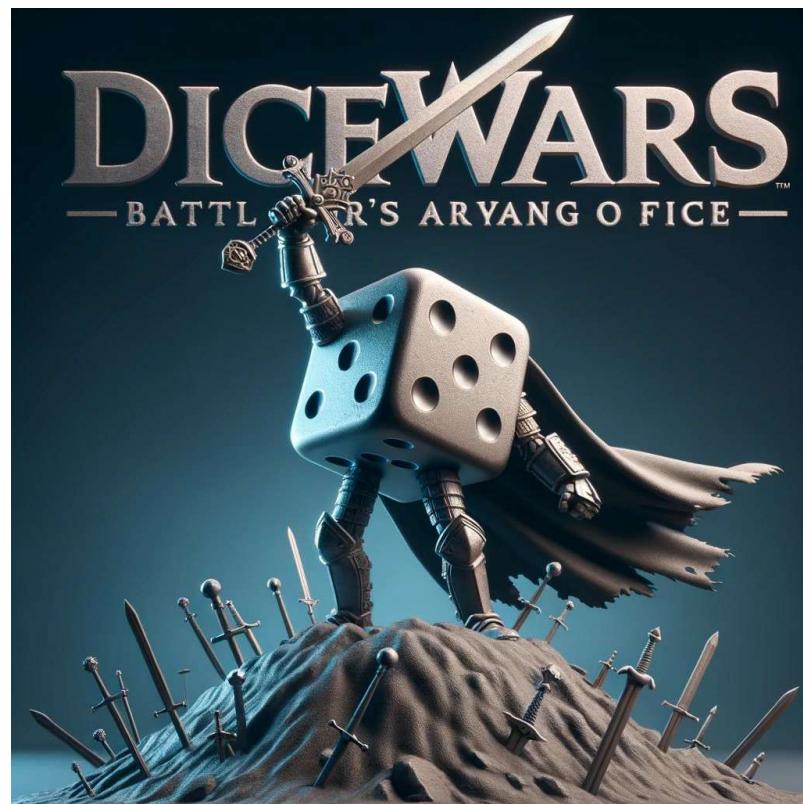


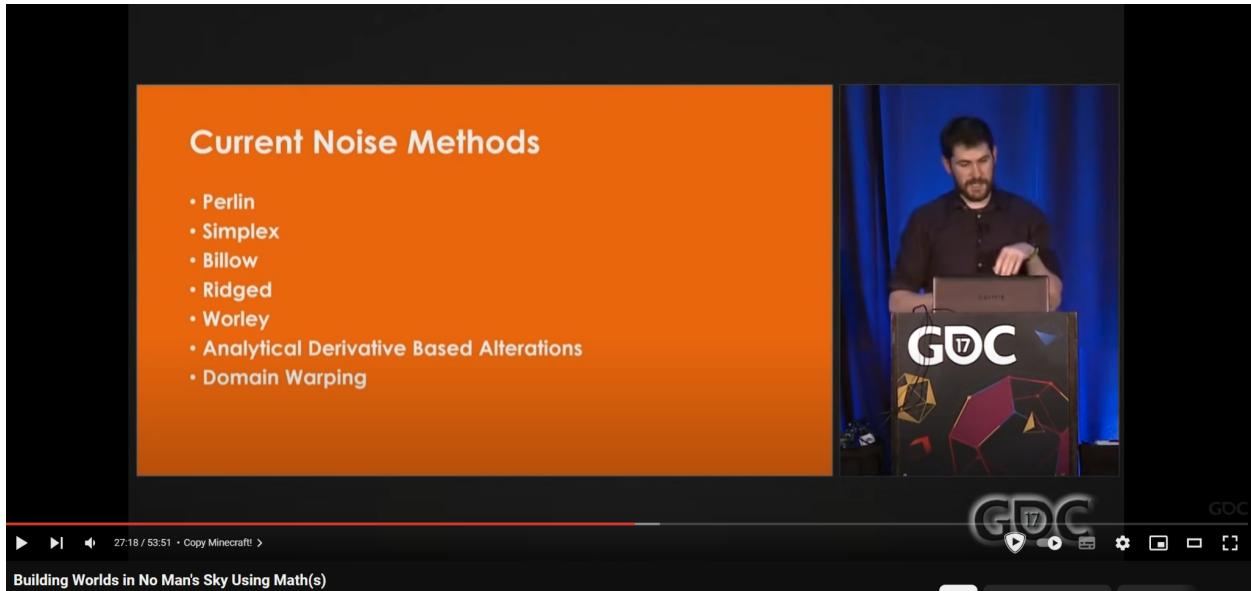
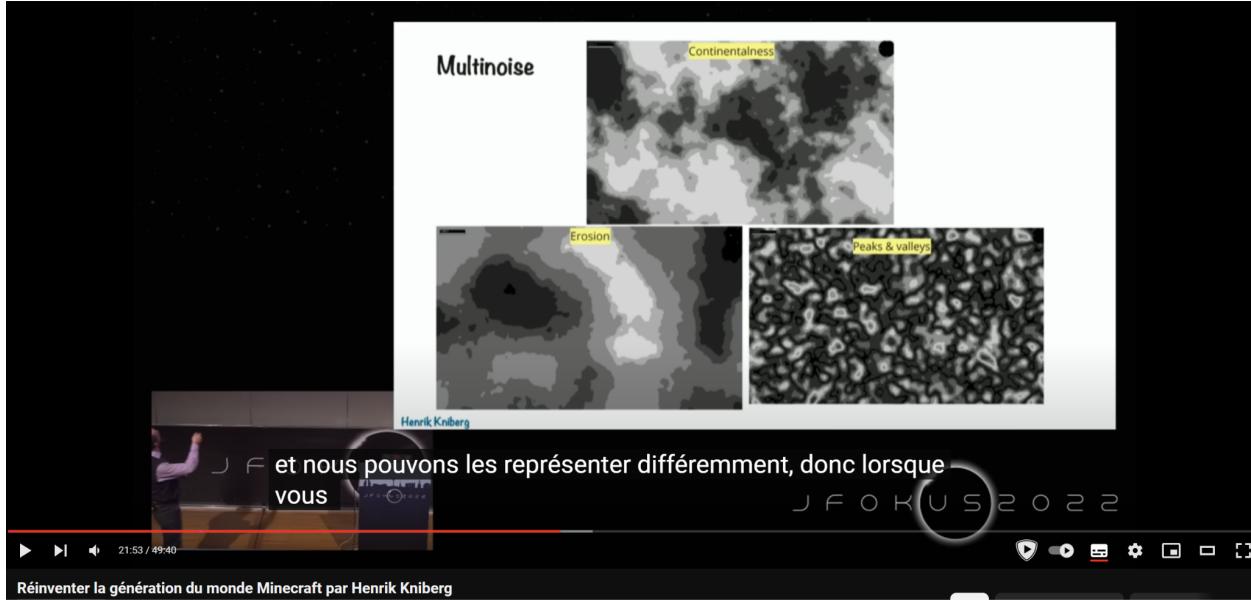
Rapport Mini-projet CPP



1. Génération de la carte

1. Inspiration et Méthodologie :

Pour la création de cartes dans 'Dice Wars', nous nous sommes inspirés des techniques utilisées dans des jeux tels que Minecraft et No Man's Sky.



