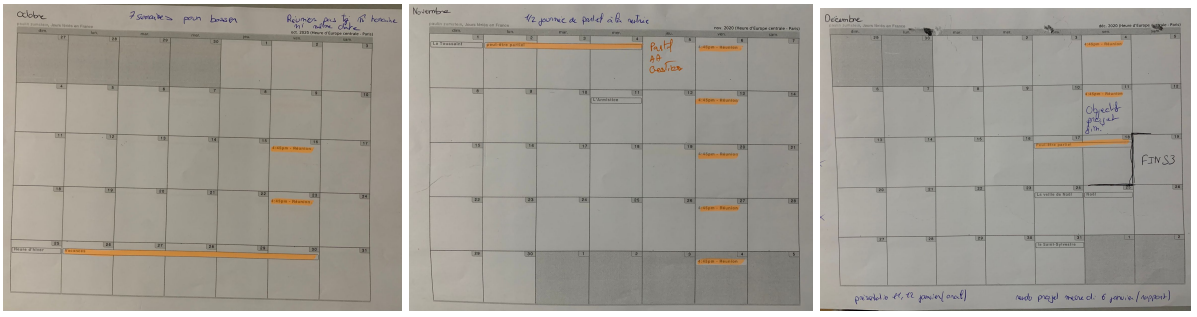

Figure 12

4.3. Bilan critique par rapport au cahier des charges

La conception d’au moins 3 niveaux de précisions n’a pas été faite bien qu'elle soit très facilement implémentable, nous avons jugé cette fonctionnalité pas cruciale dans le projet. Ainsi nous avons préféré sur les fonctionnalités essentielles

tel que la création des frontières entre biomes. Il n'y a donc que deux niveaux de précisions.

Nous n'avons pas beaucoup diversifié le contenu des biomes, ils ne contiennent que des nuages de carrés de couleurs correspondant aux voisins plus des villes (en noir).

Au début, les tuteurs ont émis la possibilité d'interagir avec la carte en la modifiant de façon absolu, de sorte à ce que même après avoir fermé la carte on puisse retrouver nos modifications quand y retournerait. Malheureusement d'autres fonctionnalités plus importantes ont pris du temps, donc nous avons abandonné cette idée.

L'implémentation de chemin entre les biomes pour rentrer dans les villes par exemple, avait été évoquée mais comme le troisième niveau de précision n'a pas été réalisé, cette idée s'en est allée avec.

Nous avons décidé d'implémenter la création de continents cohérents, en faisant des regroupement de biomes de même couleur, afin d'avoir un résultat plus agréable graphiquement.

La gestion du projet a été correcte, il y a quelques fonctionnalités qui auraient pu être traitées et réalisées, et certaines méthodes que nous aurions pu peaufiner et perfectionner si nous avions mieux géré nos travaux personnels externes au projet.

Conclusion

Pour rappel, nous avions comme objectif pour ce projet l'élaboration d'une carte générée de manière procédurale. L'utilisateur est censé pouvoir faire plusieurs actions spécifiques explicitées sur le diagramme de cas d'utilisation sur la carte (cf figure 2) comme zoomer, dé-zoomer, se déplacer d'un carré à l'autre, modifier des objets, générer une carte et quitter cette même carte. Nous devions aussi pouvoir générer une carte de dimensions souhaitée et que celle-ci ne soit pas trop gourmande en ressource mémoire, ainsi que faire un système de gestion de frontière cohérent entre les différentes zones. Le but était aussi de la générer "petit bout par petit bout" (approche top-bottom).

Finalement, nous avons réussi à réaliser une carte générée procéduralement avec une approche top-bottom. L'utilisateur peut effectuer diverses actions comme zoomer, dé-zoomer, se déplacer d'une zone à l'autre, générer une carte au dimensions modulables et être capable de quitter cette même carte. Nous avons aussi un système de gestion de frontières cohérents et nous n'avons pas rencontré de problème de ressource mémoire pour la génération.

