RAPPORT DE PROJET PSI

28/05/2007

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
- Placement 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 première personne », en Français : jeu de tir subjectif (cf. Wikipédia).

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 project to the game users.

INTRODUCTION

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

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

Les raisons qui nous ont poussés à choisir ce sujet sont nombreuses. Tout d'abord, ce sujet est l'écho d'une discussion que nous avions eue, l'année précédente, avec notre suiveur, Hubert WASSNER. Ensuite, nous voulions explorer, cette année encore, un sujet innovant et demandant une technique évoluée. Enfin, à la différence de l'an passé, nous voulions travailler sur un thème 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 qu'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 une 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 aspects 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 personnages qui peut construire et donc déplacer la base, c'est ce qui fournit le côté stratégique au jeu.

ETUDE PRELIMINAIRE A LA REALISATION

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

Générateur d'aléas

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 renvoyée est plus ou 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* fournie par le système.

Choix du langage

Le langage a 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 (langage objet).

Le format des maps

Les maps qui sont reconnues par le jeu doivent obéir à un certain format, d'extension pk3. Les pk3, nommés comme cela à cause de leur extension de fichier, ne sont en fait que des archives contenant les fichiers d'une map.

Étant encodés en zip, un format de compression inventé par Phil Katz aujourd'hui très répandu, les pk3 sont réalisables avec n'importe quel archiveur présent sur nos systèmes d'exploitations.

En fait, plus qu'un format, le pk3 est une structure précise, une façon d'organiser les éléments, qu'il faudra respecter pour que le jeu puisse retrouver les fichiers dont il a besoin et les intégrer dans le schéma final.

Voici en détail les caractéristiques de la structure d'un pk3 :

Fichiers & répertoires	Description
Readme.txt	Description complète de la map
Textures\nommap_version\	Répertoire contenant les textures
Env\	Répertoire contenant les fichiers de la Skybox
Levelshots\	Répertoire contenant une image représentant la map, destiné à être affichée dans l'écran de chargement, avec pour nom : 'nommap_version.jpeg'
Maps\nommap_version.bsp	La map compilée à partir de fichier au format map
Scripts\nommap_version.shader	Fichier contenant les shaders, i.e. des effets visuels
Scripts\ nommap_version.arena	Fichier d'information pour le jeu
Scripts\ nommap_version.particle	Fichier définissant la génération des particules utilisées dans certains effets visuels
Sounds\	Répertoire contenant des sons, des bruitages par exemple
Models\	Répertoire contenant des modèles 3D à ajouter à la map

Le fichier nommap_version.arena est très important, car il permet à la map d'être prise en compte par le jeu.

```
nommap.arena
{
map " nommap_version"
longname "Nom affiché à l'écran de chargement"
type "tremulous"
}
```

FIGURE 1: EXEMPLE DE FORMAT ARENA

Ce système est utilisé depuis plus d'une dizaine d'années par Id Software, la société à l'origine du moteur 3D de Tremulous, initialement pour Quake 3.

Il permet une facilité de création et de diffusion des maps, mais possède quand même quelques sérieux défauts. Par exemple, il n'y a aucune gestion des conflits, certains fichiers pouvant être pris en compte à la place d'autres par le jeu, sans possibilité de contrôle de ce phénomène autre qu'en utilisant des noms peu courants pour les fichiers que l'on ajoute.

Le format des fichiers map

Le format map est un format texte. Un fichier à ce format est composé d'un ensemble d'entités, décrivant les objets. Une entité est composée de ses attributs, ainsi que de sa description physique. La description physique d'une entité est une union d'intersections de plans. Un plan est défini par trois triplets de coordonnées (points) dont le premier doit appartenir au plan représenté. L'ordre des deux autres points donne le côté duquel sera visible la texture appliquée à la face (produit vectoriel entre les vecteurs $\overrightarrow{(1,2)}$ et $\overrightarrow{(1,3)}$).