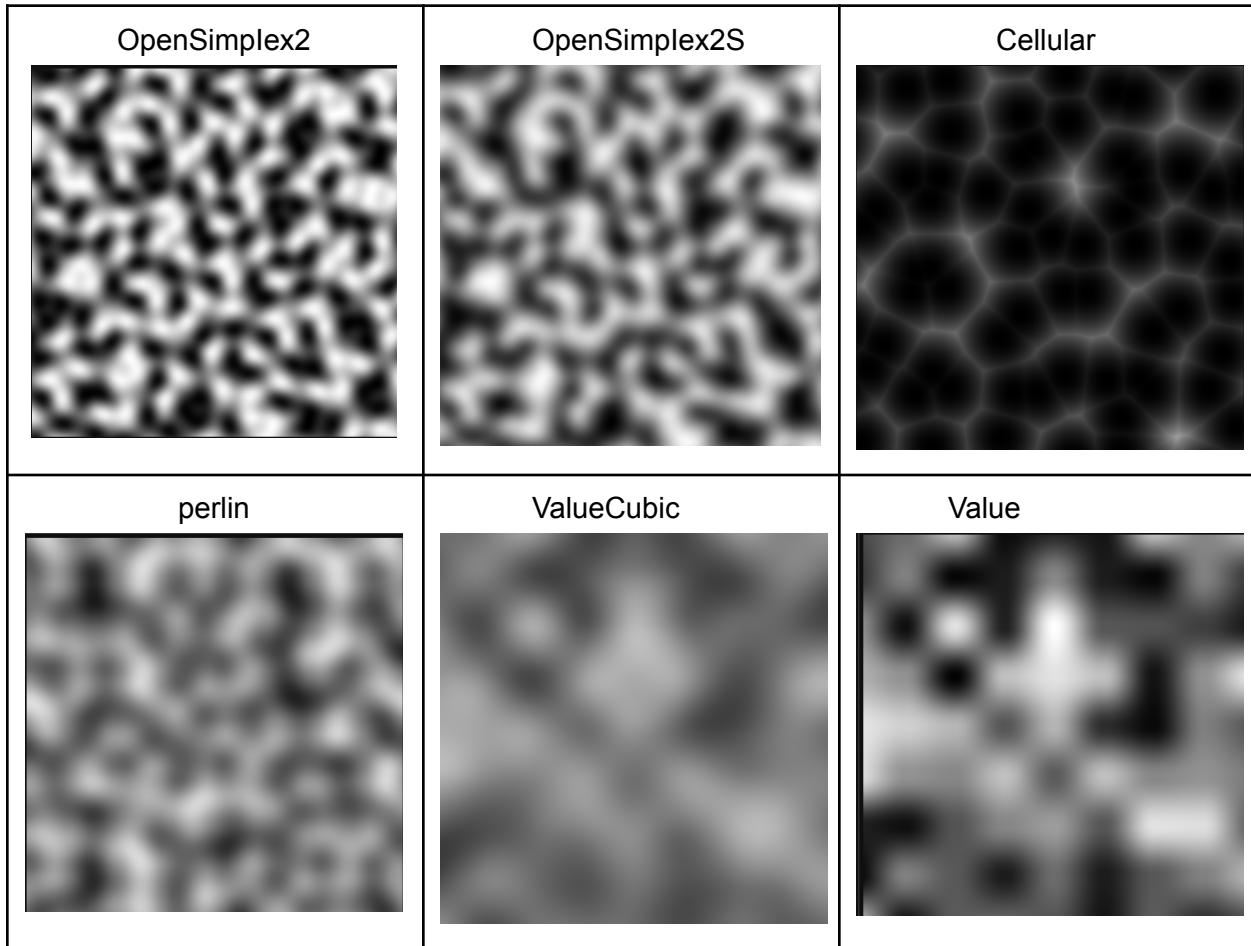
Ces jeux génèrent des mondes infinis en utilisant diverses méthodes de "bruit", permettant ainsi une variabilité contrôlée et réaliste.

Qu'est-ce que le bruit en général ?

Le bruit, dans le contexte de la génération de terrains ou de cartes, est une fonction mathématique qui produit des valeurs pseudo-aléatoires. Ces valeurs sont utilisées pour simuler des textures naturelles ou des motifs irréguliers, comme ceux que l'on trouve dans les paysages naturels.

2. Types de Bruit Utilisés :

Plusieurs types de bruit sont couramment utilisés dans les jeux vidéo, notamment :



Pour notre projet, le bruit "Cellular" s'est révélé être le plus efficace, car il permet de créer des régions bien délimitées, ce qui est essentiel pour le Dice Wars.

