ةيبعشلا ةيطارقميدلا ةيرئازجلا ةيروهمجلا
République Algérienne Démocratique et Populaire
يملعلا ثحبلا و يلاعلا ميلعتلا ةرازو
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Mohamed Khider – Biskra
Faculté des Sciences Exactes et des
Sciences de la Nature et de la vie
Département d'informatique
Réf :……

ةركسب رضيخ دمحم ةعماج
مولعلا ةيلكلا مولع و ةقيقدلاةايحلا و ةعيبط
ا مسقيلآا ملاعلإ
……. :عجرملا

Mémoire en vue de l'obtention
du diplôme de

Magister en informatique

Option : Synthèse d'image et vie artificielle

Intitulé :

Réalité Augmentée pour les Jeux Vidéo et les Serious Games

Présenté par :

GUEMOUGUI Abdessattar

Soutenu publiquement le : 26/05/2013

Devant le jury composé de :

Dr. BABAHENINI Med Chaouki	**Maître de Conférences 'A'**	**Président**	**Université de Biskra**
Pr. DJEDI NourEddine	**Professeur**	**Rapporteur**	**Université de Biskra**
Dr. BAARIR Zineddine	**Maître de Conférences 'A'**	**Examinateur**	**Université de Biskra**
Dr. CHERIF Foudil	**Maître de Conférences 'A'**	**Examinateur**	**Université de Biskra**
Dr. MELEKEMI Kamel Eddine	**Maître de Conférences 'A'**	**Examinateur**	**Université de Biskra**

Résumé

Dans ce mémoire, nous présentons l'implémentation d'un Serious Game qui permet d'apprendre les règles de la circulation routière. Notre implémentation est basée sur l'utilisation d'une interface multimodale qui utilise la réalité augmentée, la vision par ordinateur et les dernières technologies des Smartphones. En utilisant une architecture client-serveur, notre approche permet d'obtenir un rendu de haute qualité et un suivi optique en temps réel sur des Smartphones ayant des capacités de calcul réduites. Ainsi, nous combinons entre les capacités de calculs avancées d'un ordinateur de bureau (processeur, mémoire, carte graphique) et les atouts d'un Smartphone (mobilité, capteurs, écran tactile…etc.)

Mots-clés:
Réalité augmentée, Serious Games, rendu à distance, Smartphones.

Abstract

In this work, we present the implementation of a serious game that teaches the rules of the road. Our implementation is based on the use of a multimodal interface that uses augmented reality, computer vision and the latest smartphone technology. Using a client-server architecture, our approach provides a high-quality rendering and a real time optical tracking on moderate Smartphones. Thus, we combine between the advanced performance of a desktop computer (processor, memory, graphics card) and the benefits of a Smartphone (mobility, sensors, touch-screen ... etc.)

Keywords:
Augmented reality, Serious Games, remote rendering, Smartphones.

ملخص

في هذه المذكرة نقوم بعرض ما توصلنا إليه في مجال تصميم وتطوير ألعاب الفيديو الجادة، اللعبة التي قمنا بتصميمها تساعد الأطفال والشباب على تعلم قواعد المرور وضرورة احترامه. ويستند التصميم المقترح على دمج تقنيات الحقيقة المضافة، والرؤية الاصطناعية وتكنولوجيا الهواتف الذكية. باستخدام هندسة عميل ملقم، النموذج يوفر رسومات عالية الجودة، وتعقب في الوقت الحقيقي على الهواتف الذكية ذات القدرات المتوسطة. وبهذا قمنا بالجمع بين الأداء المتطور لجهاز كمبيوتر(المعالج والذاكرة وبطاقة الرسومات) ومزايا الهواتف الذكية (قابلية التنقل، أجهزة الاستشعار، شاشة اللمس ...الخ).

كلمات البحث:

الحقيقة المضافة، الألعاب الجادة، محاكات الرسومات عن بعد، الهواتف الذكية.

Remerciements

En ce moment clé de ma vie, je dois remercier tous ceux qui m'ont aidé pour l'accomplissement de ce travail, mes remercîments vont à mon encadreur, Mr NourEddine DJEDI, Professeur à l'Université Med Khider de Biskra qui m'a accordé l'honneur de diriger mon travail de Magister avec son œil avisé et son expérience, je le remercie pour ses conseils, sa disponibilité à tout moment, son aide ainsi que son sens critique. Je le remercie très particulièrement pour sa patience et son professionnalisme redoutable.

Je souhaite remercier également les membres du jury qui ont accepté de juger mon travail. Je les remercie pour le temps qu'ils ont accordé à la lecture de ce document et à l'évaluation de mon travail.

Enfin, je tiens à remercier ma famille pour le support et l'aide qu'elle m'a fournis, elle a toujours été là pour m'encourager, me soutenir sans conditions dans tout ce que j'entreprends.

Table des matières

Liste des figures

Liste des figures

Liste des figures

Liste des tableaux

Liste des extraits de code

Introduction générale

Contexte

La Réalité Augmentée (RA) désigne différentes techniques permettant d'intégrer, de façon réaliste, des entités numériques (objets 3D, images, textes, sons, etc.) dans le monde réel. La réalité augmentée peut être utilisée pour différents types d'applications, telles que l'affichage de l'itinéraire et les données de trafic directement sur le pare-brise de véhicule ou l'insertion de donnés d'un patient dans le champ de vision de son médecin.

La réalité augmentée est souvent associée à des visiocasques, qui sont également souvent utilisés pour la réalité virtuelle. Il s'agit d'un dispositif d'affichage monté sur la tête de l'utilisateur de telle sorte que ce dernier puisse voir les objets de synthèse et le monde réel en même temps. Le souci avec ces visiocasques est qu'ils ne sont pas encore couramment disponibles et commercialisés et cela a poussé les chercheurs à explorer d'autres plates-formes pour la réalité augmentée, telles que les téléphones mobiles, les PDA et les tablettes numériques.

L'une des applications intéressantes de la réalité augmentée est liée au domaine des jeux vidéo. La réalité augmentée ouvre l'horizon d'un grand nombre de possibilités dans ce domaine pour créer des jeux immersifs ou réalistes où le joueur peut interagir avec le jeu grâce à son environnement. L'utilisation des dispositifs mobiles (Smartphones, tablettes et PDA) peut aussi mettre le joueur dans un tout autre niveau de divertissement.

D'autre part, les jeux vidéo sont devenus un phénomène de société qui occupe une place prépondérante sur le marché mondial. D'après les chiffres de l'ESA (Entertainment Software Association), les consommateurs aux Etats-Unis ont dépensé 25 milliards de dollars sur les jeux vidéo en 2010 où l'industrie des jeux vidéo a généré 5,9 milliards de dollars de recettes pour l'économie américaine[26]. Toujours d'après l'ESA, l'âge moyen des joueurs est de 37 ans, ces joueur ont joué à des jeux pendant 12 ans.

dû au fait que les jeux vidéo ne peuvent plus, comme c'était encore le cas il n'y a pas si longtemps, être considérés comme un phénomène sous culturel marginal.

Depuis l'apparition des premiers jeux vidéo, de nombreuses études ont été menées sur leur impact sur le psychisme des enfants et des adolescents[21]. Ces études se sont concentrées tout particulièrement sur les effets négatifs que ces jeux peuvent avoir sur leur conception du monde et leurs attitudes: l'isolation et la violence. Les jeux vidéo sont accusés d'hypnotiser les jeunes, de provoquer une dépendance comparable à celle d'une drogue, de prôner la misogynie.

Depuis quelques années, on commence à se pencher sur une autre dimension du phénomène : les rapports entre jeux vidéo et psychologie cognitive, apprentissage, sociologie… On peut penser qu'une prise en considération de la capacité des jeux vidéo à motiver, placer l'apprenant dans des situations proches du réel et créer des situations de collaboration peut déboucher sur la conception de produits éducatifs réellement efficaces.

Cette tendance d'exploiter les jeux vidéo dans des secteurs non divertissants, résulte d'un nouveau champ de recherche en informatique : le Serious Game ou "jeu sérieux". L'objectif de cette discipline est d'immerger les joueurs, dans un système complexe et simuler ce qui masque derrière le jeu une finalité productive ou utilitaire (apprentissage, éducation, expérimentation, entraînement, concertation, création, sensibilisation, propagande…).

L'idée d'utiliser les jeux pour l'éducation n'est pas en elle-même nouvelle. Pendant longtemps, il y avait ce qu'on appelait jeux éducatifs, des jeux de mots ou de chiffres que cherchent à faire apprendre à l'enfant à compter ou à construire des mots par exemple [21, 22]. Les Serious Games imposent un changement de perspective, autrement dit, il ne s'agit pas de donner à l'enfant ou l'adolescent l'impression d' « apprendre en jouant » en conférant à des interfaces d'apprentissage de type classique un air ludique, comme c'est le cas la plupart du temps. On pourrait plutôt envisager des situations dans lesquelles l'apprentissage visé deviendrait le moyen par lequel il serait possible d'entrer et de progresser dans le jeu. Pour une génération élevée dans la technologie, l'interactivité, le contrôle et l'implication

dans la tâche.

Problématique

L'arrivée de dispositifs mobiles intelligents a augmenté de manière significative le nombre de personnes qui jouent aux jeux vidéo, quelques travaux de recherche ont étudié l'apport de la réalité augmentée pour les jeux vidéo durant ces dernières années. Néanmoins, parmi ces travaux rares, nous pouvons citer ceux qui s'intéressent à l'apport de la réalité augmentée mobile pour les serious games. Dans ce travail, nous essayons de contribuer à l'étude d'une éventuelle relation entre le domaine de la réalité augmentée mobile et celui des serious games. Pour cela nous suivons la démarche suivante :

D'abord, nous commençons par la conception et l'implémentation d'un prototype d'un serious game d'aide à l'apprentissage et le respect des règles de circulation routière, le serious game sera conçu et développé en mode réalité virtuelle.

Dans un deuxième temps, nous reprenons l'implémentation de notre prototype mais cette fois-ci en mode réalité augmentée mobile.

En proposant cette démarche, une question légitime peut se poser: pourquoi nous avons besoin de produire le prototype dans deux versions différentes (RV et RA) sachant que l'intérêt est pour la version en réalité augmentée, les points suivants peuvent servir d'éléments de réponse :

L'existence d'un même serious game en deux versions (RV et RA) permet de mener une étude comparative entre les deux et d'en déduire les avantages, les inconvénients et les éventuels problèmes relatifs à l'utilisation de RA pour les serious games.

Sur le plan conception et réalisation, cette démarche nous a permis de séparer les problèmes de différentes natures. Dans la première partie, nous nous intéressons uniquement au problème de la conception et de l'implémentation de notre serious game et en deuxième partie nous focalisons plus sur le problème d'adaptation de

notre serious game à la réalité augmentée mobile.

Plan du document

Ce document est organisé en deux grandes parties, la première tient sur deux chapitres formant un état de l'art sur la réalité augmentée et les serious games. Dans la deuxième partie nous décrivons notre démarche pour l'implémentation d'un serious game en réalité augmentée qui permet d'apprendre les règles de la circulation routière.

Le premier chapitre de notre mémoire permet de présenter la réalité augmentée, ses applications ainsi que les dernières avancées technologiques dans ce domaine. Le deuxième chapitre introduit le concept des serious games et leur applications dans l'apprentissage, une très bonne partie de ce chapitre sera consacrée à la présentation de nombreux exemples de serious games existants.

Dans le troisième chapitre, nous entamons la conception et la réalisation de notre serious game d'aide à l'apprentissage et le respect des règles de circulation routière en version RV. Nous commençons ce chapitre par l'établissement des grandes lignes de scénarios et règles de jeu, puis nous définissons l'ensemble d'outils que nous utilisons pour implémenter notre prototype. Consécutivement, le chapitre enchaîne sur les détails d'implémentation relatifs à la mise en œuvre d'un jeu vidéo, à savoir, la modélisation, l'IA, la simulation physique, l'interface utilisateur et l'interaction…etc.