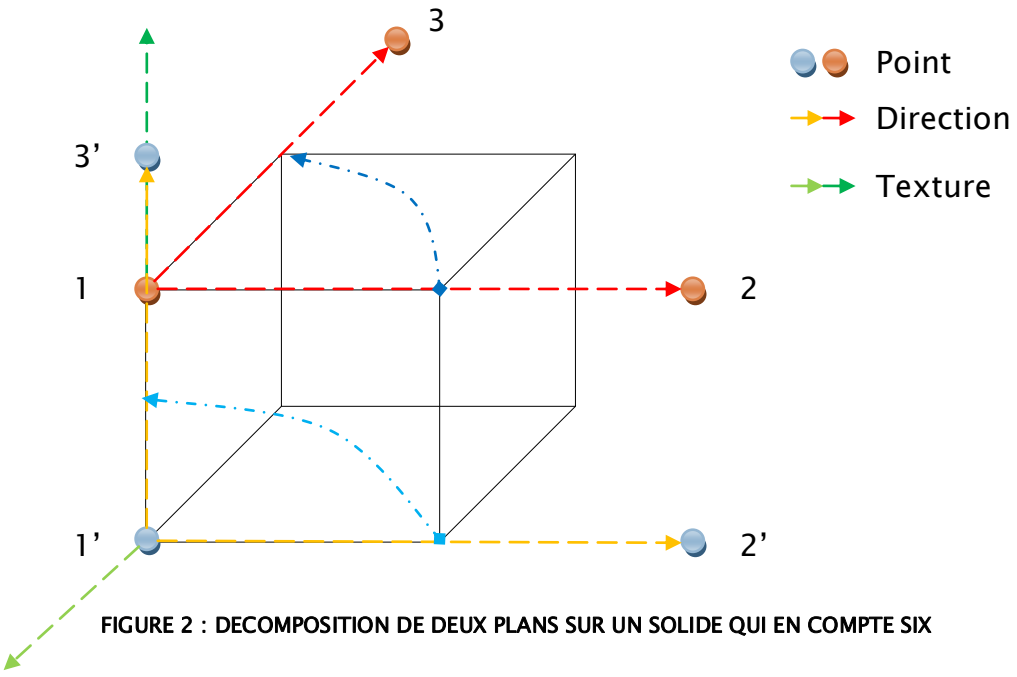


FIGURE 2 : DECOMPOSITION DE DEUX PLANS SUR UN SOLIDE QUI EN COMPTE SIX

Après la description de chaque plan, il y a une série de nombres et une chaine de caractères qui correspondent au shader/texture à appliquer à ce plan ainsi que des rotations ou homothéties à faire subir à sa texture.

Cette définition des formes est très générique et permet de rendre dans le jeu à peu près n'importe quel objet tridimensionnel. Elle n'est cependant pas aisée à comprendre et relativement peu documentée, si bien qu'au début du

projet, il nous a été plus simple de retro-ingénieurer le format que de fouiller l'Internet, à la recherche de sa description.

```
{
{
( 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 3 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 quelles seront les zone visibles de chaque point afin de simplifier la tâche du moteur graphique du jeu. Evidemment, les objets tels que les plantes et décors sont du détail et ne modifient pas la visibilité des zones qui sont derrière eux.

Cependant, si on ne fait pas la différence entre ces détails et les objets structurants (montagnes, bâtiments…), 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 4 : EXEMPLE D'ENTITE SANS PARTIE PHYSIQUE

Le format des fichiers shader

Les shaders définissent des effets visuels, parfois très complexes, associés à des textures.

Pour les utiliser dans une map, il faut les programmer. Il s'agit en effet d'un véritable langage, interprété par le compilateur de maps et le moteur 3d du jeu. Il possède une syntaxe qui lui est propre, qu'il nous a fallu en partie apprendre et surtout comprendre. Elle est en effet loin d'être simple à approcher, surtout à cause d'une documentation pauvre et élitiste.

Le domaine des exemples d'effets visuels est vaste. Dans Tremulous, le ciel, les fluides comme l'eau ou la lave, la lumière, le comportement des objets (du point de vue structurel), les sons, tout est régi par des shaders.

Voici comment créer un superbe ciel bleu, avec un magnifique soleil digne des côtes atlantiques :

