INF5153 - Génie logiciel : conception

TP Génération Procédurale de terrain

Présenté par :	Mokhtar Safir	SAFM14118605
	robitaille-larratt jean p.	ROBJ02049306
	Van Velzen, Philippe	VANP25089705
Groupe :	A	
Remis à :	mosser.sebastien	

Date : 13 décembre 2020

Département d'informatique
Université de Québec a Montréal

Table des figures

Table des matières

1 Introduction

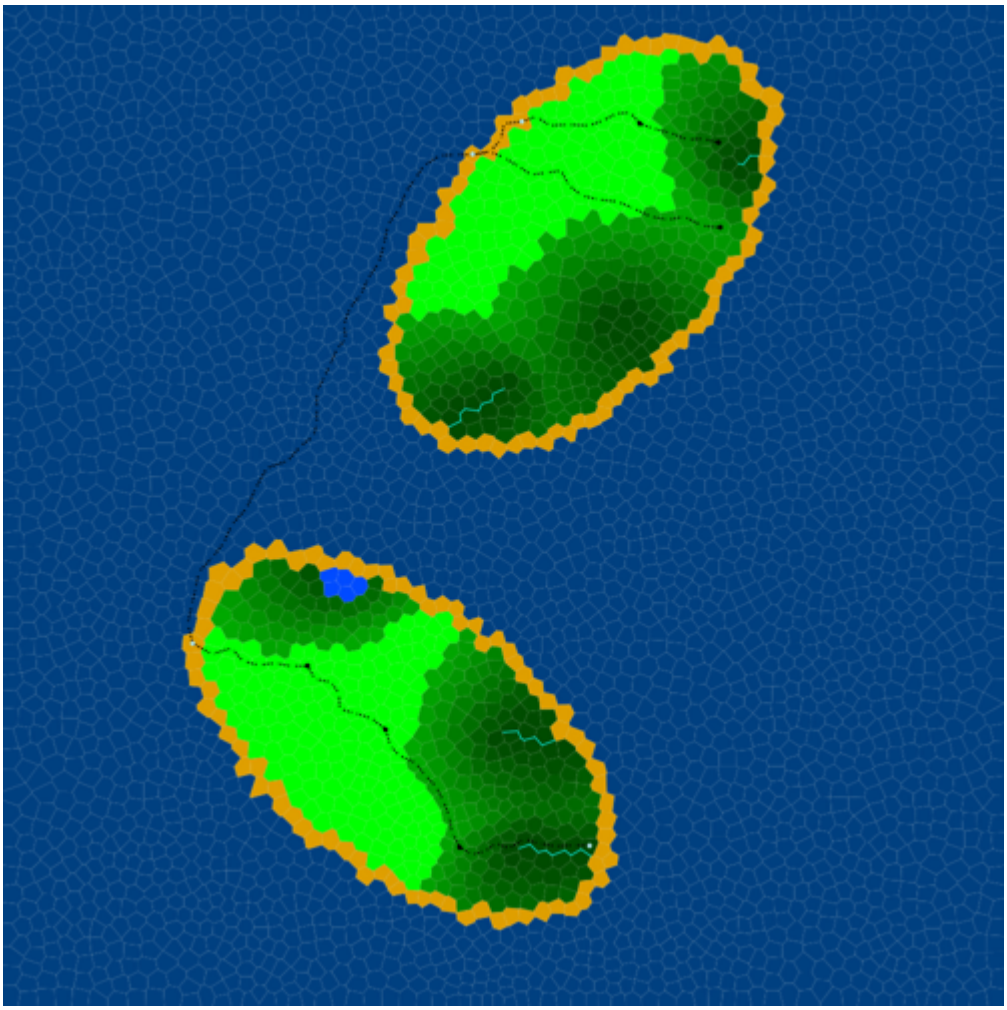
Ceci est le rapport final du projet de session de l'équipe Swarm-Colony, pour le cours INF5153, Génération procédurale de terrain. Pour résumer le travail que nous avons fait, nous avons présentement toutes les fonctionnalités de la phase 3 : produit final, bien que notre projet n'est pas sans bug. Ce rapport, présente quelques exemples de conception séparé par package, accompagné de leur diagrammes de classes. Nous avons aussi inclue un diagramme de séquence global et une liste de bugs connus et leur explication. Nous terminons par nos bons coups et nos moins bons, ainsi qu'un exemple d'ajout de fonctionnalité facile, et une plus difficile.