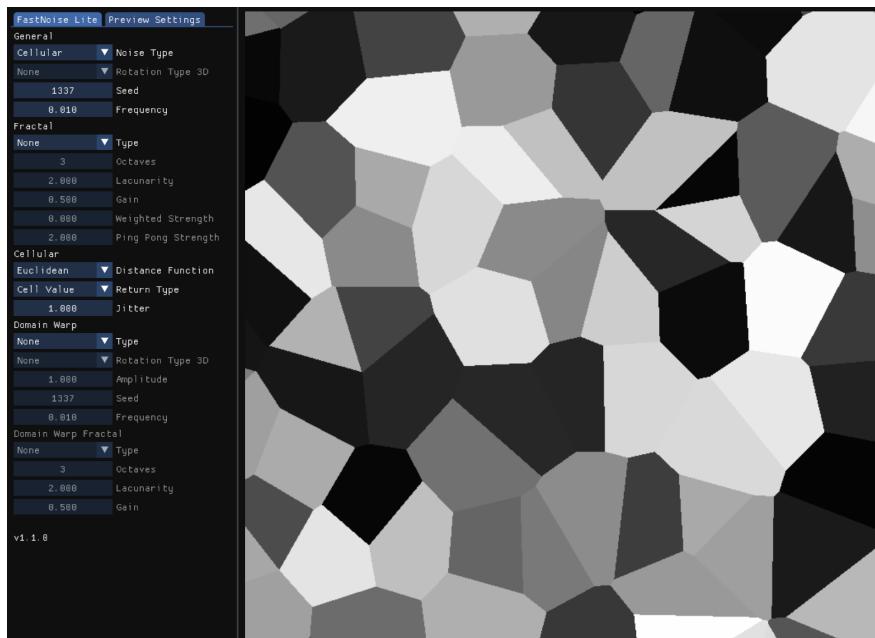
Le Bruit Cellular (aussi connu sous le nom de Worley Noise) est un type spécifique de bruit utilisé pour générer des textures qui ressemblent à des cellules ou des structures organiques. Il fonctionne en plaçant un ensemble de points (souvent appelés "sites cellulaires") dans un espace. Ensuite, pour chaque point de l'espace, la fonction de bruit calcule sa distance au site cellulaire le plus proche.

Le résultat est un motif qui ressemble à une série de cellules distinctes, chacune associée à un site cellulaire.

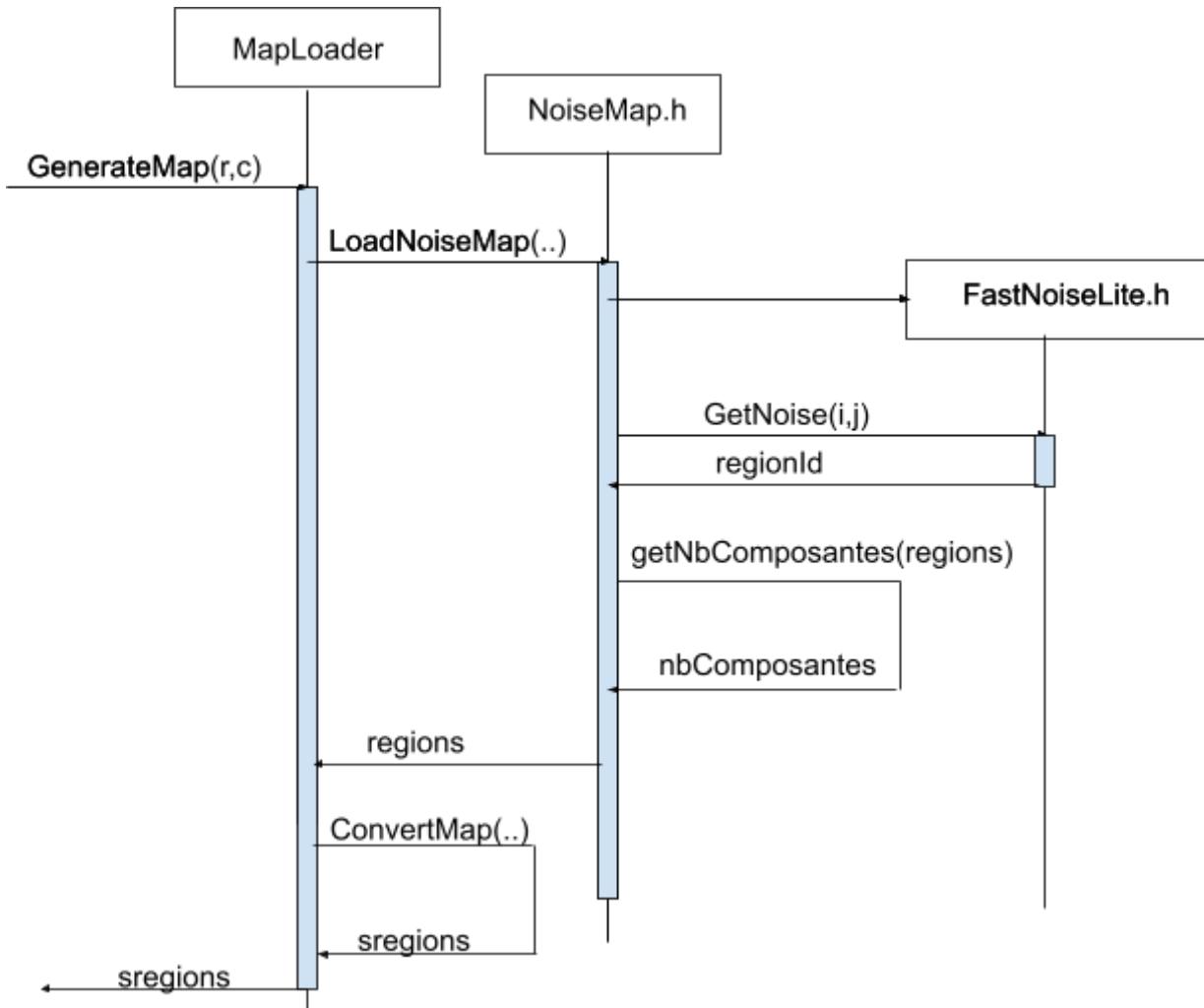
3. Implémentation via FastNoiseLite :

