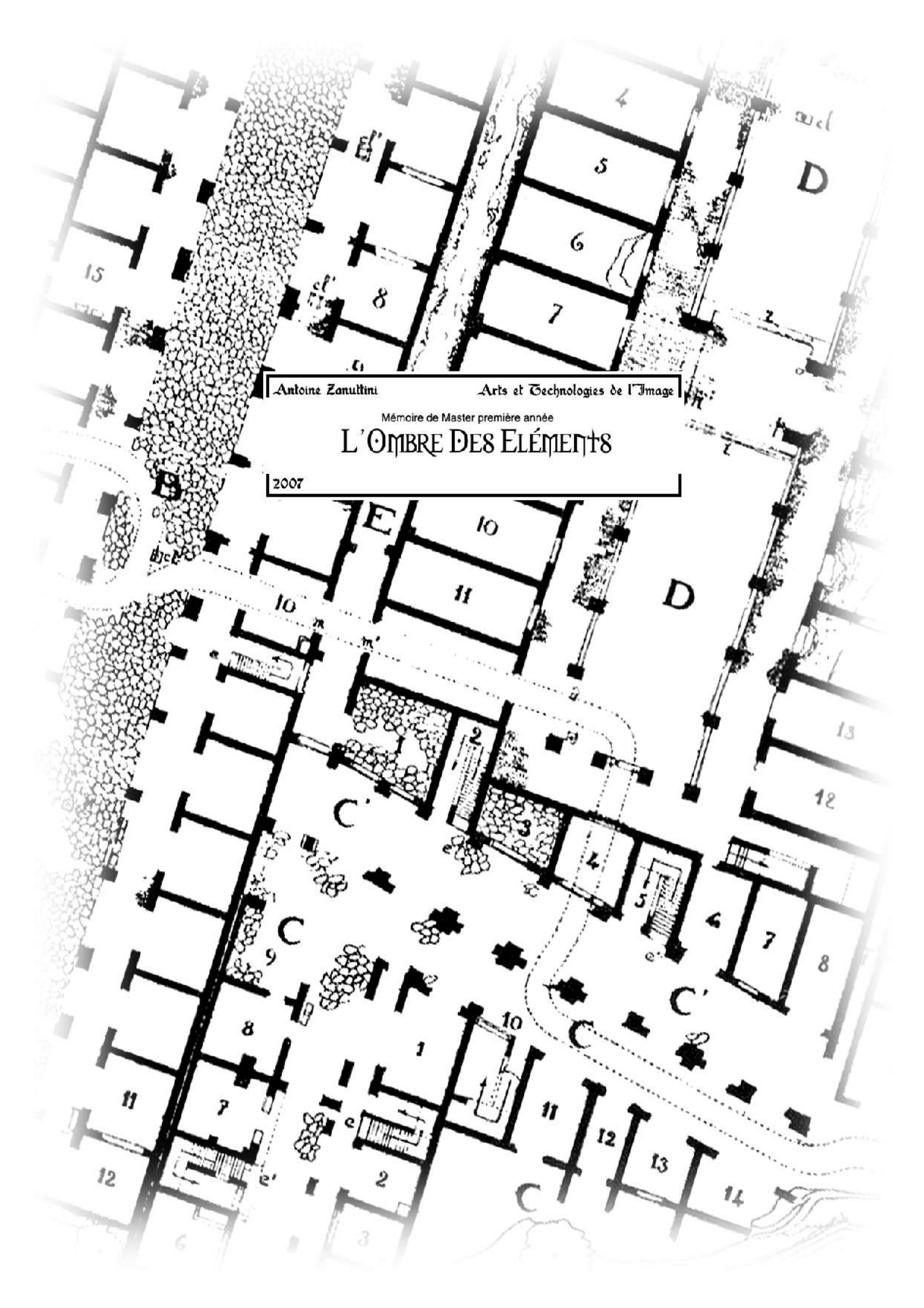


Antoine Zanuttini Arts et Technologies de l'Image

Mémoire de Master première année

L'Ombre Des Eléments

2007

Sommaire :

Introduction

Ce mémoire à été rédigé dans le cadre de l'année de Master 1 2006-2007 de la formation Arts et Technologies de l'Image (ATI) de l'université Paris 8. Il à pour but de présenter les travaux effectué sur le projet « L'Ombre des Eléments », jeu vidéo réalisé en équipe avec Dimitri Gouacide, Thomas Jeanjean, Sylvain Krasnopolski, Edwige Lelièvre et moi-même Antoine Zanuttini. Le projet est un jeu vidéo d'infiltration dans un univers médiéval fantastique dont les niveaux sont générés aléatoirement.

Les postes occupés sont les suivants :

Dimitri Gouacide	: Programmation graphique et game-play
Thomas Jeanjean	: Game-design, character-design, histoire et graphismes 2D
Sylvain Krasnopolski	: Chef de projet et animateur
Edwige Lelièvre	: Graphiste 3D
Antoine Zanuttini	: Programmation des niveaux et intelligence artificielle

Je présenterais dans ce mémoire tout d'abord un état de l'art en matière d'utilisation de l'aléatoire dans le jeu vidéo, puis j'aborderais les différentes parties du projet sur lesquelles je suis intervenu, de la génération des niveaux à l'intelligence artificielle. Pour avoir une vue globale du projet, je vous conseille de consulter les mémoires des autres membres de l'équipe, qui traiterons chacun d'une partie du projet.

Remerciements :

A Dimitri, Thomas, Sylvain et Edwige pour leur travail et leur courage durant le projet.

A Cédric Plessiet pour le temps qu'il a passé avec nous et ses conseils dans tous les domaines.

A Xavier Gouchet pour ses cours et ses encouragements.

A mes camarades de Master 1 ATI, parsqu'ils sont venus en cours ...

Partie A :

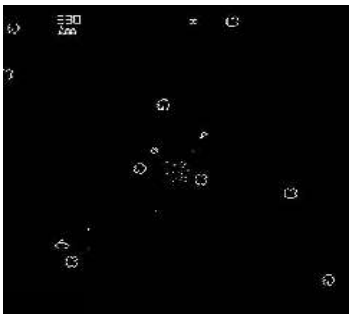
Etat de l'art dans le domaine de la génération de décors aléatoires

1) Les vénérables ancêtres

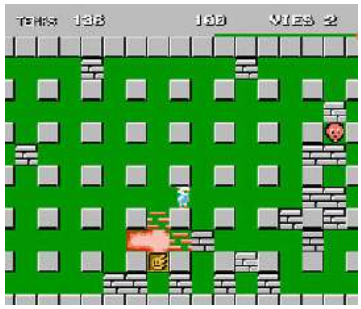
Tous les jeux comportent de l'aléatoire. C'est le moyen le plus simple qu'on ait trouvé pour donner de la diversité à un jeu, pour que les parties varient d'une fois sur l'autre. C'est également un moyen de créer beaucoup de contenu a partir d'une petite base d'éléments assemblés de manière aléatoire.

Paradoxalement, c'est peut être les premiers jeux qui ont été créés qui utilisaient le plus fortement l'aléatoire. Des jeux comme Asteroids (1979) (où l'on dirige un vaisseau pour détruire des astéroïdes arrivant de tous cotés aléatoirement) ou bien Bomberman (1983) (où le décor est constitué de caisses assemblées aléatoirement et qui donnent des items aléatoires lorsqu'on les détruit) sont des exemples marquants de l'utilisation intensive de l'aléatoire dans les premiers jeux, à la fois pour palier à la faible quantité de mémoire disponible mais aussi pour permettre à un jeu très simple comme Asteroids d'être rejoué des centaines de fois.

Par la suite l'aléatoire est bien entendu toujours utilisé mais de manière moins fondamentale pour le jeu. Il s'agit simplement de donner un comportement aléatoire aux ennemis, ou de varier les items que l'on reçois. L'aléatoire n'est alors plus utilisé pour augmenter les possibilités de manière exponentielle, il est cantonné à quelques parties du jeu, avec un impact très restreint sur le jeu en lui même.