La grande majorité de nos objectif étant remplie, nous avions quand meme penser a certaines pistes d'évolution pour le projet, comme par exemple une gestion accru des frontière, une meilleure construction en termes de cohérence pour notre Carte ou encore un zoom pratiquement à plusieurs niveaux (cf Résultat et prospective : Les perspectives de développement futur).

Ce projet de S3 nous a beaucoup apporté, en termes de connaissances techniques évidemment, mais aussi en termes de méthodes de travail ou encore de conscience et de cohésion d'équipe.

Dans un premier temps nous avons tous progresser, ne serait-ce qu'en Java ou JavaFX (connaissance algorithmique brute), mais aussi dans l'utilisation de nos divers outils de travail inhérents à la programmation, comme GitHub ou encore Intellij IDEA. Nous avons appris ce que c'était de travailler plusieurs mois sur un projet technique qui demandait une connaissance particulière sur divers sujets spécifiques (ici la génération procédurale de carte par une approche top-bottom).

Cependant, comme dans tout projet, nous avons rencontré quelques difficultés de natures variées. Tout d'abord l'utilisation de l'outil de partage de code GitHub, n'était pas intuitive, facile à comprendre et à mettre en place (beaucoup de problèmes de merge et de fusion). Le JavaFX (langage vue en 1ere année) nous a aussi posé de nombreux problèmes, notamment par son implémentation au projet sur Intellij IDEA mais aussi par le fait que nous n'avions qu'une vague idée de comment utiliser les différents composants de ce langage. Nous avons ensuite eu

plus tard des problèmes directement liés à la conception de nos algorithmes, en effet nous avions commencé par implementer la fonctionnalité de longueur et hauteur dans notre carte, cependant tous les tests de notre programme au fur et à mesure de l'avancement du projet se sont faits uniquement sur des tailles de cartes carrées (longueur = hauteur). Cela a posé un gros problème lorsque, vers le milieu du projet, nous nous sommes rendu compte que les cartes non carrées n'étaient tout simplement pas fonctionnelles. Il a donc fallu faire un énorme débogage pour régler ce problème (c'est d'ailleurs le problème le plus récurrent que nous ayons eu).

Malgré ces quelques problèmes et bugs, nous avons réussi à travailler efficacement et nous nous sommes correctement répartis les tâches au sein de l'équipe. La bonne communication de l'équipe a aussi joué dans l'accomplissement du projet.

La méthode de travail utilisée n'est pas en reste non plus. En effet, les réunions hebdomadaires avec les tuteurs pour parler de notre travail réalisé et de nos futures pistes de réflexion nous ont grandement aidés dans la réalisation du projet. Nous avons donc fourni un travail continu et constamment vérifié par les tuteurs.

Bibliographie

[1]

« Approches ascendante et descendante », *Wikipédia*. août 06, 2020, Consulté le: janv. 10, 2021. [En ligne]. Disponible sur: https://fr.wikipedia.org/w/index.php?title=Approches_ascendante_et_descendante&oldid=173602591.

[2]

« Daggerfall », *Elder Scrolls*. https://elderscrolls.fandom.com/wiki/Daggerfall (consulté le janv. 10, 2021).

[3]

« Daggerfall:Maps - The Unofficial Elder Scrolls Pages (UESP) ». https://en.uesp.net/wiki/Daggerfall:Maps (consulté le janv. 10, 2021).

[4]

« Génération procédurale », *Wikipédia*. nov. 17, 2020, Consulté le: janv. 10, 2021. [En ligne]. Disponible sur: https://fr.wikipedia.org/w/index.php?title=G%C3%A9n%C3%A9ration_proc%C3%A9durale&oldid=176702123.

[5]

« Graine (génération de carte) », *Minecraft Wiki*. https://minecraft-fr.gamepedia.com/Graine_(g%C3%A9n%C3%A9ration_de_carte) (consulté le janv. 10, 2021).