Nous avons utilisé la bibliothèque FastNoiseLite (disponible sur GitHub <https://github.com/Auburn/FastNoiseLite>) pour générer le bruit. Cette bibliothèque est simple à intégrer, car elle consiste en un seul fichier d'en-tête.

Un outil web fourni avec la bibliothèque (accessible sur FastNoiseLite Web <https://auburn.github.io/FastNoiseLite/>) facilite grandement le paramétrage du bruit. En configurant le bruit en mode "Cellular" et en choisissant le type de retour "CellValue", nous avons obtenu les résultats désirés.



4. Code d'Implémentation :



L'implémentation du bruit dans notre projet est relativement simple : en passant en paramètres les paramètres de bruit à la librairie

```

FastNoiseLite noise;
noise.SetNoiseType(FastNoiseLite::NoiseType_Cellular);
noise.SetFrequency(0.25f);
noise.SetSeed(rand() % 1000 + 1);
noise.SetCellularDistanceFunction(FastNoiseLite::CellularDistanceFunction_Manhattan);
noise.SetCellularReturnType(FastNoiseLite::CellularReturnType_CellValue);

```

On récupère un id de régions pour chaque cellule.

```
float* cellToRegionId = new float[r * c];
for (unsigned int x = 0; x < r; x++) {
    for (unsigned int y = 0; y < c; y++) {
        float n = noise.GetNoise((float)x, (float)y);
        cellToRegionId[(r * x) + y] = n;
    }
}
```

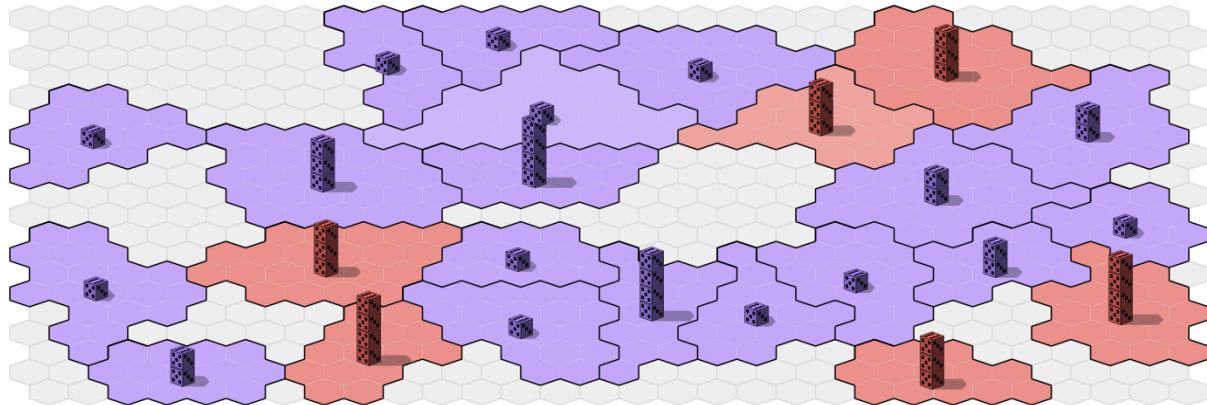
5. Optimisation et Finalisation :