Dans le quatrième chapitre, nous détaillions la démarche d'adaptation de notre serious game à la réalité augmentée mobile où nous discutons les principaux changements au niveau de l'architecture de notre jeu.

Pour terminer, nous concluons par un chapitre de synthèse des travaux effectuées afin de présenter les résultats obtenus et en dégager les apports en évoquant diverses pistes et perspectives.

Chapitre I Réalité augmentée : état de l'art

I.1 Introduction

Dans ce chapitre, nous définissons le concept de réalité augmentée (RA), nous passons en revue les principaux domaines d'applications de la réalité augmentée, puis nous présentons et analysons les déférentes configurations matérielles et plateformes logicielles couramment utilisées pour la mise en œuvre des applications de réalité augmentée.

I.2 Définition

Le concept de réalité augmentée (RA) est généralement lié à celui de réalité virtuelle (RV) pour laquelle l'utilisateur est immergé dans un monde complètement synthétique (généré et contrôlé par ordinateur) dit monde virtuel. Dans ce monde virtuel, les sens de l'utilisateur (vue, ouïe, toucher, odorat...etc.) sont contrôlés par ordinateur où il ne peut plus communiquer avec l'environnement réel qui l'entoure. En revanche, la réalité augmentée permet de combiner des objets réels et virtuels de façon à ce que ces deux types d'objets paraissant coexistant de manière cohérente au sein d'un même environnement. Les objets de synthèse enrichissent la perception que les utilisateurs ont de leur environnement en fournissant ou en facilitant l'accès à des informations inaccessibles directement. Ils permettent, par exemple, de faire apparaître des objets réels cachés, de présenter des annotations graphiques ou textuelles ou encore de superposer des informations issues de capteurs (de température par exemple).

Azuma[1] a proposé de définir la réalité augmentée comme tout un système respectant les trois critères suivants :

- Combiner réel et virtuel ;
- Fonctionner en temps réel ;
- Donner l'apparence que les objets réels et virtuels cohabitent dans le même monde tridimensionnel (alignement 3D).

Il y'a lieu, néanmoins, d'émettre une remarque non moins importante. Celle-ci est liée au fait que la majorité des applications de réalité augmentée développées à nos jours ne se sont focalisées que sur le principe d'augmentation visuelle (objet virtuel superposé sur une vidéo du monde réel). Cela n'empêche pas de dire que la réalité augmentée ne se limite pas qu'à ce type d'application mais elle pourrait potentiellement la transcender pour s'étendre à tous les autres sens, à savoir l'ouïe, le goût, l'odorat et le toucher.

I.3 Les domaines d'application de la réalité augmentée

Nous donnons ici un aperçu des différents domaines d'applications de la réalité augmentée. Le lecteur pourra se référer à la synthèse élaborée par Azuma[2] pour une liste plus compète d'applications et de travaux significatifs réalisés durant ces dernières années.

I.3.1 Le domaine médical

L'application de la réalité augmentée dans le domaine médical s'avère très prometteuse, les médecins peuvent utiliser la réalité augmentée en tant qu'outil de visualisation, d'entraînement et d'assistance chirurgicale. Ainsi, il est possible de collecter des données médicales sur le patient issues d'images IRM, d'images CT ou d'images ultrasonores, ces données peuvent être par la suite rendues et combinées avec un vue réelle du patient offrant ainsi au médecin une 'vision à rayon X'[1] intérieure du patient. Cette vision intérieure est très utile dans le cas d'une chirurgie à intrusion minimale car elle offre au médecin une meilleure vision de l'intérieur sans besoin de faire recours à des techniques plus intrusives.

La Réalité Augmentée peut être aussi utile pour l'apprentissage et l'entraînement de futurs chirurgiens, par exemple, les différentes étapes d'une opération peuvent être directement superposées sur le corps du patient.

Dans ce contexte nous pouvons citer le système Medarpa [3]. Les chirurgiens utilisant ce système sont capables de voir des informations médicales

[1]Capacité fictive de voir à travers les murs ou les objets

sous forme de modèles tridimensionnels directement sur le patient, sans porter des dispositifs d'affichage qui peuvent être encombrants. Le chirurgien pourra ainsi effectuer son intervention sans devoir détourner le regard du patient en consultant directement les informations sur une fenêtre semi transparente appelée "fenêtre virtuelle" et cela à tout instant (Figure I-1).

Figure I-1 le système MEDARPA

I.3.2 La maintenance et assemblage industriel

Les tâches d'assemblage et de maintenance des systèmes et des équipements complexes représentent l'un des domaines cibles de l'utilisation de RA, les principales perspectives attendues de cette utilisation sont le gain en productivité et une réduction des taux d'erreurs. Ainsi, les instructions de montage et de réparation apparaissent sous forme des modèles 3D superposés sur la pièce à manipuler exprimant les étapes de travail une par une. Les coûts, parfois importants, de maintenance (stockage et mise à jour) des manuels papiers sont également réduits. Plusieurs évaluations ont montré l'apport des systèmes de Réalité Augmentée pour la productivité dans les chaînes de montages et pour la maintenance et la réparation dans des environnements industriels.

Plusieurs travaux ont été développés dans cette catégorie, parmi lesquels nous pouvons citer KARMA [4] (*Knowledge-based Augmented Reality for Maintenance Assistance*) qui est un prototype du laboratoire expérimenté par l'équipe de Freiner

au début des années 1990, l'objectif de ce prototype étant la maintenance d'une imprimante laser. Elle a pour fonction deux tâches : le remplissage du bac à papiers et le changement de la cartouche de toner. Grâce au casque de visualisation, les pièces à manipuler sont mises en évidence par leur contour en fil de fer. Le monde virtuel vient en surimpression de la vue du monde réel par le biais de deux écrans, ces deux écrans étant pilotés par un ordinateur capable de repérer la position et l'axe de vue de l'utilisateur ainsi que la position de certains objets de l'entourage. Des objets tridimensionnels et en fil de fer sont synthétisés sur les deux écrans et fournissent une information virtuelle plaquée sur les objets réels (Figure I-2).

Figure I-2 Le système KARMA

I.3.3 Domaine audio-visuel.

La réalité augmentée est déjà présente dans notre quotidien télévisuel. Cela touche pour l'instant, les domaines commercial et sportif, pour lesquels l'apport de certaines informations pouvant être qualifiées d'intéressant. C'est particulièrement le cas pour les retransmissions télévisuelles de sports en direct, comme pour les courses automobiles où les images augmentées en temps réel permettent d'informer le téléspectateur sur le numéro, le nom du pilote et la vitesse de chaque voiture, ou pour des matchs de football américain en ajoutant des lignes au sol donnant des précisions sur la distance des joueurs sur le terrain (Figure I-3).

La publicité, très présente dans les grandes rencontres sportives, en général internationales, apparaît sous forme de panneaux publicitaires sur les bords du terrain. La réalité augmentée permet alors d'afficher des publicités différentes en fonction du pays dans lequel le téléspectateur se trouve.

Figure I-3 La réalité augmentée dans les matchs de football.

I.3.4 Domaine des jeux vidéo

De nouvelles recherches s'orientent sur l'application de la réalité augmentée dans les jeux vidéo. En effet, cette technique permet par exemple de donner l'impression de jouer à des jeux de sociétés classiques (et tous les avantages que cela comporte) mais avec toutes les possibilités offertes par les jeux sur ordinateur. Ainsi des recherches sont menées, au centre pour traitement des données graphiques (ZGDV, Darmstadt, Allemagne) sur le célèbre jeu Tetris version 3D et multi utilisateurs[5](Figure I-4).

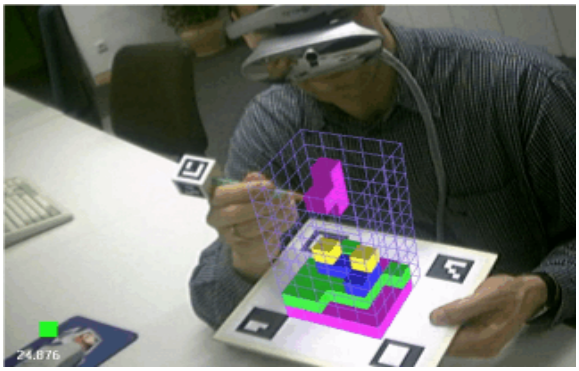


Figure I-4 le jeu Tetris3D

ARQuake[6] est l'une des premières expérimentations de Réalité Augmentée pour le jeu vidéo. Le jeu se déroule en extérieur et les éléments virtuels sont intégrés dans l'environnement physique en combinant différents capteurs (GPS, compas numérique et suivi visuel de marqueurs placés dans l'environnement). La disponibilité de caméras bon marché (Webcam) et la puissance de calcul

grandissante des consoles de jeux et des ordinateurs permettent l'essor de jeux utilisant l'interaction vidéo et la superposition d'objets virtuels.

Le terrain invisible [7] est un autre jeu développé dans les laboratoires de l'université de Vienne, Le jeu est collaboratif mobile où les joueurs contrôlent un train virtuel évoluant sur une vraie maquette en bois représentant des rails, ce train virtuel est visible seulement aux utilisateurs à travers leurs PDA[2] (Figure I-5).

Figure I-5 (a) Contrôle du train virtuel à partir du PDA (b) l'aspect collaboratif du jeu.

I.3.5 Domaine de conception

Plusieurs travaux ont montrés que la réalité augmentée peut être utilisée en tant que technique de conception très efficace et intuitive. Dans ce contexte, nous pouvons citer les travaux suivants :

Le système Build-IT (*Tangible Interaction for Collaborative Design*) [8] qui est un système permettant à un groupe de coopérer afin de concevoir l'aménagement d'objets du monde réel tel que les écoles, les usines…

Les utilisateurs peuvent interagir autour d'une table à l'aide de briques physiques leur permettant de sélectionner et de manipuler des entités virtuelles qui sont projetées sur une table (Figure I-6.a). Plusieurs briques peuvent être utilisées simultanément; ce qui permet de travailler à plusieurs en même temps mais aussi

[2]Terme anglais (Personal Digital Assistant) dont l'équivalent français est "assistant numérique personnel". Il désigne de très petits micro-ordinateurs de poche.

d'avoir des interactions à deux mains permettent des actions complexes (Figure I-6.b).

Figure I-6 Le système Build-IT (a) projection des objets virtuels sur la table du travail (b) interaction à deux mains.

Le système Ambiente[9] est un autre système pour l'aide à la conception, il propose un environnement collaboratif dédié à la conception (Figure I-7). Il est principalement composé de quatre éléments basés sur des technologies à écrans actifs : un écran mural de grande surface, une table tactile, des sièges de travail personnels et des tables connectables. Chacun des éléments définit des métaphores spécifiques et supporte l'échange de données avec les autres éléments.

Figure I-7 Plusieurs utilisateurs coopérant entre eux avec les dispositifs du système Ambiente.

I.3.6 Autres domaines d'applications

Les musées augmentés représentent une autre application de la RA. Cette application utilise le dispositif du Navicam [10], qui affiche des données dans un casque semi transparent porté par un visiteur du musée. Le système est basé sur la lecture d'un code barre et affiche dans un coin les informations relatives aux œuvres. La Figure I-8 montre la vue d'un utilisateur face à un tableau du musée. A tout moment, au cours de la visite, l'utilisateur voit les œuvres réelles ainsi que des données complémentaires ou augmentations, affichées dans le casque et restituées par l'ordinateur. Mis à part la phase de configuration du système, l'interaction entre l'utilisateur et le système est totalement transparente aux visiteurs.

Figure I-8 Le musée augmenté.

