2015/2016

Roguelike

Rapport de projet

Jérémy Bourdé – Jules Wacquier

Encadrant : julien bernard

Table des matières

Remerciement 2

Qu’est-ce qu’un Roguelike ? 3

Notre Roguelike 4

Le projet

Background et gameplay

Modélisation du moteur de jeu

Les technologies utilisées

Réalisations

La map

Bruit de Perlin

Étagement altitudinal

Les obstacles

Le personnage

Les ennemies

Les objectifs

L’interface

Bilan

Remerciement

Nous tenons tout particulièrement à remercier Julien Bernard qui a proposé ce projet pour le soutient et le temps qu’il nous aura accordé.

Nous souhaitons également remercier Jean-Michel Hufflen pour sa gestion impartiale de l’attribution des sujets de projet.

Qu’est-ce qu’un Roguelike ?

Le genre Roguelike repose sur le gameplay du jeu Rogue qui est sorti en 1980. Ce jeu reprenait des mécanismes du jeu de rôle classique comme le gain d’expérience ou les attributs, tout en y injectant une grande dose d’aléatoire.

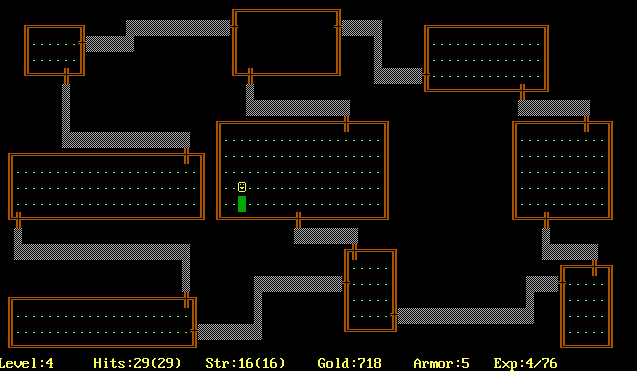


Figure 1 : Rogue, le jeu originel

Ce qui a beaucoup contribué à la démocratisation du genre et que le code source de Rogue a été largement distribué. On a ainsi très vite vu émerger de nombreux jeux gratuits similaire.

En 2008 à Berlin, la première International Roguelike Conference a tentée d’établir une définition du genre Roguelike à travers sept caractéristiques :

* Le joueur joue tour par tour.
* Tous les éléments du jeu sont placés sur une grille.
* Si le joueur échoue, il doit tout recommencer depuis le début.
* L’environnement est généré de manière procédurale et aléatoire à chaque nouvelle partie.
* Le résultat des actions de combat possède une part d’aléatoire avec des chances de critique ou d’échec.
* Le joueur peut stocker des items dans un inventaire.
* Le joueur n’est représenté que par un seul personnage dans le jeu.

Cependant, l’appellation est assez souple et tous les Roguelikes ne satisfont pas chacune des règles. Dans notre projet, la règle primordiale imposée par le sujet est la génération procédurale de l’environnement.

Notre Roguelike

Le sujet

L’objectif de notre projet est la familiarisation avec la programmation d’un jeu-vidéo à travers le C++ et la bibliothèque Simple and Fast Multimedia Library (SFML). Le sujet du projet est le développement d’un jeu vidéo de type Roguelike.

Les deux aspects d’un roguelike qui sont le plus exploités à travers ce projet sont la mort définitive du personnage qui entraîne un redémarrage à zéro lors de la partie suivante, ainsi que la génération aléatoire de l’environnement.

Pour cela, notre encadrant nous a demandé d’utiliser l’algorithme du bruit de Perlin pour générer une carte d’altitude. Puis nous avons dû créé un étagement longitudinal pour définir les différentes régions de la carte.

Background et Gameplay

Dans notre jeu, le joueur incarne un jeune druide sans défenses qui part à la recherche d’ingrédients pour ses potions à travers un monde remplis de chauve-souris particulièrement agressives.

