Rapport TER

Zink Tom - Saperès Clément - Nigh Kai - Bonetti Timothée

april 2025

Table des matières

1 Introduction

L'objectif de ce projet est de réaliser un jeu vidéo massivement multijoueur sur navigateur. Celui ci se joue de deux façons bien distinctes :

- *Mode "Péon"* : Le joueur contrôle un personnage et s'occupe des actions élémentaires telles que récupérer des ressources, construire des bâtiments, etc.. Ces cations seronts fait a traves des mini-jeux simples.
- *Mode "Seigneur"* : Le joueur ne contrôle pas de personnage directement, mais a une vision plus globale et donne des ordres aux péons en fonction des besoins du royaume.

Il y aurait plusieurs royaumes gérés par des seigneurs différents, en concurrence pour devenir le royaume le plus puissant. L'idée est inspirée de la "pixel war", l'événement temporaire sur r/place de Reddit où tous les utilisateurs du site pouvaient dessiner sur une page blanche. Ce principe très simple avait créé des dynamiques de groupes autour de communautés d'Internet et de streamers notamment qui voulaient leur place sur ce tableau géant, et nous avions assisté à des guerres d'alliances et des trahisons. Là où le but de la pixel war était purement esthétique, notre jeu serait plus concurrentiel. L'objectif de notre projet est de recréer ces dynamiques de groupes dans les royaumes.
// Image Pixelwar (commentée) ca prend du temps a compiler
Ce projet peut se suffire a lui même mais il est fait en collaboration avec un autre groupe de TER qui a crée un Backend scallable pour accueillir des applications comme la notre. Nous allons donc lier nos projets pour déployer notre frontend sur leur backend.
Les défis que nous avons identifiés avant le debut du developement sont les suivants :

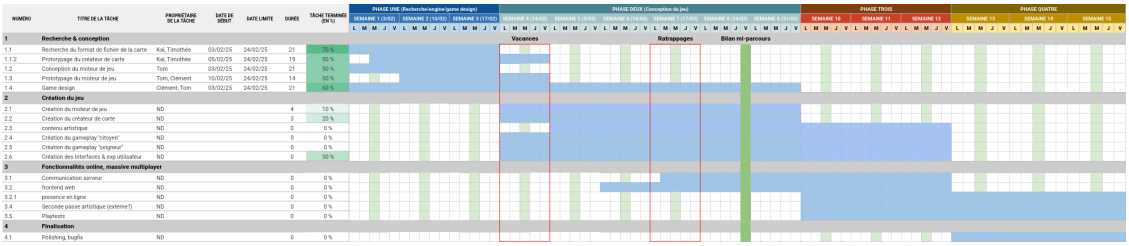
- Générer une carte procéduralement avec ressources.
- Choisir la bonne technologie pour faire tourner le projet sur un navigateur.
- Créer un moteur de jeu simple.
- Comment créer une interface utilisateur.
- Créer les minijeux.
- Créer et équilibrer les 2 façons de jouer différents.
- Gérer la communication avec le Backend.
- Créer (ou trouver) des assets.

trouver une problématique

2 Conception et recherches

2.1 Game design

2.2 Planification



NUMÉRO	TITRE DE LA TÂCHE	PROPRIÉTAIRE DE LA TÂCHE	DATE DE DÉBUT	DATE LIMITE	DURÉE	TÂCHE TERMINÉE (EN %)
1	**Recherche & conception**					
1.1	Recherche du format de fichier de la carte	Kai, Timothée	03/02/25	24/02/25	21	70 %
1.1.2	Prototypage du créateur de carte	Kai, Timothée	05/02/25	24/02/25	19	50 %
1.2	Conception du moteur de jeu	Tom	03/02/25	24/02/25	21	50 %
1.3	Prototypage du moteur de jeu	Tom, Clément	10/02/25	24/02/25	14	50 %
1.4	Game design	Clément, Tom	03/02/25	24/02/25	21	60 %
2	**Création du jeu**					
2.1	Création du moteur de jeu	ND			4	10 %
2.2	Création du créateur de carte	ND			3	20 %
2.3	contenu artistique	ND			0	0 %
2.4	Création du gameplay "citoyen"	ND			0	0 %
2.5	Création du gameplay "seigneur"	ND			0	0 %
2.6	Création des interfaces & exp utilisateur	ND			0	30 %
3	**Fonctionnalités online, massive multiplayer**					
3.1	Communication serveur	ND			0	0 %
3.2	frontend web	ND			0	0 %
3.2.1	presence en ligne	ND			0	0 %
3.4	Seconde passe artistique (externe?)	ND			0	0 %
3.5	Playtests	ND			0	0 %
4	**Finalisation**					
4.1	Polishing, bugfix	ND			0	0 %

FIGURE 1 – Diagramme de Gantt

2.3 Recherche documentaire

3 Developement

3.1 Génération de la carte

Pour ce projet nous voulions créer un outil pour créer une carte procéduralement. Pour cela nous avons écrit un programme en C++ qui crée un fichier .OBJ a partir de cartes de bruits.

Cartes de bruits Une carte de bruit, ou "noisemap", dans le contexte de la génération procédurale, est une technique utilisée pour créer des textures, des terrains ou des environnements de manière algorithmique. Les noisemaps dans la génération procédurale sont utilisées pour introduire des variations naturelles et réalistes dans les modèles générés par ordinateur.
La génération d'une noisemap implique l'utilisation de fonctions de bruit, telles que le bruit de Perlin, le bruit de Simplex ou d'autres algorithmes de bruit procédural. Ces fonctions génèrent des valeurs pseudo-aléatoires qui varient de manière continue et douce à travers l'espace. L'utilisation la plus elementaire de ce genre de cartes et de lier l'élévation du terrain aux valeurs de la carte de bruit.