2 Exécution

Commençons par l'exécution. Vous pouvez exécuter le programme à l'aide des options input et output. Si ces options ne sont pas incluses, le programme lance une erreur. Ensuite, les options suivantes peuvent être utilisées en les combinant de différentes façons et avec les type d'arguments convenable selon les spécifications : shape, water, soil, altitude, heatmap, seed, archipelago, rivers, location, production, pois et roads. L'option pois, support aussi la possibilité de pouvoir être appelé plusieurs fois. Un exemple de d'exécution du programme est la suivante :

./ptg.sh -i samples/sample-1000-1000-4096.mesh -o demo.mesh –shape tortuga –archipelago 2 –pois cities :3 –pois ports :2 –roads –water 1 –rivers 2 –seed 150

Qui devrait générer la carte suivante après quelques secondes (l'attente est normale) :



Les heatmaps (Altitude, humidity et ressources) sont offerts en Annexe.

Cette carte montre nos îles à leur meilleur, toutes les fonctionnalités y sont, et se génèrent correctement. Ceci n'est pas le cas de toutes les cartes générés. En effet, il arrive par exemple que les rivières n'atteignent pas l'océan. Ceci n'est qu'un exemple des problèmes qui seront discuter plus loin dans ce rapport. Malheureusement, nous n'avons pas développer un système de rejet d'île en cas d'anormalité.

3 Conception

3.1 Diagramme de séquence Global

Nous incluons ici un diagramme de séquence qui montre bien la structure du projet, d'un point de vu global. Nous élaborerons sur les différentes classes plus loin dans le rapport, mais ce que nous voulons souligner ici est la structure abstraite qui nous guidait lors de la conception du projet. Cette structure venait du patron de conception ligne de montage.

L'idée était de passer notre objet principal, ici **World**, d'un générateur à l'autre. Chaque générateur modifie le world selon les fonctionnalités qu'il implémente. Cette idée uniformisait l'ajout de fonctionnalité (les fonctionnalité qui modifiait la carte). Ainsi l'ajout des rivières implique l'ajout d'un générateur de rivière qui s'occupe de modifier notre objet **World** de la façon voulu.

Ceci permettait aussi la division du travail, quelque chose que nous n'avons pas réellement tester avant la fin, quand nous avons fusionner deux branches contenant 2 fonctionnalités différentes, soit localisation et les points d'intérêts. La fusion à été relativement facile.

Toutefois, cette conception implique que l'objet world, bien qu'il offre des services comme des getter spécifique qui prouvent être très utile lors d'ajout de fonctionnalité, il nous semble que l'objet world sert surtout de base de données. Elle permet aux autres objets de la modifier mais n'exécute pas beaucoup de fonctionnalité. Nous croyons que ce choix est justifié tout de même, puisque qu'il nous a permis d'ajouter de nouvelles fonctionnalités rapidement, facilement et sans conflit avec les autres fonctionnalités.

Nous passons maintenant à quelques exemples de conceptions de plus bas niveau.

3.2 La forme du terrain

Commençons par la forme du terrain. Les terrains peuvent avoir n'importe quelle forme. Pour ce projet, on traite deux formes spécifiques, **un atoll** (forme circulaire) et **une tortuga** (forme elliptique).

Pour cela nous proposons :
Une **interface Shape** qui doit être réalisé par une forme quelconque.
Dans notre cas, la forme **Circle** et **Ellipse** implémente la methode :

```
isInShape(c: Coordinate): boolean
```

qui valide l'appartenance d'une tuile à la forme en se basant sur l'équation mathématique qui définie la forme.

Ensuite, une classe abstraite **IslandShape** qui à son tour est **spécialisé** par une classe définissant la forme finale de l'ile. Dans notre cas, la classe **CircularIsland** et **EllipticIsland** mettent en évidence la relation de spécification avec **IslandShape**.

Il est facile de voir que nous pourrions ajouter une île de n'importe quelle forme mathématique.

