RAPPORT DE PROJET PSI

13/05/2007

Génération automatisée et aléatoire de maps intéressantes pour le jeu Tremulous

Arnaud DESCHAVANNE, Benoît LARROQUE, Cédric TESSIER

Suiveur : Hubert WASSNER

Rapport de projet PSI

GENERATION AUTOMATISEE ET ALEATOIRE DE MAPS INTERESSANTES POUR LE JEU TREMULOUS

SOMMAIRE

MOTS CLES & RESUMES

Mots Clés

- Génération de terrain fractale
- Placements d'objets
- 3D
- Mathématiques
- C++
- Compilation

Keywords

- Fractal terrain generation
- Object laying
- 3D
- Mathematics
- C++
- Compilation

Résumé

Le but du projet est de générer des environnements de jeu (appelés maps) pour le FPS[1] Tremulous. La génération est complètement automatisée, et produit des maps intéressantes à jouer. L'ensemble du projet représente plus de 2000 lignes de C++. Nous avons pu le démarrer dans le cadre du PSI mais nous prévoyons de le continuer sur notre temps libre.

Le projet a été construit en plusieurs phases. Première phase : la génération de terrain qui est accomplie par un algorithme récursif utilisant des propriétés fractales. Deuxième phase : placement d'objets sur le terrain généré par la première phase en utilisant notamment des algorithmes probabilistes mimant un placement naturel (pour les forêts, par exemple). Troisième phase : génération de chemins et de leurs représentations physiques associées (couloirs, cavernes).

[1] FPS : First Person Shooter, Jeu de tir à la en vue première personne

La programmation s'est accompagnée de réalisation de matériel de communication pour expliquer le concept aux utilisateurs du jeu.

Summary

Our project aims to generate Tremulous' maps. This generation is fully automated and the produced maps are interesting to play with. Up to day, our project is composed of more than 2000 lines of C++. The Engineer Science Project enabled us to get started with the project. It will be continued during our free periods.

The work was discomposed in steps. Step one: implementing a recursive algorithm for terrain generation using a fractal method. Step two: laying objects on the generated terrain using probabilistic algorithms which can be nature like (for forests). Step three: paths creation and building the related coves and corridors in the maps.

In the mean time, we've produced videos to explain the projects to the game users.

INTRODUCTION

L'industrie du jeu vidéo est un des secteurs de l'informatique qui génère le plus de profit (7,1 milliard de dollars en 2005). De fait, le secteur est fortement concurrentiel et les produits qui marchent sont de très haute qualité. Pour maintenir à la tête du marché les créateurs de jeu vidéo sont obligés d'innover en permanence.

C'est aussi un secteur qui a une grande diffusion auprès du grand publique. Il est très facile de trouver des gens potentiellement intéressés par un projet dans le domaine.

Les raisons qui nous on poussé à choisir ce sujet sont nombreuses. Tout d'abord ce sujet est l'écho d'une discussion que nous avions eu l'année précédente avec notre suiveur, Hubert WASSNER. Ensuite, nous voulions travailler, cette année encore, sur un sujet innovant et demandant une technique évoluée. Enfin, à la différence de l'an passé, nous voulions travailler sur un sujet qui pouvait avoir un fort retentissement.

REFLEXIONS A PARTIR DU SUJET

Le sujet, génération automatisée et aléatoire de maps intéressantes pour le jeu Tremulous, comporte plusieurs parties que il convient d'aborder.

Génération automatisée

La génération d'environnements de jeu est un domaine particulier de l'industrie du jeu que l'on appelle : mapping. Cette génération est normalement effectuée sur un modeleur.

Il s'agit, pour nous, de générer ces environnements de façon automatique, i.e. à l'aide d'un programme.

Génération aléatoire

Ce programme ne doit pas avoir exécution linéaire et complètement prédictive. Chaque environnement de jeu doit être différent

Maps intéressantes

Cependant le générateur doit être biaisé pour donner des cartes qui ont un intérêt pour les joueurs.

Tremulous

Tremulous est un FPS stratégique. Il y a donc deux aspect fondamentaux à prendre en compte.

Le moteur de jeu est un descendant de celui de Quake 3 il faut donc générer un environnement tridimensionnel de jeu.

Dans Tremulous, il ya deux équipes qui s'affrontent, les humains et les aliens. Chaque équipe démarre avec une base, et a pour but de détruire la base adverse. De plus, dans chaque équipe il existe une classe de personnage qui peut construire et donc déplacer la base, c'est ce qui fourni le coté stratégique au jeu.

