Cahier des Charges

Buendia Sandrine Gallien Coralie Almes Romain
Richard Romain

9 février 2009

Table des matières

1 Besoins utilisateur

1.1 Expression initiale des besoins - Rappel du sujet

L'objectif de ce TER est d'implémenter des PNJ (Personnages Non Joueurs) dans un FPS (*First-Person Shooter* ou Jeu de tir subjectif) par des processus de décision markoviens, ou machines markoviennes, leur permettant d'avoir un comportement stochastique : le choix d'action est déterminé selon une distribution de probabilités, d'avoir un comportement évolutif : en fonction du renforcement de la carte (ex : pondérations sur les waypoints) les probabilités sur les actions sont modifiées.

FIG. 1 – Exemple de FPS : Unreal Tournament 2004

Nous partirons de l'hypothèse que la carte est connue des PNJ : la machine markovienne peut donc être incrémentalement mise à jour par programmation dynamique.

Dans ce TER il est demandé de construire une maquette permettant :

- de charger une carte avec son bâtiment 3D, ses objets et PNJ,
- de permettre à un humain de jouer un PJ,
- de construire les IA des PNJ avec des machines markoviennes.

1.2 Exigences fonctionnelles

Le projet devra regrouper toutes les fonctionnalités permettant donc, comme indiqué précédemment, de charger une carte complète en 3D (terrain, objets et PNJ) et surtout de construire les intelligences artificielles des personnages non joueurs avec des machines markoviennes.

Suite à un entretien avec notre responsable, M. Koriche, nous n'implémenterons pas la fonction d'intégration d'un personnage joueur : trop de complexité et trop peu d'intérêt par rapport à l'UE correspondante.

1.2.1 Charger une map

Un terrain (ou map) devra être constitué d'un environnement simple dans un premier temps : un terrain plat sans étage avec peu d'obstacles, tout cela en 3D. L'intégration des PNJ se fera plus tard. La modélisation du terrain et des personnages ainsi que l'animation de ces derniers seront entièrement réalisées par nos soins.

1.2.2 Construire l'intelligence

Les PNJ devront donc se comporter de manière intelligente : nous souhaitons que les personnages se déplacent de manière rationnelle. Pour cela, nous allons implémenter les comportements des PNJ à l'aide de machines markoviennes (comportement probabilistique) qui pourront être incrémentées (variations des probabilités) en fonction d'évènements (passage sur un checkpoint ou évènement déclenché manuellement).

Un travail important de pathfinding devra donc être réalisé pour que les PNJ se déplacent de manière optimale tout en prenant en compte l'environnement (objets et autres PNJ).

1.3 Exigences non-fonctionnelles

1.3.1 Exigences de qualité

Nous avons choisi de ne pas investir beaucoup de temps dans le moteur de jeu (regroupant les moteurs graphique et physique entre autres) pour se consacrer plus intensément sur la partie de l'IA des PNJ (ou bots).

Nous devrons être capable de modifier les paramètres initiaux de manière efficace (sans avoir à "plonger" dans le code), en utilisant par exemple un fichier de configuration.

Nous ne nous focaliserons pas trop sur l'aspect design du jeu ; cela ne veut pas dire pour autant une absence de recherche dans l'ambiance du jeu ou encore la gestion des graphismes (nous développerons d'avantage cette partie si nous arrivons à un résultat

satisfaisant concernant l'IA des bots rapidement).

Le shooter devra néanmoins rester attirant et simple, permettant aux "spectateurs" de profiter pleinement du spectacle (le combat des bots).

1.3.2 Exigences de performance

Le programme devra être fluide, c'est à dire que les personnages devront se déplacer sans à-coups.

On devra pouvoir mettre 2 PNJ (ou plus dans l'idéal) ayant un comportement distinct (on créera différentes "classes" de personnages) sans diminuer la performance du programme.

1.4 Contraintes de conception

Nous essaierons de faire en sorte que notre projet fonctionne sur un maximum de machines (au niveau des performances et non de l'OS).

Nous ferons aussi en sorte que le temps de chargement reste raisonnable.