Asteroids (1979)



Bomberman (1983)

2) L'aléatoire fait peur

Une des raisons pour ces restrictions dans l'utilisation de l'aléatoire est que les créateurs du jeu ont peur de perdre le contrôle sur le jeu. Avec l'industrialisation des jeux vidéos, le mode de création et surtout de production des jeux a considérablement changé, et cela n'a pas favorisé l'utilisation de l'aléatoire. En effet, la production d'un jeu est maintenant principalement confié à des financiers, qui recherchent un profit conséquent mais surtout une prise de risque minimum, avec des garanties sur le résultat. Alors qu'une petite équipe, à l'aube du jeu vidéo, pouvait se permettre de tester des concepts de jeux, et même devait trouver des moyens originaux de créer des jeux plus long et plus intéressants avec des moyens dérisoires, les productions actuelles de jeu cherchent simplement à appliquer une recette garantie et compensent le manque de concept fort par des graphismes très poussés. La majorité des jeux actuels sont envisagés de la même manière que des films. On y travaille principalement le scénario, l'ambiance graphique, on se place par rapport aux autres productions du genre, puis on passe à la réalisation concrète, ou il s'agit de créer successivement les différentes phases du jeu.

C'est une manière très linéaire de créer des jeux vidéo, et qui est très éloignée des possibilités qu'offre réellement ce média.

Les jeux devenant de plus en plus imposants, grossissant en taille, en durée, en diversité, il devient impossible de créer chaque élément séparément selon la manière traditionnelle. La plupart des jeux utilisent tout de même l'aléatoire durant la création du jeu, mais en interne à l'entreprise, puis cet aléatoire disparaît lorsqu'il sort dans le commerce. Il n'est en effet pas possible de créer à la main tout les arbres dans une carte de Warcraft III par exemple, à la place on utilise une sorte de pinceau qui va placer des arbres par dizaine, agencés de manière aléatoire, aux endroits ou l'on veut. Mais tout ça est ensuite figée, cela permettant de contrôler la qualité et éventuellement de faire des retouches à la main si besoin.

3) Les succès commerciaux :

Le jeu qui à le plus exploité l'aléatoire, prouvant l'étendue des possibilités offertes par ce biais la, est Diablo et sa suite Diablo II développés par Blizzard. Ce jeu est un « rogue-like », c'est à dire un jeu inspiré du jeu Rogue (1980). Dans ces types de jeu, on dirige un héros dans un donjon (ici médiéval), affrontant des monstres pour accomplir des quêtes et récupérer des trésors. Les donjons sont générés entièrement aléatoirement. A partir de divers éléments (différents murs, colonnes, ...) le donjon est créé au démarrage du jeu, puis remplit de monstres, avec eux aussi des caractéristiques aléatoires (force, compétences, objets à récupérer à sa mort ...).

Ces donjons sont intégré dans une histoire, qui elle n'a rien d'aléatoire, et les extérieurs aux donjons, ou l'on trouve les quêtes, sont totalement fixes. Le voyage du héros consistera donc a chercher des quêtes au village, à aller dans un donjon aléatoire les remplir, puis à refaire la même chose pour un autre donjon. Cette association d'éléments aléatoires et non aléatoires permet de conserver le contrôle sur le jeu et de donner tout de même une progression à l'histoire. Les donjons aléatoires sont classée en 4 ambiances différentes, ainsi même si le donjon n'est plus pareil la fois suivante, il conserve la même ambiance, on le reconnaît, et il reste intégré à l'histoire (la tanière d'un monstre restera une grotte, mais la structure en sera différente).

Lorsque l'on sait que le jeu Diablo comporte 60 donjons, tous conséquents, et permet de jouer des centaines d'heures, on comprend que les donjons ne puissent pas tous être créés à la main, et même ne doivent pas être créés a la main, si l'on veut que le joueur puisse refaire une partie qui soit unique.

Diablo I

Diablo II

Paradoxalement, la même entreprise, Blizzard, n'a plus utiliser ces techniques pour son jeu massivement multi-joueur (MMO) : World Of Warcraft. Ce jeu conserve à peu prés le même principe que celui de Diablo, avec l'ajout d'un univers persistant en plus. Mais ici, plus rien n'est aléatoire dans les décors. Le monde est totalement fixe (ce qui est un peu contradictoire pour un monde « persistant » censée évoluer en permanence, mais en fait qui persiste toujours pareil).

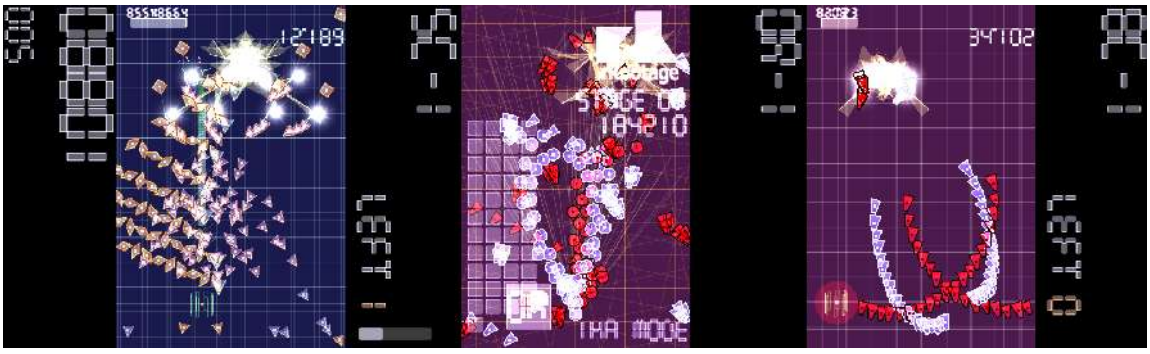
Certaines parties du jeu sont vraiment proches du concept de donjon à la Diablo, ce sont les Instances, qui sont des endroits qui peuvent être exploré en même temps par différents groupes, dans des sortes de mondes parallèles. Mais ces endroits, qui pourraient être recréés à chaque fois qu'un groupe pénètre à l'intérieur, restent désespérément fixes. Les seuls éléments aléatoires sont les « drops » (les objets qui tombent quand on tue un monstre). Quand on voit la difficulté qu'il y a

déjà pour Blizzard, d'équilibrer cet aléatoire la, on comprend peut être qu'ils n'ont pas voulu rajouter de la complexité et de l'aléatoire sur le reste.

Pourtant dans des jeux comme les MMORPG, ou la taille de l'univers est tellement énorme, ou la diversité et surtout la durée de vie (certaines personnes y jouent plusieurs heures par jours pendant des années), on ne peux s'empêcher de penser qu'il faudra bien un jour intégrer des moyens de diversifier automatiquement le contenu à partir d'un nombre réduit d'éléments. La construction complète de mondes aussi grand demande en effet l'intervention de plus en plus de personnes, ce qui fige également beaucoup l'originalité, l'équipe étant concentré sur la production de tous le contenu du monde, et pas sur des concepts plus globaux et plus généraux.

4) Les jeux amateurs :

Beaucoup de créateurs de jeux vidéos amateurs, en dehors des contraintes commerciales habituelles, utilisent intensément l'aléatoire dans la création des niveaux. On trouve ainsi des « mods » (modifications d'un jeu original par des amateurs) de jeux comme le vénérable Doom qui proposent des niveaux aléatoires. Il s'agit en général simplement de composer aléatoirement les niveaux a partir de quelques bouts, en modifiant les couleurs des matériaux et les monstres rencontrées. On trouve également un Mod du jeu Counter Strike, ou des simples caisses sont dispersées aléatoirement dans le niveau, cela rendant les combats entre les joueurs plus intéressants. Les Doujin (jeux amateurs japonais) de ABA Games sont des shoot'em up (jeux ou il faut tirer sur tout ce qui bouge) dans lequel les ennemis sont crées par un algorithme génétique. Le mouvement et la fréquence des tirs ennemis (communément appelée « patern » dans ce type de jeu) sont également soumis à un algorithme génétique. Et au final, un algorithme génétique est un moyen très intéressant d'ordonner et de restreindre l'aléatoire.



rRootage et ses Patern aléatoires (ABA Games)

5) L'avenir de l'aléatoire dans les jeux :

Le prochain jeu vidéo commercial qui exploitera véritablement l'aléatoire dans la création des niveaux est le jeu vidéo Hellgate – London, qui reprend les principes de Diablo mais en trois dimensions cette fois ci, en construisant les niveaux a partir de blocs. Le jeu se déroule dans un Londres alternatif, complètement délabré par d'arrivés de monstres. Certaines parties des niveaux seront fixes, par exemple les bâtiments célèbres (Big Ben par ex.) de Londres seront placés aux bons endroits sur la carte, mais les chemins pour se déplacer entre eux sera lui construit aléatoirement, avec différentes ambiances (métros, rues pleines de décombres ...).