ETUDE PRELIMINAIRES A LA REALISATION

Les environnements du jeu Tremulous ne sont pas générés à la volé. Ils sont écrit, le plus souvent par le modeleur, dans un format texte et ensuite compilé par un compilateur qui a été fourni par les créateur de Quake 3. Nous allons aborder ici la forme des différents fichiers. Cependant nous discuterons d'abord de la réflexion sur le coté aléatoire et le choix du langage.

Aléatoire

L'aléatoire en informatique est un vaste sujet. En effet il est théoriquement impossible de générer quelque chose de purement aléatoire à partir d'un système déterministe tel qu'un ordinateur. Cependant il existe des algorithmes mathématiques qui renvoient des valeurs pseudo-aléatoires. En fonction de l'algorithme la valeur renvoyé est plus où moins sûre (non prévisible). Pour le projet nous n'avons pas besoin d'un générateur d'aléas sécurisé, nous utilisons donc simplement la fonction rand fournit par le système.

Choix du langage

Le langage à été une de nos premières réflexions. Pendant les tests de faisabilité nous programmions en Ruby car c'est un langage de haut niveau et d'une grande souplesse (issue des méthodes agiles). Pour le produit final nous sommes revenus au C++ qui est le langage qui donne les meilleures performances tout en étant bien adapté à la problématique.

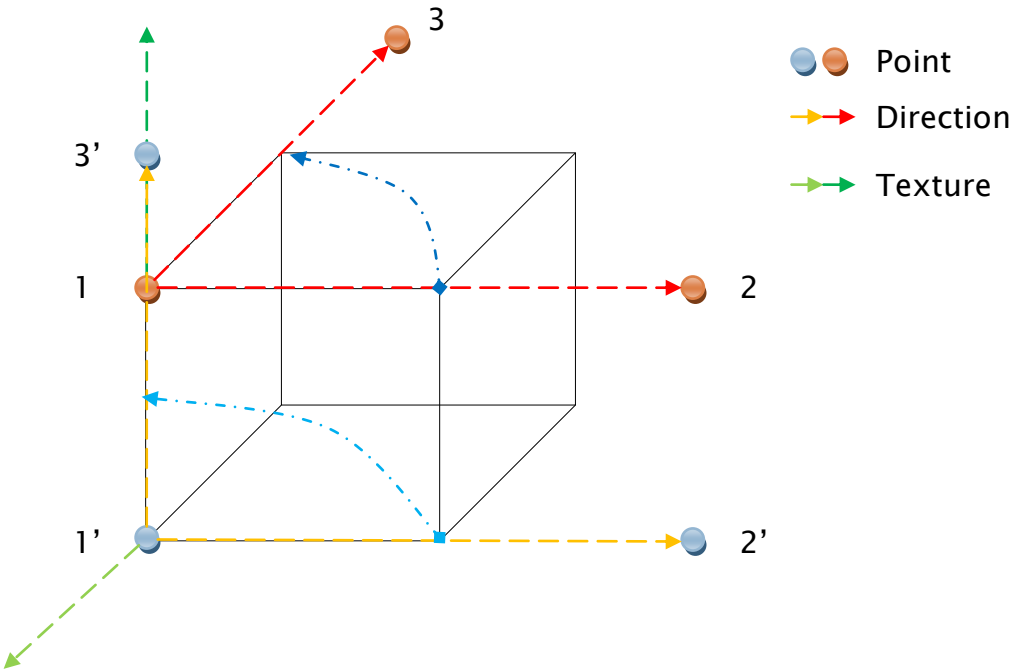
Le format des maps

Les maps qui sont reconnues par le jeu doivent obéir à un certain format, d'extension pk3. Les fichiers pk3 sont en fait des zip d'un dossier ayant une certaine architecture. Il y a en effet plusieurs fichiers/dossiers dont voilà les utilisations

Le dossier map contient les fichiers bsp (Binary Space Partition) qui décrivent l'environnement physique. Ces fichiers sont le résultat de la compilation des fichiers au format map que nous gêneront plus particulièrement.

Le format map

Le format map est un format texte. Un fichier à ce format est composé d'un ensemble d'entités qui peuvent avoir des paramètres. Une entité est composée de ses attributs ainsi que de se description physique. La description physique d'une entité est une union d'intersection de plans. Un plan est définit par trois triplet des coordonnées (points) sont le premier doit appartenir au plan représenté. L'ordre des deux autres points donne le coté duquel sera visible la texture appliquée à la face (produit vectoriel entre les vecteurs $\overrightarrow{(1,2)}$ et $\overrightarrow{(1,3)}$.



Après chaque plan, il y a une série de nombre et une chaine de caractère qui correspondent au shader/texture à appliquer au plan ainsi que des rotations ou homothéties à lui faire subir.