Le système ARLibrary utilise le dispositif *Ubiquitous Talker* qui est une version étendue de Navicam. Il permet d'aider l'utilisateur à sélectionner et localiser des publications dans une librairie. En lançant une recherche d'un livre ou d'un magazine spécifique à travers une interface de recherche intuitive, le système fournit à l'utilisateur la position du livre ou du magazine (Figure I-9).

Figure I-9 Le dispositif Ubiquitous Talker utilisé en tant que guide dans une librairie.

I.4 Architecture des systèmes de réalité augmentée

Dans cette partie, nous allons présenter en détail les différents composants d'un système de réalité augmentée.

I.4.1 Le suivi et l'alignement

Le mixage est la composante essentielle d'un système de réalité augmentée, il consiste en l'intégration (d'un point de vue visuel) entre le monde réel et le monde virtuel. L'objectif est de maintenir les objets virtuels (souvent des éléments graphiques 3D) alignés avec les objets physiques du monde réel dans le repère de visualisation de l'utilisateur: c'est le recalage (registration). Pour parvenir à réaliser cet alignement de façon correcte et consistante, le système doit déterminer la position et l'orientation de l'utilisateur ainsi que les objets dans l'environnement: c'est le suivi (en anglais Tracking). Ainsi, on peut définir le suivi et le recalage comme l'un des problèmes fondamentaux de la réalité augmentée [11].

I.4.1.1 Le suivi (tracking)

Le suivi précis, rapide et robuste de l'utilisateur et des objets de l'environnement et un élément déterminant dans toute application de réalité augmentée. Nous présentons et analysons ci-après les différentes approches existantes relatives au processus de suivi.

Le suivi à base de capteurs

Il s'agit d'utiliser des capteurs pour déterminer la position de l'utilisateur dans l'environnement, il existe plusieurs types de capteurs : magnétiques, acoustiques, inertiels, accéléromètres, compas, GPS, …etc. On peut distinguer deux types de suivi [12]: Le suivi du type *outside-in* et le suivi *inside-out*. Le premier type (*outside-in*) fait référence aux systèmes qui fixent des capteurs dans l'environnement pour suivre des émetteurs placés sur des cibles mobiles. Le second type, (*inside-out*), utilise des capteurs attachés aux objets en mouvement qui sont capables de déterminer leur position par rapport aux émetteurs fixés dans l'environnement.

Ces techniques ont l'avantage d'être robustes aux mouvements abrupts (mouvement rapide de la caméra), mais ils ont aussi l'inconvénient d'être sensibles aux perturbations de l'environnement.

Le suivi visuel

Grâce aux progrès dans le domaine de la vision par ordinateur, des approches de suivi dites visuelles basées sur la détection des positions par des caméras sont devenues très populaires. Une première approche de suivi dite par marqueurs utilise des motifs facilement reconnaissables [13] dont l'avantage majeur est qu'ils ne sont pas excessifs en terme de puissance de calcul car l'analyse d'une seul image permet de détecter la position des marqueurs. La Figure I-10 montre quelques exemples de marqueurs utilisés

Figure I-10 Exemples de marqueurs.

Une autre approche de suivi visuel dite 'sans marqueurs' consiste à calculer la position et l'orientation de la caméra en se basant sur la taille et la déformation

d'un objet connu identifié par analyse d'image [14]. Cette approche impose qu'au moins un objet de taille suffisante soit présent dans le champ de vision de la caméra. Formellement, déterminer la position et l'orientation de la caméra se traduit en la détermination de la matrice de projection qui permet d'obtenir l'image 2D (de la caméra) à partir de la forme 3D de l'objet (connu au préalable). Les systèmes de suivi visuel sans marqueurs artificiels restent toujours les plus difficile à mettre en œuvre mais aussi les plus prometteurs pour les applications futures da la réalité augmentée.

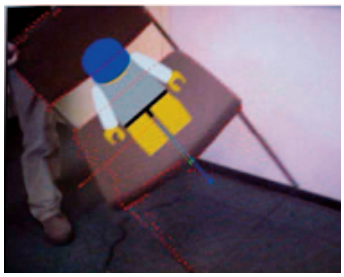


Figure I-11 suivi d'une chaise sans utilisation de marqueurs

Ces approches sont moins robustes aux mouvements abrupts de la caméra car ces mouvements ne permettent pas de suivre les indices 2D (primitives géométriques 2D extraites de l'image) entre des images consécutives.

Les approches hybrides

En absence d'une solution optimale, la tendance actuelle est d'utiliser des approches de suivi hybride dans lesquelles une approche basée vision est couplée avec une assistance basée capture [15].

I.4.1.2 L'alignement

L'un des problèmes les plus importants dans les applications de la réalité augmentée est celui de l'alignement virtuel/réel. Les objets dans le monde virtuel doivent être superposés correctement sur les objets réels, de telle manière à ce que l'utilisateur ait l'impression que les deux mondes réel et virtuel coexistent ensemble.

Les erreurs d'alignement sont difficiles à contrôler et constituent un des

principaux verrous de la RA. Azuma[1] définit deux types d'erreurs: statique et dynamique. L'erreur statique apparaît quand le point de vue de l'utilisateur et l'objet dans le monde restent fixe. L'erreur dynamique apparaît quand le point de vue de l'utilisateur ou de l'objet est en mouvement dans l'environnement.

Dans les systèmes actuels, les erreurs dynamiques sont généralement la source principale des erreurs d'alignement.

Alignement spatial : erreurs statiques

On définit l'erreur statique par l'erreur d'alignement entre objet réel et objet virtuel, déterminée lorsque l'utilisateur est en position statique. Selon Holloway [16], on peut distinguer trois types d'erreurs statiques qui sont définies comme suit :

– la distorsion optique,

– l'erreur du système de suivi,

– et la mauvaise estimation des paramètres (champ de vue, distance œil/capteur, distance inter-pupillaire, etc..).

La distorsion optique est un problème bien connu qui est dû aux propriétés des lentilles des caméras. L'erreur du système de suivi est générée par les capteurs électromagnétiques sensibles aux distorsions du champ magnétique. La mauvaise estimation des paramètres est l'erreur la plus étudiée en réalité augmentée. Une estimation fiable nécessite de mettre en œuvre des méthodes robustes basées sur une définition de contraintes dans la procédure de calibration. À partir d'une modélisation du système, on s'intéresse alors à la calibration de ces différentes parties. On peut alors utiliser des méthodes automatiques (par vision, auto-calibrage) ou des méthodes basées utilisateurs (méthodes simples et efficaces).

Alignement temporel : temps de latence

Le recalage dynamique des images de synthèse sur le monde réel est une étape cruciale dans l'élaboration d'un système de réalité augmentée. Les données des différents systèmes d'acquisition, tels que les dispositifs de suivi d'objets et de formation d'images, doivent être enregistrées sous un aspect spatio-temporel avec le point de vue de l'utilisateur. Chaque dispositif induit un retard associé à

l'observation du monde et à l'affichage sur le système de visualisation. La différence des retards entre les divers dispositifs d'acquisition ou systèmes de traitement est appelée : latence relative. Elle représente une source du mauvais repérage et doit être réduite. Les sources de retard sont classifiées en six catégories et représentées par : le retard d'arrivée des données, le retard de traitement, le retard du rendu, le retard d'affichage, le retard de synchronisation et le retard de mise à jour.

I.5 Le système de visualisation de RA.

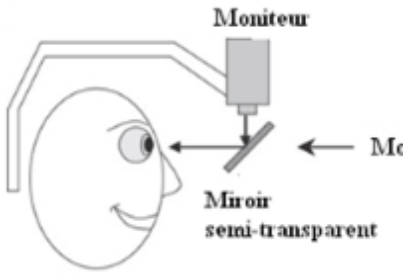
La technologie de visualisation représente également un élément important dans le domaine de la réalité augmentée, elle permet de visualiser le résultat de superposition réel/virtuel. On peut distinguer différents types de systèmes.

I.5.1 Les visiocasque.

Les visiocasques ou HMD (*Head Mounted Display*) sont les dispositifs d'affichage les plus utilisés actuellement pour les applications de réalité augmentée des systèmes couplés à la tète de l'utilisateur. On peut distinguer entre trois types de visiocasques: les visiocasque semi transparent (*optical see-through head-mounted displays*), les visiocasque vidéo (*video see-through head-mounted displays*) et les visiocasques rétiniens.

I.5.1.1 Les visiocasques optiques semi transparents

Ces dispositifs sont équipés de miroirs semi transparents qui permettent à l'utilisateur de voir par transparence le monde réel, l'augmentation virtuelle provient de petits moniteurs situés latéralement ou au-dessus des miroirs.



(a)

Figure I-12 Visiocasque semi transparent : (a)principe de fonctionnement, (b) prototype expérimental

Ce type de visiocasque présente l'avantage de garder une vision naturelle du monde réel (résolution et angle de vue de la vue naturelle). Néanmoins[17], il pose des problèmes dans la gestion d'occultation car le mixage est difficilement contrôlable, il nécessite aussi une calibration par l'utilisateur et un suivi précis de l'orientation de la tête. Un autre problème est connu aussi avec ce type de dispositifs, il s'agit du problème de focal fixe car le monde réel et les objets virtuels sont perçus dans deux profondeurs différentes, cela force l'œil humain de changer constamment le focus entre les deux niveaux de profondeur.

I.5.1.2 Les visiocasques vidéo

Ce type de casque est équipé d'une ou deux caméra filment le monde réel, les objets virtuels son par la suite superposé sur la vidéo et le résultat et affiché sur de petits moniteurs de type LCD situés devant l'œil de l'utilisateur.

Figure I-13 Visiocasque vidéo : (a) principe de fonctionnement, (b) prototype expérimental.

Lié aux casques semi transparents[17], le problème d'occultation peut être géré lors du mixage. Ainsi, le problème de focus ne se pose plus car il y'a une seule profondeur, celle du plan image. Néanmoins, le problème qui se pose avec les visiocasques vidéo réside dans la résolution relativement médiocre des écrans LCD utilisés, ces systèmes souffrent aussi d'un angle de vue relativement limité.

I.5.1.3 Le visiocasque rétinien

Récemment, un troisième type de visiocasque appelé VRD (*virtual retinal display)*ou RSD*(retinal scan display)*est proposé[18]. Un tel dispositif projette les informations directement sur la rétine de l'utilisateur en balayant cette dernière par trois faisceaux lumineux. Ce dispositif fonctionne d'une façon comparable aux écrans CRT. En dépit de sa complexité, cette technologie présente les avantages suivants :

- Un champ de vision plus large comparé aux deux visiocasques précédents;
- Les objets virtuels apparaissent avec une grande luminosité et une haute résolution;
- Possibilité de miniaturisation (intégration dans des lunettes légères);
- Vision stéréoscopique.

Figure I-14 Visiocasque rétinien : (a) principe de fonctionnement, (b) prototype expérimental.

I.5.2 Les systèmes d'affichage de type écran

Un écran est généralement couplé avec une caméra pour former une configuration minimale d'un système de réalité augmentée. Un écran d'un ordinateur de bureau avec une caméra peut être vu comme une plateforme de RA. Néanmoins, il n'est pas envisageable de réaliser des applications de RA avec cette configuration. Les Tablettes PC, les assistants personnels (PDA) et les téléphones portables semblent des technologies plus prometteuses pour la réalité augmentée [19], car ils regroupent dans un seul équipement le traitement, la mémoire, les mécanismes d'interaction, de communication (Bluetooth, GPRS, Wifi) et même de suivi (GPS, accéléromètre et boussole) (Figure I-15). Mis à part ces caractéristiques

techniques, ces équipements présentent l'avantage d'être mobiles, légers et disponibles pour le grand public. Pour ce type d'équipement, une approche dite (*videosee-throught*) est la plus adaptée, la caméra embarquée capte le flux vidéo et le processeur lui superpose des augmentations et le résultat est affiché à l'écran.

Figure I-15 L'IPad 2 utilisé comme dispositif de réalité augmentée.