[6]

jewelsea et u_kami, « input - What is the recommended way to make a numeric TextField in JavaFX? », *Stack Overflow*, mai 02, 2020. https://stackoverflow.com/questions/7555564/what-is-the-recommended-way-to-make-a-numeric-textfield-in-javafx/12851162#12851162 (consulté le janv. 10, 2021).

[7]

S. Lee, « Introduction to JavaFX for Game Development », *Game Development Envato Tuts+*, mai 19, 2015. https://gamedevelopment.tutsplus.com/tutorials/introduction-to-javafx-for-game-development--cms-23835 (consulté le janv. 10, 2021).

[8]

« No Man’s Sky », *Wikipédia*. déc. 06, 2020, Consulté le: janv. 10, 2021. [En ligne]. Disponible sur: https://fr.wikipedia.org/wiki/No_Man's_Sky.

[9]
« Overview (JavaFX 11) », *JavaFX 11*. https://openjfx.io/javadoc/11/ (consulté le janv. 10, 2021).

[10]
« The Elder Scrolls II: Daggerfall », *Wikipedia*. nov. 30, 2020, Consulté le: janv. 10, 2021. [En ligne]. Disponible sur: https://en.wikipedia.org/w/index.php?title=The_Elder_Scrolls_II:_Daggerfall&oldid=991435533.

[11]
« Tutoriel JavaFX pour débutant - Hello JavaFX », *devstory*. https://devstory.net/10623/tutoriel-javafx-pour-debutant (consulté le janv. 10, 2021).

Annexe

Rapport Réunion Compilation

Projet S3 : Génération procédurale de cartes par une approche top-bottom

Rapport 06/10

Le Mardi 06 Octobre 2020 à 12 h 30 dans l'enceinte de l'IUT Informatique de Montpellier s'est tenu la réunion avec les participants suivant :

LEBRETON Romain : tuteur
POUPET Victor : tuteur
MARTINEZ Gabriel : étudiant
ZUMSTEIN Paulin : étudiant
MARIN Leo : étudiant
ROURE Elie : étudiant

L'ordre du jour : Présentation du projet S3 :
Génération procédurale de cartes par une approche top-bottom.

La réunion s'est déroulée en deux parties, dans un premier temps une partie logistique du projet puis une deuxième partie qui explique le sujet de manière globale.

Premièrement, le projet se déroulera sur GitHub. Il y a six collaborateurs, deux tuteurs de projet (LEBRETON Romain et POUPET Victor) et quatre étudiants (MARTINEZ Gabriel, ZUMSTEIN Paulin, MARIN Leo et ROURE Elie). Le projet GitHub regroupe le tableau de suivi avec les différentes fonctionnalités ainsi que le code qui est divisé en plusieurs branches. Une branche "master" est réservée au code propre et fonctionnel. Les commits doivent être explicites pour pouvoir se retrouver avec le versionning. Le choix du langage de programmation est libre, il faut faire des tests des différents choix pour se diriger vers le plus adapté.
De plus, les réunions se feront tous les vendredi à 16 h 45. Un calendrier regroupant les jours de partiel, les vacances et la date final de rendu devra être établie et envoyée.

Deuxièmement, le but du projet est de créer une carte du monde type Daggerfall de The Elder Scrolls II, ou plus récemment une carte qui reprend les principes de Minecraft dans sa génération, mais en deux dimensions.

C’est-à-dire qu’il faut générer une carte d’un monde aléatoirement ou chaque carré de cette région effectue un zoom sur ce carré laissant découvrir une nouvelle zone toujours généré aléatoirement. Cependant cet aléatoire doit répondre à une certaine cohérence. La transition entre chaque carré doit se faire de manière douce. S’il y a un carré de désert à côté d’un carré d’eau, la transition ne doit pas se faire de manière nette et brutale. Pour cela, un carré va être divisé en plusieurs parties. Le centre sera exclusivement du type du carré, autour de celui-ci, il y aura une frontière qui mélange les deux types de case pour avoir une transition cohérente.
De plus, cette aléatoire n’est pas réellement aléatoire et nous pouvons la contrôler grâce à une graine. On rentre une graine de départ qui peut être un nombre ou une chaîne de caractère à la génération de notre monde. Cette graine va définir l'aléatoire ce qui permet que chaque monde créé avec la même graine est identique. Les régions zoomées utilisent le même principe cependant nous pouvons utiliser les coordonnées de chaque case de chaque région pour créer leur région zoomée. Grâce à ce principe, nous pouvons générer une infinité de régions.
Pour la réalisation de ce projet, nous pouvons voir les régions comme des matrices d’entier, ou chaque entier correspond à un type de climat.