Cette définition des formes est très générique et permet de rendre dans le jeu à peu prêt n'importe qu'elle objet tridimensionnel. Elle n'est cependant pas aisée à comprendre et relativement peu documenté, si bien qu'au début du projet il nous a été plus simple de retro-ingénieuré le format que de fouiller l'Internet.

FIGURE 1 : DECOMPOSITION DE DEUX PLANS SUR UN SOLIDE QUI EN COMPTE SIX

```
{
{
( 2700 300 1312 ) ( 2700 0 1670 ) ( 2400 300 1203 ) example2/ter_rock_mud 32 16
144 0 0 13.8222 3.48546 11.1248
( 2400 0 1500 ) ( 2400 300 1203 ) ( 2700 0 1670 ) example2/ter_rock_mud 64 64 12
0 0 0 12.6466 18.8398 2.10653
( 2700 300 1312 ) ( 2400 300 1203 ) ( 2700 300 1302 ) common/caulk 0 0 0 0 0 0 0
0
( 2700 300 1312 ) ( 2700 300 1302 ) ( 2700 0 1670 ) common/caulk 0 0 0 0 0 0 0 0
( 2400 0 1193 ) ( 2700 0 1193 ) ( 2400 300 1193 ) common/caulk 0 0 0 0 0 0 0 0
( 2400 0 1490 ) ( 2400 0 1500 ) ( 2700 0 1660 ) common/caulk 0 0 0 0 0 0 0 0
( 2400 0 1490 ) ( 2400 300 1193 ) ( 2400 0 1500 ) common/caulk 0 0 0 0 0 0 0 0
}
{
( 3000 300 1371 ) ( 3000 0 1697 ) ( 2700 300 1312 ) example2/ter_rock_mud 16 16
28 0 0 12.066 1.24327 18.0548
( 3000 300 1371 ) ( 2700 300 1312 ) ( 3000 300 1361 ) common/caulk 0 0 0 0 0 0 0
 0
( 3000 300 1371 ) ( 3000 300 1361 ) ( 3000 0 1697 ) common/caulk 0 0 0 0 0 0 0 0
( 3000 300 1361 ) ( 2700 300 1302 ) ( 3000 0 1687 ) common/caulk 0 0 0 0 0 0 0 0
( 2700 300 57 ) ( 3000 0 1697 ) ( 3000 0 1687 ) common/caulk 0 0 0 0 0 0 0 0
}
}
```

FIGURE 2 EXEMPLE DE PARTIE PHYSIQUE D'ENTITE

A la compilation, le compilateur reproduit en mémoire les zones solides et les zones vides établissant un partitionnement de l'espace. A partir de ce partitionnement le compilateur prévoit quelle seront les zone visible de chaque point afin de simplifier la tâche du moteur graphique du jeu. Evidement les objets tels que les plantes et décors sont du détail et ne modifie pas la visibilité des zones qui sont derrières eux.

Cependant si on ne fait pas la différence entre ces détails et les objets structurant le calcul se complexifie grandement. Ainsi pour des raisons de performance une entité peut ne pas avoir de partie physique mais inclure des modèles tridimensionnels d'objets. Ceux-ci sont alors placés via des attributs (position, angle) et peuvent être déformés.

```
{
"classname" "misc_model"
"origin" "4702.75 2836.26 834.446"
"model" "models/mapobjects/ctftree/ctftree2.md3"
"spawnflags" "2"
"modelscale" "2.40433"
"angle" "192.95"
}
```

FIGURE 3 : EXEMPLE D'ENTITE SANS PARTIE PHYSIQUE

Le format shader

PLANIFICATION DE LA REALISATION

FIGURE 4 : ORGANISATION DE LA REALISATION

Nous avons choisi de réaliser le projet en plusieurs phases telles qu'indiquées sur le schéma ci-dessus.

1. La génération d'un terrain sur lequel les joueurs puissent se promener sans risque ainsi que la création de lacs qui peuvent servir de repères et de cachettes. Le tout bien sûr en essayant de texturer le terrain au mieux.

2. Le placement d'objets tels que des arbres selon un schéma qui fasse un semblant de sens pour le joueur (par exemple regrouper arbres en les forêts).

3. La création de couloirs et caves pour donner au jeu une nouvelle dimension. Ces couloirs ayant la capacité de raccorder des pièces on peu imaginer de raccorder deux « terrains » entre eux.