Ces dispositifs souffrent de quelques problèmes qu'on peut résumer dans les points suivants :

- La taille de l'écran est généralement petite, ce qui conduit à une résolution insuffisante ainsi qu'un champ de vision limité ;
- L'impuissance des processeurs utilisés surtout en traitement en virgule flottante peut conduire à des temps de réponse relativement longs. Pour contourner ce problème, la solution alternative consiste en l'utilisation d'une architecture client-serveur où un serveur peur prendre en charge la tâche de calcul et renvoie les résultats au dispositif de l'utilisateur ;
- Comparés à des visiocasques, les dispositifs portés à la main ne permettent pas une application complètement mains-libres.

I.5.3 Les systèmes de visualisation par projection

Dans ce type d'affichage, l'augmentation est projetée directement sur les objets réels. On peut ici distinguer entre deux types de projections, le premier type est utilisé dans le cas où l'augmentation affecte la surface de l'objet réel (couleur, texture ou illumination), dans ce cas une projection simple est suffisante pour donner l'effet nécessaire. Le deuxième type de projection est utilisé si l'augmentation est une forme tridimensionnelle qui doit apparaître devant ou au-dessus de la surface par exemple : dans ce cas, une stéréo-projection qui prend en considération la position et l'orientation de l'utilisateur est nécessaire (Figure I-16).

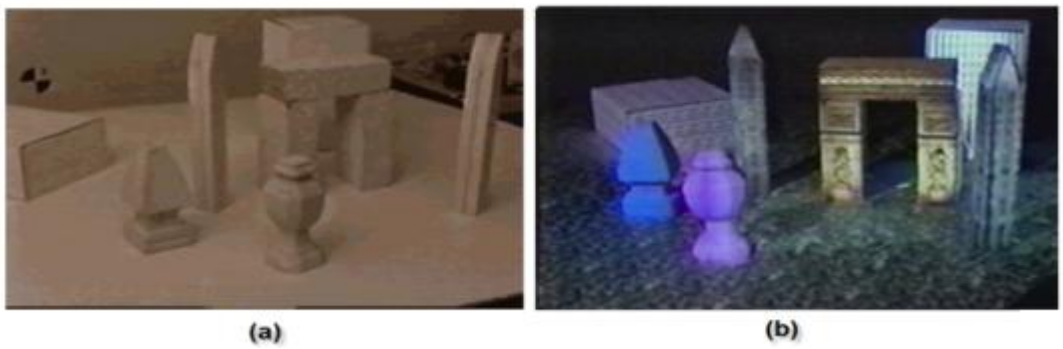


Figure I-16 (a) Scène réelle (b) augmentation par stéréo projection

Les systèmes de visualisation basés sur la projection souffrent aussi de quelques problèmes [11] :

- La projection frontale n'est pas efficace s'il y a des objets physiques ou des utilisateurs entre le projecteur et la surface de projection, dans ce cas une configuration multi projecteur peut résoudre le problème ;
- Les projecteurs conventionnels fonctionnent avec un seul plan focal, la projection sur une surface non plane peut générer un effet de bavures, les projecteurs laser ne souffrent pas de ce problème ;
- La projection est limitée par la forme, la taille et la couleur de la surface de l'objet réel.

I.6 Conclusion

La réalité augmentée est un domaine en plein essor et présente un domaine où le potentiel d'exploitation est bien réel, notamment dans le domaine éducatif et médical. Dans ce chapitre, nous en avons présenté un état de l'art et nous avons tenté de mettre en évidence les bases matérielles qui y sont associées ainsi que son principe de fonctionnement. Dans un second temps, nous avons présenté les domaines d'application liés à la réalité augmentée.

Chapitre II Les serious games : état de l'art

II.1 Introduction

Dans ce chapitre, nous présentons un état de l'art sur les serious games. Dans une première partie, nous essayons d'adopter une définition du terme serious game, en nous basant sur les différents définitions proposées dans la littérature. Dans une seconde partie, nous donnons un historique relativement cours sur l'apparition des serious games. Nous finissons ensuite en présentant la différence entre un jeu vidéo ordinaire et un serious game. Enfin, nous allons conclure ce chapitre par une classification des serious games.

II.2 Définition des Serious Games

Afin de trouver une définition du terme serious game, nous examinons quelques définitions proposées dans la littérature. Michael Zyda directeur du laboratoire Game Pipe consacré à l'étude des serious games proposa la définition suivante:

« *Un défi cérébral, joué avec un ordinateur selon des règles spécifiques, qui utilise le divertissement en tant que valeur ajoutée pour la formation et l'entraînement dans les milieux institutionnels ou privés, dans les domaines de l'éducation, de la santé, de la sécurité civile, ainsi qu'à des fins de stratégie de communication.* » [20].

Amoto[21] définit les serious games comme étant des jeux vidéo utilitaires, c'est-à-dire productifs, dont la conception vise à opérer une transformation chez leurs destinataires allant dans le sens d'une amélioration des compétences (entraînement), de l'adaptation au milieu (traitement des phobies), de la compréhension d'un phénomène (éducation) ou d'une plus grande adhésion au message véhiculé (promotion, publicité, jeux vidéo idéologiques, dits aussi *political games*).

Julian Alvarez combine les deux premières définitions et propose la définition suivante du serious game: *«C'est une application informatique dont l'objectif est de combiner à la fois des aspects sérieux (Serious) tels, de manière non exhaustive, l'enseignement, l'apprentissage, la communication, ou encore*

telle association a donc pour but de s'écarter du simple divertissement» [22].

II.3 Le concept de scénario pédagogique

Le conseil supérieur de l'éducation québécois a proposé la définition suivante du didacticiel : un logiciel ou programme, spécialisé dans l'enseignement d'une discipline, d'une méthode, de certaines connaissances et utilisé en enseignement assisté par ordinateur.

Après la définition des serious games et du didacticiel, ils semblent donc qu'ils partagent à eux deux les fonctions d'enseignement et d'apprentissage. Cependant, d'un part les serious game semble adopter un panel plus large par l'intégration des autres domaines de la communication et de l'informatique.

D'autre part, si le didacticiel peut revêtir facultativement un aspect ludique, défini par le genre ludoéducatif, le serious game, quant à lui, au niveau de sa conception, d'après [20], fait appel nécessairement à des références liées au jeu vidéo. Cependant, pour établir une comparaison entre ces deux concepts, nous nous référons aux écrits de Zyda[20] : celui-ci part du postulat qu'un jeu vidéo est défini par "l'histoire, l'art et le logiciel". Puis, il précise que les serious games "sont cependant, plus qu'une histoire, de l'art et du logiciel, ils impliquent la pédagogie : *des activités qui éduquent ou instruisent, diffusant de ce fait de la connaissance ou de la compétence*. Cet ajout rend les jeux sérieux."

Après cette définition, nous constatons que les serious games ajoutent le scénario pédagogie dès la conception de l'application pour répondre à un objectif pédagogique, ce qui permet de spécifier les serious games sur le plan informatique.

Tricot propose une définition du scénario pédagogique appliqué au serious game. Le scénario pédagogique est une fonction dédiée à un "objectif pédagogique", dont la propriété est de susciter l'envie d'apprendre et dont la réalisation dépend d'un jeu vidéo avec lequel elle puisse s'intégrer[23].

Et nous concluons cette partie par une définition du serious game qui se propose d'ajouter l'objectif pédagogique,

Un serious game est une application informatique, dont l'objectif est de combiner à la fois des aspects sérieux (Serious) tels, de manière non

l'information, avec des ressorts ludiques issus du jeu vidéo (Game). Une telle association, qui s'opère par l'implémentation d'un "scénario pédagogique", a donc pour but de s'écarter du simple divertissement.

La Figure II-1 exprime le schéma établi par Zyda pour illustrer ses propos, il montre la dimension pédagogique qui vient compléter le jeu vidéo pour donner naissance à un serious game.

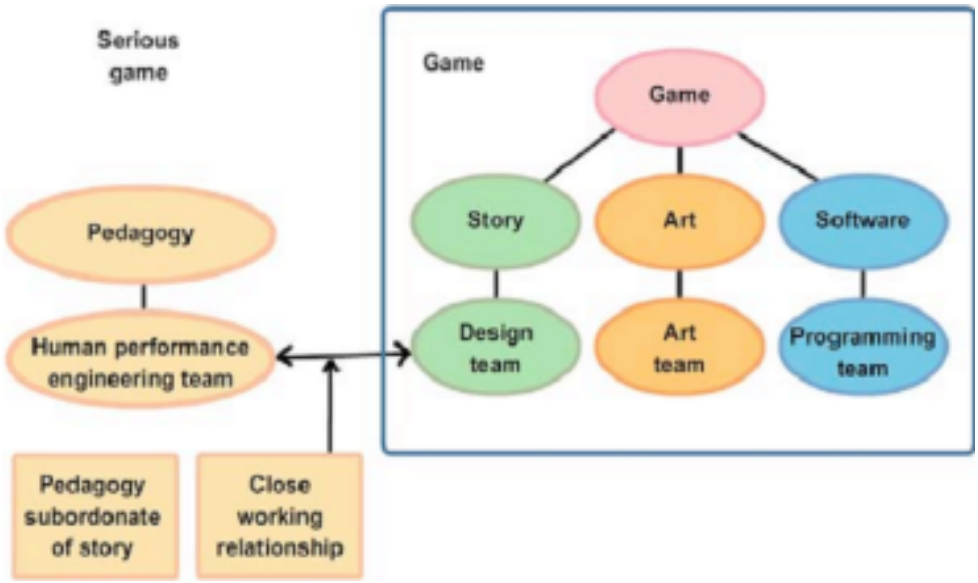


Figure II-1 Serious game : schéma représentant le lien entre le jeu vidéo et la composante pédagogique en vue d'élaborer un Serious Game[20]

II.4 Apparition des serious games

Dès le XV^{ème} siècle, avec le mouvement humaniste en Italie, on recense l'oxymore "Serio Ludere". Ce terme renvoie à l'idée de traiter un sujet "sérieux" avec une approche "amusante". Cela se retrouve ainsi dans le style littéraire où un ton léger et humoristique peut dénoncer des problèmes de société. Par exemple, en France, Montaigne est un humaniste qui fait notamment usage du Serio Ludere. Vers la fin du XVIIIe siècle et le début du XIXe, l'amirauté britannique et l'armée prussienne s'intéressent sérieusement à l'emploi de la simulation ludique pour développer de nouvelles tactiques et former leurs futurs cadres. Ainsi, en 1820, la Prusse va adopter le 'Kriegspiel' (ou 'wargame' en anglais et 'jeu de guerre' en français) comme outil de formation de ses officiers. Jusqu'à l'émergence et le développement récents du concept de 'Serious Games' dans le domaine informatique, 'Wargame' sera le principal jeu sérieux employé par la quasi-totalité

des armées du monde.

Pour trouver le concept moderne du Serious Game, il faut attendre les années 70, avec l'œuvre éponyme de Clark Abt [28]. Dans ses écrits, ce chercheur américain, voit dans le jeu de société, le jeu de plein air, le jeu de rôle et le jeu sur ordinateur (très peu développé à cette époque), des supports pour diffuser des messages éducatifs, politiques, de marketing, etc. Sa participation au développement de TEMPER, l'un des premiers wargames informatisés destiné à prendre en compte le contexte de la guerre, ne doit pas être étrangère à l'intérêt de ce chercheur pour tous les types de serious game.

L'approche actuelle du serious game date de 2002 et a démarré avec *America's Army*[24]. Ce titre est représentatif de l'ensemble des fonctionnalités que l'on peut associer à un serious game : diffuser un message, dispenser un entraînement, permettre l'échange de données. Cependant, on recense des applications vidéo ludiques qui associent de telles fonctions utilitaires bien avant cette date.