Une prochaine réunion est programmée le vendredi 16 octobre à 16 h 45, à l’IUT Informatique de Montpellier ou un travail est attendu pour la prochaine réunion. Nous devons définir un langage de programmation adapté ainsi que la création d’une matrice d’entier entre 0 et 5 générée aléatoirement accompagnée d’une interface graphique. Cette génération ne doit pas être forcément cohérente pour la prochaine réunion.

La réunion est levée à 13h10.

Secrétaire de réunion
ROURE Elie

Rapport 16/10

Le Mardi 16 Octobre 2020 à 16 h 45 dans l'enceinte de l'IUT Informatique de Montpellier s'est tenu la réunion avec les participants suivant :

LEBRETON Romain : tuteur
POUPET Victor : tuteur
MARTINEZ Gabriel : étudiant
ZUMSTEIN Paulin : étudiant
MARIN Leo : étudiant
ROURE Elie : étudiant

L'ordre du jour : Présentation de l'avancé effectué ainsi que la planification de nouvelles tâches.

La réunion s'est déroulée en deux parties, dans un premier temps une partie ou l'on a présenté le travail réalisé depuis la dernière réunion puis une deuxième partie l'on a traité des liaisons entre les différentes régions.

Premièrement, nous avons présenté une fenêtre où l'on peut modifier la taille et la graine de la carte. Actuellement, nous ne pouvons pas générer une carte de plus de 50 régions de large et de longueur sinon le programme se coupe de lui-même. Cette coupure est due à une méthode qui fonctionne récursivement et que l'on appelle trop de fois ce qui provoque une multitude d'appels de la fonction. Pour pallier ce problème, nos tuteurs nous ont conseillé de remplacer la récursivité par une double boucle "for" pour parcourir la carte. Après avoir rentré les informations nécessaires ou laissé les paramètres par défaut nous pouvons cliquer sur un bouton générer pour créer la carte. Une carte carrée avec plusieurs régions carrées de différentes couleurs apparaît alors sur une nouvelle fenêtre.

Deuxièmement, nous avons traité comment ajouter une liste de commandes que l'utilisateur pourrait utiliser. Quand on clique sur une région, cela fait un zoom sur celle-ci. Dans cette vision zoomée, nous pourrions avoir des flèches directionnelles pour se déplacer dans les régions voisines ou utiliser les flèches directionnelles pour effectuer la même tâche. Un bouton retour serait nécessaire pour retourner à la vue précédente. Par la suite, nous avons parlé de comment créer une liaison plus douce visuellement. Cette frontière devra être générée en fonction de la région actuelle ainsi que la région voisine. Pour l'instant, on peut faire un mixte des deux régions de manière non-linéaire. C'est-à-dire qui ne fait pas une frontière bien délimitée, mais floue. Pour cela,

il faut générer les zones de la région de manière ordonnée. Les quatre angles en premier, les quatre frontières en second et pour finir la partie centrale de la région. Il faudra par la suite en plus de gérer le système de frontière créé des continent, c'est-à-dire des régions qui font plus d'une case.

Une prochaine réunion est programmée le vendredi 23 octobre à 16 h 45, à l'IUT Informatique de Montpellier ou un travail est attendu pour la prochaine réunion. Nous devons gérer les frontières afin d'avoir une carte moins "carrée". De plus, nous pouvons aussi faire l'interface de l'utilisateur avec plusieurs boutons différents.