```
textures/skies/normallight
{
    qer_editorimage textures/skies/sky.tga
    q3map_surfacelight 75
    q3map_lightsubdivide 512
    sun 0.75 0.79 1 250 0 65
    surfaceparm sky
    surfaceparm noimpact
    surfaceparm nomarks
    q3map_nolightmap
    skyParms textures/skies/blueskyofarcachon 512 -
}
```

Évident n'est-il pas ?

Ce shader est pourtant un exemple simple, celui-ci demandant pas plusieurs passes et ne faisant appel à aucune fonction avancée.

Les shaders posent un problème majeur qu'il a fallu prendre en compte pour notre projet. Certains effets visuels demandent de gros calculs, qui ont tendance à ralentir la compilation et l'affichage d'une map. Il a donc fallu très souvent trouver le juste milieu entre l'apparence, le réalisme, et les limites des machines des joueurs.

PLANIFICATION DE LA REALISATION

FIGURE 5 : 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 d'erreur (passage au travers du terrain, par exemple), ainsi que la création de lacs qui peuvent leur 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 sens minimal pour le joueur (par exemple regrouper les arbres en 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 peut imaginer de raccorder deux « terrains » entre eux.

De manière coordonnée avec l'avancement du projet nous avions planifié de communiquer sur celui-ci vers l'extérieur de l'Ecole, 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 imparti au PSI, en pensant mener à terme le projet durant notre temps libre via esiea-labs[2]. Enfait, la première phase est totalement achevée ; la seconde est fin de test et la troisième déjà entrée en conception. Notre objectif d'avancement est donc largement atteint.

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

REALISATION

Phase un : génération de terrain

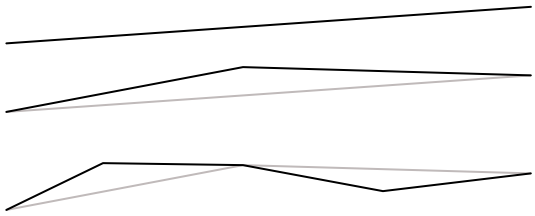


FIGURE 6 : EXEMPLE DE SUBDIVIDE & DISPLACE

La première phase est la génération de terrain jouable. Cette génération est effectuée à l'aide d'un algorithme fractal, *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 central est élevé (ou abaissé) d'une certaine valeur, aléatoire. 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 l'a vu au-dessus, pour obtenir une forme dans le jeu, il faut la décrire sous forme d'une réunion d'intersections de plans. Pour passer de la carte des hauteurs du terrain à cette représentation, nous avons testé plusieurs formules successives.

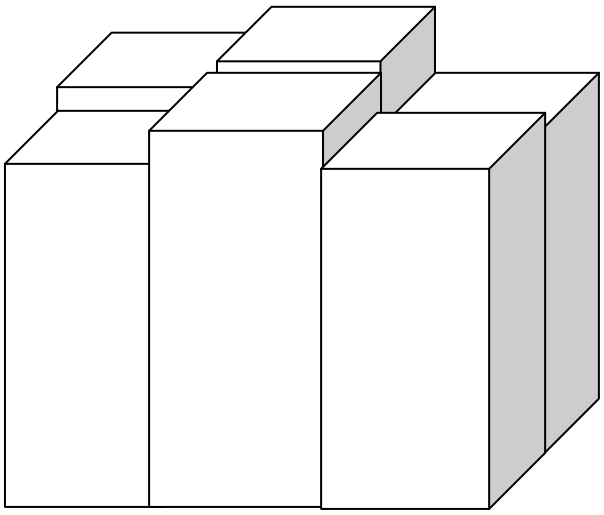


FIGURE 7 : PREMIERE VERSION DU TERRAIN

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.

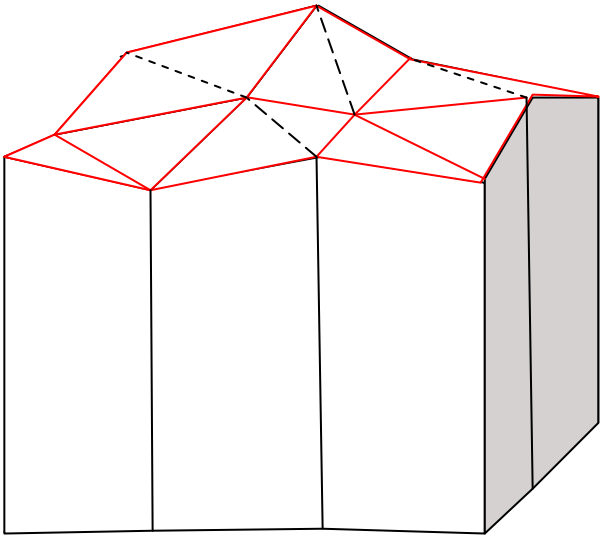
