Génération Procédurale de Terrain

Diapo/Canva :

https://www.canva.com/design/DAGRxIcjRyU/xvylla_9ZwB9j0eJifwGxQ/edit?utm_content=DAGRxIcjRyU&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton

Github :

https://github.com/GamingCampus-MillieBourgois-24-25/programmation-avancee-destruction-procedurale

Ressources :

1. https://gamedevacademy.org/what-is-procedural-generation/
2. https://gamedevacademy.org/procedural-2d-maps-unity-tutorial/
3. https://www.baeldung.com/cs/gameplay-maps-procedural-generation
4. https://christianjmills.com/posts/procedural-map-generation-techniques-notes/
5. https://gamedevacademy.org/complete-guide-to-procedural-level-generation-in-unity-part-1/
6. https://hacdias.com/2023/06/24/procedural-map-generation-with-noise/
7. https://www.tomstephensondeveloper.co.uk/post/creating-simple-procedural-dungeon-generation
8. https://www.numerama.com/sciences/133517-la-generation-procedurale-ou-comment-le-jeu-video-devient-infini.html
9. https://game-ace.com/blog/procedural-generation-in-games/#:~:text=Games%20that%20use%20procedural%20generation,speeding%20up%20the%20development%20cycle.
10. https://bfnightly.bracketproductions.com/chapter_39.html
11. https://cochoy-jeremy.developpez.com/tutoriels/2d/introduction-bruit-perlin/
12. Série de vidéos de génération procédurale de Brackeys
13. https://www.numerama.com/sciences/133517-la-generation-procedurale-ou-comment-le-jeu-video-devient-infini.html

I. Définition :

La *génération procédurale* est la création de contenu (modèles, textures, sons, musiques, niveaux, etc.) de manière automatisée par un ou des algorithme(s). A l'aide d'informations, les algorithmes vont pouvoir créer du contenu. Dans les jeux vidéo, la génération procédurale sert de manière générale à créer un monde virtuel cohérent.

II. Jeux vidéo utilisant la génération procédurale de terrain :

- Minecraft
- Terraria
- No Man's Sky
- Starbound
- The Binding of Isaac
- Rogue Legacy
- Dead Cells
- Don't Starve
- Valheim
- Spelunky
- Hades
- Subnautica
- ARK: Survival Evolved
- Planet Nomads
- Celeste
- Factorio
- 7 Days to Die
- Oxygen Not Included
- Journey to the Savage Planet
- Dwarf Fortress
- Farming Simulator (certaines versions)
- Garry's Mod (avec des addons)
- Eldritch
- Axiom Verge
- Starbound
- RimWorld
- Worlds Adrift
- Caves of Qud
- Mordhau
- Core Keeper

III. Méthodes de génération procédurale :

** Présent dans la démo Unity

1. **Simple Room Placement****

- Algorithme :
    1. Démarrer avec une map solide (que des murs).
    2. Choisir un rectangle aléatoire pour une salle
    3. Si les rectangles ne sont pas superposés à des salles existantes, la tailler.
    4. Répéter les étapes 2-3 jusqu'à ce que le nombre de salles désirées soit placé.
    5. Connecter les salles avec des couloirs
- Dogleg Corridors : Une génération simple de couloirs qui alterne entre des segments verticaux et horizontaux.
- Possibilité d'utiliser d'autres algorithmes de couloirs existant.
- Limitations : Peut résulter en des salles peut espacées et un agencement inefficace des salles.

  Cette méthode est peu coûteuse en performance et est une des plus simples à programmer, mais sa simplicité limite son utilisation.

2. **Binary Space Partition (BSP) Rooms : Récursivement diviser l'espace en deux parties.****

- Algorithme :
    1. Diviser la map en deux (verticalement ou horizontalement).
    2. Récursivement diviser chaque partie jusqu'à ce que le nombre de salle désiré soit atteint..
    3. Optionnellement ajouter une bordure autour de chaque salle pour éviter qu'elles ne fusionnent.
- Benefices :
    - Meilleur espacement des salles comparé à la méthode Simple Room Placement.
- Limitations : Peut conduire à un agencement rectangulaire prévisible.

  Tout comme le Simple Room Placement, c'est une méthode simple qui possède ses propres faiblesses.

3. **Cellular Automata****

- Basé sur le principe du jeu de la vie de Conway.
- Algorithme :
    1. Initialiser la map aléatoirement avec des murs et des sols (ratio ~50/50).
    2. Itérer sur chaque case (sauf les bords) :
        - Compter le nombre de murs voisins (diagonales comprises).
        - Appliquer une règle baser le nombre de voisins :
            - 0 voisin: Devenir un mur.
            - 1-4 voisins : Devenir un sol.
            - 5+ voisins : Devenir un mur.

3. Répéter l'étape précédente pour un certain nombre d'itérations.

- Bénéfices :
    - Créer une map au look naturel, organique à partir d'un état initial aléatoire.
    - Algorithme simple et rapide.
    - Déterministe : La même seed produit la même map.
- Limitations : Peut être difficile de contrôler les formes spécifiques générées.