De manière coordonnée avec l'avancement du projet nous avions planifié de communiquer sur le projet vers l'extérieur de l'école (notamment au moyen de vidéos). Nous avions tablé sur la complétion de la première phase ainsi qu'une partie de la deuxième phase pendant le temps impartit au PSI. Le complétion finale du projet se fera pendant notre temps libre via esiea-labs[2]

[2] Association d'élève dont le slogan est : « La recherche et développement pour le plaisir »

REALISATION

Phase un : génération de terrain

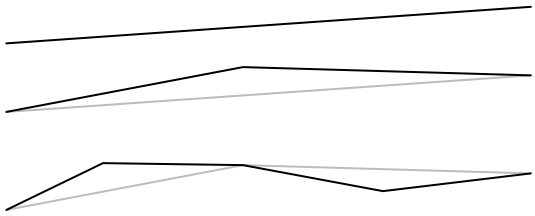


FIGURE 5 : EXEMPLE DE SUBDIVIDE & DISPLACE

La première phase est la génération de terrain jouable. Cette génération est effectuée à l'aide d'un algorithme fractale, subdivide & displace. Cet algorithme appelé aussi « random midpoint displacement algorithm » est assez simple et donne d'excellents résultats. On part d'une surface plane, le point centrale est élevé d'une certaine valeur. On obtient alors une surface brisée, il suffit ensuite de relancer l'algorithme sur les sous-surfaces qui sont planes.

Cette génération effectuée nous avions donc en mémoire une carte des hauteurs du terrain (heightmap) qu'il nous fallait donc ensuite transformer effectivement en terrain dans le jeu.

On la vu au dessus pour obtenir une forme dans le jeu il faut la décrire sous forme d'une réunion d'intersection de plans. Plusieurs versions se sont succédées.

Dans la première version nous lisions directement l'heightmap pour la traduire en parallélépipèdes rectangles accolés Ainsi pour une heightmap de taille n par n on obtient n^2 parallélépipèdes.

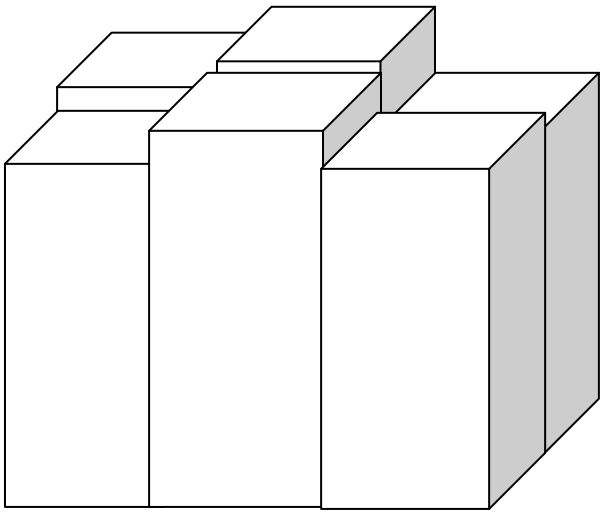


FIGURE 6 : PREMIERE VERSION DU TERRAIN

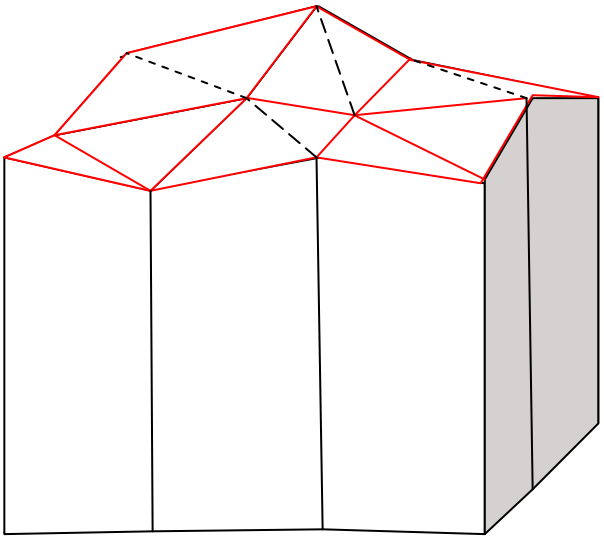
Cette version bien que pouvant sembler triviale à réalisée n'a pas été des plus simple à réalisé, en effet nous découvrions le format en même temps

La deuxième version du terrain a été obtenue suite à un changement de méthodologie. L'idée cette fois est de dire qu'un terrain 3D est un objet qui se représente bien par le collage de triangle rectangle. Ainsi au lieu de prendre chaque point de la heightmap et d'en faire un solide on a besoin de trois point de

la heightmap qui donnent la hauteur de chaque point du triangle. En réutilisant les points on arrive ainsi à un terrain continu et assez réaliste.