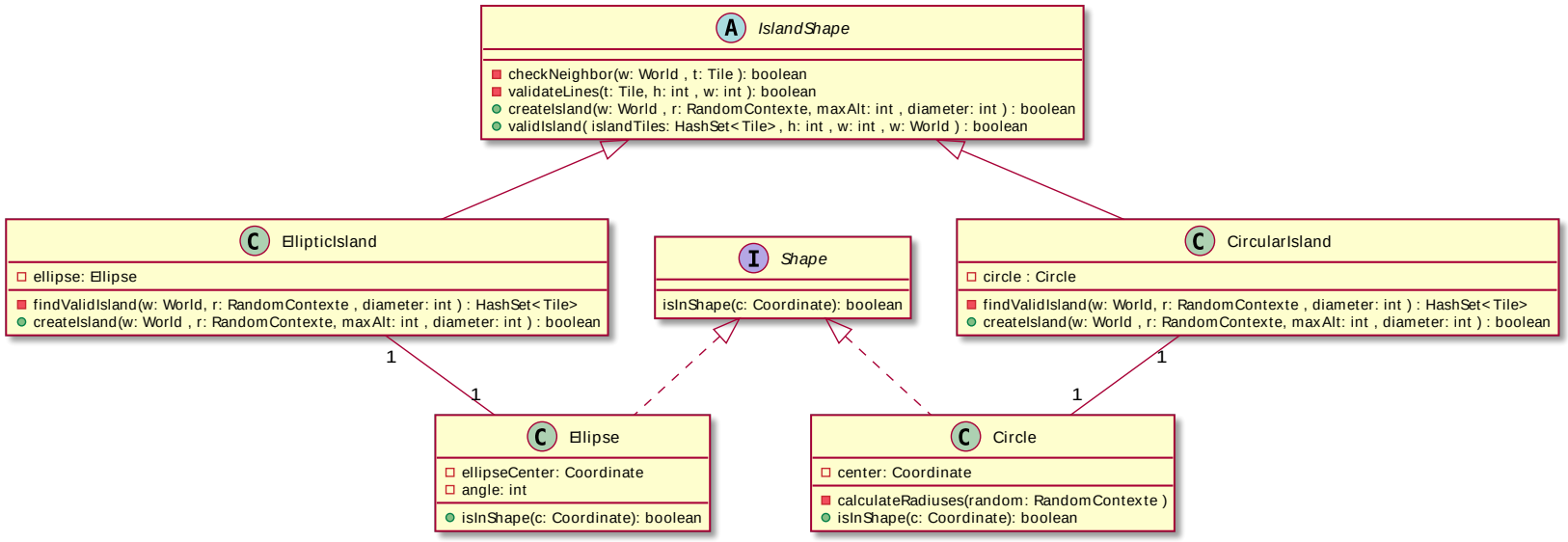


FIGURE 1 – Relation de spécification et de réalisation lors la création d'une forme de terrain

3.3 Le monde

World est notre objet qui continent les informations pertinentes à la créations d'un nouveau mesh. Mais l'objet offre aussi des services pertinent à nos générateurs. Par exemple, le monde contient les informations qui différencie les différentes îles les unes des autres. Ceci facilite l'ajout de caractéristiques, comme des rivières, à chacune des îles de manière indépendante.

La tuile, et les lignes et points qui la composent, sont les unités de base qui permettrons de construire le mesh final. Le monde est un contenant de tuiles et s'occupe d'être le lien entre le mesh (composé de tuiles) et les générateurs (qui modifie les tuiles).

Une autre fonctionnalité démontré dans se diagramme est la structure du seed, que nous proposons comme étant une stratégie. Cette stratégie étant instancier comme étant seeded ou seedless, si nous aurions besoin d'un autre random, pour d'autre fonctionnalité (non associé à la génération d'île) il suffirait d'en faire un autre.