Cette version, bien que pouvant sembler triviale à réaliser, n'a pas été des plus simples à mettre au point car nous découvrions les subtilités du format des fichiers map en même temps.

La deuxième version du terrain a été obtenue suite à un changement de méthodologie. L'idée cette fois est de considérer qu'un terrain 3D est un objet qui se représente bien par le collage de triangles rectangles. Ainsi, au lieu de prendre chaque point de la heightmap et d'y faire correspondre un solide, on a besoin de trois points de la heightmap qui donnent la hauteur de chaque sommet du triangle. En réutilisant les points de proche en proche, on arrive ainsi à un terrain continu et de ce fait, beaucoup plus réaliste que le précédent.

Cette interprétation amène toutefois à générer un terrain comportant

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

formes.



Ici il y a 8 blocs au lieu de 12

FIGURE 8 : TROISIEME METHODE DE GENERATION

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

Cependant, le format des fichiers map n'impose pas de limite au nombre de plans dont on calcule l'intersection. Donc pour les solides convexes, nous avons pu fusionner les solides adjacents, afin de limiter le nombre de formes à compiler. Ce qui est la troisième version de la génération du terrain physique à partir de l'heightmap

Enfin, afin de faciliter le travail pour la troisième phase (génération de couloirs), nous avons rendu le terrain moins massif en enlevant les volumes pleins qui étaient dessous. Il a cependant fallu veiller à ne pas rendre concaves les briques (qui sont toutes convexes). Cette version est la dernière et la plus aboutie

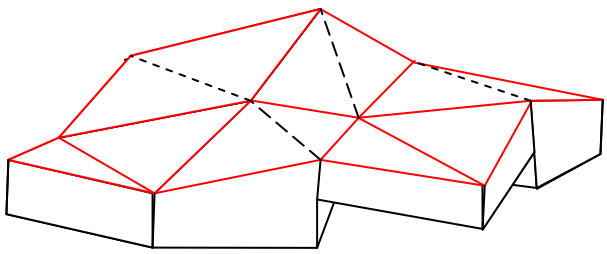


FIGURE 9 : DERNIERE VERSION DU GENERATEUR

Une fois le terrain généré nous avons ensuite créé un algorithme qui place un lac sur la carte. Cet algorithme est récursif et il est lancé sur les points de la

carte d'altitude minimum (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, ayant pour texture *water,* aux points qu'il renvoie.

Phase deux : placement d'objets

A la fin de la première phase, nous avions donc un terrain texturé et ayant une forme plus ou moins naturelle en fonction des différents paramètres des algorithmes. Nous avons pu constater que le terrain était solide en parcourant celui-ci dans tous les sens. Au cours de cette phase de test, 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.

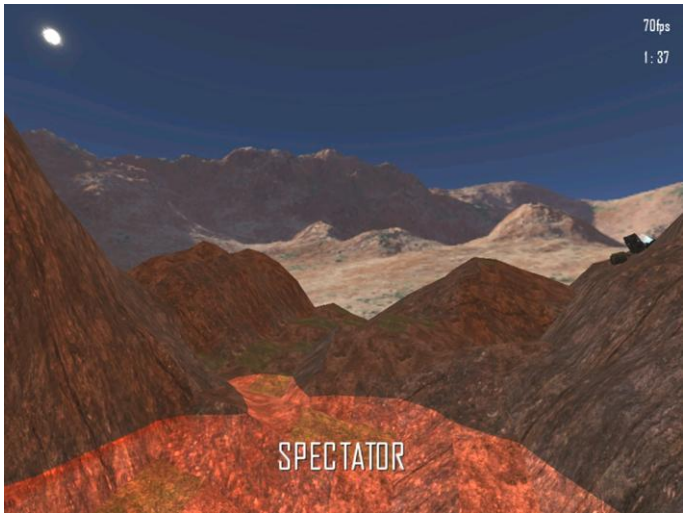


FIGURE 10 : TERRAIN GENERE (COMPILE EN LUMIERE)

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