Le joueur devra donc récupérer différents ingrédients dans les quantités qui lui sont indiquées aux début du jeu. A travers sa quête, il rencontrera des chauve-souris qui essayeront de le mordre qu’il devra absolument fuir !

Technologies utilisées

Le langage qui nous a été imposé qui nous a été imposé pour le développement du jeu est le C++. Cela nous a permis de revoir et d’améliorer les compétences acquises en Modélisation et Outils pour la Programmation au cours du semestre cinq de licence informatique.

Pour la partie graphique du jeu nous devions utiliser SFML (Simple And Fast Multimedia Library). C’est une bibliothèque libre de droit écrite en C++. Elles permettent de gérer facilement les parties graphique, audio et les évènements d’une application au sein de la même bibliothèque ce qui permet d’éviter d’avoir plusieurs modules à gérer ce qui entraine souvent des problèmes de compatibilité.

Elle possède également l’avantage de fonctionnée sur la plupart des systèmes d’exploitations Windows et Unix.

Pour le travail en équipe, nous avons utilisé Gitlab qui est un gestionnaire de version décentralisé. Cela nous a permis de travailler chacun de notre côté sur des éléments différents et de pouvoir assembler notre travail sans conflits de version. Cet outil nous a également permis d’éviter la perte de données car tout notre travail est sauvegardé sur un serveur et nous avons accès aux versions antérieurs.

Réalisations

La map

Pour la réalisation de notre carte générée aléatoirement nous avons combiner deux algorithmes. Tout d’abord, nous avons généré une carte d’altitudes avec l’algorithme du bruit de Perlin, aussi appelé bruit à base de gradients. Nous avons ensuite habillé cette carte grâce à un étagement longitudinal.

Le bruit de Perlin

Ken Perlin a créé l'algorithme du bruit de Perlin en 1985 par dans le but d'améliorer l'aspect des textures. Trois ans plus tôt, il avait travaillé sur les scènes en images de synthèse du film Tron et s'était retrouvé confronté aux limitations mémoire des ordinateurs de l'époque.

Il devait donc trouver une façon d'avoir des textures plus belles mais qui ne prenais pas plus d'espaces mémoire. Il eut donc l'idée de générer des textures par le calcul. Aujourd'hui, son algorithme est toujours utilisé pour générer certaines textures mais une autre utilisation lui a été trouvé : la génération de carte aléatoire.