II.5 Évolution des serious games

Le concept qui consiste à utiliser des jeux à des fins éducatives remonte à bien avant l'apparition des ordinateurs. On considère que le premier serious game est *Army Battle zone*, élaboré par Atari en 1980 pour entraîner les militaires, et qui n'a pas eu un grand succès.

Ces dernières années le gouvernement américain ainsi que son département de défense se sont adressés à plusieurs reprises aux développeurs de jeux vidéo pour la création de simulateurs à faible coût à la fois réalistes et impliquant. Ces jeux devaient permettre aux militaires d'appréhender les réalités de la guerre en utilisant les infrastructures déjà existantes.

L'intérêt de l'armée américaine dans cette démarche est double, d'une part, les développeurs bénéficient d'une grande expérience dans les jeux vidéo classiques. Et d'autre part, la création de ce type de simulations coûte des millions de dollars de moins que les simulations traditionnelles qui bien souvent requièrent un équipement spécifique.

Les coûts de développement de jeux « sur mesure » sont abordables pour de

américain. Devant le succès de ces premiers serious games, d'autres secteurs se sont montrés intéressés. Aujourd'hui, les serious games trouvent des applications dans l'éducation, la formation professionnelle, le secteur médical, la publicité et même dans la politique.

Le mouvement des serious games a débuté en 2002 avec la sortie du jeu vidéo *America's Army* téléchargeable gratuitement. Selon Ben Sawyer, ce jeu fut le premier à avoir suscité autant d'intérêt auprès du public. En effet, plus de 5 millions de personnes s'étaient passionnées pour ce jeu qui leur permettait d'appréhender le secteur de l'armée.

Les universitaires commencèrent à reconnaître la portée potentielle des technologies des jeux vidéo et parallèlement les conférences sur ce thème se multiplièrent.

En 2003, pour la première fois, a lieu une journée des serious games: "Serious Game Day" organisée par le Woodrow Wilson International Center for Scholars.

C'est en 2004 que s'est tenu le premier sommet des serious games: «Serious Game Summit" organisé lors de la conférence des développeurs de jeux vidéo. La même année, MIT comparative media studies contribue au sponsoring de *Education Arcade: Games in Education*, un congrès qui a eu lieu à Los Angeles deux jours avant le congrès annuel "E3" sur l'industrie des jeux vidéo.#

II.6 La conception d'un serious game

Les serious games constituent un nouveau phénomène dans le monde des jeux vidéo. Ils pourraient, à moyen terme, éclipser les jeux vidéo « classiques » dont le seul objectif est de divertir l'utilisateur[29].

Le succès d'un serious games repose, comme celui des autres jeux, sur l'originalité et l'utilisation de technologies de plus en plus élaborées. Mais celles-ci doivent permettre de développer des jeux réalistes dans lesquels la mission peut, par exemple, être de conduire un gouvernement lors d'une crise internationale ou de réaliser les objectifs d'une entreprise internationale.

Les concepteurs de serious games doivent poursuivre des objectifs

autant supprimer l'aspect créatif et divertissant du jeu. Il s'avère donc nécessaire de proposer aux futurs créateurs de jeux, des formations particulières dans le domaine des serious games.

Tenant compte du fort potentiel de développement des serious games, certaines écoles de conception se sont spécialisées dans ce domaine. On peut par exemple citer la USC Viterbi, Ecole d'ingénieurs qui a créé le célèbre laboratoire interdisciplinaire de recherche *Game Pipe*. Cette structure est consacrée au développement et à l'enseignement des technologies nécessaires à la création des futurs serious games et à leurs applications.

La mission du laboratoire est de rechercher et de développer des technologies de jeux interactifs qui permettent des gains de temps considérables dans la production, de fournir des technologies efficaces renforçant le côté innovant des jeux. Ce laboratoire crée enfin des jeux sérieux et ludiques pour le gouvernement et les entreprises.

Le laboratoire Game Pipe s'investit fortement dans la recherche et le développement des jeux de la nouvelle génération. Il produit des serious games ainsi que des jeux prototypes comme des jeux sponsorisés. Il assure aussi le développement de jeux à programmes éducatifs et des jeux liés au secteur de la recherche. Enfin, il déploie des équipes interdisciplinaires sur plusieurs sites universitaires afin de résoudre les problèmes technologiques et de développement de ces nouveaux types de jeux.

En mettant en œuvre tous leurs moyens en matière de recherche et développement pour le développement des jeux interactifs dans un environnement académique, les développeurs peuvent influencer non seulement l'avenir des jeux interactifs divertissants mais aussi celui de jeux dont la vocation éducative, et enfin, et surtout, le développement à grande échelle de tous types de serious games. Avec la formation au sein du laboratoire Game Pipe et les programmes éducatifs qui s'y rapportent, l'USC se projette dans l'avenir et compte devenir une référence dans le domaine des serious game.

II.7 Les domaines d'application des serious games

Les serious games sont des actions interactives qui peuvent être utilisées dans

l'industrie, l'administration, la sécurité civile et bien d'autres domaines. Ils permettent d'apporter des solutions à de nombreuses problématiques liées à la formation, à la simulation ou encore à la communication[29].

Voici un petit aperçu des principaux domaines dans lesquels les serious games peuvent représenter des outils d'apprentissage et d'expérimentation intéressants. Les serious games permettent aux professionnels de ces différents secteurs de se former d'une manière un peu moins traditionnelle. Au-delà du côté ludique de ce type de support, les serious games s'avèrent être très efficaces auprès des apprenants. Ils leur permettent de mieux appréhender les difficultés auxquelles ils peuvent être confrontées et les impliquent davantage dans la résolution de celles-ci.

II.7.1 L'éducation

Déjà largement implantés au Royaume-Uni, les serious games ont naturellement vocation d'être massivement utilisés dans le cadre de l'enseignement, qu'il soit primaire, secondaire ou universitaire, théorique ou technique. En effet, les jeunes sont aujourd'hui particulièrement à l'aise avec les nouvelles technologies et les univers virtuels qu'ils se sont déjà appropriés en dehors des écoles dans le cadre de leurs loisirs. Les serious games permettent ainsi d'impliquer fortement les apprenants et de capter durablement leur attention. C'est aussi un outil efficace de représentation de concepts abstraits permettant, notamment de manipuler virtuellement l'infiniment petit ou l'infiniment grand, d'expérimenter sans risque, de vivre la réalité de lointaines époques de l'histoire et bien d'autres champs d'applications[29].

II.7.2 Les administrations

Outre les possibilités de formation interne que l'on retrouve en filigrane de tous les domaines d'application des serious games, ceux-ci offrent aux administrations des possibilités spécifiques qui pourraient leur faciliter grandement la vie quand il s'agirait de prendre des décisions délicates voire vitale concernant les populations qu'ils ont en charge. Les serious games peuvent en effet être de précieux outils d'aide à la décision par leur capacité à simuler et représenter des phénomènes complexes, comme par exemple l'impact de travaux de voierie sur l'encombrement

transports collectifs, les risques environnementaux liés à l'exploitation de zones industrielles et bien d'autres. C'est aussi un outil de communication efficace et novateur vers les administrés qui pourront mieux comprendre et appréhender les projets de leurs élus au travers d'applications ludiques et interactives|29].

II.7.3 La santé

C'est, d'après Ben Sawyer, co-fondateur de l'initiative Serious games aux Etats-Unis, le domaine d'application des serious games qui connaîtra la plus forte croissance dans les années à venir. Permettant à la fois d'apporter de réelles avancées en terme de formation (enseignement général, anatomie, entraînement des chirurgiens...), et d'accompagnement thérapeutique (diagnostic, suivi des patients, médecine sportive, traitement des pathologies et des phobies...), les « Games for Health » connaissent un véritable engouement tant auprès des praticiens que des patients qui ont pu en mesurer l'efficacité. Comme dans beaucoup de domaines des serious games, de très nombreuses applications sont encore à inventer[29].

II.7.4 La défense

Terre d'origine des premiers serious games, la défense reste encore aujourd'hui un de ses principaux financeurs. Ce sont bien sûr les simulateurs qui viennent tout d'abord à l'esprit quand on évoque le rapprochement entre jeux vidéo et applications militaires, mais les serious games répondent à d'autres besoins, comme les entraînements individuels ou collaboratifs, les enseignements généraux ou spécifiques (écoles militaires), la documentation technique électronique interactive pour assurer une maintenance rapide et efficace des matériels, le recrutement, les outils de commandement[29].

II.7.5 Les entreprises

Les industriels sont souvent confrontés à des problématiques de formation concernant la sécurité des sites dangereux, l'utilisation de machines-outils ou de véhicules spéciaux, l'application de procédures de qualité et bien d'autres auxquelles les serious games peuvent apporter des réponses bien plus évoluées et satisfaisantes que les méthodes traditionnelles. En entreprise, on pourra par exemple former les cadres lors de simulations représentant certains aspects de leur vie professionnelle,

marketing, transmettre rapidement et efficacement les caractéristiques d'un nouveau produit aux forces de vente dispersées sur le terrain[29].

II.7.6 La sécurité civile

Quand il s'agit de préparer des hommes pour intervenir en milieu hostile, les serious games montrent rapidement qu'ils seront bientôt incontournables dans ce type d'enseignement et d'entraînement. La possibilité de simuler des catastrophes naturelles ou industrielles de toutes sortes, des attaques terroristes, puis d'entraîner des centaines d'hommes connectés simultanément à interagir efficacement entre eux et en bonne intelligence avec leur commandement, voilà ce que proposent aujourd'hui les serious games dédiés à ceux qui veillent sur la sécurité de la population civile[29].

II.7.7 Les sciences

Pouvoir représenter des phénomènes non observables, simuler des interactions chimiques ou physiques complexes que l'on peut manipuler en temps réel, visualiser et présenter de manière claire les résultats d'expériences en vue d'une publication, donner aux étudiants des outils leur permettant de mieux comprendre et assimiler toutes les subtilités cachées de la nature, tels sont les réponses que peuvent apporter les serious games aux scientifiques[29].

II.8 Des exemples des serious games

Nous concluons ce chapitre par un tour d'horizon sur les différents serious games qui ont trouvés un succès ces dernières années.

II.8.1 Virtual University

Ce jeu a pour objectif de familiariser les joueurs avec les pratiques de management des collèges et universités américaines[30].

Il donne l'unique opportunité aux étudiants, professeurs et parents de se mettre dans la peau d'un directeur d'université. Ce rôle leur permet de prendre conscience de la difficulté de gérer une institution universitaire et des responsabilités du directeur. Les joueurs doivent exercer un contrôle global qui s'étend de l'établissement des salaires des professeurs à l'organisation des places de parking.

II.8.2 Food Force

Ce jeu a été créé par la WFP(World Food Program), à l'initiative de l'ONU[30]. Son but étant de sensibiliser les adolescents aux problèmes de la faim dans le monde mais aussi de leur faire prendre conscience de l'importance du travail des aides humanitaires. Disponible par simple téléchargement, il a déjà été testé par 4,5 millions de personnes depuis sa sortie.

Il propose 6 missions tout à fait réalistes qui invitent à se mettre dans la peau d'un bénévole travaillant pour une ONG. Le joueur doit être capable d'analyser les sites géographiques et les besoins alimentaires des différentes populations, ceci passant par le calcul des rations à distribuer. Il doit aussi se charger de l'acheminement des vivres qui seront lâchés par avion sur les sites.

Food Force est la preuve qu'avec un moyen d'expression adapté, un problème aussi invisible et distant soit-il qu'est la faim dans le monde dans les pays en développement, peut susciter l'intérêt et le soutien dans les pays où l'excès de la nourriture consommée est le réel problème.

Figure II-2 Quelques missions du serious game Food Force.