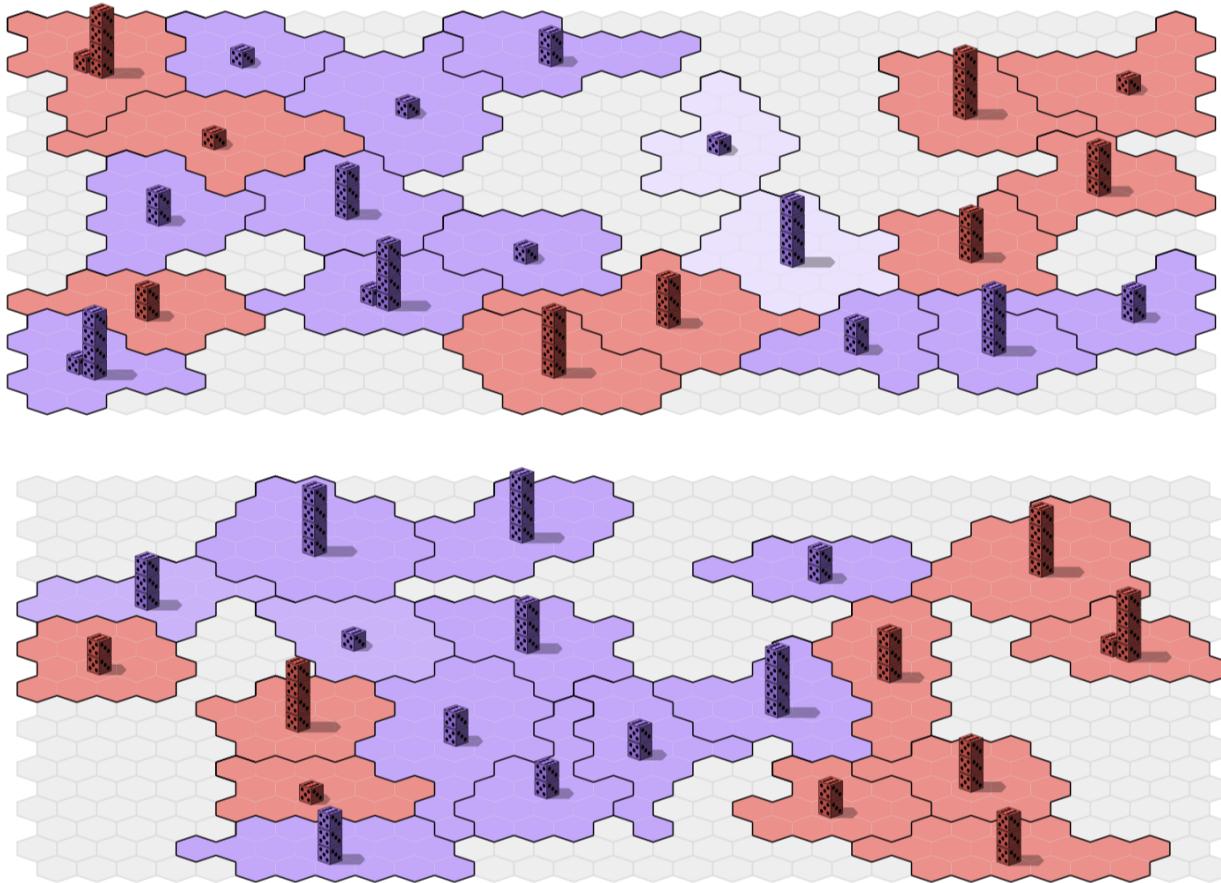
Nous avons également mis en place un algorithme pour calculer le nombre de composantes et éliminer les plus petites régions, souvent fragmentées par les limites du monde.