FIGURE 2 – Visualisation de Carte de bruit

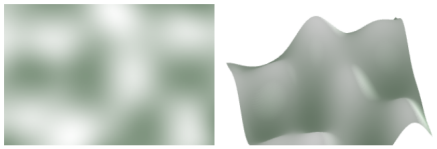
Nous pouvons manipuler l'aléatoire des fonctions qui generent ce genre de carte grace a differents parametres :

- La fréquence dans le contexte des fonctions de bruit, comme le bruit de Perlin, détermine la granularité ou la finesse des détails dans la carte de bruit générée. Une fréquence élevée signifie que les variations de bruit se produisent plus rapidement, ce qui entraîne des détails plus fins et plus nombreux dans la carte. À l'inverse, une fréquence basse produit des variations plus lentes et plus douces, résultant en des caractéristiques plus larges et moins détaillées.
- Les octaves sont utilisées pour ajouter de la complexité et du réalisme aux cartes de bruit. Chaque octave est une couche supplémentaire de bruit générée à une fréquence différente. En superposant plusieurs octaves, on peut créer des textures plus riches et plus variées. Les octaves sont généralement ajoutées avec des fréquences de plus en plus élevées et des amplitudes de plus en plus faibles.

Effet sur la Carte Générée

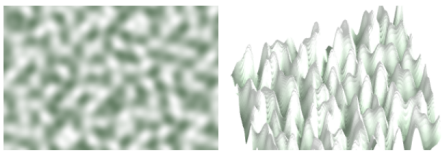
- *Fréquence Basse* : Une fréquence basse produit des caractéristiques larges et douces. Par exemple, dans la génération de terrains, une fréquence basse peut créer des collines et des vallées larges et douces.
- *Fréquence Élevée* : Une fréquence élevée produit des détails fins et nombreux. Par exemple, dans la génération de terrains, une fréquence élevée peut ajouter des rochers, des crevasses et d'autres détails fins à la surface.
- *Octaves Multiples* : L'utilisation de plusieurs octaves permet de combiner des caractéristiques larges et douces avec des détails fins. Par exemple, une première octave avec une fréquence basse peut créer les grandes formes du terrain, tandis que des octaves supplémentaires avec des fréquences plus élevées ajoutent des détails fins comme des rochers et des textures de surface.

Voici des exemples en images de la différence.



(a) Exemple carte et élévation avec une fréquence basse

image



(b) Exemple carte et élévation avec une haute fréquence

FIGURE 3 – Comparaison de niveau de fréquences

Génération de biomes Sur notre carte nous voulons différents biomes c'est a dire différents environnements repartiimages le plus naturellement possible. Pour cela en plus de la carte de bruit qui va gérer l'élévation du terrain nous allons générer une autre carte de bruit qui va simuler l'humidité du milieux. En combinant ces deux cartes on peut obtenir des biomes plutôt cohérents. En plus de cela nous pouvons définir une élévation minimale qui va permettre de créer des océans. Enfin nous avons défini une aire de jeu circulaire au milieu de la carte celle ci sera plate mais utilise quand même les noisemap pour gérer les biomes.

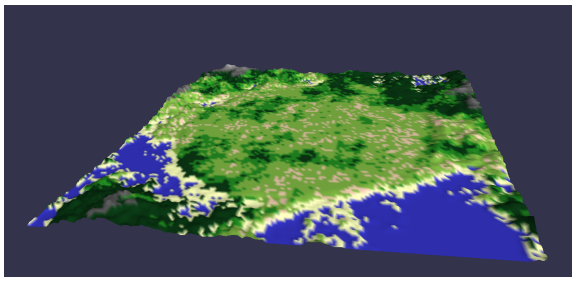
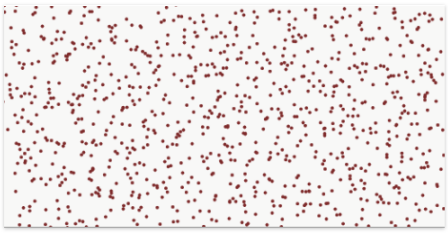


FIGURE 4 – Une carte générée avec notre code

Placer des ressources Un autre défi de la création de la carte est de placer des ressources sur la carte en favorisant les potentielles interaction entre les royaumes, c'est a dire créer des ressources qui serons rares et ne pas les répartir de manière uniformes pour obliger les royaumes a interagir entre eux. Si on prend simplement des positions aléatoires sur la carte par exemple on se retrouve avec un résultat trop uniforme on aimerait avoir une génération avec des trous et des endroits plus concentrés. La solution que nous avons trouvé pour cela ce sont les "Jittered grid" on peut traduire cela par "grille perturbée". Le principe est de générer une grille carrée ou hexagonales et d'appliquer une perturbation aux points de la grille. Voici le résultat.

(a) Grille avant la perturbation



(b) Gille après la perturbation

FIGURE 5 – Visualisation de jittered grid

On voit apparaître des zones avec peu de points et d'autres avec beaucoup de points.

Résultat sur la carte Les triangles violets représentent les positions des ressources.

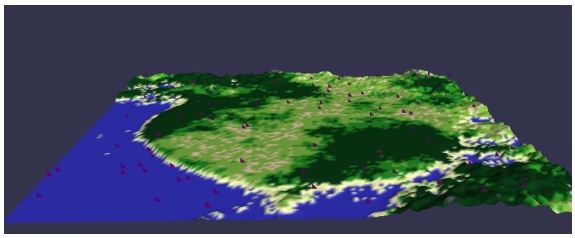


FIGURE 6 – Carte avec position de ressources générés

Il suffira de re-exécuter ce code le même nombre de fois que de ressources.