Un autre jeu vidéo qui vient de sortir est Loki, du studio français Cyanide. Proposant un principe de jeu très similaire à Diablo, on retrouve ici aussi des décors aléatoires pour la plus grande partie, mais cette fois ci nous avons également droit à des extérieurs. Même si le jeu est en trois

dimensions, le gameplay est totalement en deux dimensions. Le jeu est composé de plusieurs aires (ou niveaux) qui ont chacune un style et des buts fixes, mais dont les éléments internes sont aléatoires. C'est en fait un simple carré, sur lequel est placé des points spéciaux tels que les villages, qui sont ensuite reliés entre eux par des routes qui serpentent aléatoirement de l'un a l'autre. En dehors de ces éléments, qui suffisent à donner un cadre à la zone, des forets, des lacs, des montagnes sont générées aléatoirement, ainsi que les ennemis bien sur.


Les mondes aléatoires de Loki (2007)

6) Les algorithmes utilisées :

La plupart des algorithmes utilisée dans la génération de niveaux de jeux aléatoires sont des algorithmes en deux dimensions seulement. De plus, le monde est entièrement discrétisé, structurant les éléments sous la forme d'une grille. Le but est alors de générer une sorte de labyrinthe de salles et de couloirs, comme ceux que l'on trouve dans les magazines télé, et où il s'agit de retrouver la sortie. Bien sur, ces algorithmes rajoutent tout de même quelques subtilités pour créer portes, coffres et monstres, mais la base est simplement de creuser un chemin qui aille d'un point A a un point B.

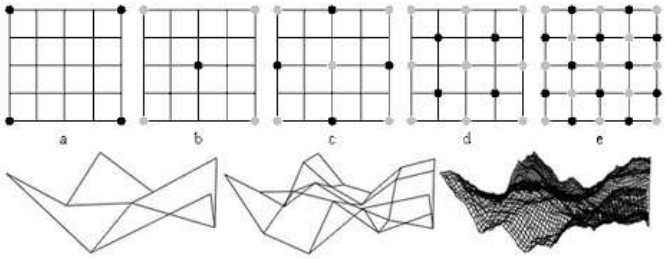
Le fait que l'univers soit en deux dimensions avec des coordonnées discrètes (entières), donne des résultats souvent assez peu naturels, avec des salles très rectangulaires et des objets placés toujours de la même manière, sur la grille 2D. Les jeux qui les utilisent profitent de leurs graphismes travaillés pour briser un peu la rigidité des structures, mais ils sont tout de même limité dans leurs possibilités d'aléatoire.

Dans un jeu comme Diablo, ce qui va donner les différentes ambiances, c'est le style des murs, ainsi que des pierres au sol. L'architecture des salles ne varie elle que très peu même si l'algorithme permet de gérer certains paramètres, tels que la taille moyenne des salles, la longueur moyenne des couloirs, le degré d'embranchement, et donc de créer des styles architecturaux différents. Tout cela reste dans le cadre de la 2D, ce qui est flagrant dans Diablo puisque les passages entre niveaux se fait par des escaliers qui sont en fait de simples portails, il n'y a aucune relation entre les différents niveaux.


Le moteur de génération de terrains aléatoire : Accidental Intricaty
Une génération intéressante et une autre beaucoup moins ...

Lors d'un précédent projet, en Licence à ATI, j'ai réalisé un jeu de stratégie avec des décors aléatoires. Le principe est très simple, puisqu'il s'agit juste de créer du relief, puis de positionner les deux équipes sur le terrain, en les reliant par un chemin. Le reste du terrain est remplit d'arbres ou de plaines. Les textures du sol sont des mixes de différentes textures, avec des filtres aléatoires pour les mélanger, le tout étant généré au démarrage du jeu. Ce projet m'a permit d'explorer certaines voies dans le domaine de l'aléatoire, des voies essentiellement en deux dimensions. Ce type de technique est très utilisé pour les jeux de stratégie.

La méthode de subdivision de terrain nommée « Diamond-square » pour créer du relief de manière aléatoire utilise, comme pour notre jeu, une notion de hiérarchie importante. Il s'agit simplement de diviser le terrain en quatre parties, et de modifier aléatoirement la hauteur du point médian de ces quatre parties (phase « diamond »). L'algorithme est récursif et s'applique donc ensuite sur chacune des parties, en veillant à garder des connections douces entre les parties (phase « square »). (Voir Fig. 1) Il existe ensuite des méthodes pour « éroder » le terrain, comme par le passage du vent et des années, mais aussi pour y creuser des rivières, qui devront serpenter de manière logique (en coulant du haut vers le bas). Vous trouverez des liens vers des sites traitants de ces sujets en annexe.



L'algorithme de subdivision Diamond-Square

Fig. 1

Partie B :

Le projet « L'ombre des éléments »

1) Introduction :

Dans le cadre d'un projet d'étudiants à ATI, le principal est d'expérimenter de nouvelles approches, que ce soit graphique ou ludique.
Nous nous sommes donc réparti les taches et chacun à essayé d'apporter de l'originalité à sa partie. J'ai pour ma part apporté l'idée de décors aléatoires complexes et intimement lié au game-play. Beaucoup de jeux utilisent de l'aléatoire pour rendre chaque partie d'un jeu unique. Mais la plupart du temps, cela se limite à la position des armes et autres items a collecter, ainsi qu'a la position des ennemis. Je souhaitais aller plus loin, et créer entièrement la structure d'un niveau de manière aléatoire, en tentant de créer une partie unique a chaque fois que l'on lance le jeu. Un type de jeu qui repose pleinement sur les décors est le jeu d'infiltration. En effet, on y incarne en général un personnage dont le but est de se déplacer discrètement à travers les décors en évitant gardes et pièges, en trouvant la sortie, qui parfois est difficile à dénicher, le personnage ayant de nombreuses possibilités d'actions (grappins, sauts, murs secrets ...).

Je vais tout d'abord présenter mes premiers idées, puis la direction que nous avons décidé de prendre une fois le groupe constitué.

2) Les premières idées :

Le jeu qui à servit de référence au départ est le jeu « Thief » sortie en France sous le nom de « Dark Project ». On y incarne Garret, un voleur engagé par diverses factions pour s'infiltrer dans des manoirs affin de récupérer des objets précieux ou bien d'autres missions tel que faire évader une personne, en tuer une autre.

Le but était donc de réussir à approcher le plus possible le principe de ce jeu, en ajoutant l'aléatoire des niveaux. Le principe le plus efficace à mon sens pour obtenir un aléatoire intéressant, est de hiérarchiser au maximum les décisions, et ensuite de définir le degré d'aléatoire à chaque niveau.

Ainsi dans un jeu de type Thief, nous pouvons découper les décisions selon le schéma suivant :

D'abord création d'un scénario à la Thief, avec en général un ou plusieurs buts, qui seront atteint en passant par plusieurs sous buts. On peux ainsi avoir le scénario « Voler le sceptre dans le coffre du compte » qui est constitué des sous-buts : « voler la clef du coffre », « s'introduire dans le manoir » et « trouver l'emplacement du coffre ». Chacun de ces sous-but pourrait faire l'objet d'une mission a part entière, ou-bien être regroupé dans une même mission, commençant dans la ville par exemple, ou il faudrait tout d'abord retrouver le comte dans une auberge, ou il est venu se distraire, lui subtiliser la clef, ainsi qu'une carte de son manoir sur laquelle on voit le coffre, et enfin se diriger vers le manoir, s'y introduire discrètement en évitant gardes et pièges, trouver clefs et autres leviers pour arriver au coffre.