https://fr.wikipedia.org/wiki/Algorithmes_de_connexit%C3%A9_bas%C3%A9s_sur_des_pointeurs

6. Conclusion

Le résultat est très satisfaisant. Les cartes générées présentent des cellules de taille similaire, avec des formes variées et originales, offrant ainsi une expérience de jeu cohérente et immersive. Le code est aussi très simple à modifier et paramétrier pour modifier la forme de la carte





2. Stratégie

Concernant la stratégie nous avons d'abord essayé de réaliser la stratégie la plus simple possible qui était de prendre toute les case et dans celle qui était à nous, prendre la case avec le plus de dés qui à un voisin avec au moins un dé de moins.

Une fois cette stratégie mise en place correctement nous avons essayé d'en développer une autre plus abouti, pour cette stratégie le but était:

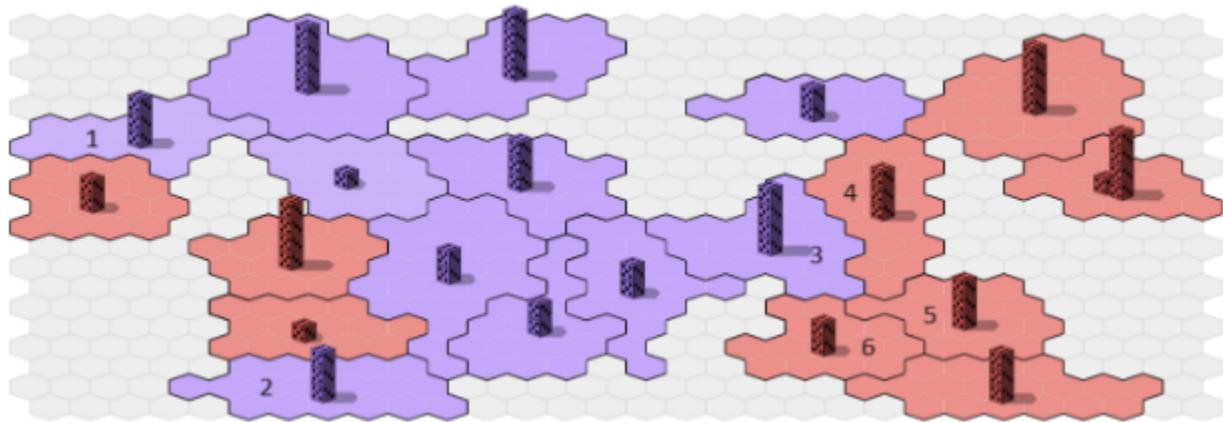
Attaquer si :

- Nombre de dés > nombre de dés adverse et si le dé qu'on laisse tout seul n'est pas en danger et si on ne se met pas en danger (cad avoir au moins autant de dés que les prochains/futurs voisins)
- Avoir une règle spéciale pour les 8, si nb dé = 8 attaquer
- Si on a le choix d'attaquer plusieurs zones, attaquer le plus gros d'abord
- Si on a le choix d'attaquer une même zone avec deux tours différentes choisir la plus grosse

Mais cette stratégie ne s'est pas avérée payante car les résultats n'étaient pas meilleurs que la stratégie précédente ainsi nous avons donc réfléchis à une autre stratégie. Pour la stratégie

finale nous avons décidé d'attaquer seulement si une tour à au moins 3 dés. Ainsi une case était considérée comme jouable si elle à au moins 3 dés et un voisin avec moins de dé ou si elle à 8 dés et dans ce cas là elle peut forcément attaquer. Ensuite pour déterminer laquelle de ces cases jouable allait être sélectionnée on choisit celle qui à le plus faible écart de dé avec son adversaire ainsi que le plus grand nombre de dés initialement . Cette méthode est celle qui nous a donné le meilleur taux de réussite