Cette interprétation mène cependant à générer un terrain comportant (nombre de formes) :

$$(2(n-1))^2 = 4 * (n^2 - 2n + 1) = 4n^2 - 8n + 4$$



Ici il y a 8 blocs au lieu de 12

FIGURE 7 : TROISIEME METHODE DE GENERATION

Ce qui est supérieur bien supérieur à n^2 pour les valeurs de n que nous utilisons (entre 10 et 100)

Cependant le format n'impose pas de limite au nombre de plan dont on calcule l'intersection. Donc pour les solides convexes nous avons pu fusionner les solides adjacents. Ce qui est la troisième de la génération du terrain physique à partir de l'heightmap

Enfin, afin de faciliter le travail pour la troisième phase nous avons rendu le terrain moins massif en enlevant les volumes plein qui était dessous. Il a cependant fallut veiller à ne pas rendre les briques convexes concaves. Cette version est la dernière et la plus aboutie

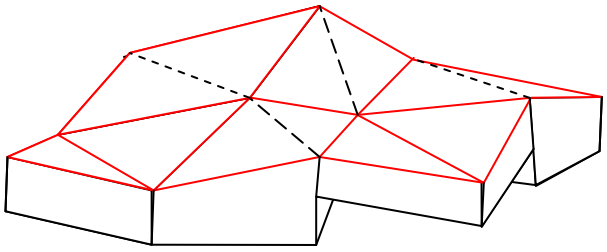


FIGURE 8 : DERNIERE VERSION DU GENERATEUR

Une fois le terrain généré nous avons ensuite créé un algorithme qui place un lac sur la carte. Cet algorithme est récursif et est lancer sur les points d'altitudes minimum de la carte (les hauteurs sont discrétisées). A partir de son point de départ l'algorithme cherche la plus grande zone telle que le volume de terrain placé à l'intérieur soit une cuvette. Lorsque l'algorithme s'arrête, on place un parallélépipède aux points qu'il renvoi ayant pour texture water

Phase deux : placements d'objets

A la fin de la première phase nous avions donc un terrain texturé et ayant une forme plus où moins naturel suivant les différents paramètres des algorithmes. Nous avons pu constater que le terrain était solide en la parcourant dans tous les sens. Au cours de cette phase de tests nous sommes bien rendu compte qu'il est très facile de se perdre sur un tel terrain sans aucun repère visuel autre que le relief.

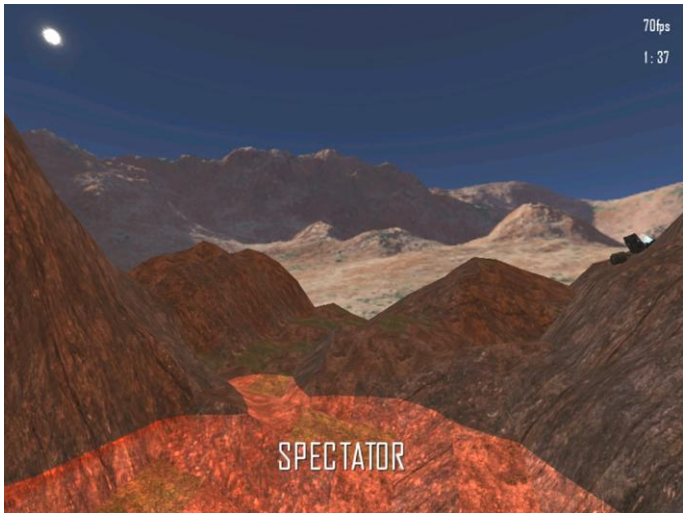


FIGURE 9 : TERRAIN GENERE (COMPILE EN LUMIERE)

Nous pouvions donc passer à la partie placement d'objet afin d'avoir des repère sur la carte.

Pour pouvoir tester, nous avions déjà du nous intéresser à la représentation au format map des objets (comme vu au paragraphe « Le format map ») dans la première phase de développement. En effet le but du jeu étant de détruire une des bases, celui-ci s'arrête dès qu'une base absente. Il nous avait donc fallut placer un minimum d'objet sur la carte. Ce positionnement était cependant fixe à la différence de ce qui a été réalisé en phase 2.

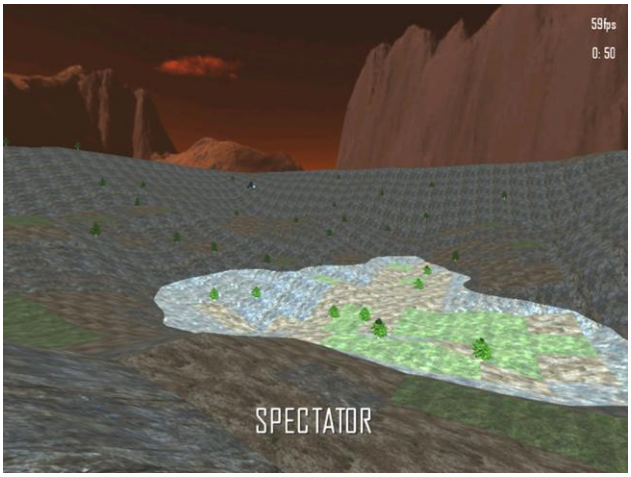


FIGURE 10 : PREMIER PLACEMENT D'ARBRES

Après réflexion, la meilleure façon de rendre le terrain encore plus naturel et de rajouter dans le même temps des repères visuels est d'y placer des arbres. Nous avons créé un premier algorithme distribuant aléatoirement des arbres sur la carte.