4. Drunkard's Walk

- Algorithme :
    1. Démarrer avec une map solide (que des murs).
    2. Placer un "ivrogne" (une entité) à un point aléatoire.
    3. L'ivrogne se déplace aléatoirement, créant un chemin en se déplaçant.
    4. Définir une distance maximum pour les déplacements de l'ivrogne pour qu'il s'arrête.
    5. Répéter les étapes 2-4, faire apparaître un nouvel ivrogne dans les l'espace ouvert (créé par les entités précédentes) jusqu'à ce qu'un pourcentage désiré de la map soit ouvert.
- Benefices:
    - Garanti une map contiguë (pas d'endroit inaccessible).
    - Créer une map qui ressemble à une formation naturelle comme des cavernes.
- Limitations: Peut provoquer des chemins inefficaces .

5. Diffusion-Limited Aggregation (DLA)**

- Algorithme :

1. Démarrer avec une map solide (que des murs).
2. Ajouter une "seed", un petit regroupement de cases de sol proche du centre de la map
3. Créer un marcheur sur la map qui bouge aléatoirement.
4. Déplacer le marcheur jusqu'à ce qu'il rentre en collision avec une case de sol.
5. Répéter les étapes 3-4 jusqu'à un certain nombre d'itération ou autre..

- Bénéfices :
    - Créer des espaces ouverts.
    - Garanti une map contiguë (pas d'endroit inaccessible).
- Symmetry: Appliquer de la symétrie peut créer une map plus structurées.

6. Voronoi Diagrams

- Algorithme :

1. Placer des points aléatoirement ou de manière délibérée à travers la map.
2. Pour chaque tuile de la carte, déterminer le point de seed le plus proche à l'aide d'un algorithme de distance (par exemple, Euclide, Manhattan, Chebyshev).
3. Assigner la tuile à la région appartenant à son point le plus proche.

- Bénéfices :
    - Créer des régions qui représentent des zones d'influence autour de chaque point de départ.
    - Peut être utilisé à diverses fins (par exemple, génération de villes, placement de monstres). (en référence à Apocalypse Taxi)
- Algorithmes de distance :

    - Euclidien (Pythagore): Ligne droite de distance standard, ce qui permet d'obtenir des bords lisses.
    - Manhattan: Distance mesurée comme la somme des pas horizontaux et verticaux, créant des bords nets, semblables à des grilles.
    - Chebyshev: Distance mesurée comme le maximum des distances horizontales et verticales, produisant un mélange entre Euclidien et Manhattan.
- Utilisations :
    - Placement des monstres basé sur les relations (par exemple, garder les alliés ensemble, séparer les ennemis).
    - Génération de villes (routes le long des bords, différentes régions pour différentes zones de la ville).

7. Perlin and Simplex Noise**

- Bruit de Perlin et bruit de Simplex : Algorithmes qui génèrent des valeurs continues, variant de manière régulière dans un espace.
- Propriétés :
    - Les valeurs de sortie sont généralement comprises entre -1 et 1 ou 0 et 1.
    - Les valeurs adjacentes sont liées de manière régulière.
    - Continu : un zoom sur une partie du bruit produit un modèle similaire à une échelle plus fine.
- Variables de lissage :
    - Octaves : Nombre de fonctions de bruit différentes mélangées ensemble, affectant les détails.
    - Gain : contribution de chaque octave au résultat final, affectant l'amplitude.
    - Lacunarité : Ajuste la fréquence de chaque octave, introduisant un caractère aléatoire et des détails.
    - Fréquence : La vitesse à laquelle les valeurs de bruit changent dans l'espace, ce qui affecte l'échelle des caractéristiques.
    - Persistance : Facteur venant modifier l'amplitude de chaque fonction de « bruit lissé »
- Utilisations :
    - Génération de l'overworld : Création de cartes de hauteur de terrain.
    - Génération de nuages.
    - Effets de particules.

8. Combining Techniques

IV. Pourquoi utiliser la génération procédurale ? :

- Peut réduire le temps et le coût associé au design des niveaux et du contenu.
- Permet aux développeurs de créer des espaces divers, variés et/ou de grandes tailles en peu de temps, accélérant donc le cycle de développement.
- Permet aux développeurs de concentrer leurs efforts sur d’autres aspects d’un jeu.
- Utilisation systématiques d’algorithmes qui crée des décors et de puzzles, assurant de la variabilité sans sacrifier la qualité des designs.

- Améliore la rejouabilité, l'environnement peut différer à chaque nouvelle session, permettant au joueur de vivre à chaque fois une nouvelle expérience.
- Rends le gameplay engageant et imprévisible.
- Contribue à la dynamique de gameplay avec par exemple des environnements imprévisibles.
- Provoque une variété de gameplay et de surprises pour les joueurs.
- L'usage de seed permet de reproduire l'aléatoire généré par la génération procédurale à des fins de test ou de compétition (eg. Speedrun).
- Permet de répondre à la demande pour des mondes complexes et immersifs.
- Consistance : Les processus automatisés garantissent que tous les éléments du jeu respectent des normes de qualité prédéterminées, ce qui réduit le besoin de correction.
- Flexibilité. Les modifications apportées à la conception du jeu peuvent être mises en œuvre rapidement en ajustant les algorithmes au lieu de retravailler manuellement de grandes parties d'un jeu.

V. Points négatifs :

- Peut produire de la répétitivité ou du contenu inintéressant. Peut être résolu avec de la variabilité dans les algorithmes de génération et/ou la combinaison avec des designs fait à la main (eg. Structures dans Minecraft).
- Peut aussi produire des bugs, de l'incohérence dans le gameplay ou du déséquilibre.
- Les mondes générés peuvent paraître fades (eg. Début de No Man's Sky).
- La génération peut être coûteuse en performance et prendre plus ou moins de temps selon les machines.

Démonstration : La Démo Unity