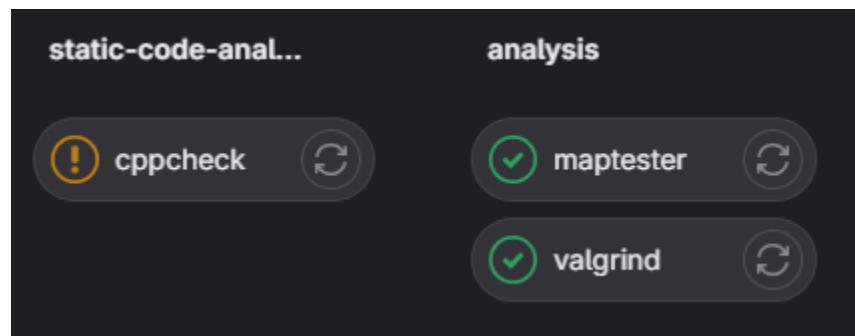
Ainsi sur la carte suivante si nous sommes violet, seul les cases numérotées 1 2 et 3 seront considérées comme jouable. Mais celle qui sera choisie est la 3 car peut attaquer avec une différence de 1 dé la case 4 ou la case 5 tout comme la case 1 mais la case 3 comporte plus de dé donc c'est elle qui sera sélectionnée ce tour ci.



Gestion de projet

Pipelines gitlab

Nous avons instauré un pipeline GitLab-CI pour évaluer le projet à l'aide de cppcheck et valgrind, en plus d'effectuer une étape de test sur le générateur de cartes pour 500 cartes. L'intégration continue nous assure un code valide tout au long du projet, simplifiant ainsi le rendu du code.



En amont, nous réalisons une analyse statique avec `cppcheck`, suivie d'une analyse dynamique avec `valgrind` et le `maptester` inclus dans le squelette de code.

Nous avons opté pour l'image Ubuntu plutôt que l'image Alpine (un système d'exploitation beaucoup plus léger, particulièrement adapté à l'intégration continue). Cependant, nous avons rencontré un problème lors de l'exécution de Valgrind et du maptester. C'est dû à une version plus ancienne de la bibliothèque standard du langage C sur Alpine par rapport à Ubuntu. Nous sommes donc restés sur `ubuntu:latest`.

Issues et branches

Durant l'évolution du projet, nous avons généré des issues pour chaque aspect à traiter, accompagnées de branches dédiées à chaque fonctionnalité ou ajout significatif. Nous avons cherché à émuler un processus professionnel similaire à celui de PTrans, garantissant ainsi une organisation structurée tout au long du développement, ce qui nous prépare efficacement au monde professionnel.

Voici une liste des issues que nous avons créées pour le projet :

- **Makefile pour vscode**
#9 · created 2 weeks ago by Thomas LEBRETON
- **Vérifier que le cpp check passe + valgrind**
#8 · created 3 weeks ago by Theo LE BAIL
- **Fonction pour connaître le "ratio Empty"**
#7 · created 3 weeks ago by Theo LE BAIL
- **Ajout d'une fonction de calcul du nombre de composantes connexes**
#6 · created 3 weeks ago by Theo LE BAIL
- **Update binaries**
#5 · created 3 weeks ago by Theo LE BAIL
- **Stratégie simple**
#4 · created 1 month ago by Theo LE BAIL
- **Étude des différentes stratégies**
#3 · created 1 month ago by Theo LE BAIL
- **Générateur de carte basique**
#2 · created 1 month ago by Theo LE BAIL
- **Intégration du modèle au projet**
#1 · created 1 month ago by Theo LE BAIL

Nous avons aussi créé plusieurs branches quand le besoin s'en faisait sentir :

The screenshot shows a detailed commit history from a Git repository. A vertical red line indicates the main 'master' branch, while several green lines represent other branches that have been merged into it. The commits are listed in chronological order, starting from the top:

- ci: use alpine
- fix: valgrind exit on error
- Ajout de la fct pour mettre à jour les voisins et avoir un meilleur ratio empty
- strategy
- update windows dll
- update bin maptester
- A simple strategy
- missing delete
- The big ONE - fix all problem
- Add function to know the ratio empty
- fix memory leak
- utilisation du nombre de composantes
- calcul du nombre de composantes connexes #6
- Merge branch 'rpi-map' into 'master'
- Fix noise map generator
- remove cached
- Add macos support
- Ref: build-essential
- ref: g++
- Fix: make and valgrind
- Merge branch 'master' of gitlab.univ-nantes.fr:E22B055P/labatailledude
- Ref: valgrind en CI