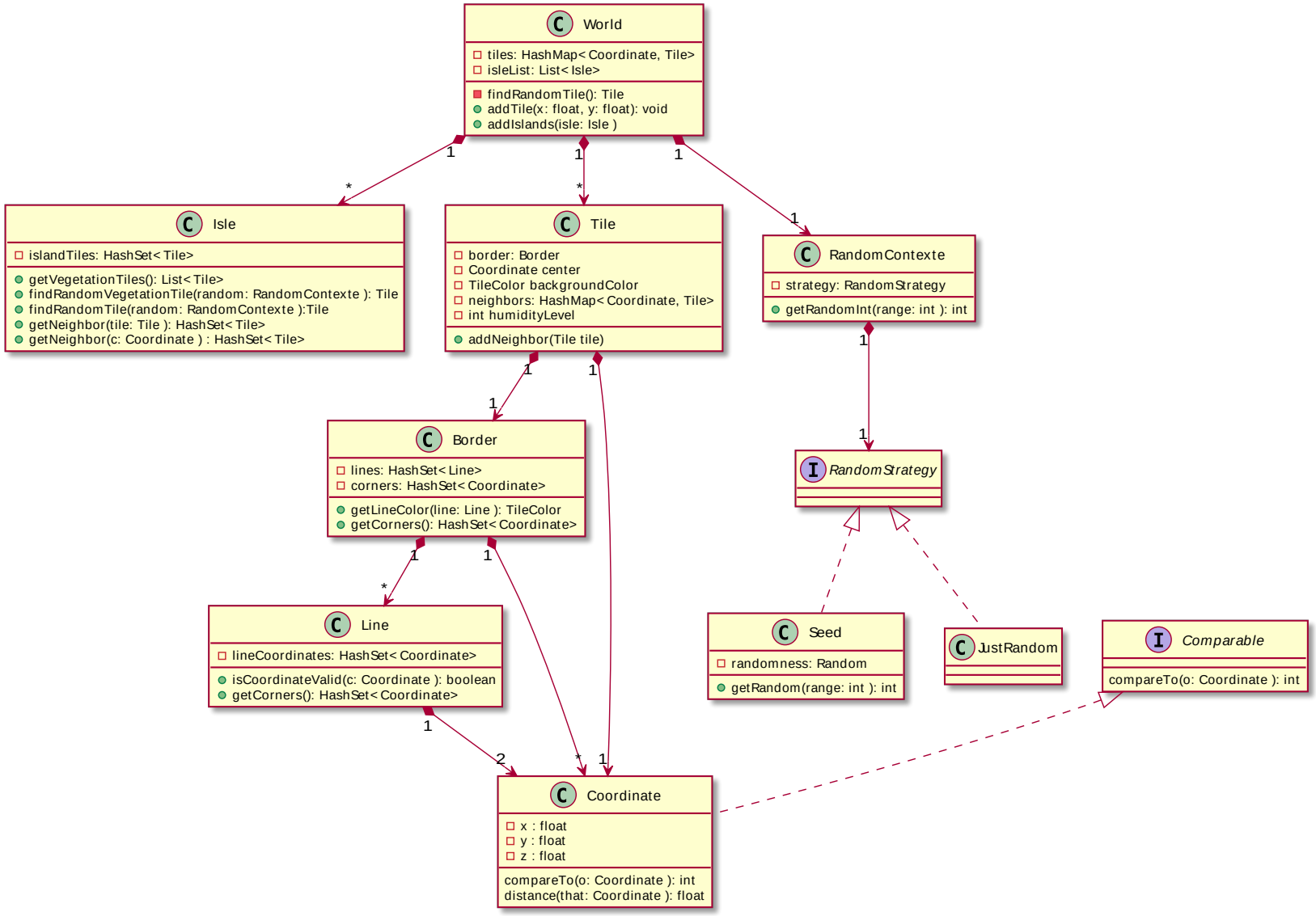


FIGURE 2 – Les différents composant du monde

3.4 Les aquifères

Les aquifères sont des exemples de générateur, qui sont de type IslandProcessor et qui utilise la méthode clef de tous les générateures, la méthode apply(). Ainsi, tous les instances d'aquifère, nape, lake et river, implémente apply, mais aussi la méthode spécifique aux acquifères, applyHumidityEffect().

Ce diagramme montre aussi que les lacs et napes implémentent WorldItem, un ajout de la toute fin du projet et qui abstrait la couleur (bleu ou vert foncé) et le type (lake ou Nape). Cette interface intervient aussi avec les biômes, qui sont aussi représenté par une couleur (jaune) et un type (plage).

Tous les générateurs sont directement ou indirectement appelés dans WorldGénérator. ainsi pour ajouter un générateur, il s'agit d'ajouter une classe qui instancie générateur et l'ajouter dans WorldGénérator. Cette uniformité facilite énormément l'ajout de fonctionnalité.