II.8.3 Darfur is Dying

Un autre jeu qui, par ailleurs, a fait l'objet d'une polémique est le serious game *Darfur is Dying*. Ce jeu conçu par des étudiants de *l'University of Southern California* (lui aussi à l'initiative de l'ONU) a pour but de sensibiliser les jeunes à la situation dramatique qui sévit en Afrique. En quinze minutes, ce jeu permet d'appréhender la vie que mènent les habitants du Darfour contraints de survivre à une guerre. Par ce jeu, les joueurs peuvent expérimenter le fait d'être coincés dans un camp de réfugiés, ils peuvent aussi se rendre compte que les gestes les plus anodins tels que le fait d'aller chercher de l'eau au puits deviennent des actions très difficiles

Figure II-3 Le Serious game Darfur is Dying

Ce jeu inspiré de Food Force a lui aussi rencontré un vif succès. Néanmoins, certains on vu dans ce type de jeu une exploitation de la misère humaine. On peut se demander si les plus jeunes joueurs comprennent la différence entre ces jeux réalistes et des jeux de guerres plus classiques.

II.8.4 Incident Commander

Ce serious game est un logiciel de simulation de crise destiné aux responsables des services d'urgence de villes de taille moyenne n'ayant pas les moyens d'organiser des entraînements pour des évènements tels que des actes de terrorisme, prises d'otage ou catastrophes naturelles qui peuvent survenir mais qui entraîneraient de grosses dépenses compte tenu de leur relativement faible fréquence. Ce logiciel a reçu l'aval du National Institute of Justice, branche spécialisée dans la recherche et le développement du ministère américain de la justice(DOJ). Celui-ci Of Justice aurait contribué au développement du jeu à hauteur de 400 000 de dollars et l'éditeur a, quant à lui, dépensé 1,5 millions de dollars. Ce jeu a été commercialisé en 2007, néanmoins, avant sa sortie, on pouvait se le procurer sur simple demande auprès des responsables de "public safety agencies".

Figure II-4 Le Serious game Incident Commander

II.8.5 Brain Training

Certains serious games s'adressent à des cibles spécifiques. C'est par exemple le cas du jeu Brain training qui a été élaboré par Nintendo. Ce jeu qui a connu un véritable succès au Japon, on parle même de phénomène de société à déjà été vendu à plus de 2 millions d'exemplaires sur l'archipel depuis sa sortie en mai 2005.

Ce programme d'entraînement cérébral a été conçu spécialement pour les personnes âgées qui sont soucieuses de conserver leur santé mentale. Il est en effet particulièrement recommandé aux personnes âgées de stimuler leur mémoire et d'exercer leur logique. Ce jeu constitue déjà un outil de travail dans certaines maisons de retraite japonaises.

Il s'agit d'une série de tests ludiques qui permettent au joueur d'évaluer et de faire travailler ses aptitudes cérébrales. Ce concept nouveau dans le domaine des jeux vidéo est inspiré des travaux du docteur Ryuta Kawashima. Brain training est basé sur 15 exercices développés par ce célèbre neurophysiologiste.

Figure II-5 Le Serious game Brain Training

Le principe du jeu est de s'exercer tous les jours par une évaluation effectuée à travers trois exercices choisis de manière aléatoire parmi les quinze. Après avoir été soumis à cette évaluation (de 10 minutes) le joueur peut ensuite découvrir son âge cérébral qui se base sur le temps ainsi que le nombre d'erreurs qui ont été commises. Le but étant dans l'idéal de se rapprocher le plus possible de l'âge de vingt ans. Il est également possible de se livrer à des phases d'entraînement ainsi qu'à des jeux de type sodoku permettant de développer l'esprit logique.

Certains jeux font directement travailler le calcul par des séries d'additions, soustractions et multiplications tandis que d'autres font appel à la mémoire (liste de mots à mémoriser et à retranscrire). Il peut aussi s'agir de créer des suites logiques, de compter des syllabes, de dessiner divers éléments, repérer des couleurs… et bien d'autres encore.

Ces jeux sont générés aléatoirement afin de ne pas tomber dans la routine et le parcœurisme qui pourraient fausser les évaluations.

La commercialisation d'un jeu d'entraînement cérébral peut paraître risquée pour une entreprise comme Nintendo qui s'adresse généralement à de jeunes joueurs.

Les personnes âgées sont cependant nombreuses au Japon et dans les pays industrialisés et possèdent souvent un pouvoir d'achat plus important que les

jeux vidéo de toucher une population en expansion et encore novice dans ce domaine.

Néanmoins, la cible visée n'a pu être atteinte que parce que la génération de japonais visée apprécie les technologies. On peut se demander comment les européens de la même classe d'âge auraient accueilli ce serious game.

II.9 Conclusion

Dans ce chapitre, nous avons abordé le concept de serious games. En insistant sur la dimension pédagogique, nous avons pu établir la différence principale entre un jeu vidéo ordinaire et un serious game. Nous avons présenté quelques exemples des serious games développés ces dernières années. Dans la suite de ce document, nous voulons développer un serious game en utilisant les techniques de réalité augmentée, l'objectif principal est d'étudier l'apport de RA pour les jeux vidéo et les serious games.

Chapitre III Mise en œuvre d'un Serious Game en RV pour l'apprentissage des règles de circulation routières

III.1 Introduction

Dans ce chapitre nous présentons le processus d'implémentation d'un serious game d'aide à l'apprentissage et le respect des règles de circulation routières. La mise au point de notre jeu nécessite, dans un premier temps, de préciser l'idée et le scénario du jeu ainsi que la manière d'intégration de la dimension pédagogique dans ce scénario. Ensuite, nous présentons nos choix en matière de moteur de jeu et d'outils de développement utilisés pour la suite de ce travail. Puis, nous présentons l'architecture et les principales composantes de notre prototype. Enfin, nous décrivons plus en détail la mise en œuvre de chaque composante.

III.2 Un Serious Game pour la sensibilisation à la sécurité routière

Sur les routes du monde entier, les accidents de la circulation font partie des évènements de la vie courante. Des milliers de personnes y perdent la vie tous les jours. Des millions d'autres en conservent des handicaps définitifs ou des blessures émotionnelles qui les hanteront toute leur vie. Les enfants et les jeunes adultes sont parmi les plus vulnérables. Ainsi, et l'échelle mondiale, chaque heure qui passe voit mourir quarante jeunes dans des accidents de la circulation[25].

À la recherche de nouveaux moyens pour sensibiliser les jeunes conducteurs et piétons aux risques liés à la circulation, nous proposons un serious game de sensibilisation à la sécurité routière. Le jeu doit faire apprendre aux jeunes d'être des conducteurs prudents, coopératifs et responsables. L'accent est davantage mis sur le respect des règles de la circulation, également sur l'anticipation des risques, les comportements dangereux (vitesse, manœuvres dangereuses, distractions) et le partage de la route.

III.2.1 Définition du contenu « sérieux »

Après avoir choisi le thème de notre serious game, nous devons spécifier le contenu sérieux du jeu que nous nous sommes proposé de développer, il s'agit de définir avec précision les messages à transmettre à travers le jeu. Dans notre cas, par exemple, nous devons définir quelles sont les règles de la route et les bons

comportements que nous devons faire apprendre aux joueurs à travers le serious game.

Il est clair que le nombre de règles de code de la route et les mesures de sécurité routière sont nombreux, comme nous le verrons plus tard. Durant la conception d'un serious game, chaque élément de contenu sérieux est transformé en scénario, il est donc, très difficile d'inclure toutes les règles et les mesures de sécurité routière dans un seul jeu, sinon on risque de finir avec un logiciel d'apprentissage de code de la route et non pas un serious game. Alors, à ce stade il va falloir choisir parmi ces règles et mesures celles qui sont jugées importantes et prioritaires. À titre démonstratif, nous avons choisi d'intégrer les regèles et les mesures de sécurité suivantes :

- Respect de la vitesse maximale autorisée ;
- Respect de la distance de sécurité avec le véhicule précédent ;
- Application des règles de priorités dans les intersections ;
- Respect des panneaux d'obligation (arrêt et sens interdit).

III.2.2 Définition d'un type et d'un scénario de jeu

A partir du contenu « sérieux » défini dans l'étape précédente, la seconde étape consiste à inventer un concept de jeu permettant de transmettre ce contenu sérieux. Le contenu sérieux regroupe les éléments suivants :

- Les routes, intersections et panneaux de signalisation ;
- Les véhicules ;
- Les piétons.

Dans le monde des jeux vidéo, ce sont les jeux de course qui ont le potentiel de transmettre ce contenu sérieux, on parle des jeux qui placent le joueur aux commandes d'un véhicule et celui-ci doit effectuer un nombre déterminé de tours de piste et lutter contre d'autres pilotes en vue d'obtenir une place sur un podium. Chronologiquement, dans les premiers jeux, il fallait simplement atteindre la ligne d'arrivée ou réaliser le meilleur temps. Ce type de vidéo ludique s'inspire directement des sports automobiles. Ce type de jeu demeure très populaire aujourd'hui et continue de pousser toujours plus loin le réalisme visuel et physique. On distingue deux sous-genres distincts : le jeu de course d'arcade et le jeu de course de simulation. Le premier tend à exagérer la conduite automobile et à s'éloigner fortement de la réalité (exemples : passer une épingle à 200 km/h en dérapant, tirer

un missile, moteur surpuissant, etc.). A contrario, le jeu de course de type simulation essaye, autant que faire se peut, de respecter la réalité et les caractéristiques de véhicules réels. Certains jeux appartenant à ce sous-genre permettent au joueur de participer à des évènements de course automobile réels tels que l'Indianapolis 500 ou le rallye Dakar.

Afin d'être capable de transmettre notre contenu sérieux de manière efficace, notre jeu sera dans la seconde catégorie (jeu de course de simulation) avec une modification significative du scénario du jeu. Autrement dit, nous voulons garder le même réalisme visuel et physique pour mieux attirer l'attention du joueur. Pour le scénario, le jouer prend le rôle d'un conducteur de taxi en ville dont la mission sera d'acheminer des passagers à leurs destinations dans un temps précis, le joueur sera confronté à des situations de conduite qu'un conducteur peut rencontrer dans la vie réelle, sa réaction par rapport à ces situations ainsi que son comportement sur la route vont influencer son parcours tout au long du jeu.

III.3 Conception du jeu

La conception d'un jeu consiste en la définition de ce qu'on appelle le Gameplay, le terme Gameplay signifie les règles du jeu, la manière dont le joueur est censé y jouer, la fluidité de ces règles une fois appliquées à l'environnement du jeu, et également la manière dont le joueur peut jouer.

La conception d'un Gameplay est composée de deux phases principales (Figure III-1): le « Game Design » et le « Level Design ». Chacune de ces composantes est constituée de plusieurs parties qui seront spécifiées distinctement : les modes de jeu, les mécanismes d'évaluations, les assistants de jeu et les évènements scénaristiques.