Evidement les premiers résultats étaient assez moyens, en effet les arbres n'étaient placés qu'à des positions entière de la heightmap soit donc aux points de jointure des blocs de terrain. Il en ressortait une impression d'alignement des arbres tout à fait artificielle. De plus certain arbres arrivaient au milieu de l'eau…

Nous avons donc retravaillé l'algorithme pour qu'il place les arbres à des coordonnée non entière de la heightmap et en vérifiant que le point est hors de l'eau. Pour ce faire il nous a fallut stocker une carte des lacs et faire une approximation linéaire bidimensionnelle de la position des plans de terrain. En effet en phase un, nous nous contention de donner des séries de point sans jamais nous inquiéter des équations des plan ainsi formés. Pour la phase deux, il nous fallait pouvoir donner l'altitude de n'importe qu'elle point sur la carte.

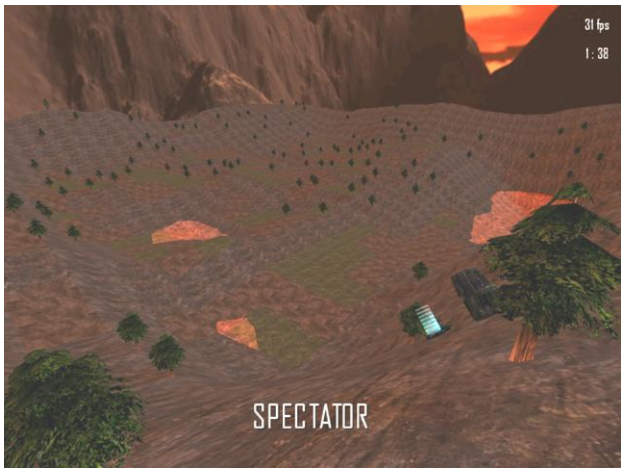


FIGURE 11 : DEUXIEME PLACEMENT D'ARBRES

Le rendu visuel est cette fois meilleur, mais les arbres sont encore trop épars. Nous avons donc, sous la suggestion, de Monsieur WASSNER, créé un algorithme de placement de fôret. Le fonctionnement de celui-ci est assez simple.

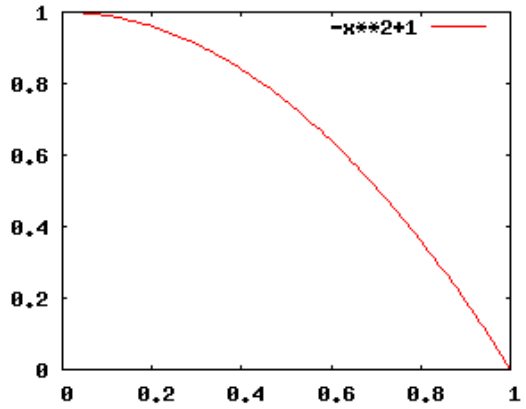


FIGURE 12 : REPARTITION AU CENTRE DE LA FORET

On choisit au hasard (algorithme de la deuxième méthode) la position du centre C de la fôret. On tire ensuite deux coordonnées polaires (r,θ) au hasard avec $r = -k * rand(0,1)^2 + 1$. On transcrit ces coordonées polaires de centre C dans la base cartésienne de la map et on vérifie que l'arbre ne soit ni dans l'eau ni hors du terrain, si tel n'est pas le cas on retire (r,θ). On place ensuite l'arbre à l'altitude requise.

De plus à chaque nouvelle itération il y 5% de chance d'aller fonder une nouvelle forêt avec la moitié des arbres restant à placer. Enfin pour éviter d'avoir des fôret trop dense, si on est trop souvent amener à retirer des (r,θ) (1000 fois de suite) on refonde une nouvelle forêt avec tous les arbres à placer ; ce cas arrive par exemple quand la forêt est placée trop proche d'un bord de la carte.

Le rendu final est très bon, les forêts générées sont suffisamment dense pour que les aliens puissent se cacher à loisir dedans mais pas trop pour que les humains puissent encore les traverser.