Pour pouvoir tester, nous avions déjà dû nous intéresser à la représentation au format map des objets (comme vu au paragraphe « Le format des fichiers 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 est absente. Il nous avait donc fallu placer un minimum d'objets sur la carte. Ce positionnement était cependant fixe à la différence de ce qui a été réalisé en phase 2.

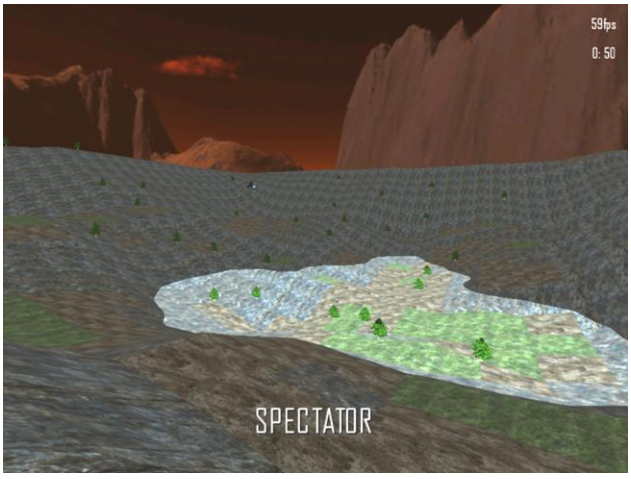


FIGURE 11 : PREMIER PLACEMENT D'ARBRES

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

carte.

Evidemment les premiers résultats étaient assez moyens. En effet les arbres n'étaient placés qu'à des positions entières 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, certains d'entre arbres arrivaient au milieu de l'eau…

Nous avons donc retravaillé l'algorithme pour qu'il place les arbres à des coordonnées non entières de la heightmap et en vérifiant que le point de placement se trouvait hors de l'eau. Pour ce faire, il nous a fallu 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 contentions de donner des séries de points sans jamais nous inquiéter des équations des plans ainsi formés. Pour la phase deux, il nous fallait pouvoir donner l'altitude de n'importe quel point sur la carte.

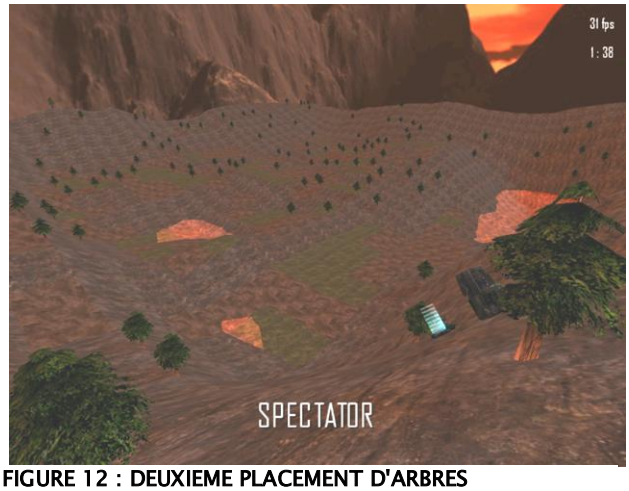


FIGURE 12 : DEUXIEME PLACEMENT D'ARBRES

Le rendu visuel était cette fois meilleur, mais les arbres étaient encore trop épars. Nous avons donc, sur la suggestion de Monsieur WASSNER, créé un algorithme de placement de forêts.

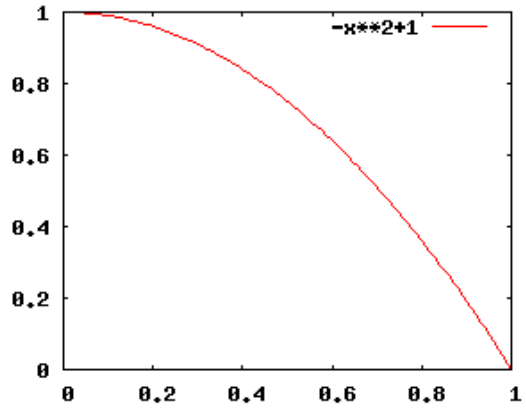


FIGURE 13 : REPARTITION AU CENTRE DE LA FORET

Le fonctionnement de celui-ci est assez simple. On choisit au hasard (algorithme de la deuxième méthode) la position du centre C de la forêt. On tire ensuite deux coordonnées polaires (r,θ) au hasard avec $r = -k * rand(0,1)^2 + 1$. On transcrit ces coordonnées 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, nous avons implanté un mécanisme donnant 5% de chance d'aller fonder une nouvelle forêt avec la moitié des arbres restant à placer. Enfin pour éviter d'avoir des forêts trop denses, si on est trop souvent amené 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 semble très bon, les forêts générées sont suffisamment denses pour que les aliens puissent se cacher à loisir dedans mais pas trop pour que les humains puissent encore les traverser.

FIGURE 14 : PLACEMENT DES ARBRES FINAL

Après le placement d'arbres, il ne nous restait plus qu'à 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 répartir des caisses sur le terrain.