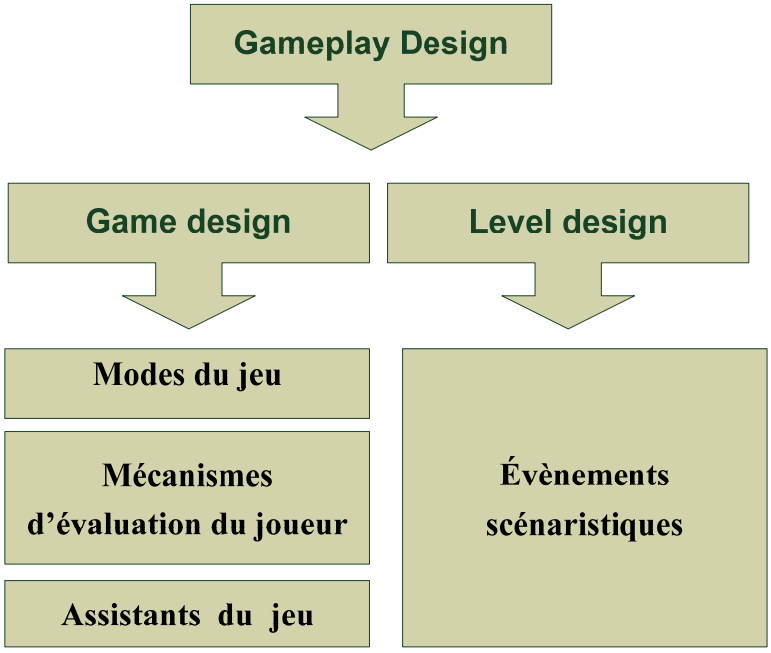


Figure III-1 Principaux composants d'un Gameplay

III.3.1 Elaboration d'un Gameplay pour le serious game

Voici la spécification en format textuel du Gameplay de notre serious game :

- *Le joueur est placé aux commandes d'un véhicule léger et a pour mission d'atteindre une destination précise dans un temps limité. La mission du joueur est d'amener son client à sa destination.*
- *Si le joueur n'atteint pas la destination dans le temps imparti, il a perdu la partie.*
- *S'il l'atteint, on lui propose alors un score, calculé à partir du temps du trajet qu'il a effectué, des erreurs de conduite ainsi que des infractions à la loi routière commises durant le trajet. Plus le nombre d'infractions aux règles se conduite routière diminue, plus son score augmente.*
- *Si le joueur endommage son véhicule, les véhicules des autres ou touche un piéton, il a perdu la partie.*

- *Le véhicule du joueur dispose d'un indicateur de santé (health level) au début de la partie qui est égal à 100%. A chaque fois que le joueur entre en collision avec des éléments de l'environnement (panneau, poteau, édifice…etc.) l'indicateur est décrémenté en fonction du choc, si l'indicateur atteint 0% la partie est perdue.*

- *Le client de chaque conducteur sera visible dans le rétroviseur du véhicule : il sera en train de lire un livre ou joue avec son téléphone. La "dangerosité" de la conduite du joueur influera directement sur le comportement du client. Tant que le joueur roule " prudemment" , son client sera à l'aise. Si le joueur a des réactions brusques, par exemple face à des événements scénaristiques, cela va gêner son client et lui fera éventuellement peur. Si le joueur atteint (ou n'atteint pas) l'objectif dans le temps imparti, un bonus (ou malus) de score sera appliqué selon la satisfaction de son client.*

A ces règles générales s'ajoutent des règles spécifiques aux différents modes du jeu proposé. Pour notre jeu ils sont au nombre de deux : un mode « mission », basé sur le franchissement de niveaux, et un mode « score », dans lequel le joueur doit réaliser le meilleur score possible sur un niveau unique.

Mode	**Objectifs**	**Moyens**	**Contraintes**
Mission	- Atteindre un point donné - Réaliser un score minimal pour passer au niveau suivant. - Satisfaire son client	- Conduire un véhicule - Respecter le code de la route - Conduire prudemment	- Limite de temps - Commettre le moins d'infractions possible aux règles de circulation routière.
Score	- Atteindre un point donné - Réaliser un meilleur score	- Conduire un véhicule - Respecter le code de la route - Conduire prudemment	- Commettre le moins d'infractions possible à la loi de la route.

Tableau III-1 Exemple des règles de Gameplay de notre serious game.

III.3.2 Mécanismes d'évaluation du joueur

Les mécanismes d'évaluation du joueur ou (*game scoring system*) sont des systèmes de notation permettant d'attribuer des points au joueur selon sa progression dans le jeu et les actions qu'il effectue. Les scores sont importants car d'un côté, ils donnent au joueur l'impression d'avancer dans le jeu et ils lui permettent de se comparer avec les autres joueurs (course au plus haut score) et donc le rester motivé. D'un autre côté, les scores peuvent contribuer dans la transmission de contenus sérieux en étant que forme de récompense ou de punition à son comportement dans le jeu. Dans le cas de notre serious game, nous utilisons un score principal qui sera lié à la réalisation de la tâche principale du jeu, et un score secondaire sous forme de bonus (récompense) malus (punition) lié au comportement du joueur sur la route. Faire de la sorte, est très important car nous ne voulons pas trop mettre l'accent sur l'aspect sérieux du jeu, autrement dit, il est très important que pour le joueur, la mission et l'objectif soient de ramener le client le plus rapidement possible d'où le score principal est calculé en fonction de la réalisation de cette tâche. Respecter les règles de circulation sera pour le joueur un moyen pour faire de bons scores est d'éviter de perdre des points ou d'être en retard (à cause des accidents…etc.).

Une fois le principe d'évaluation fixé, la deuxième étape consiste en la proposition des algorithmes qui permettent de calculer les scores du joueur, ces algorithmes seront appliqués en temps réel ou à la fin de la mission.

Pour le score principal, il est calculé à partir de l'écart entre le temps du trajet et le temps limite de la mission, plus le joueur gagne du temps et arrive tôt à la destination plus on lui attribue des points supplémentaires.

Score = TempsLimiteEnSecondes –TempsTrajetEnSecondes *50

Pour 5 secondes d'écart du temps, le joueur gagne 150 points, pour 3 minutes il en gagne 9000. Il n'existe pas une logique particulière derrière les points attribués au joueur, par contre, les joueurs favorisent des scores élevés (3,000 39,000) avec de pleins de zéros et de séparateurs que des scores entre (0 et 100) par exemple. C'est le même principe que d'attribuer des notes sur 10 ou sur 20 aux étudiants: la plupart des étudiants préfèreraient qu'on leur attribue une note de 18/20 qu'une note de 9/10 !

Pour les scores liés aux bonus/malus, ils sont calculés selon le comportement du joueur sur la route, ce comportement est évalué selon les règles et les dispositions énumérées par le tableau ci après :

<table>
<tr><td>Comportement</td><td colspan="3">Score attribué</td></tr>
<tr><td>Excès de vitesse</td><td colspan="3">-1000 points pour chaque 10km/ h supplémentaires en vitesse moyenne par rapport à la vitesse autorisée.</td></tr>
<tr><td rowspan="3">Respect de loi de la route.</td><td>règle</td><td>respect</td><td>Non-respect</td></tr>
<tr><td>S'arrêter dans les stops et les passages piétons</td><td>+300 points</td><td>-1500 points</td></tr>
<tr><td>Respecter les sens interdits</td><td>+300 points</td><td>-1000 points</td></tr>
<tr><td>Conduite prudente</td><td colspan="3">Le niveau de santé de véhicule restant à la fin de mission sera transformé en points de bonus soit 300 points pour chaque 10%</td></tr>
</table>

Tableau III-2 Points d'évaluation.

La seule remarque à faire sur ce tableau concerne les points attribués aux règles de circulation: on peut se demander pourquoi les points attribués au respect de la règle sont moins importants que les points (négatifs) attribués au non-respect. D'un côté, cela est très proche de la réalité où un conducteur reçoit une amende s'il commet une infraction, tandis qu'il ne reçoit aucune rémunération sur son bon comportement sur la route. D'un autre côté, si on attribue des points au respect des règles de la route de manière similaire aux infractions, ces derniers vont passer inaperçus et n'auront aucune influence sur le jeu dans le cas où le joueur réalise trois cas de respect de règles de circulation et une infraction par exemple.

III.3.3 Conception des niveaux

Une fois les mécanismes de jeu définis, vient la seconde phase de conception dite "Level Design". Elle consiste à inventer des scénarios mettant en scène le Gameplay du jeu. Concrètement, il s'agit de construire les différents « niveaux » du jeu, d'où le nom donné à cette phase.

Pour notre jeu, les différents niveaux se déroulent dans une ville virtuelle, chaque niveau du jeu est constitué d'une ou plusieurs missions, une mission consiste à ramener le client d'un point de départ à un point d'arrivé

Pour chaque niveau, nous devons spécifier les éléments suivants :

- Un plan du réseau routier où se déroule le niveau ;
- Le parcours que doit suivre le joueur en indiquant le point de départ et le point d'arrivée et en surlignant le ou les chemins à parcourir ;
- Ce parcours est ensuite annoté avec des numéros correspondants aux différents évènements pouvant survenir durant le niveau.

Figure III-2 Conception d'un niveau du jeu.

indice	Conditions	événements
départ	-	Le client lui communique la destination.
1	Le client est monté à bord	Le joueur doit démarrer sa voiture.
2	-	Le joueur doit faire demi-tour, chemin coupé.
R1	Le joueur n'a jamais été confronté à la situation d'un feu rouge.	Le feu passe au rouge quand le joueur arrive : incitation à s'arrêter et attendre qu'il passe au vert.
	Le joueur a été déjà confronté à la situation d'un feu rouge.	Le feu reste vert ou passe au vert s'il est déjà en rouge.
R2	le joueur s'arrête sur le panneau de stop sur place.	Le système game lui donne une occasion de passer en minimisant le nombre de véhicules qui passent.
	le joueur grille le stop.	moteur game essaye de provoquer un accident pour montrer au joueur la dangerosité de son action.
L1	Si le joueur dépasse la vitesse limite de 50 km/h.	Si le joueur sera informé qu'il été contrôlé au radar et qu'il perdra des points suite à son excès de vitesse.
	Si le joueur ne laisse pas une distance de sécurité de 15 m	le moteur game essayera de freiner la voiture la plus proche du joueur.

Tableau III-3 Exemple d'événements sénaristiques d'un niveau de jeu.

III.4 Réalisation d'un prototype de Serious Game en mode réalité virtuelle.

En général, la phase de réalisation d'un jeu vidéo est une tâche laborieuse qui prend du temps, elle est assurée par une équipe de développement composée d'un nombre important de personnelles.allant de 20 à 100 selon la complexité du jeu. On distingue parmi eux des artistes, des programmeurs, des concepteurs de niveaux et des spécialistes des effets sonores. Le tableau suivant recense l'ensemble de personnes impliquées dans le processus de développement de jeux vidéo ainsi que le rôle associé à chacun.

Métier	spécialité	Rôle
Programmeur	Programmeur outil	il développe des outils (programmes) qui serviront directement à la réalisation du jeu.
	Programmeur moteur	Le programmeur du moteur graphique s'occupe plus particulièrement du rendu

	graphique	graphique du jeu, ce qu'on appelle le moteur de rendu. Que le jeu soit en 3D ou en 2D, le rendu dans les jeux vidéo reste un problème fondamental, et le programmeur de moteur graphique doit assurer un compromis entre la qualité visuelle et la performance du jeu.
	Programmeur moteur physique	Le programmeur du moteur physique travaille sur les interactions dans le jeu : le déplacement des personnages, les collisions, l'utilisation d'objets...
	Programmeur IA	Le programmeur intelligence artificielle a en charge le développement des comportements des personnages, qu'ils soient humains ou animaux.
Designer	Game designer	le Game Designer imagine le jeu dans son ensemble : il définit le genre, le thème, l'histoire, les règles, les décors, les personnages, la structure logique et l'interactivité du produit. Il assure également la cohérence globale de tous ces éléments. Son travail est de créer un Game play qui rendra l'expérience du joueur plaisante
	Level designer	En accord avec le game designer, le level designer aura plus particulièrement en charge la gestion des différents niveaux du jeu : leur complexité, les évènements qui se produisent durant ces niveaux. etc.
Artiste	Roughman	Du travail du Roughman va naître l'image du jeu. Les ambiances décrites par le scénariste et le game designer vont émerger de ses croquis et esquisses (d'où son nom de Roughman).
	Graphiste 2D	Le graphiste 2D réalise les images fixes du site, à commencer par les dessins préparatoires qui auront été esquissés par le Roughman. Il aura également en charge la réalisation de l'interface du jeu et les textures qui habilleront les personnages ou objets 3D mais également les fonds ou décors (matte painting).
	Graphiste 3D / modélisateur	Le modélisateur met en volume les personnages, éléments ou décors du jeu. Il travaille en étroite collaboration avec l'animateur 3D afin de faciliter l'animation des différents éléments et avec le graphiste

		cheveux, vêtements...).
	Animateur	A partir des éléments fournis par les graphistes, l'animateur s'occupera des animations des personnages ou objets.
Sound designer		Sur la base du travail du compositeur et du musicien, le sound designer s'attache à mixer les différentes sources musicales et sonores pour les intégrer au jeu. il définit l'ambiance générale du jeu, comme la musique, les bruitages et les dialogues.
Testeur		le testeur est chargé de repérer les différent types de défauts du jeu. Cela va de la recherche de bugs et les propositions de corrections appropriées aux programmeurs à l'évaluation de la justesse du Game play en anticipant les remarques des friture joueurs

Tableau III-4 Liste des métiers du jeu vidéo[26].