Ce scénario typique peux être facilement écrit de manière aléatoire, en définissant de nombreux objectifs possibles, et en les associant de manière aléatoire. Pour que le scénario soit intéressant, cela ne suffit cependant pas. Il faut que l'on sente que chaque scénario est cohérent, que chaque sous-but contribue effectivement à l'accomplissement du but final. Pour cela, il faut effectuer une vrai travail de liaison de tous les scénarios que l'on arrive à imaginer, en liant tous les buts possibles pour chaque thème.

Une approche hiérarchique est encore une fois intéressante. Pour chaque but, on définit les sous-buts a atteindre. Chaque sous-but est lui même subdivisé. Ainsi le programme choisirait d'abord le but « Voler le sceptre » parmi tous les buts définis puis choisirait la circonstance « coffre » en ajoutant donc les sous-but « voler clef » et « trouver coffre ». De même le choix pour « voler clef » serait : « clef sur le comte » ce qui mènerait à « comte dans l'auberge » et le but « trouver le coffre » y serait associé ce qui ajouterait « trouver la carte » et également « s'introduire dans le manoir ». (Voir Fig.2)

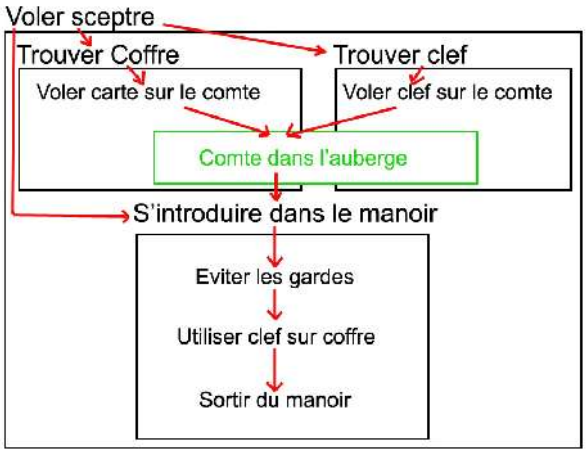


Fig. 2

Une fois le scénario définit et tous les buts liés les uns aux autres, nous pouvons passer à la création du décor proprement dite. Le scénario spécifie déjà les éléments principaux qui devront apparaître. Ainsi nous devons avoir une auberge, un manoir, une salle du coffre au minimum. A partir de la on commence à définir les contraintes extérieures telles que cours d'eaux, montagnes, falaises .... Celle-ci n'ont aucune influence sur l'histoire mais permettent de donner un caractère et une structure très typé à chaque niveau. Ensuite, on place les bâtiments principaux, ici tout d'abord le manoir et l'auberge.

Une idée intéressante est de pouvoir choisir les choses à différent niveau hiérarchique. Ainsi dans le scénario il est pour le moment simplement prévu les différents liens entre les buts, mais c'est au moment de la création concrète du décor, que l'on défini un bâtiment qui selon les contraintes extérieures est plus ou moins grand et qui donc prendra le nom, dans le scénario, de « manoir » ou de « château » selon sa talle, de même pour l'auberge.

Une fois les bâtiments nécessaires crées, on passe aux secondaires. On va créer toute la ville autour. On peux alors prévoir plusieurs chemins pour aller d'un endroit à l'autre ou bien des buts annexes qui peuvent être atteint. (voir Fig. 3)

Une fois l'équipe constitué, nous avons beaucoup discuté de ce que chacun avait envi d'apporter au projet et en partant de cette idée de jeu d'infiltration médiéval fantastique, nous avons beaucoup évolué, notamment en ajoutant une histoire forte et un gameplay beaucoup plus travaillé, avec de nombreux sorts et une évolution du personnage principal. La manière d'envisager l'aléatoire à donc pas mal changé, et nous voulions une structure plus fixe, qui permette d'intégrer une histoire. Celle-ci ce déroule principalement dans un château, et le personnage parcourt les différentes ailes du château en cherchant à réunir des clés pour pénétrer dans le donjon. (pour de plus amples détails sur l'histoire, une partie du mémoire de Thomas y est consacré).

Nous avons rapidement décidé de nous consacrer tout d'abord à une des ailes particulière du donjon, à savoir l'aile de la Terre. Nous sommes donc parti plus sur des niveaux intérieurs, avec une ambiance un peu fermé, de cave éclairé uniquement à la torche. Mes idées de départ ont donc bien changé, et je me suis principalement concentré sur les niveaux en intérieur.

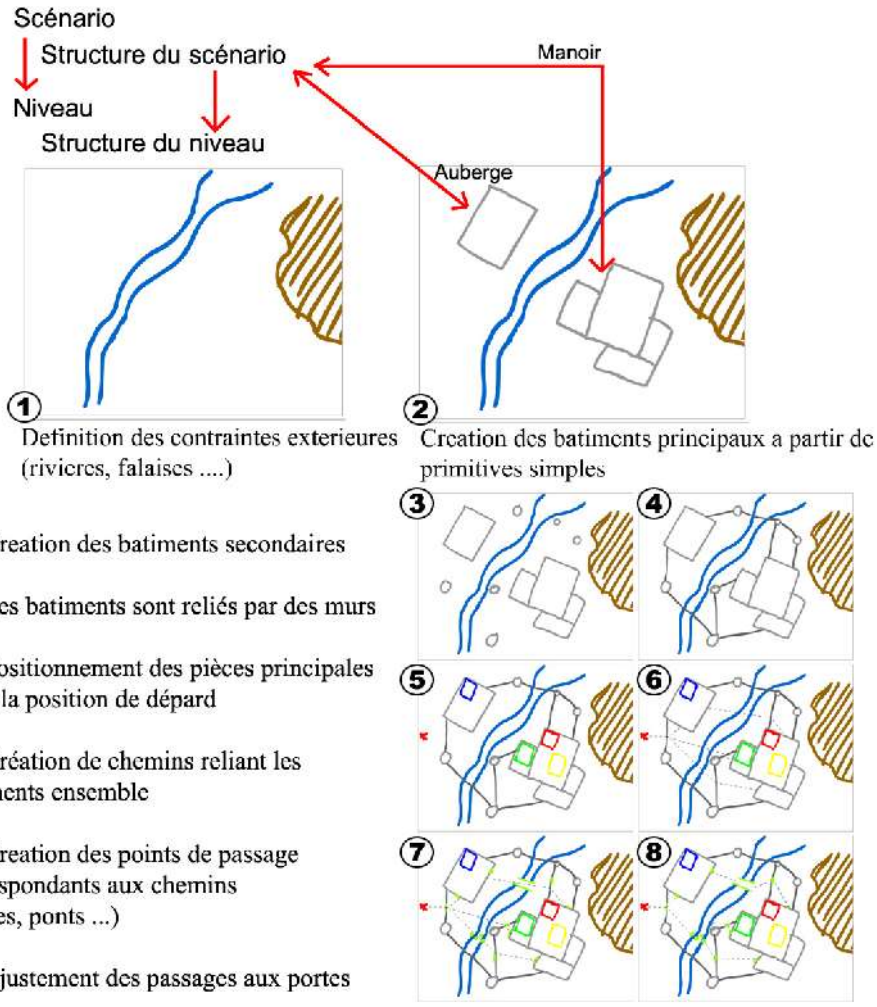

Scénario
Structure du scénario
Niveau
Structure du niveau
Manoir
Auberge
1
2
Definition des contraintes exterieures (rivieres, falaises ....)
Creation des batiments principaux a partir de primitives simples
3 : Creation des batiments secondaires
4 : Les batiments sont reliés par des murs
5 : Positionnement des pièces principales et de la position de dépard
6 : Création de chemins reliant les batiments ensemble
7 : Creation des points de passage correspondants aux chemins (portes, ponts ...)
8 : Ajustement des passages aux portes
3
4
5
6
7
8

Fig. 3

3) La construction des niveaux :

a) La génération des niveaux :

Ce que je souhaitais, dans la génération aléatoire des pièces, était de le pas discrétiser le niveau, mais au contraire de travailler avec des valeurs continues, en travaillant sur l'adaptation des objets aux niveaux. Beaucoup des idées que j'avais n'ont pas pu être mis en oeuvre sur le projet, par manque de temps, mais aussi parce qu'elles n'étaient pas nécessaires sur ce projet précis.