Après le placement d'arbres il ne nous restait plus qu'a placer d'autres objets pour obtenir des endroits où s'abriter pendant le jeu. Nous avons donc réutilisé le deuxième algorithme de placement des arbres pour répatir des caisses sur le terrain.

Après avoir placé ces objets décoratifs nous sommes revenus sur le code de départ. En effet à ce stade les bases étaient encore placées à deux angles du terrain, ce qui n'était pas trop en accord avec le coté « aléatoire ». Afin de placer ces bases nous avons donc développé un autre algorithme prenant en compte la spécificité des deux classes. En effet, les bases aliens sont plus défendables dans des zones de faibles volumes (cavernes par exemple), les humains sont pour leur par avantagé en terrain où ils peuvent voir les aliens arrivé de loin.

Ainsi nous avons écrit un algorithme qui place la base alien dans un endroit de basse altitude ressemblant le plus possible à une cuvette cette position est évalué via une fonction de mesure. On cherche ensuite une position de même valeur de mesure pour la fonction de mesure de type humain. Cette dernière prend en compte les critères d'une bonne base humaine mais aussi l'éloignement à la base alien qui doit être le plus grand possible. Enfin il a fallut faire quelques ajustements pour empêcher, par exemple, que les arbres poussent au milieu des bases.

Phase trois : couloirs

A ce stade nous avons un environnement de jeu inscrit dans un cube. Le cube contient, un terrain sur lequel sont placés des objets. Lidée en phase trois est de donner une autre dimension au jeu. L'environnement après ne devrait plus être cubique mais centré sur un cube ou encore un réseau de cubes.

Pour ceci il nous faut créer des couloirs qui s'accordent avec la zone de jeu. Il nous donc réaliser des couloirs de type humain, aliens voire terrain et plus important savoir bien les relier à la zone de jeu.

Les couloirs de type humain sont les plus simples à faire, en effet ils se composent de surfaces parallélépipédiques. Faire de tels couloirs est assez simple et créer des salles du même type aussi. Les liaisons inter couloirs se font à angle droit. Un prototype créant ce type de couloirs existe déjà.

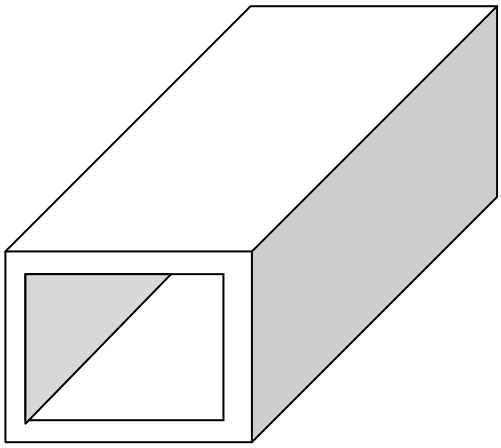


FIGURE 13 : COULOIR DE TYPE HUMAIN

Les couloirs de type alien doivent, quant à eux, ressembler à des cavernes. L'algorithme de génération sera donc plus compliqué. Nous avons pensé à une solution implémentant un algorithme récursif qui génère untel type de couloirs. Cet algorithme fonctionne en séparant chaque couloir en plusieurs sections successives.

Pour commencer on prend au hasard N point sur un cercle centré au point central du début du couloir et de rayon la taille voulu. On va ensuite reprendre une série de N±2 points sur un cercle placé à L (taille d'une section du cercle) et orienté sur un plan quasi parallèle et presque de même taille. On relit ensuite les

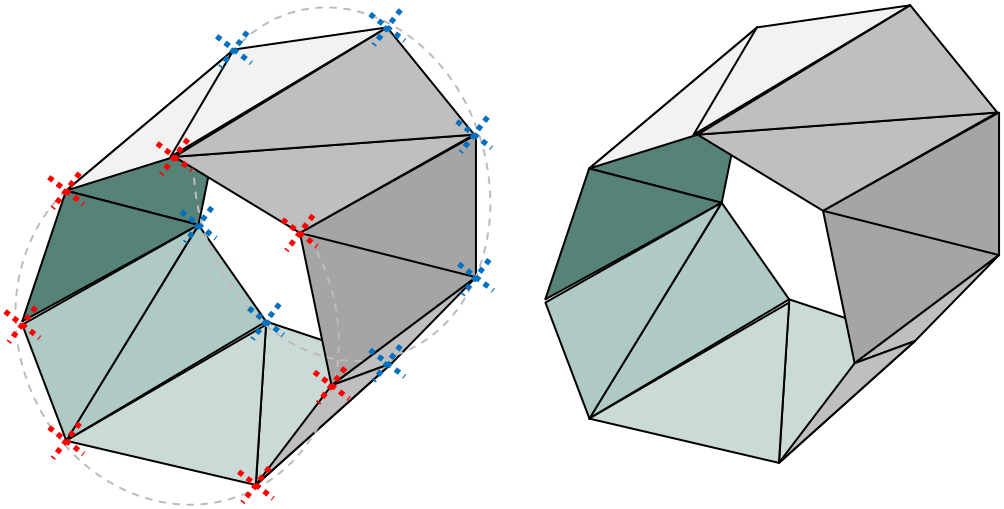


FIGURE 14 : EXEMPLE DE SECTION DE COULOIR ALIEN (SIMPLIFIE)