La réunion est levée à 17h30.

Secrétaire de réunion
ROURE Elie

Rapport 23/10

Le Vendredi 23 Octobre 2020 à 16 h 45 dans l'enceinte de l'IUT Informatique de Montpellier s'est tenu la réunion avec les participants suivants :

LEBRETON Romain : tuteur
POUPET Victor : tuteur
MARTINEZ Gabriel : étudiant
ZUMSTEIN Paulin : étudiant
MARIN Leo : étudiant
ROURE Elie : étudiant

L'ordre du jour : Diagrammes UML, Biomes, Frontières :

Premièrement, nous avons présenté le diagramme de classes illustrant ce que nous envisageons pour le projet. Des modifications sont possibles, en effet la profondeur de zoom et les différents niveaux de précision devraient être flexibles, et donc potentiellement supérieure à 3. Pour cela les tuteurs ont proposé de faire en sorte que Map puisse générer des Map, au lieu de générer des biomes, puis villes…
De plus, la classe Aléatoire semble ne pas être hyper pertinente, il faut réfléchir à comment s'en passer.

Ensuite nous nous sommes penchés sur le code, et avons présenté les Biomes, qui consistent à afficher le second niveau de précision formant les frontières, tout en restant au premier niveau de précision. Ce point fut mal compris dans le groupe, mais a été éclairci cette fois, il ne faut pas pouvoir visualiser plusieurs niveaux de précision en même temps. Mais les méthodes conçues seront très utiles pour la suite.
Les frontières sont réalisées en fonction de la couleur du biome juxtaposé, créant un nuage de point issu de cette couleur. Dont chaque biome, appelle une méthode frontièreGauche, frontièreDroite…
Le problème étant que les frontières doivent être générées de la même façon des deux côtés, et donc ce qu'il faudrait c'est que chaque frontière soit généré en collaboration entre les 2 (ou 4 pour les coins) biomes concernés en même temps.
Revoir la méthode remplir, qui se subdiviserait en plusieurs méthodes, remplirCoin, remplirCoté qui font appel aux voisins en question (2-3) et une direction (haut-bas..).

Les frontières des limites de la carte, il faut que nous choisissons entre supposer qu'il y a de l'océan au-delà des limites, rendre impossible d'accéder aux biomes des extrémités, ou donner une valeur par défaut.

Respecter l'ordre de génération: coins -> côtés -> centre.

Par la suite, les tuteurs nous ont proposé de raccourcir les méthodes de formation de frontières et les couleurs associées. Paulin a proposé d'agencer les couleurs sous forme de tableau où il suffirait de se rendre à l'indice correspondant. On pourrait refaçonner les méthodes de frontières avec une méthode frontièreGlobal qui utiliserait des coordonnées comme paramètres au lieu de méthodes excessivement explicites (frontiereDroite …), où il suffirait de faire varier les coordonnées y et x, pareil pour getColorBas… en une méthode getColor avec une variation de coordonnées (y+1, x-1 …)

Après nous avons présenté la fonctionnalité permettant de récupérer les coordonnées d'un biome, et de créer une Map à partir des coordonnées et de la graine initiale lorsqu'on clique gauche. Par contre il faut faire en sorte que ça supprime l'ancienne Map et empêcher l'accumulation des Map inutiles.

Enfin des conseils plus généraux sur la conception et la programmation: éviter les héritages non essentiels, en particulier MapProcedurale qui ne devrait pas être un objet JavaFX et donc éviter de le faire hériter de Parent mais plutôt lui donner un attribut JavaFX ou même un sous-objet.
Ajouter la grille et non l'objet MapProcedurale lors de l'affichage (dans le groupe de la scène).

La prochaine réunion n'a pas encore été programmée mais un travail est attendu. Appliquer les conseils sur le code, et mettre en place le zoom dans un biome, le changement de niveau de précision.

La réunion est levée à 18h00.