Après avoir placé ces objets « décoratifs » nous avons retravaillé 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 côté « 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 avantagés en terrain où ils peuvent voir les aliens arriver 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 équivalente 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 fallu 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. L'idée en phase trois est de donner une autre dimension au jeu. L'environnement final ne devrait plus être limité à un cube mais étendu, soit dans un monde de galeries débouchant sur un cube, soit encore grâce à un réseau de tunnels reliant des cubes.

Dans ce but, il nous faut créer des couloirs qui s'accordent avec la zone de jeu. Il nous faut donc réaliser des couloirs de type humain (surfaces perpendiculaires, etc.) ou de type alien (cavernes…) voire de type terrain. Et surtout, il faudra savoir bien les relier à la zone de jeu.

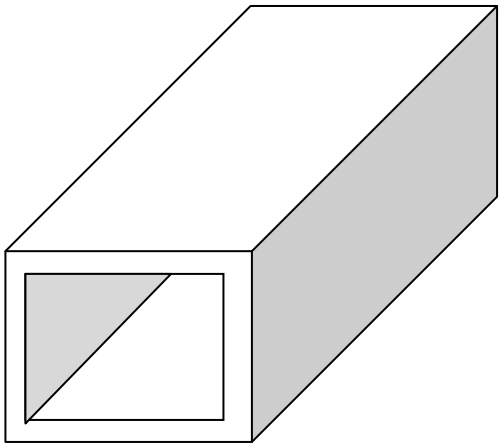


FIGURE 15 : COULOIR DE TYPE HUMAIN

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.

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 un tel type de couloirs. Cet algorithme fonctionne en séparant chaque couloir en plusieurs sections successives.

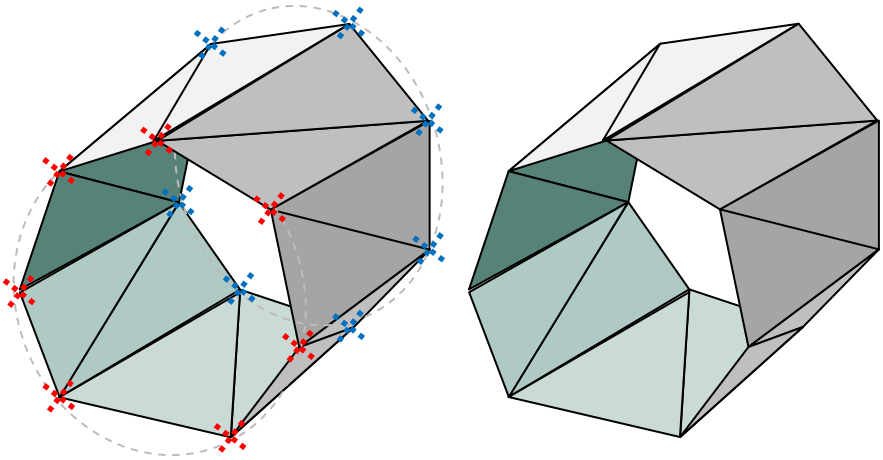


FIGURE 16 : EXEMPLE DE SECTION DE COULOIR ALIEN (SIMPLIFIE)

Pour commencer on prend au hasard N points sur un cercle centré au point central du début du couloir et de

rayon la taille voulue. On va ensuite reprendre une série de N±2 points sur un cercle, presque de même taille, placé à L (taille d'une section de couloir) et orienté sur un plan quasi parallèle. On relie ensuite les points entre les deux cercles par des surfaces planes (en fait des volumes pour créer un plancher solide). On reprend le travail sur la section suivante en prenant le deuxième jeu de points comme départ. On devrait obtenir ainsi des couloirs irréguliers 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 côtés, des surfaces 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 ici 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 carrée. Ce carré est enlevé du mur et le couloir va pouvoir installer ses propres décors. Par exemple une porte pour les humains et une cassure dans le mur pour les aliens.

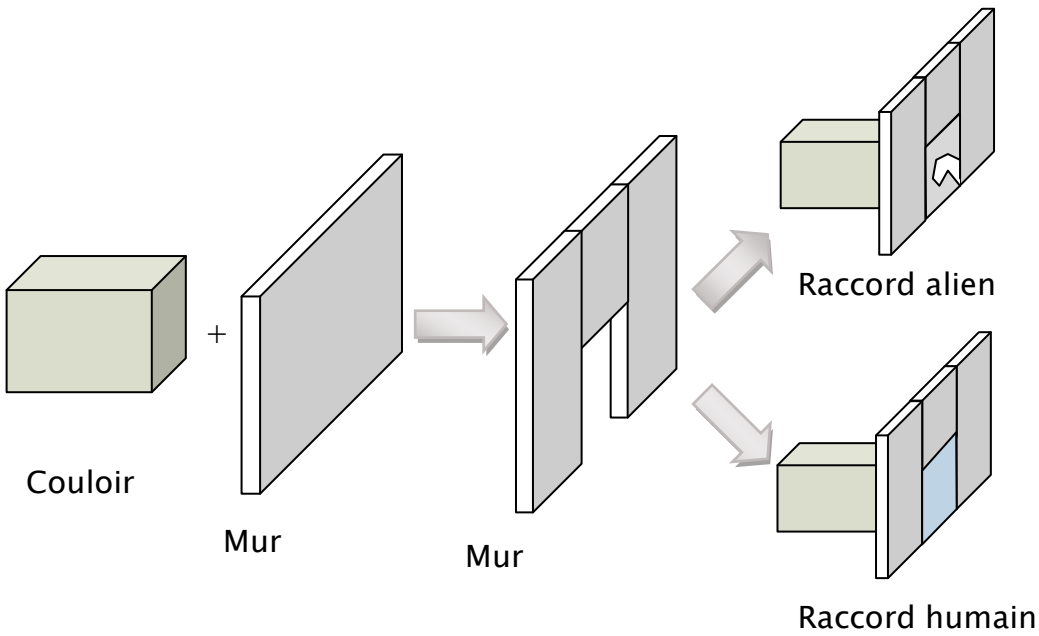


FIGURE 17 : PRINCIPE DU RACCORDEMENT DES COULOIRS AUX SALLES

Retour sur la compilation des maps

La compilation des maps est un processus compliqué. En effet, le fonctionnement du compilateur est dépendant d'un grand nombre d'options qui peuvent lui être passées en argument. Il y a trois niveaux de compilation : physique, visibilité et lumière. De plus, une option (« rapide » ou non) permet de moduler le niveau d'optimisation voulu dans le format de sortie.