Ainsi, j'ai beaucoup réfléchi aux moyens pour adapter les objets à leurs environnements. Une idée toute simple est par exemple de pouvoir définir pendant la création de l'objet (dans Maya par exemple) certains points comme devant prendre toute la hauteur de la pièce par exemple. On peux ainsi faire des objets qui vont « stretcher » ou « tiller » c'est a dire s'étirer ou se répéter sur toute la hauteur de la pièce. C'est particulièrement utile pour créer des colonnes ou bien des motifs répétitifs sur les murs. Le même principe peux être appliqué à pas mal d'objets pour réaliser par exemple des tables que l'on va pouvoir étirer en programmation pour lui donner la longueur et la largeur que l'on souhaite. Les objets deviennent alors très facilement adaptables à leur environnement.

Par manque de temps, ce procédé n'a pas été utilisé, mais j'ai mis en place d'autres fonctions qui vont faciliter la création d'objets qui s'adaptent à l'environnement (ou qui adaptent l'environnement à leur présence). Ainsi l'on peux définir (directement dans le logiciel de création : Maya) certains objets comme creusant des trous dans les parois. C'est pratiquement indispensable pour les portes, puisque cela permet de faire des ouverture de n'importe quelle forme, simplement en donnant la forme que l'on souhaite à l'objet qui va découper le mur. On peux créer également des fenêtres ou bien de petites alcôves avec des bougies, ou bien encore des petits bassins creusés dans le sol. Cette fonctionnalité, qui paraît assez simple, est en fait plutôt complexe, puisque la découpe du mur doit être faite au moment de la génération du niveau, donc il m'a fallu programmer entièrement des CSG (Constructive Solid Géometry), également appelées « Opérations booléennes » et qui permettent de creuser un objet avec l'autre (le logiciel utilisé, Virtools, ne permettait pas cela de base);

b) La génération des objets :

Pour pouvoir régler l'aléatoire de manière très fine, j'ai construit une structure hiérarchique, chaque niveau de la hierarchie pouvant faire appel à d'autres objets. Par exemple, dans un cas typique, lorsque l'on crée une pièce, on va décidé de mètre une table dans un coin. Cette table, une fois mise place, va décidé de mettre des chaises sur ces cotés, et également de porter un contenu de table. Quand on fait appel à ce contenu de table, il décide de placer une corbeille de fruit par exemple. Cette corbeille décide elle-même de contenir deux pommes et une poire. Chaque objet va donc pouvoir décider des objets dont elle aura besoin pour se constituer. On peux alors introduire des dizaines de variations par exemple sur le contenu d'une table, et chaque table pourra y faire appel, mais aussi d'autre éléments (armoires, buffets ...) sans avoir a redéfinir ce contenu. (Fig 4/5)

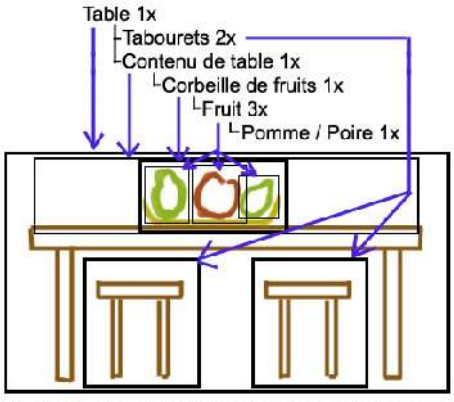


Fig. 4 Hierarchie des objets dans l'exemple "table-corbeille"

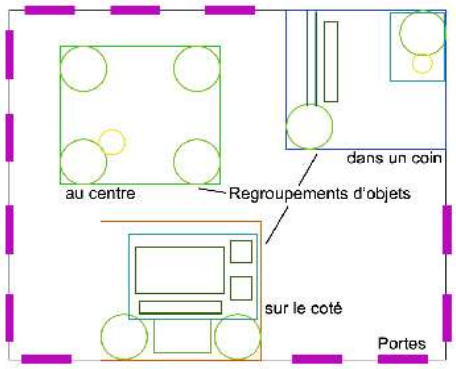


Fig. 5 Découpage d'une pièce et hiérarchie des objets

c) Les différents formats de fichiers :

Afin de permettre une génération des objets avec un aléatoire hiérarchique, j'ai mis en place un système séparant la description des objets en plusieurs fichiers.
Pour pouvoir les identifier, les sélectionner, mais aussi les typer, chaque « objet » reçoit, tout au long de sa hiérarchie, diverses informations, de deux sortes :

Tout d'abord les catégories auxquelles appartient l'objet, qui peuvent être des catégories basées sur le type (coffre, armoire, porte ...) sur la fonction (conteneur, piège, ...) ou bien encore sur la taille (gros objets, petits objets ...).

Ensuite l'objet défini tout un tas de variables par rapport auxquelles il se positionne. Ainsi on peux avoir la variable « rouge » qui sera définie à 100 pour un objet rouge et ne sera pas définie pour un objet bleu par exemple. Dans notre cas, la corbeille appartiendrait par exemple aux catégories : {accessoire, contenu de table, conteneur de fruits, corbeille} et aurait les variables suivantes : { jolie 80%, drôle 10%, utile 30%, précieux 5%}. La table va par exemple chercher un contenu en formulant sa requête avec comme catégories {contenu de table}, et comme variables { jolie 100%, utile 50%} (donc la table souhaite avoir un contenu qui soit très jolie mais aussi assez utile), et la corbeille sera pré-sélectionné (car ses catégories sont compatible) puis ensuite sélectionné, sauf si on définit quelque part un objet « Chandelier » qui aura comme catégories { accessoire, arme, contenu de table, lumière} et comme variables { jolie 90%, utile 60%, précieux 60%} et qui sera donc préféré pour être placé sur la table. Bien sur, ce n'est pas une décision fixe, divers paramètres permettent de sélectionner le plus souvent les objets qui correspondent le mieux, mais aussi de temps en temps des objets qui correspondent moins bien mais qui vont apporter de la diversités aux décors.

Ce principe est également appliqué aux sons dans le jeu, ainsi qu'aux textures des sols et des murs.

Pour les objets, nous en commençons la description au niveau de la pièce, qui comporte l'emplacement (position et rotation) de différents objets désigné par leurs catégories. On choisira plus tard l'objet concret qui sera placé à cette endroit la et il devra appartenir à toutes les catégories définies. Le fichier de description de pièce comporte aussi des indications de tailles, les catégories auxquelles la pièce elle-même appartient, ainsi que son positionnement par rapport a des variables (comme pour les objets). Au moment de choisir quelle pièce concrète doit aller à tel emplacement, on choisira la pièce qui correspond aux catégories et qui est le plus proches des variables choisies.
Il existe également des fichiers décrivant les murs, sur le même principe que pour les pièces, et qui s'occupent principalement des décors sur les murs (poteaux, tableaux ...)

d) La découpe des salles :

Avant de pouvoir remplir les pièces, il faut d'abord définir leurs emplacements, leurs tailles, ainsi que les portes qui y mènent. La solution utilisée ici est assez simple, mais donne de bons résultats pour des décors intérieurs. Elle permet en outre d'obtenir des salles de tailles et de positions en coordonnées continues, et non discrètes comme le sont la plus-part des algorithmes cités auparavant. Le principe est de subdiviser récursivement chaque pièce selon un axe (vertical ou horizontal) en deux parties qui seront elle-même subdivisées, et ainsi jusqu'à ce que l'on estime avoir atteint une taille, ou bien une proportion intéressante pour la pièce. (voir Fig. 6)
Selon la taille de la pièce, ainsi que diverses variables (voir la partie c sur les variables), un fichier de description de pièce est sélectionné, ainsi que des fichiers de description de murs. Ces fichiers contiennent les emplacements de tous les objets principaux, qui feront eux même appel à d'autres objets. Une technique simple pour augmenter l'aléatoire à ce niveau est de créer des regroupements

d'objets. Par exemple la pièce va demander un objet de type « Coin de taille 2x4m ». Des dizaines de ces objets existent, l'algorithme en choisi un. Celui choisi pourra contenir des tables, des colonnes, des lumières, l'essentiel sera qu'il ne dépasse pas la taille de 2x4m.