Secrétaire de réunion
MARIN Léo

Rapport 06/11

Le Vendredi 06 Novembre 2020 à 13 h sur la plateforme de communication BigBlueButton s'est tenu la réunion avec les participants suivant :

LEBRETON Romain : tuteur
MARTINEZ Gabriel : étudiant
ZUMSTEIN Paulin : étudiant
MARIN Leo : étudiant
ROURE Elie : étudiant

L'ordre du jour : Zoom et Remodelage du code :

Présentation des nouvelles implémentations, tels que le zoom et la génération cohérente de régions.
Avec la grande amélioration de la classe Biome, nous avons pu concevoir la possibilité de cliquer sur ces derniers afin de les afficher en plus gros, faire un "zoom". L'affichage de ce biome zoomé se fait avec l'ouverture d'une nouvelle fenêtre (scene, groupe), elle prend la couleur du zoom précédent comme couleur majoritaire et à cela est ajouté d'autres cases pour l'instant que des cases noires (qui pourraient correspondre à des villes par exemple).
De plus, nous avons amélioré la méthode de remplissage des grilles, afin de rendre cohérente la construction de la carte. C'est à dire si un biome se trouve entouré de 2 biomes bleus alors il a une chance plus élevée d'être lui même du bleu, cela varie donc en fonction du nombre de voisins de même couleurs..

Plusieurs conseils et astuces nous ont été suggérés par le tuteur, comme l'utilisation de canvas à la place des GridPane (ce qui serait plus adapté à notre projet)
L'utilisation d'un tableau pour stocker les voisins d'un biome, évitant ainsi la répétition de code.
Passer les attributs lz et hz d'un biome en static.
Fixer la taille des fenêtres et empêcher la redimension de ces dernières
Supprimer les fenetre précédentes lorsqu'on zoom
Et enfin fait en sorte de garder une constance dans la génération des biomes zoomés, en effet même si on clique sur la même case, entre deux génération le résultat est différent, ce qui est à éviter.
Et enfin, l'affichage des biomes alentour en imaginant des biomes 4x4 avec un centre de 2x2, et ce quel que soit le niveau de zoom (sauf à l'affichage global de la carte évidemment).

Une prochaine réunion est programmée le vendredi 13 octobre à 16 h 45, sur la session BBB du groupe TD-Q3 ou un travail est attendu pour la prochaine réunion. Nous devons essayer de gérer la création des frontières (coins puis côtés et centre).
La réunion est levée à 13h45.

ABSENCE EXCUSÉE: Le tuteur POUPET Victor nous a prévenu de son impossibilité à se rendre à cette réunion qui a vu son horaire déplacé.

Secrétaire de réunion
MARIN Léo

Rapport 13/11

Le Mardi 13 novembre 2020 à 16h45 sur la plateforme de communication BigBlueButton s'est tenu la réunion avec les participants suivant :

LEBRETON Romain : tuteur
POUPET Victor : tuteur
MARTINEZ Gabriel : étudiant
ZUMSTEIN Paulin : étudiant
MARIN Leo : étudiant
ROURE Elie : étudiant

L'ordre du jour : Passage à Canvas S3 :
Génération procédurale de cartes par une approche top-bottom.

Premièrement, le changement majeur cette semaine c'est le passage de GridPan à Canvas, en effet cette initiative a été donnée par les Tuteurs, qui ont pensé qu'il serait plus adapté d'utiliser des canvas à la place des GridPan, et à raison. Les canvas nous permettent d'avoir une sorte de feuille blanche sur laquelle on met ce qu'on désire, objets ,carrés, biomes, images…
Et c'est ce qu'on a fait, on a juste ajouté des carrés associés à des couleurs particulières. Ce qui simplifie le code au niveau du système, et le rend plus rapide à se lancer, avant nous ajoutions des biomes dans chaque cases de la grille GridPan. Donc plus efficace et plus flexible (pourquoi pas mettre des images de cartes faites à la main, plus tard ?).