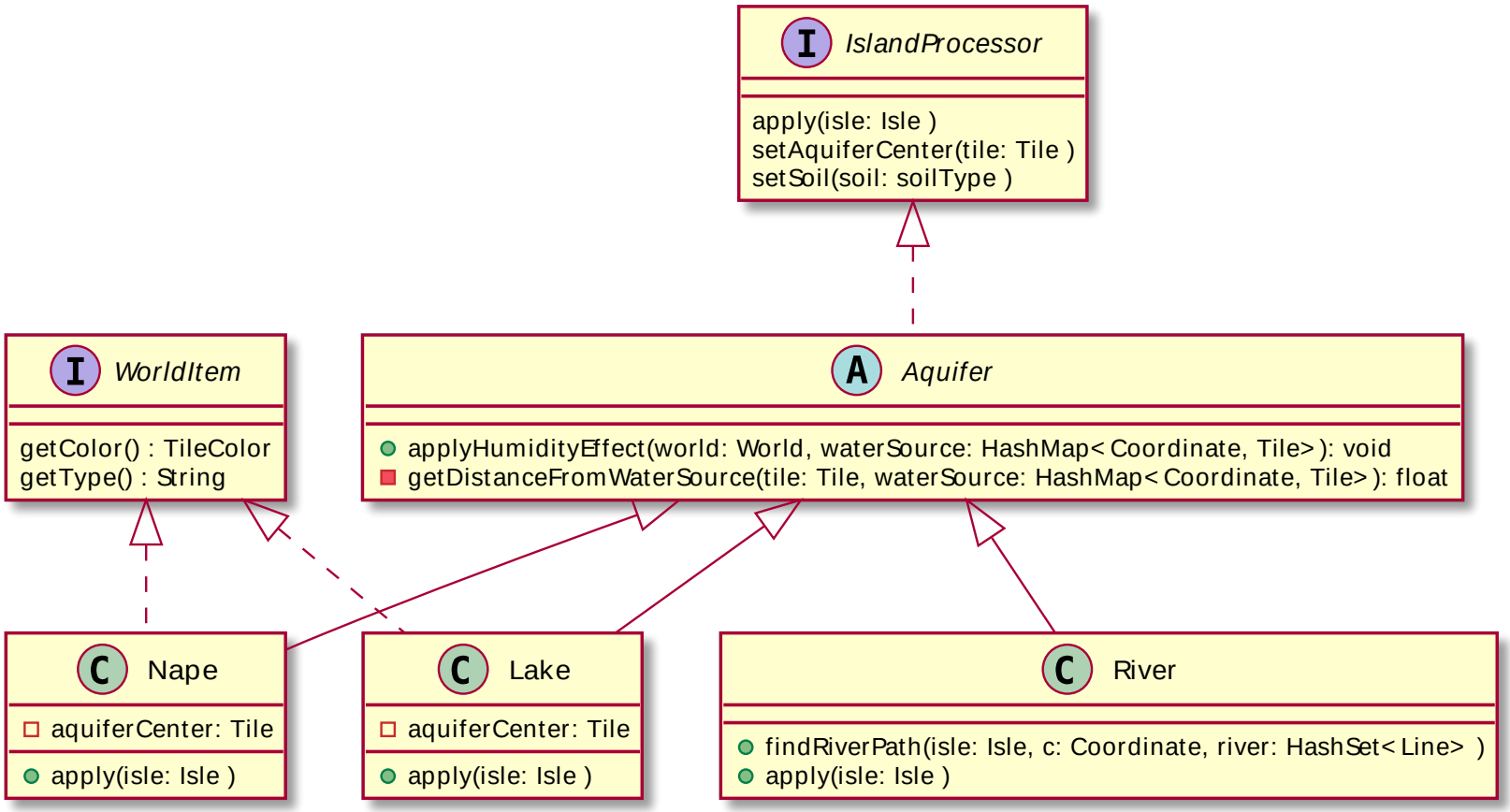


FIGURE 3 – Relation de spécialisation

4 Points forts et moins fort

4.1 Points forts

Le Mode

Un exemple de conception bien réussi, simple et indépendant du reste, est la conception relier aux Heatmap. Elle n'est pas parfaite, mais l'idée est là. Lors de la synchronisation du mesh concret avec l'objet World, nous passons par le Mode, qui est un type de heatmap. Ainsi nous appelons mode.getColor() d'une couleur passer en paramètre et qui est transformé en la couleur du heatmap approprié. Il n'y a pas de répétition du code pour chacun des Heatmap.

Prenons par exemple la couleur d'une tuile lac dans un heatmap d'humidité. Le Mode (humidité ici) s'occupe de demander à la tuile ce que ce type de mode veut avoir comme information (l'humidité de la tuile). Elle renvoie ensuite la couleur en lien avec ce heatmap là (bleu) augmenté d'un facteur relier à l'humidité reçu par la tuile.

Les Générateurs

Cette exemple de point fort est déjà décrit plus haut.