1 : Creation de l'interieur des batiments

2 : Séparation récursive de l'interieur des batiments (découpe)

3 : Fin de la séparation récursive

4 : Création des pièces interieures finales en combinant plusieurs pièces

5 : Creation des chemins principaux entre les pièces importantes et les sorties

6 : Creation de portes correspondantes aux chemins

7 : Creation de chemins reliant les pièces isolées et les portes associées

8 : Définition des types de pièces en fonction de leur position par rapport aux pièces principales, de leurs tailles et de leurs formes

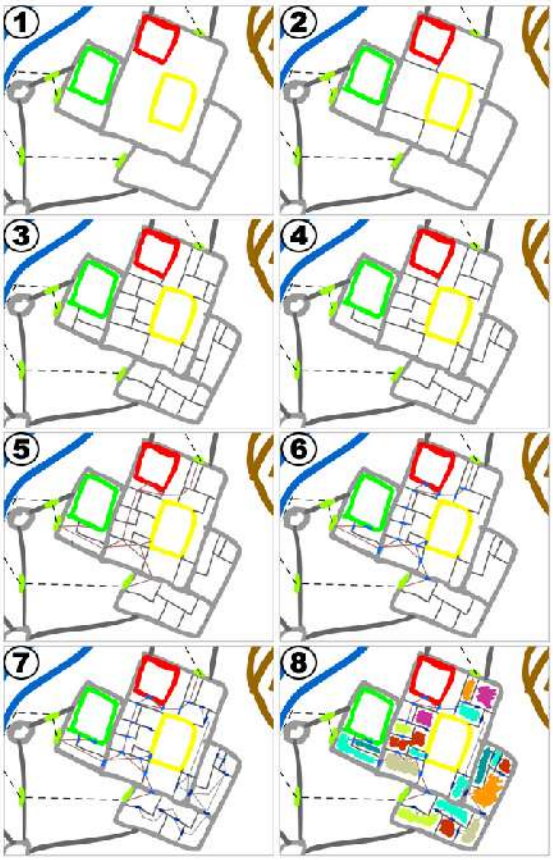


Fig. 6

e) La création d'un chemin :

Ensuite, après la création du bâtiment, avec ses multiples niveaux, et remplit de salles, il s'agit de tracer un chemin, un labyrinthe, qui relira une pièce A a une pièce B en connectant toutes les autres pièces en passant. Le premier but est donc de créer un arbre qui relie une et une seule fois chaque pièce. C'est ce que l'on appelle un arbre couvrant. L'ensemble de toutes les connections possibles entre les pièces forme un graphe connexe non-orienté. Il s'agit donc de sélectionner un sous-ensemble de ce graphe qui forme un arbre complet et son utilisation garanti qu'il n'y a qu'un seul chemin pour aller d'une pièce à une autre. Pour tracer cet arbre, on peux utiliser l'algorithme de Prim, qui a partir d'un sommet sélectionné aléatoirement dans le graphe, construit l'arbre couvrant minimal (c'est a dire dont la longueur des branches est le plus petit possible).

Avec cet arbre, on peux maintenant trouver facilement la branche qui présente la plus grande longueur, en partant de la pièce choisit comme la pièce finale du niveau (par exemple une pièce du dernier étage) et en rejoignant le bord du rez-de-chaussé du bâtiment en cherchant à maximiser la longueur du chemin. Ce chemin la sera le chemin principal que le personnage devra forcement visiter pour arriver à la fin. On peut maintenant placer aléatoirement des portes entre les pièces sur ce chemin principal. Ces portes sont fermé a clés, et les clés placées dans le niveau, avec pour seule contrainte qu'elles puissent être atteintes avant la porte qu'elles doivent ouvrir. Elles doivent donc ce trouver simplement du bon coté de leurs portes.

Une fois ces portes principales posées, nous avons délimité différentes zones, séparées par des portes a clés. (voir fig. 7) On peut maintenant rajouter, a l'intérieur de ces zones, des nouveaux

passages, qui créerons donc des boucles dans le graphe. Le personnage aura plusieurs chemin possibles entre deux portes a clés. Un ajout intéressant qui pourrait être fait à ce niveau, serais de pouvoir placer des portes à clés sur les chemins secondaires du graphe, et pas seulement sur le chemin principal. Ceci rend plus complexe le placement des clés, et il faut alors décider d'un ordre précis pour la découverte des clés par le personnage. Avec l'algorithme actuel, on peux trouver les clés dans n'importe quel ordre.
Un autre ajout possible, serais de retirer la contrainte non-orienté du graphe, ce qui nous permettrait d'avoir des chemins a sens unique, tels que des sauts d'un étage à celui d'en-dessous.

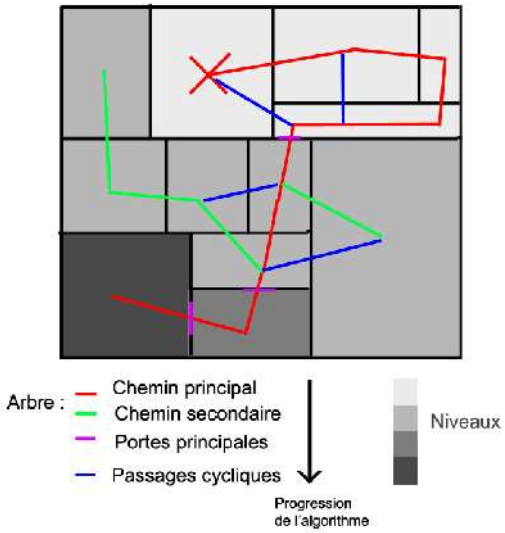


Fig. 7

f) Le système sonore :

Dans un jeu d'infiltration, l'ambiance sonore et la précision des sons sont des éléments essentiels pour l'implication du joueur mais aussi pour le game-play, puisque les indications sonores permettent de se diriger dans les lieux sombres, et de connaître la positon des gardes sans les voir.

Nous avions donc besoin d'un son en trois dimensions, c'est a dire que l'on puisse, grâce au stéréo, ou même à un système avec plus de deux haut parleurs; savoir d'où viennent les bruits alentours. Une autre caractéristique essentielle est le fait que le son ne traversent pas les murs, mais passent par les chemins, les portes, pour que les gardes ne nous entendent pas à travers les murs, et que nous aillons vraiment la bonne direction donnée aux sons. Pour cela, chaque son bénéficie du système de path-finding, et provient donc de la bonne direction par rapport au joueur.

De la même manière que les objets, les sons sont organisées dans des fichiers qui leurs donnent certaines caractéristiques, et qui permet de les regrouper et de les sélectionner aléatoirement selon certains critères. Ainsi, chaque pas d'un garde va prendre un son de pas au hasard parmi tous ceux qui sont définis. De plus le son sélectionné ne sera pas le même en fonction de l'endroit où se place le joueur. Ainsi selon les salles, nous pouvons définir des sons de pas différents, et même, le son changera si l'on passe sur certains objets spéciaux, comme les tapis, et nous aurons alors un son plus étouffé.

4) L'intelligence artificielle :

a) La recherche de chemin :

Le premier problème pour créer une intelligence artificielle dans un jeu de ce type, est celui de la recherche de chemins (pathfinding). Il s'agit de pouvoir trouver les differentes positions (waypoints) par lequels un garde devra passer pour aller d'un point A à un point B sans traverser le décors, et bien sur sans faire de détours inutiles.