Nos maps compilent en général, en 30 secondes en physique et 4minutes en visibilité rapide (non optimisé) et 6 minutes 30 en lumière rapide. Lorsqu'on demande les optimisations, il faut 14heures environ pour faire compiler la map en lumière. Ce dernier temps est passablement long mais reste encore dans la marge acceptable pour réaliser une de nos idées directrices : « un jour → une map ». Ce qui signifie que nous pourrions distribuer une nouvelle maps par jour à un public de joueurs.

COMMUNICATION

Réaliser un projet qui vise à toucher de nombreux joueurs, nécessite une période de test assez importante. Et surtout il faut qu'il y ait le plus de testeurs possible.

En effet, si l'intérêt d'avoir un générateur de maps est évident pour tout ce que cela peut apporter au gameplay, il faut garder à l'esprit que les maps créées doivent être en accord avec l'équilibre original du jeu. Par là, il faut comprendre que nous devons nous plier au fait que certaines armes ont une très longue portée, qu'il existe des jet pack qui permettent de se déplacer dans le ciel, etc. Ainsi les maps créées pour l'instant désavantagent le camp alien d'après les retours que nous avons eus, et il faudrait remédier à ce problème en rajoutant par exemple du brouillard. Ceci n'est qu'un exemple pris parmi les retours que nous avons eus.

Mais pour avoir ces retours, il a fallu créer un « buzz » autour de notre projet.

Le public visé par notre projet est relativement large, celui des joueurs ; cependant les joueurs sont habitués à des productions de qualité, et il y a peu de projets libres qui suscitent un réel engouement de la part des joueurs, Tremulous est une des rares exceptions.

Nous avions besoin de l'avis de ces joueurs, afin de respecter l'équilibre entre les deux camps et connaître la jouabilité effective de nos maps, pour améliore les algorithmes de génération.

Communiquer et faire parler de notre projet a donc été une étape importante.

Étant donnée la nature scolaire du projet, nos moyens financiers étant nuls, la campagne de promotion n'a donc eu lieu qu'au travers d'Internet. Cela ne nous a pas pour autant empêchés de recevoir de nombreux retours.

Nous avons diffusé deux vidéos de promotion du projet, la première pour expliquer le concept du projet, et la deuxième pour illustrer le projet en situation, lors de la 2ème Lan esiea.

Ces deux vidéos ont été diffusées sur les deux plus grandes plates-formes de vidéo en ligne : Youtube et Dailymotion. À ce jour elles ont été visualisées plus de 290 fois sur Dailymotion et plus de 660 fois sur Youtube. Nous avons également diffusé les vidéos au format h264 compatible pour les Podcasts.