4.2 Points faibles

La qualité et l'uniformité du code ont toutefois été légèrement mises de côté dû à un projet compliqué, plusieurs changement de conception et un manque de temps. Ainsi, plusieurs conceptions ne sont pas implémentés uniformément (WorldItem pour les biômes et aquifères) et des conception ont été laisser à leur première état imparfait puisque ça fonctionnait déjà (Mode pour les Heatmap). Il arrive aussi que nous utilisons des nombres magiques dans notre code.

À cela s'ajoute des bugs qui ne sont pas toujours traités correctement. C'est le thème de la prochaine section. Ajoutons aussi que la gestion des maps imparfaites (qui devrait être rejeté) n'est pas implémentée. Ceci fera partie d'une section prochaine, où nous essaieront d'implémenter une fonctionnalité facile et une fonctionnalité difficile à notre projet.

5 Bugs connus et potentiel

Nous savons que notre programme comporte déjà quelques bugs. L'exemple le plus évident sont les rivières qui n'atteignent pas l'océan, restant coincé dans des tuiles d'altitudes similaires. Parfois la couleur des lacs est influencé par l'humidité des lacs et rivières avoisinantes.

Nous pensons aussi qu'il y a des bugs potentiels, que nous avons jamais rencontrer. Par exemple, nous croyons que si certains arguments sont utilisés de façon absurde (–rivers 1000 disons) cela pourrait être un problème, nous n'avons pas eu le temps de tout tester. Nous garantissons les fonctionnalité avec des nombres acceptables. Souvent des îles trop petites où il n'y a que de la plage empêche l'ajout d'autres caractéristiques aux îles.

6 Exemple d'ajout de fonctionnalité

6.1 Fonctionnalité facile

Nous croyons qu'il serait relativement facile d'ajouter un log et un chronomètre pour chaque étape de la génération des îles, comme les load screen de jeux. Ceci est un exemple de fonctionnalité que M. Mosser proposait d'ajouter de façon théorique lors de la présentation oral de notre projet.

Puisque notre projet est divisé en générateur, il serait facile de logguer chacune des étapes dans un fichier ou au terminal. Cependant, tout dépendant de la précision qu'on désire dans les logs, on aura soit à ajouter ces lignes de codes dans notre WorldProcessor (facile) ou dans chacun de nos petits générateurs (plus de code).

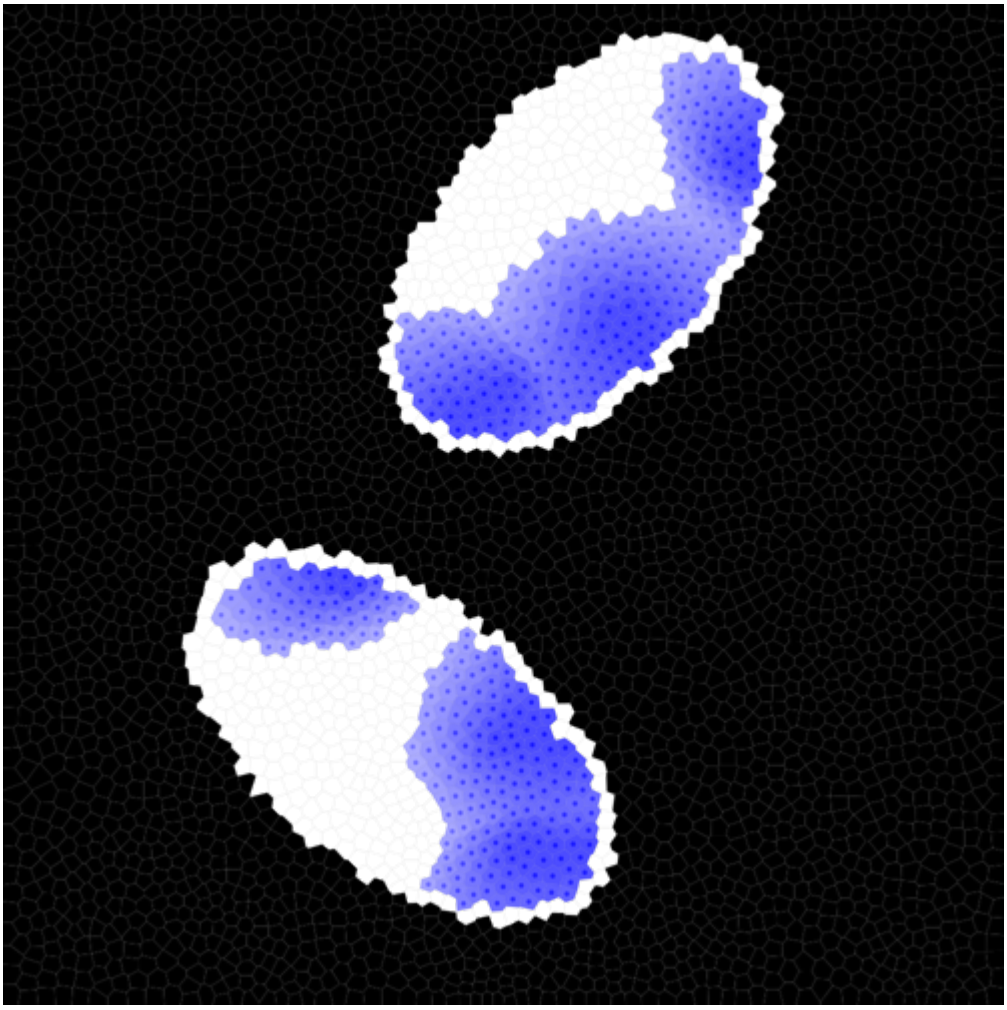
Nous croyons donc que notre projet permet l'ajout de cette fonctionnalité.