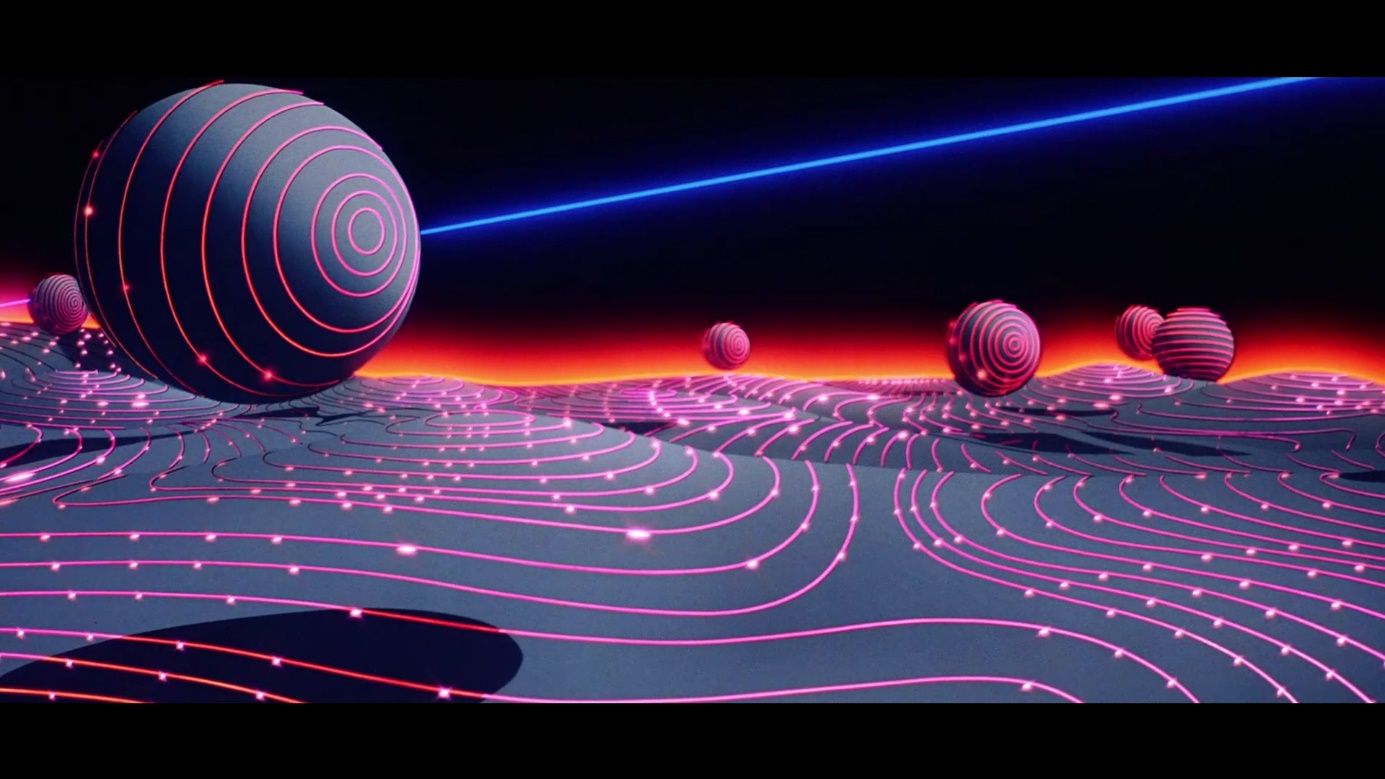


Figure 2 : Tron, 1982

Le bruit de Perlin et aussi appelé bruit à base de gradient, c’est-à-dire qu’il utilise le bruit à base de gradient.

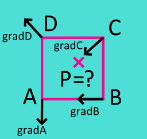
L’algorithme du bruit de Perlin possède deux grandes étapes :

* La génération d’un bruit à base de gradient.
* La combinaison de plusieurs bruits avec des amplitudes et des fréquences différentes.

Qu’est-ce qu’un bruit à base de gradient ?

Avec une fonction de bruit on cherche à avoir une table de valeurs qui nous servirons de carte d’altitudes. Le bruit à base de gradient est un bruit cohérent, c’est un bruit qui simule l’aléatoire. Ce genre de bruit est parfait pour la simulation de terrain car il va nous permettre de générer des terrains tous différents les uns des autres mais qui se ressemblent.

Il existe différent types de bruits cohérents mais le bruit de à base de gradient utilise des vecteurs pour générer des valeurs. En réalité, on utilise même quatre vecteurs pour calculer une valeur de la table.

Pour créer un bruit à bases de variant on a donc besoin d’un tableau de vecteur et d’une table de permutation qui ne contient que des valeurs d’index mélangées de manière totalement aléatoire. On va ensuite imaginer que chaque valeur est un carré avec un vecteur à chaque angle.

On va donc sélectionner quatre vecteurs dans notre tableau de vecteur. La sélection se fait de manière aléatoire car l’on passe par notre table de permutation pour choisir l’index du vecteur à utiliser pour chaque angle.

On fait ensuite le produit scalaire du vecteur à chaque point avec le vecteur entre ce point et le point central.

La dernière étape consiste à interpoler toutes ses valeurs pour obtenir la valeur finale. Pour cela on interpole ces valeurs deux par deux selon l’axe des abscisses, pois l’on interpole les valeurs obtenues selon l’axe des ordonnées.

Il existe plusieurs fonctions d’interpolation qui offrent des résultats plus ou moins lissés, mais pour ce projet nous avons utilisé une interpolation de type linéaire.

Comment ça se combine ?

Nous avons maintenant un bruit à base de Gradient en deux dimension.