Ainsi nous avons pu créer un petit buzz autour du projet, et nous avons pris contact avec des membres du forum officiel de Tremulous[3] aux US, ainsi qu'avec une équipe de joueurs français « the Hell with campers » (tHc) qui dispose de serveurs de jeux. Ces derniers semblent très intéressés[4] par notre projet et nous avons d'ores et déjà pu réaliser une petite phase de test sur un de leurs serveurs.

Le site web d'esiea labs comporte également une partie consacrée au projet, en finalisation. Ainsi, quand le projet sera disponible officiellement au public, toutes les informations relatives à celui-ci ainsi que les dernières nouveautés seront disponibles sur ce site.

Le projet a également été testé lors de la seconde LAN esiea, et ce fut le premier test d'envergure, trois maps ont étés dévoilées aux étudiants, qui les ont appréciées durant une nuit. Cependant au vu du public[5] fréquentant cette lan, les retours avaient étés bien moins construits que lors du test avec les thc.

[3] http://tremulous.net/phpBB2/viewtopic.php?t=4268

[4] http://forum.thcteam.org/viewtopic.php?id=255

[5] Elèves esiea, joueurs à leurs rares moments de libre

CONCLUSION

Ce projet a pour nous été l'occasion de passer de l'autre côté du rideau du monde des jeux vidéos. Cette expérience s'est avérée enrichissante sur tous les plans. Aussi bien au niveau Technique où nous avons pour parfaire nos connaissances en algorithmique et améliorer notre pratique du C++ qu'au niveau humain où nous avons du créer une petite campagne de communication qui a permis de faire parler de l'esiea au-delà de nos frontières.

ANNEXE : BIBLIOGRAPHIE

Voici quelques sources qui nous ont le plus aidés :

Kurim. Le Format .map. *Game-Lab.* [En ligne] Mini-tutoriel expliquant les bases de la création d'objets structurels en utilisant le format de description .map. http://www.game-lab.com/index.php?section=tutorials&section_id=1&p=tutorial&action=showtut&id=57.

O'Callaghan, Simon "sock". Creating Terrains. *Simland.* [En ligne] Article décrivant la mise en place des fonctions avancées de texturing des terrains pour les jeux basés sur le moteur quake3. http://simland.planetquake.gamespy.com/pages/articles/terrain1_1.htm.

Tremulous Team. Tremulous.net :: forums. *Tremulous.net.* [En ligne] Forums du site officiel de Tremulous, on y trouve les informations spécifiques au jeu. http://tremulous.net/phpBB2/.

Ydnar. Q3map2 shader manual. *Shaderlab.* [En ligne] Manuel relatif au compilateur de maps, il décrit le langage de scripting des shaders, point d'ultime référence. http://shaderlab.com/q3map2/manual/.