6.2 Fonctionnalité plus difficile

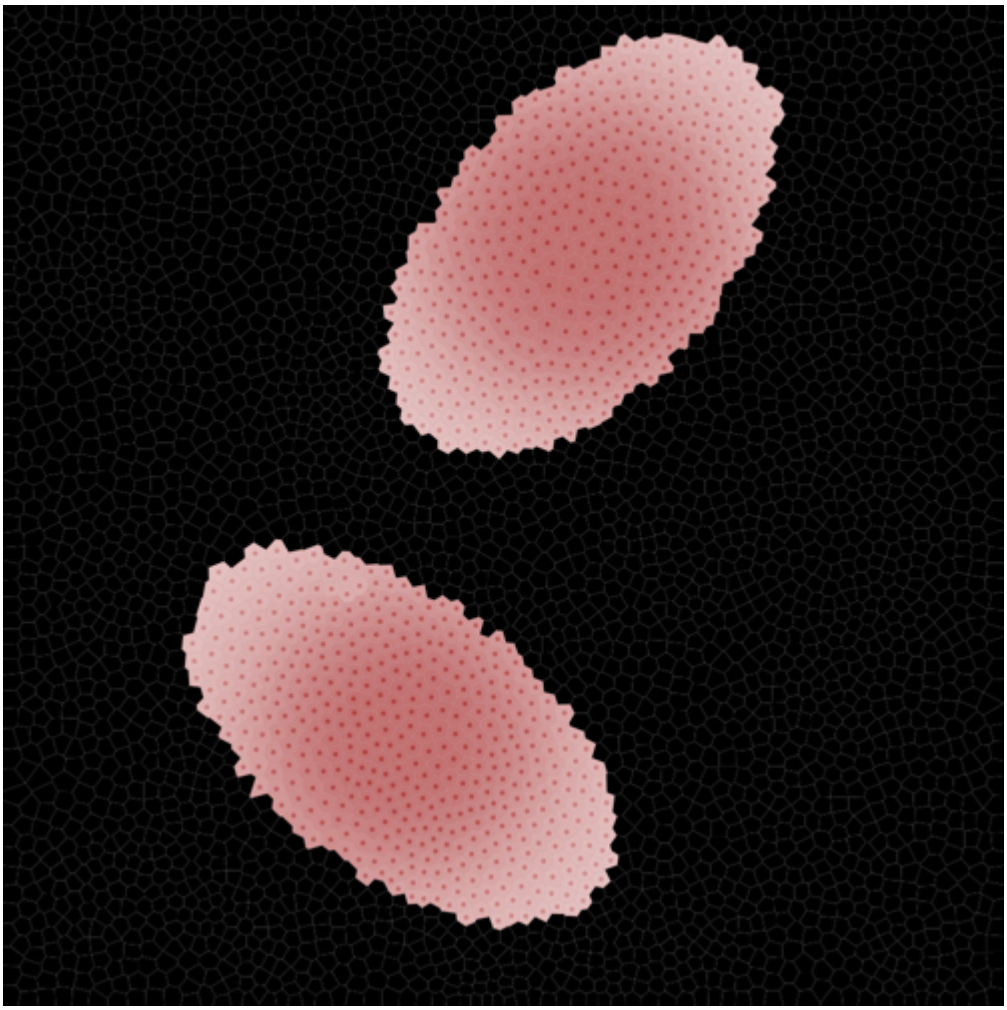
Ajouter une fonctionnalité qui trie les bonnes des mauvaises cartes serait une fonctionnalité que nous pourrions ajouter mais non sans difficultés. En effet, il faudrait déterminer une façon de trier les cartes. Cependant, nous avions eu l'idée d'ajouter à notre interface générateur, en plus de la fonction apply(), la fonction verify() qui vérifierait, de façon plus globale, l'aspect réaliste de notre carte.