2 Solutions envisagées

2.1 Intelligence des PNJ

Nous allons nous investir d'avantage dans cette partie. Les machines markoviennes seront étudiées cette année donc nous nous appuierons sur nos cours pour implémenter les comportements des PNJ que nous créerons. L'avantage principal des machines markoviennes est de pouvoir ajouter de la jouabilité à un jeu en augmentant l'effet de surprise. En effet, vu que le comportement d'un bot sera probabiliste, il ne réagira pas forcément de la même manière à une situation.

Le moteur de jeu incorpore entre autre un langage de script, UnrealScript, qui permet aux modeleurs de paramétrer les niveaux au-delà du modélisme :

- Élaboration d'automatismes
- Modification des propriétés des décorations
- Modification des propriétés des armes

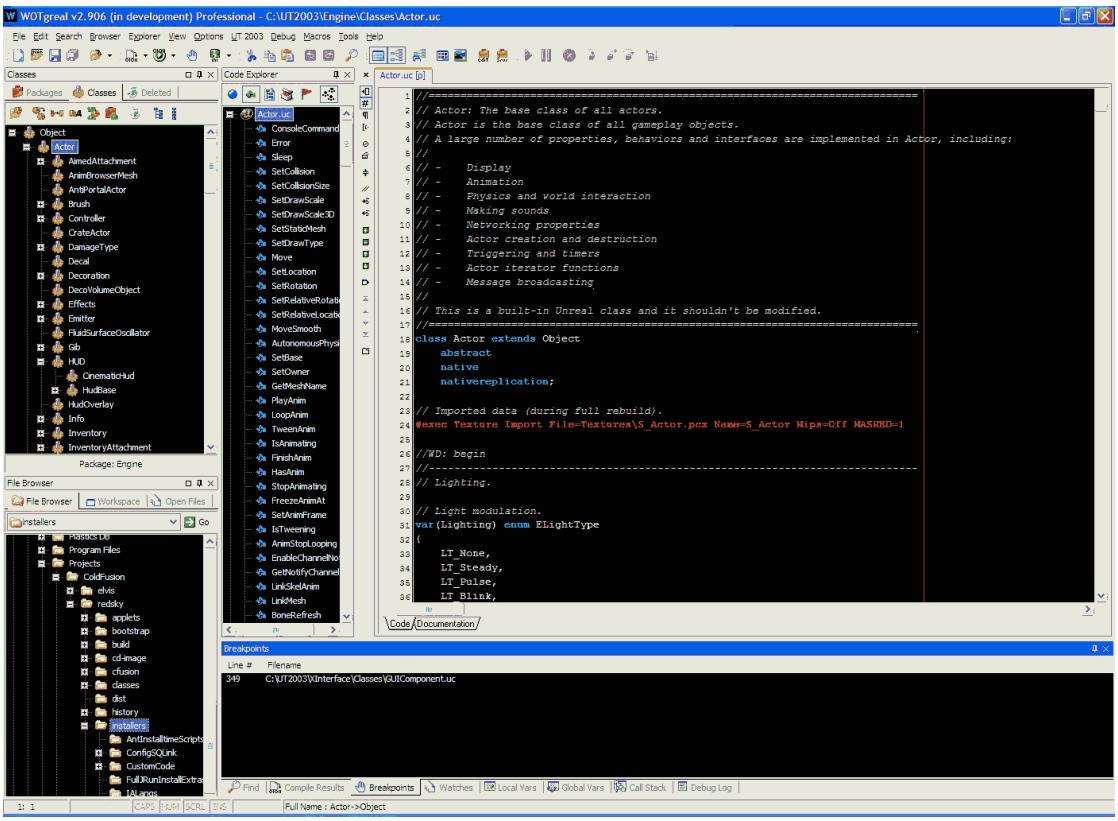


FIG. 2 – Interface de Wotgreal, un IDE pour l'UnrealScript

Nous verrons par la suite que le moteur UnrealEngine2, celui que l'on pense utiliser est composé en plus d'un moteur graphique, physique et aussi d'un éditeur de niveaux.

Pour arriver à un tel résultat, nous nous servirons d'abord de machines à états finis puis nous nous essaierons aux machines markoviennes. On pourra ainsi commencer à maîtriser l'éditeur de script du moteur, donc avancer dans l'IA des PNJ, tout en travaillant avec l'éditeur de niveaux.

Nous essaierons de faire dans un premier temps deux PNJ s'affrontant puis, si cela se déroule bien, deux équipes de bots (avec un comportement différent chez les PNJ d'une même équipe : un bot qui soigne, un bot trés offensif, un qui explore et qui avertit ses coéquipiers, etc.).