Ensuite l'optimisation du code: tous les biomes ne sont en effet pas créés en même temps. Lorsqu'on clique sur un carré le biome correspondant se crée et s'affiche.

Après nous avons l'implémentation de l'interface joueur, en d'autre termes la capacité de se déplacer autour du biome sélectionné, à l'aide de boutons illustrants des flèches directionnelles (droite, gauche, haut, bas, milieu) et le zoom/dézoom.

De plus, l'amélioration de la carte principale qui dorénavant, permet d'obtenir une carte plus cohérente, avec la formation de régions/continents, et cela en augmentant les chances de colorisation spécifique en fonction du nombre de carrés juxtaposés de même couleur.

Par la suite quelques indications à propos du code notamment sur la récupération des coordonnées du biomes, le getx() et gety() qui n'opère pas de la bonne façon. En effet il faudrait diviser le nombre de pixels à partir de l'origine et le diviser par la taille d'une case, et non pas par un getlongueur() en évitant les +1.
Et une remarque à propos de la formation des continents qui suivent un schéma en cascade surement dû à leur ordre de création. Plusieurs solutions ont été proposées, comme avoir une fond de couleur , ou faire la construction des continent autour de certains points au lieu de le faire en ligne.

Une prochaine réunion est programmée le vendredi 20 novembre à 16 h 45, sur la session BBB du groupe TD-Q3 ou un travail est attendu pour la prochaine réunion. Nous devons concevoir l'affichage des biomes périphériques à celui zoomé (seulement leur bordure) afin de visualiser les frontières cohérentes qui devront, elles aussi, être implémentées, du moins avoir commencé dans le but qu'on en discute.
La réunion est levée à 17h00.

Secrétaire de réunion
MARIN Léo

Rapport 20/11

Le vendredi 20 novembre 2020 à 16h45 sur la plateforme de communication BigBlueButton s'est tenu la réunion avec les participants suivant :

LEBRETON Romain : tuteur
POUPET Victor : tuteur
MARTINEZ Gabriel : étudiant
ZUMSTEIN Paulin : étudiant
MARIN Leo : étudiant
ROURE Elie : étudiant

L'ordre du jour : Interface et Frontière S3 :
Génération procédurale de cartes par une approche top-bottom.

Pour commencer l'interface a été complètement revue, en effet à présent tout se fait sur une et même fenêtre, les boutons et la carte tout se fait sur la même fenêtre. La possibilité de mettre en plein écran, et de mettre une carte aux dimensions maximum adaptées à notre écran. Les Tuteurs proposent d'ailleurs plutôt, de faire varier la tailles des carrés en fonction de la taille de la carte, mais elle devrait dans tous les cas prendre la place maximum.
De plus, désormais les voisins sont en partie affichés, bien que la création des frontières ne soit pas encore gérée.
Dans un autre temps, la correction d'un problème lié au déterminisme des zooms, c'est à dire qu'à présent lorsqu'on zoom sur le même carré l'affichage reste le même.
L'ajout des autorisation au niveau des boutons (zoomable ou pas, impossible de regénérer une carte sans quitter le programme…).
Les déplacements, et les contrôles en général devront être possible à l'aide des touches (flèches directionnelles par exemple)

Par la suite les deux tuteurs nous ont exposé leur stratégie afin de concevoir la création de frontières, qui est un élément clé de ce projet.
Graphiquement nous pourrions avoir l'affichage du biome et d'une partie de ses voisins, mais en réalité, ce serait des grilles différentes, le biome courant, et l'affichage d'une partie des grilles correspondant aux voisins. Donc cela implique lors d'un zoom de calculer entièrement le biome et ses voisins. Cette méthode qui remplit les 9 biomes (le courant et ses 8 voisins) serait divisée en 3 méthodes avec une mission particulière. La première à intervenir devrait calculer les coins, en prenant 4 arguments pour les 4 biomes voisins, en

deuxième une méthode qui calculerait les côtés avec 2 arguments et enfin celle qui calculerait le biome courant avec 1 argument.
Le tout forcément en gardant la contrainte de déterminisme aléatoire, en ayant le même contenu de part et d'autre des frontières lorsqu'on se déplace.