point entre les deux cercle par des plans (en fait des volumes pour créer un

plancher solide) on reprend le travail sur la section d'après en prenant le deuxième jeu de point comme départ. On devrait obtenir ainsi des couloirs irrégulier ressemblant, nous le pensons à des cavernes mais ayant des caractéristiques de volume contrôlées (pour éviter de désavantager des humains par exemple).

Le dernier type de couloir serait un type qui permettrait de relier ensemble deux zones de terrains pour donner l'illusion de la continuité du terrain sur une plus grande taille. Ces couloirs reprennent le principe de création de terrain pour des zones allongées avec sur les coté des surface verticales de type canyon pour justifier le rétrécissement de la zone de jeu.

Le dernier défi pour cette partie est de relier les couloirs avec les salles. Il faut donc établir des jonctions qui « collent » avec le type de salle/couloir concerné. On ne traitera que les cas des couloirs humains et aliens. Pour relier ces types de couloirs aux salles nous pensons définir une interface commune entre salle et couloir. Le principe serait de dire qu'un couloir se raccorde à un plan de mur sur une surface carré de taille N. Ce carré est enlevé du mur et le couloir va pouvoir installer ces propres décors. Par exemple une porte pour les humains et une cassure dans le mur pour les aliens.

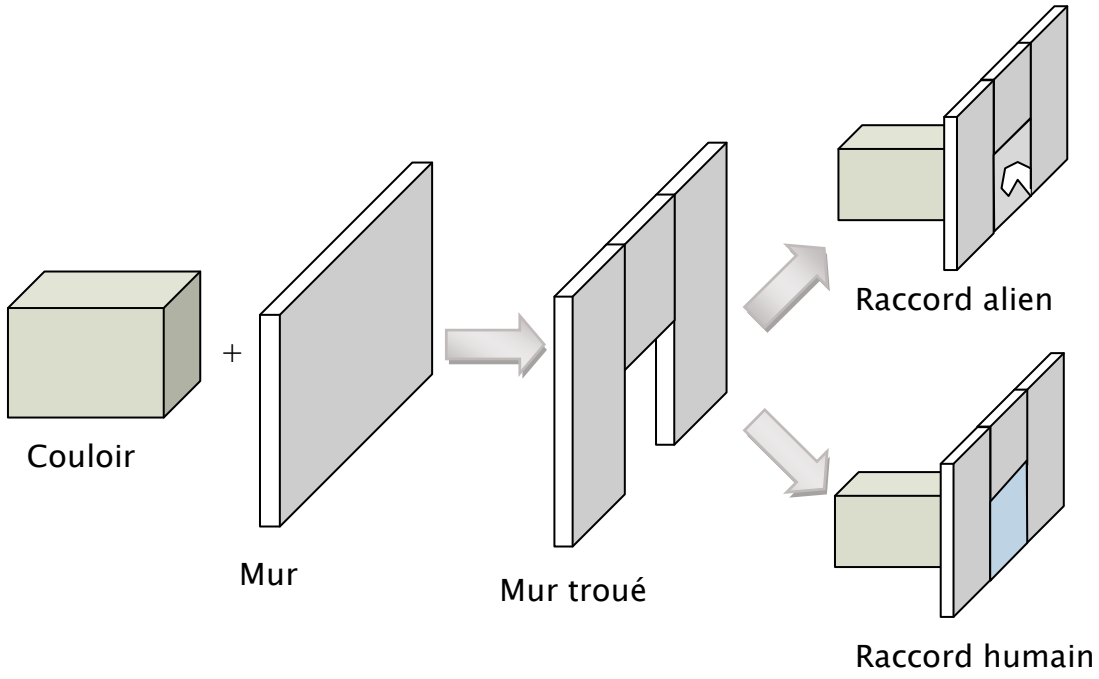


FIGURE 15 : PRINCIPE DU RACCORDEMENT DES COULOIRS AUX SALLES

Retour sur la compilation des maps

COMMUNICATION

CONCLUSION