On décompose ces algorithmes en deux phases : en premier la construction des points par lesquels les gardes peuvent passer (waypoints), puis la recherche d'un chemin entre ces différents points.

L'algorithme le plus utilisée, et qui est même « parfais » en recherche de chemins est le A* (A star) qui part du point de départ et progresse d'un point à un autre jusqu'à arriver au point d'arrivé. L'idée est de tester les différents points selon la distance parcouru depuis le départ et en estimant la distance jusqu'à l'arrivée, afin d'envisager d'abord les points les plus proches puis les autres. Le seul point de l'algorithme qui peux varier est l'heuristique qui va nous donner une approximation de la distance à parcourir pour atteindre l'arrivée. Cette distance doit bien sur être la plus proche de la réalité pour que l'algorithme soit efficace. Dans mon cas, l'utilisation d'une simple mesure de distance à vol d'oiseau suffit.

L'autre phase, la construction des points de passage, est bien plus problématique. En effet, dans un jeu classique où les décors sont fixes, ces points de passage le sont aussi et sont pré-calculées puis enregistrées avec le niveau. On peux donc utiliser des algorithme très gourmands en temps, voir même placer les points à la main, puisque cela ne sera fait qu'une seule fois. Mais pour notre jeu, nous sommes forcés de générer les points au démarrage du jeu, selon la forme du niveau. Les éléments importants à prendre en compte dans le placement de ces points sont tout d'abord qu'il faut que de n'importe quelle position atteignable du niveau on puisse accéder en ligne droite à au moins un waypoint, et également que l'on puisse aller de chaque waypoint à tous les autres (un graphe connexe). Ensuite, il faut réduire au maximum le nombre de way-points, puisque chaque ajout va rendre l'algorithme plus lent. Il faut enfin que ces points soient placés de manière logique pour que les gardes ne fassent pas de détours mais que leur chemin paraisse naturel. (voir Fig. 8)

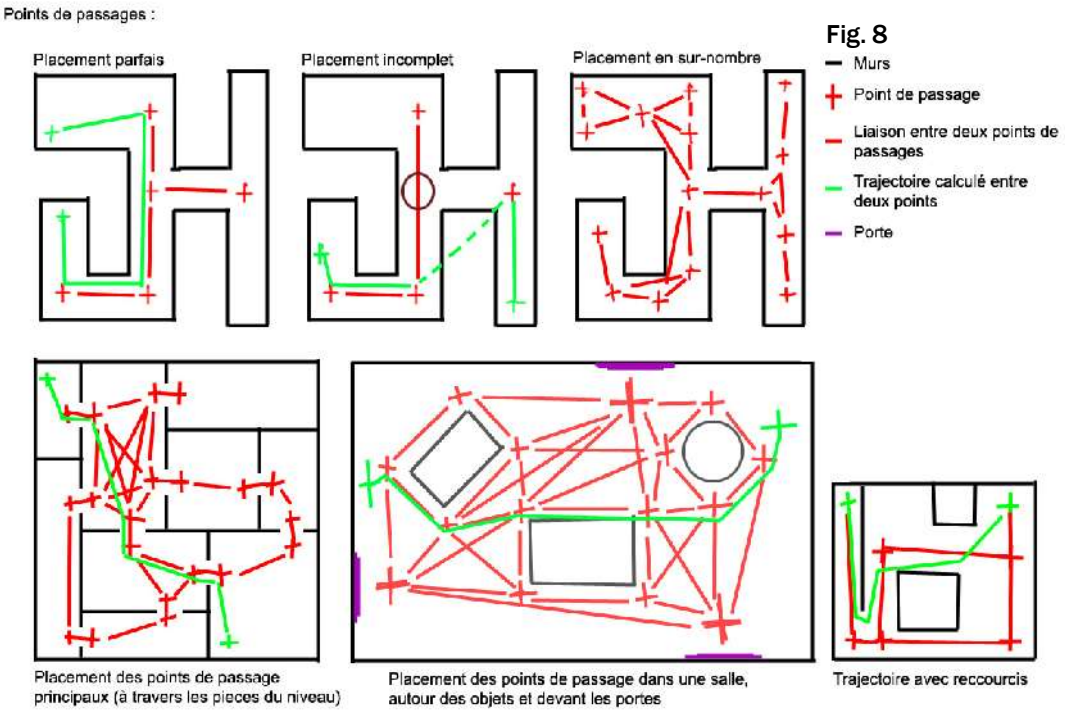


Fig. 8

Pour la construction des waypoints, j'ai tester plusieurs solutions :
Une solution est de quadriller le niveau de points de passages, puis de supprimer ceux qui sont à l'extérieur du décor. On peux également mètre en place des algorithmes pour supprimer les points redondants. Cette technique est utilisée en majorité par les jeux de stratégie. Dans notre cas, cela ne donne pas de bons résultats, surtout parce que nous avons fait le choix de ne pas placer les objets à des emplacements discret (sur une grille) ce qui demande de quadriller le décor de beaucoup trop de points de passages. L'algorithme est alors bien trop lent pour être efficace.

La solution que j'ai retenu au final est basée sur un pathfinding à deux niveaux.
Tout d'abord nous avons des waypoints principaux qui permettent de trouver un chemin d'une pièce à un autre pièce, en passant par les portes, mais sans faire attention aux objets dans les salles. Ensuite le deuxième niveau permet de passer d'une porte à une autre porte à l'intérieur d'une salle, en évitant les objets. On cherche donc tout d'abord le chemin principal vers la destination, puis pour chaque salle visitée, on cherche comment la traverser. La seconde recherche peux être fait au début du trajet du garde, afin d'avoir un chemin complet, ou bien calculé au cour du trajet du garde, au moment ou il entre dans une nouvelle pièce, ce qui permet de distribuer le calcul, et surtout d'éviter des calculs si finalement le garde change d'avis et ne va pas au bout de son chemin.

La génération des points de passage principaux est très simple, puisqu'il suffit de mettre un point de passage de chaque coté de chaque porte pour que le chemin soit complet et minimal.
Pour générer les points de passage à l'intérieur d'une pièce, j'ai choisi d'utiliser les objets eux même. En fait, chaque objet qui comporte des collisions comporte également des points de passage qui permettent de contourner l'objet. Quand on place l'objet dans la pièce, on ajoute également les points de passage, en veillant à supprimer ceux qui sortent de la salle ou qui entrent en collision avec d'autres objets. Cette manière de placer les points permet de s'assurer d'avoir suffisamment de points pour contourner les objets tout en gardant un nombre faible de points.

L'algorithme final ce présente ainsi :

- Recherche du point de passage le plus proche du point de départ, et qui soit atteignable en ligne droite.
- De même pour le point de passage le plus proche du point d'arrivé
- On cherche le chemin principal pour traverser les salles entre les deux points de passage trouvé.
- On cherche ensuite le chemin pour chaque salle traversée.
- Le garde suit le chemin en allant a chaque fois au point de passage suivant
- On recherche continuellement des raccourcis pour obtenir un chemin plus naturel.

b) Le comportement des gardes :

Maintenant que le garde peux se rendre où il le souhaite, il faut lui donner différents comportements selon la situation. Un prendra l'un ou l'autre de ces comportements selon par exemple qu'il à vu ou non le joueur.