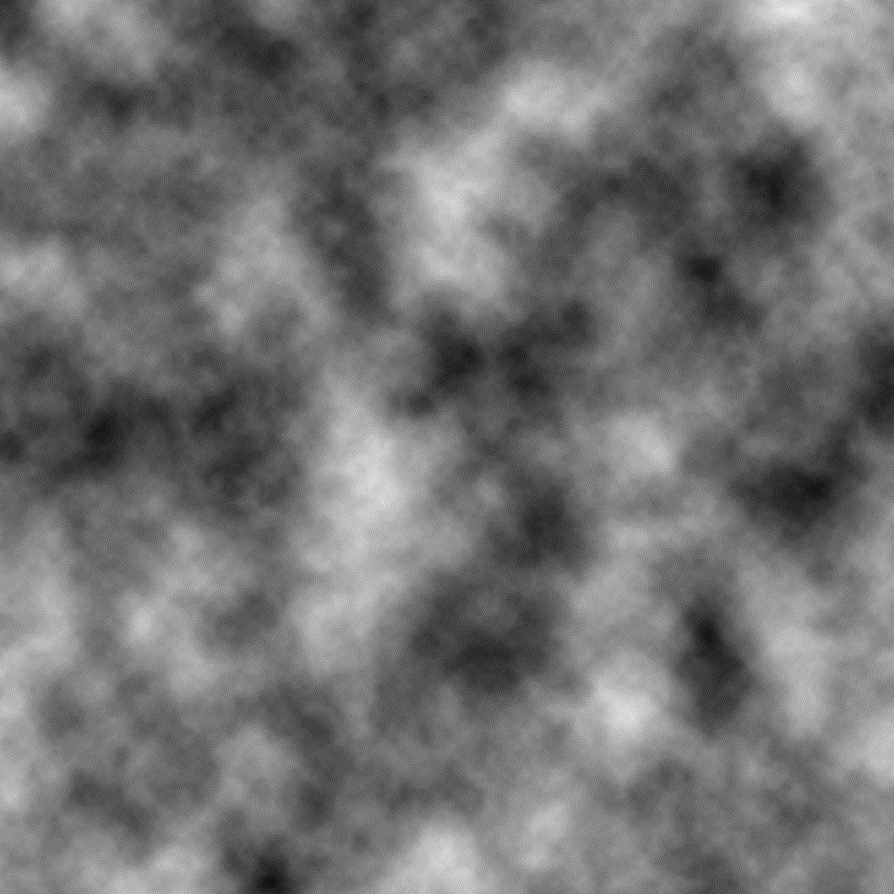


Figure 3 : Bruit à base de gradient simple

Nous allons maintenant combiner plusieurs bruits afin d’obtenir un résultat mieux réparti. Pour cela, il faut additionner plusieurs bruits tout en jouant sur la fréquence d’appel des bruits et sur l’amplitude de chaque bruit.

Entre chaque bruit, on multiplie la fréquence par deux et on divise par deux l’amplitude du bruit. Ces deux paramètres ont des impacts bien distinct sur le rendu final de notre bruit à base de gradients :

* La fréquence joue sur la diversification des valeurs. Plus la fréquence sera élevée, plus il y aura de de variation dans les valeurs obtenues.
* L’amplitude est l’écart entre les valeurs minimales et maximales. Plus l’amplitude est élevée, plus les valeurs seront diversifiées.

Ce type de procédure se retrouve dans différents types de bruits, qu’on qualifie de « fractal ».

Voilà à quoi ressemble notre bruit final.

Photo du bruit

Étagement altitudinal

Grâce au bruit de Perlin nous avons obtenus une carte d’altitude. Il faut maintenant établir un étagement longitudinal pour pouvoir l’habiller. Cela consiste à établir différents intervalles de valeurs. Chaque intervalle correspond à un environnement qui sera représenté par la même texture.

Nous avons donc créé une fonction qui prends en valeur

Image de l’étagement

Nous avons utilisé des if imbriqués pour « filtrer » les différentes valeurs de la table.