Nous pouvons donc facilement placer les vérification, mais l'algorithme de vérification lui même serait la partie compliqué. Nous croyons tout de même que ce serait possible.

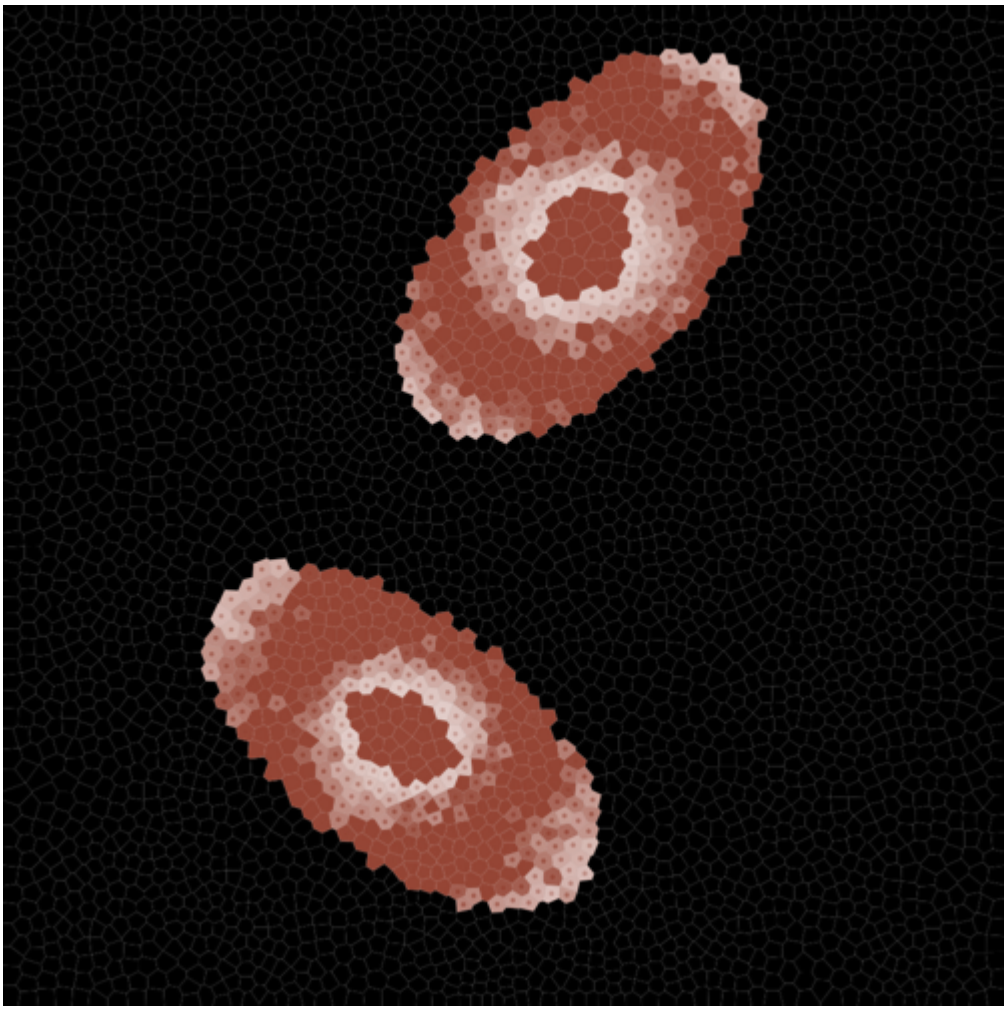
7 Annexe



Heatmap d'humidité de la carte présentée au début.



Heatmap d'altitude de la carte présentée au début.



Heatmap des ressources de la carte présentée au début.