Une prochaine réunion est programmée le vendredi 27 novembre à 16 h 45, sur la session BBB du groupe TD-Q3 ou un travail est attendu pour la prochaine réunion. La conception des méthodes de frontière est la priorité, étant donné la date butoire de plus en plus proche. En effet il nous reste 3 rendez-vous. La réunion est levée à 17h15.

Secrétaire de réunion
MARIN Léo

Rapport 30/11

Le vendredi 30 novembre 2020 à 16h45 sur la plateforme de communication BigBlueButton s'est tenu la réunion avec les participants suivant :

POUPET Victor : tuteur
MARTINEZ Gabriel : étudiant
ZUMSTEIN Paulin : étudiant
MARIN Leo : étudiant
ROURE Elie : étudiant

L'ordre du jour : Frontière :
Génération procédurale de cartes par une approche top-bottom.

Pour commencer l'ajout majeur a été l'implémentation des frontières avec des pointillés. Les frontières d'un biomes représentent 5*2 carrés chacune. Les différentes parties sont calculées indépendamment, effet il y a une méthode pour les coins, pour les côtés et pour le centre. De plus, les frontières sont biens identiques lorsqu'on la regarde depuis différents biomes.
Petite correction du code, dorénavant faire un zoom in-out-in affiche bien la même chose.
Par la suite des améliorations ont été proposées par le tuteur à propos du nbAleatoire d'un biome qui par moment peut être répété, donc améliorer le hachage du nombre.
Après des indications quant à l'implémentation des frontières, il faut que les deux côtés d'une frontière soient calculés simultanément afin d'avoir une cohérence entre les deux.
Donc pour cela, nous pourrions construire le biomes courant et ses frontières, sans calculer les frontières des biomes voisins, sinon ca calculerait toutes les frontières de la map à la chaîne. Les biomes de gauche calculent leurs frontières de droite, et les biomes à droite auraient juste à l'afficher et calculer leur frontières de droite (pareil pour le haut et le bas). Les méthodes de calcule de frontières (coin, côtés) devrait à présent envoyer des matrices correspondants aux frontières calculer afin de les copier dans celle courante.

Un travail est attendu pour la prochaine réunion : L'application des conseils et indications afin d'améliorer les frontières et mieux correspondre aux attentes du sujet. La réunion est levée à 17h30.

ABSENCE EXCUSÉE: Le tuteur LEBRETON Romain nous a prévenu de son impossibilité à se rendre à cette réunion qui a vu son horaire déplacé à cause de partiels.

Secrétaire de réunion
MARIN Léo

Quatrième de couverture

RESUME EN FRANCAIS

Le projet de création de carte procédurale par approche top-bottom est un programme informatique dont le but est de permettre la création d'une immense carte géographique virtuelle, imaginaire, automatique et unique avec la possibilité de visiter cette dernière. Bien que la taille de cette carte soit gigantesque, la stocker n'est pas du tout un problème. Ceci en créant les cartes par une approche descendante. La carte respecte une contrainte de déterminisme aléatoire primordial dans ce projet. Le programme a été développé en Java et en JavaFX avec sa librairie pour l'interface graphique.

Mots clés : JavaFX, Top-Bottom, Procédural, Cartographie, 2D, aléatoir, Java, Approche Descendante, déterminisme

RÉSUMÉ EN ANGLAIS

The procedural map creation project using a top-bottom approach is a computer program whose goal is to allow the creation of a huge virtual, imaginary, automatic and unique geographic map with the possibility of visiting it. Although the size of this card is gigantic but storing it is simple. This by creating maps through a top-down approach. In this project an essential constraint is respected, indeed the map respects the random determinism building. The program was developed in Java and JavaFX with its library for the graphical interface.

Mots clés : JavaFX, Top-Bottom, Top-Down, Procedural, castrography, 2D, random, Java, determinism