Le premier état du garde est l'état « ronde » dans lequel le garde passe simplement de position en position selon la ronde qu'il doit faire. Il fait des poses régulièrement.
S'il voit le joueur, il passe dans un état suspect, dans lequel il surveille. S'il ne voit plus le joueur, il reprend sa ronde, mais si au bout que quelques instant le joueur est toujours visible, il se lance à sa poursuite. Il se rend toujours au dernier endroit où il vu le joueur. S'il perd vraiment le joueur, il va se mettre à patrouiller dans les environs pendant quelques temps, puis reprendre sa ronde. S'il arrive à rejoindre le joueur et à s'approcher suffisamment près, il l'assomme et le jette en prison. Les même états se retrouvent à peu prés au niveau du son. Si un garde entend un bruit suffisamment fort, il va se diriger vers l'endroit où il s'est produit. Il va faire une patrouille dans le coin, puis s'il n'a vu personne, il va retourner sur sa ronde.
D'autres états pourront être ajouter, notamment pour faire interagir les gardes entre eux, pour qu'ils cherchent dans des directions différentes par exemple.
Mais dans un jeu d'infiltration comme le notre, il est surtout important pour le joueur qu'il ne soit pas découvert, et s'il l'est, c'est quasiment la fin pour lui, s'il ne se cache pas assez vite.

c) Ajouts possibles au game-play :

Certains éléments de game-play sont maintenant faciles à rajouter. Ainsi nous pouvons donner la possibilité de jeter un objet qui va servir de leurre, et attirer les gardes à un autre endroit. Au niveau programmation, il s'agit simplement de produire un son très fort à un endroit, et le système de gestion des sons prend en charge l'avertissement des gardes à proximité. Il est également envisageable de permettre au personnage d'éteindre les torches qu'il rencontre, afin de progresser dans le noir. Les sorts du type « assommer un garde » que ce soit au corps à corps ou a distance ne sont pas vraiment difficiles à implémenter avec l'architecture en « machine à état » des gardes.

5) Bilan sur le résultat :

Avec les systèmes que j'ai mis en place au niveau de l'aléatoire et au niveau intelligence artificielle, je crois qu'il est possible d'arriver à créer des niveaux pratiquement aussi intéressants à jouer que ceux de Thief par exemple. Mais nous n'y sommes pas véritablement parvenue. En effet, il ne suffit pas de programmer les systèmes pour obtenir un jeu, il faut ensuite beaucoup travailler sur les données du jeu en elle-même, c'est à dire dans notre cas sur les fichiers de description de salles et de regroupement d'objets.

Créer des salles intéressantes et diversifiées en conservant l'aléatoire est difficile, il faut bien penser la manière dont sont regroupées et choisis les objets, il faut s'assurer que les objets rentrent dans les endroits spécifiés. De plus, créer des niveaux aléatoires devient surtout intéressant lorsque l'on à beaucoup d'objets et encore plus de variantes de ces objets, que l'on va pouvoir regrouper dans différents packs de types de salles et de types d'objets en utilisant mon système de variables, ce qui peux permettre, avec un type de salle, d'avoir une ambiance de cuisine par exemple, ou bien de salle de garde, simplement en faisant varier quelques variables, qui vont influencer les styles des objets choisit au final. Mais nous ne sommes parvenue qu'a faire une seule ambiance, un seul style de salle, ce qui rend l'aléatoire complètement répétitif.

Le choix de l'aléatoire à peut-être également un peu bloqué notre graphiste, Edwige, qui ne se sentait pas vraiment libre dans la création des objets. Je pense que je me suis également un peu trop concentré sur la technique, et que je me suis pris trop tard sur la création concrète des salles et des objets. Je pensais confier cela au game-designer, Thomas, et à Edwige, mais eux aussi manquaient de temps pour travailler dessus, ce qui fait que nous n'avons pas eu beaucoup de temps pour tester et intégrer les salles et les objets.

Paradoxalement, peux être qu'une solution à ce problème était de rajouter encore un peu de technique. En effet, j'ai rapidement écrit des outils en mel script pour Maya afin de pouvoir exporter et importer facilement des types d'objets et des types de salles, avec leurs propriétés. J'ai aussi écrit un petit programme Virtools pour voir une seule salle, généré aléatoirement, en fixant sa taille. Mais ces outils n'étaient pas faciles à prendre en main, et ne suffisaient pas pour tester complètement et efficacement les modifications que l'on fait sur les données du jeu. Peut être qu'un outil plus évolué, un véritable éditeur, comme dans les jeux de stratégies par exemple, aurait été d'un grand secours pour contrôler le résultat.

Un exemple de génération aléatoire d'un niveau dans le jeu « L'ombre Des Eléments »

Disposition d'objets aléatoire dans une salle

Concernant le jeu en lui même, il est vrai que moi et Dimitri, lorsque nous nous sommes partagée les taches, nous n'avons pas vraiment mis l'accent sur le game-play et sur la programmation des pièges et des sorts dont dispose le joueur. Moi je souhaitais travailler sur la création aléatoire du niveau ainsi que sur l'intelligence artificielle, et lui voulait principalement programmer des effets graphiques, en utilisant des shaders et autres techniques, pour travailler les lumières, les ombres, le rendu. Au final, nous avons vraiment laissé de coté le game-play, ce qui est évidement une erreur. Je pense que c'est aussi un erreur de méthode, de commencer à programmer ce qui est le plus intéressant et le plus difficile en laissant le reste pour plus tard.

Une bonne méthode aurait peut être été de ne commencer la programmation concrète du jeu que vers le milieu de l'année, et passer les premiers mois à préparer une maquette du jeu, avec quasiment tous les éléments du jeu fini mais sans la complexité de la programmation derrière. Virtools est de plus idéal pour ce genre de choses. Nous aurions ainsi pu avoir une version dans laquelle le personnage se déplace, utilise quelques sorts, évite quelques gardes. Nous aurions pu donner un aspect « fini » à cette démo, ce qui aurais dirigé le groupe dans la bonne direction en fixant certains éléments qui sont restés obscur très longtemps, tel que le style visuel, l'éventail d'action du joueur ...

Une démo de ce genre est également très importante pour la motivation, qui peut très vite faire défaut dans un groupe qui travaille pendant plus d'un an sur un projet.

Une prison dans « L'Ombre Des Eléments »

Une pièce vu du dessus

Gestion des ombres et des lumières

Un étage complet vu du dessus

Conclusion :

Ce projet à été intéressant à de nombreux points de vue. Il m'a permis d'expérimenter beaucoup en matière d'aléatoire et en matière d'intelligence artificielle, notamment au niveau path-finding. C'est le premier grand projet sur lequel j'ai été programmeur, et j'ai beaucoup appris.

Un projet en groupe est très difficile à maîtriser, surtout lorsque l'on est encore étudiant. Définir un chef de projet parmi nous n'a pas suffit a donner une bonne structure au groupe. Une raison également est le manque de suivi du corps enseignant à ATI. Les quelques séances de suivit de projet n'ont pas été d'un grand secours.

Au final, nous ne sommes pas parvenu à atteindre les buts que nous nous étions fixé en début d'année. Nous n'avons pas un véritable jeu, puisque l'histoire et le game-play créée par Thomas ne sont pas vraiment intégrés. Nous avons différents bouts de jeux qui ne forment pas un tout.

Mais pour moi, dans un projet universitaire, le résultat n'est pas la seule priorité, la recherche et l'expérimentation l'est également, et dans ce domaine la, nous sommes allés assez loin, que ce soit dans les niveaux aléatoires, dans le graphisme (programmation et conception), dans l'histoire et le game-play (resté à l'état papier malheureusement).

Annexes :

Références web :

Slige http://www.doomworld.com/slige/
Génération aléatoire de niveaux pour le jeu Doom

DOOM Boardgame Random Level Generation System
http://pages.infinit.net/lone/doomrlgs.htm
Paterns aléatoires pour le jeu de plateau Doom

Accidental Intricaty : http://members.gamedev.net/vertexnormal/
Moteur expérimental de génération de terrains aléatoires, par fractales essentiellement.

Algorithme « Diamond-square » pour la génération de terrains aléatoires.
http://www.gameprogrammer.com/fractal.html

ABA games : http://www.asahi-net.or.jp/~cs8k-cyu/
Création de jeux amateurs exploitant énormément l'aléatoire

Rogue : http://www.rots.net/rogue/index.html
Guides, liens, sources, tout pour jouer à Rogue en ascii

Bibliographie :

« Intelligence artificielle » de Stuart Russel et Peter Norvig :
Théorie des graphes, algorithmes de recherches, heuristiques ...

« L'Aléatoire » de Marcel Conche
Etude des influences de l'aléatoire dans tous les domaines (histoire, cosmos, nature)

« Push Start : 30 ans de jeux vidéo » de François Houste
Historique des jeux vidéo