2.2 Moteur de jeu

Nous avons choisi d'utiliser un moteur de jeu déjà existant, cela étant dû à l'importance d'un tel module (contrainte de temps). Le fait d'en créer un nous aurait contraint à passer moins de temps sur la conception de l'IA des PNJ, ce qui nous intéresse principalement. Ce choix est, de plus, guidé en grande partie par nos connaissances actuelles dans les domaines de la 3D et de l'IA.

Nous sommes actuellement en train de tester différents moteurs de jeu pour en choisir un qui se rapprochera le plus de notre idéal (sachant qu'il y aura toujours certaines fonctionnalités non implémentées).

Allonger la phase d'analyse nous permettra de ne pas subir les conséquences d'un choix trop hâtif.

À l'heure actuelle nous travaillons encore sur deux moteurs tombés dans le domaine public :

- l'Id Tech III (aussi connu sous le nom de Quake 3 Engine), développé par Id Software et à l'origine de nombreux jeux.
- l'UnrealEngine 2 (ou Warfare Engine), à l'origine de Unreal Tournament 2003 et (aussi) de nombreux jeux.

Ces deux moteurs de jeu (regroupant les moteurs graphique et physique) ont été largement utilisés et le fait qu'ils soient aujourd'hui dans le domaine libre nous permettra d'avoir de nombreuses aides (tutoriels et forum par exemple).

Un avantage cependant pour l'UnrealEngine 2 étant donné qu'il existe, pour ce moteur, des outils de développement fournis par Epic Games (les créateurs de ce moteur) comme par exemple un éditeur de niveau mais aussi tout l'éditeur de scripts permettant de programmer facilement l'IA des bots. Nous pensons utiliser cette solution car, d'une part, elle est plus confortable et, d'autre part, ce moteur reste encore utilisé dans le domaine professionnel (une très bonne expérience pour nous donc).

2.3 Environnement et mapping

On parle de mapping pour désigner le level design. Nous allons nous servir principalement de l'éditeur fourni avec l'UnrealEngine 2. Cela nous permettra d'être sûr de la pleine compatibilité avec le moteur.

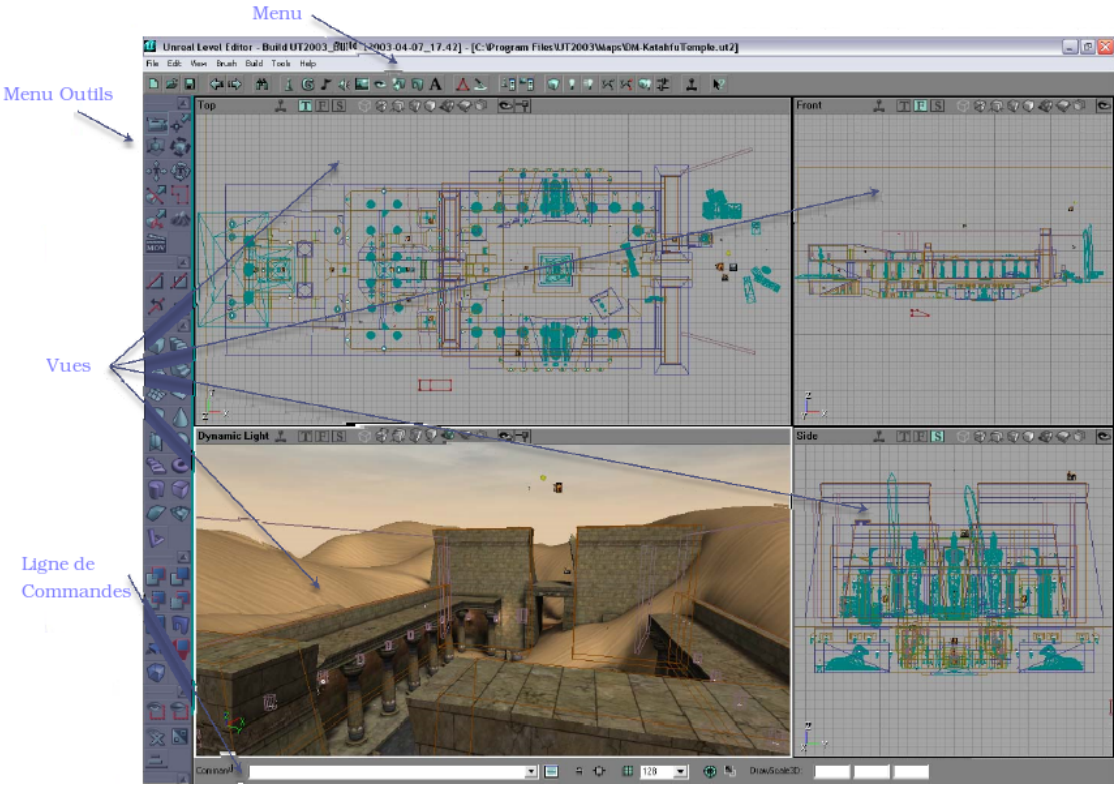


FIG. 3 – Interface de l'UnrealEd

Nous pensons aussi utiliser le logiciel Maya (animation, modélisation et rendu 3D) qui, bien que payant, reste le plus utilisé par les professionnels et nous permettrait d'acquérir une bonne expérience. On rappelle aussi que certains PC de l'université seront bientôt équipés de ce logiciel et mis à notre diposition.

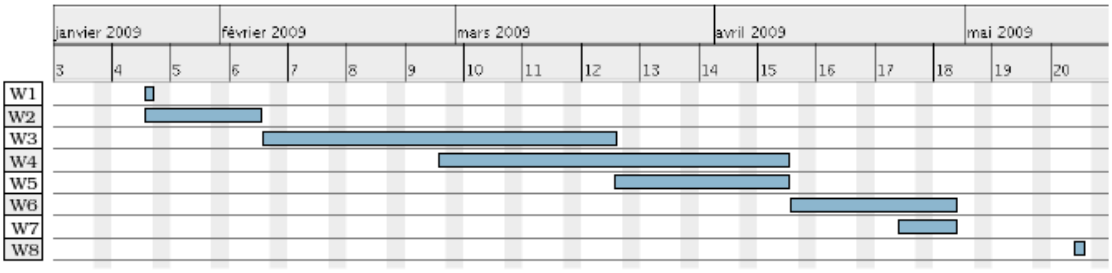
Nous ne nous servirons de cette solution que si nous n'arrivons pas à trouver notre bonheur dans le vaste éditeur inclus dans l'UnrealEngine 2 (il existe de base une grande quantité d'armes, de personnages et de textures).

Bien que cette partie ne soit pas celle qui nous prendra le plus de temps, nous essaierons de concevoir les environnements et décors qui peuplent le jeu de manière plausible, cohérente et variée.

3 Organisation du travail

À l'heure actuelle, l'avancement ne nous permet pas de savoir précisément qui s'occupera de quoi. Néanmoins, on peut d'ores et déjà fournir un diagramme de Gantt cohérent.

3.1 Diagramme de Gantt



W1	Prise de contact avec l'enseignant et démarrage du projet
W2	Rédaction du cahier des charges
W3	Prise en main du moteur choisi
W4	Implémentation de l'IA
W5	Design de quelques éléments graphiques necessaires au jeu (arme, perso...)
W6	Débug et finitions
W7	Rédaction du rapport
W8	Soutenances orales

FIG. 4 – Diagramme de Gantt

3.2 Possibilité d'organisation

La majorité des travaux devant être réalisée dans l'ordre, nous pensons les faire ensemble. Le seul moment où nous nous diviserons réellement sera au moment de la réalisation de l'IA. Comme le montre le diagramme de Gantt ci-dessus, et si l'avancement le permet, une équipe se concentrera d'avantage sur quelques éléments liés aux graphismes (animation des personnages, design d'armes et de personnages, etc.) tandis que l'autre finalisera l'IA.

Concernant nos méthodes de travail, nous nous sommes réservés plusieurs créneaux de temps libre, commun à tous, pour travailler sur le TER. De plus, nous allons créer un FTP pour mettre en commun tous les fichiers pour être sûr que tout le monde travail avec les bonnes versions (notamment durant les week-ends et vacances).

Suite à notre mésaventure récente (crash total d'un ordinateur avec impossibilité de récupérer les données), il nous semble qu'aujourd'hui, cette solution est indispensable.