Le Tableau III-4 montre que le processus de réalisation d'un jeu vidéo est un travail d'équipe qui exige des compétences dans différents domaines. Vu que ce travail sera assuré par une seule personne dans le cas de notre serious game, nous mettons l'accent sur les parties importantes du processus de développement en éliminant des parties comme les effets sonores et la réalisation de plusieurs niveaux du jeu par exemple.

III.4.1 Présentation de la plateforme de développement

Pour implémenter notre Serious Game dans sa version RV, nous utilisons une plateforme de développement composée d'un ensemble d'outils. Nos choix ont été influencés par les paramètres suivants :

- Un langage de programmation de haut niveau ;
- Des outils gratuits et compatibles les uns avec les autres ;
- Une plateforme ouverte et extensible avec des outils et interfaces de programmation.

Nous avons étudié et essayé différents plateformes et outils, notre choix final est fixé sur la boite à outil décrite dans la table suivante :

Outil / Plateforme	**Description**	**Licence**
XNA 4.0	Moteur graphique 3D dédié au développement des jeux vidéo pour les plateformes Windows, X-BOX 360 et Windows Phone 7.	Proposé gratuitement pour les jeux Windows, payant pour les jeux XBOX 360 et Windows phone 7.
Engine 9	Bibliothèque d'animation serious games dédié aux jeux vidéo.	Open source sous licence GPL.
Newton	Moteur physique.	
3DS max	Logiciel de modélisation 3D.	Autodesk.
FBX exporter	Pour convertir les modèles de 3DS MAX vers le format FBX.	Autodesk.
Unity3D	Outil de développement de jeux vidéo 3D.	

Visual studio 2010	Environnement de développement intégré (IDE) exigé par XNA.	

Tableau III-5 Liste des outils utilisés pour la réalisation de la version RV.

Dans cette boite à outils, figurent deux plateformes qui méritent un peu plus de présentation, on parle de XNA game studio et Unity 3D. Ces plateformes sont très utiles pour débuter dans le développement des jeux vidéo car ils montrent un très bon rapport simplicité/fonctionnalité, nous les présentons brièvement dans les deux paragraphes qui suivent.

III.4.1.1 Microsoft XNA Game Studio 4.0

Microsoft XNA Game Studio 4.0 est un environnement de programmation qui permet d'utiliser Microsoft Visual studio pour développer des jeux vidéo pour Windows, Xbox 360 et Windows Phone 7. XNA est un ensemble de classes et d'outils permettant de simplifier la programmation des jeux vidéo. Nous pouvons résumer les fonctionnalités de base de XNA dans la liste suivante :

- Création des jeux pour différentes plateformes : Windows, Xbox 360 et Windows phone 7 ;
- Support et gestion automatique et simplifiée de fichiers multimédia (images, sons, musiques), textures, modèles 3D…etc.
- Gestion automatique de la boucle de jeu ;
- Rendu et effets avec des Shader model 2.0 et 3.0 (actuellement pour PC et Xbox 360 uniquement) ;
- Programmation basée sur la plateforme .NET et le langage C# ;
- Gestion simplifiées des transformations géométriques et support natif des mathématiques de base pour l'infographie (Calcul matriciel, Calcul vectoriel, Quaternions, ...etc.).

La Figure III-3 représente la structure d'un jeu sous XNA, le jeu est représenté sous forme d'une classe C# qui implémente 4 méthodes: Initilize(), LoadContent(), Update() et Draw().

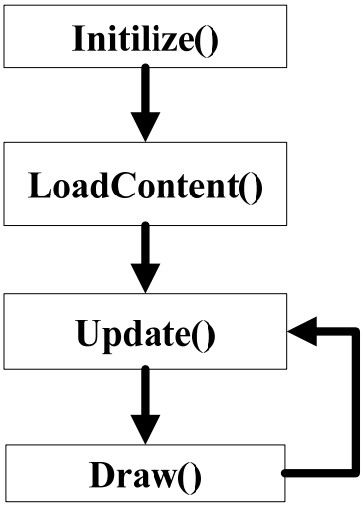


Figure III-3 Structure d'un jeu XNA

- **LoadContent() :** Utilisée pour charger les modèles 3D, les textures, les sons …etc.
- **Initilize**() : Cette méthode est utilisée pour initialiser le jeu, placer les objets dans l'espace 2D/3D, ajouter des sources lumineuses, initialiser le caméra...etc.
- **Update**() : Cette méthode est responsable de la mise à jour du monde virtuel, des transformations géométriques pour les objets animés, elle permet également d'appliquer les algorithmes tels que les algorithmes de navigation,

de planification de chemins (A*), de détection de collision, de simulation physique…etc.

- **Draw**() : Cette méthode est utilisée pour le rendu du jeu, à ce niveau nous spécifions quel objet il faut afficher, avec quel effet, comment l'objet interagit avec la lumière, quel est le Shader à utiliser pour dessiner l'objet…etc. Une très grande partie des instructions de cette méthode sont exécutées directement au niveau de la carte graphique(GPU).

III.4.1.2 Unity 3D

Unity est une plateforme de création des jeux vidéo 3D, il s'agit d'un outil intégrant toutes les fonctionnalités nécessaires pour créer un jeu vidéo. Ce moteur de jeu est exclusivement réservé aux jeux 3D et il ne permet d'utiliser que les fonctionnalités permettant de créer ce type de jeux. Dans Unity 3D, le jeu est piloté par des scripts qui peuvent être exprimés soit dans le langage C# mono, javascript ou Bodo. Unity 3D dispose également d'une interface graphique permettant de travailler sur le jeu de manière interactive, cette interface permet par exemple de placer les objets dans l'espace 3D, appliquer des transformations géométrique sur les objets de la scène à l'aide de la souris, modifier les textures, matériaux, propriétés physiques, paramètres de collision…etc.

Cette plateforme ne permet pas d'implémenter notre jeu car ce type de plateformes est généralement de nature fermé est non extensible. Alors il est impossible d'implémenter un jeu de RA selon l'architecture proposé dans le chapitre précédent avec Unity 3D. Par contre, nous avons décidé de tirer profit de quelques fonctionnalités de cette plateforme que nous avons trouvées très utiles pour notre projet. Par exemple, nous pouvons utiliser Unity 3D pour la conception de la scène, prototypage rapide game etc. La figure III-4 représente une capture d'écran d'un jeu

en cours de développement avec Unity 3D.

Figure III-4 Un jeu 3D en cours de réalisation avec l'outilUnity 3D.

III.4.2 Création de l'«assets» du jeu

Une partie très importante de tout jeu vidéo est ce qu'on appelle l'assets du jeu, ce sont les éléments du jeu non programmés, la liste suivante regroupe les assets les plus utilisés dans les jeux vidéo :

- Les modèles 3D pour les jeux 3D ;
- Les spirites pour les jeux 2D ;
- Les séquences d'animation ;
- Les fichiers audio de musique et d'effets sonores ;
- Les textures et les fichiers de matériaux ;
- Les fichiers de cartes (maps) et de description des niveaux ;
- Les séquences vidéo.

La Figure III-5 montre l'exemple d'un personnage qui utilise plusieurs types d'assets de différentes natures.

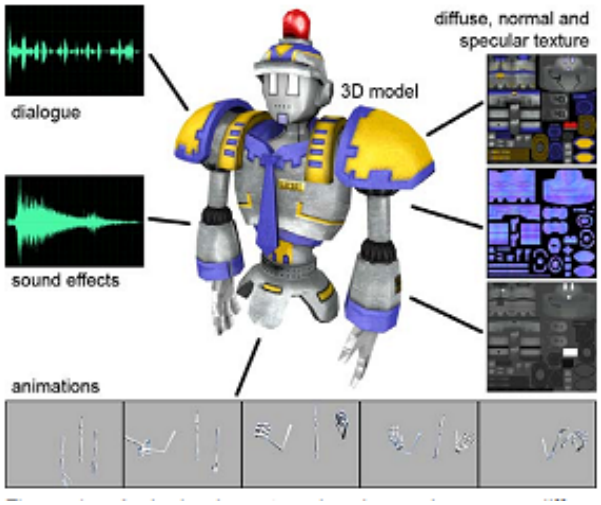


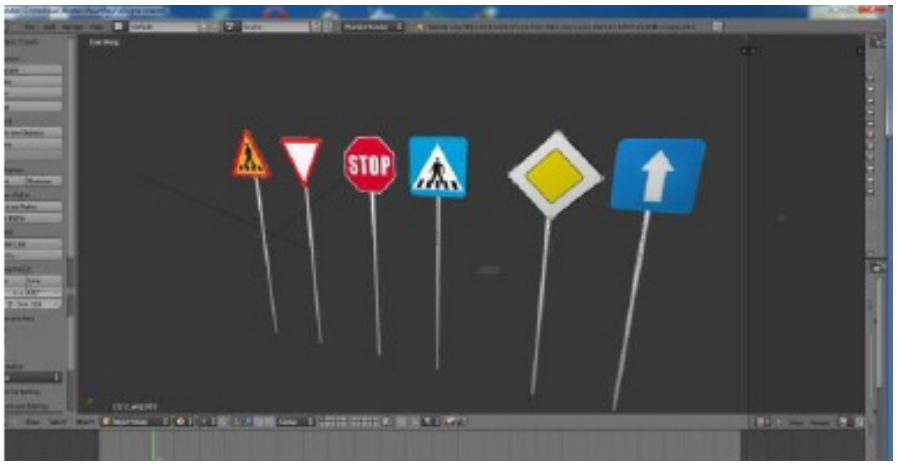
Figure III-5 Un seul personnage qui utilise plusieurs assets.

Comme nous l'avons déjà expliqué auparavant, chaque type de ces assets est créé par des spécialistes en utilisant des outils dédiés. A titre d'exemple, les modèles 3D et les textures sont créés par des artistes, les effets sonores sont créés par des ingénieurs de son, les fichiers de description de niveaux sont créés par des concepteurs de niveaux…etc.

III.4.2.1 Collection et modélisation des modèles 3D

Le jeu se déroule dans les pistes d'une ville virtuelle, nous avons besoin des modèles suivants pour construire cette ville virtuelle :

- Modèles d'édifices ;
- Modèles des routes ;
- Des panneaux de signalisation ;
- Des modèles de voitures ;
- Modèles de piétons.



o

Figure III-6 Modélisation de panneaux de signalisation avec Blender.

Figure III-7 Quelques modèles utilisés pour la construction de jeu.

III.4.3 Chargement des modèles dans le jeu

Les modèles 3D sont initialement disponibles dans le format de l'outil avec lequel ils ont étés modélisés. Par exemple, l'outil 3DS MAX utilise le format .max, Blender utilise le format. blend …etc. il s'agit des formats propriétaires qu'on ne peut pas utiliser directement. Ces modèles doivent être transformés dans le format supporté par le moteur de jeu utilisé.

XNA supporte nativement deux formats pour les modèles 3D, le format X et le format FBX. Dans notre cas, nous utilisons le format FBX car il permet d'intégrer à la fois, la géométrie, les textures, les matériaux et l'animation.

Pour Blender, le format FBX est nativement supporté, dans le cas de 3DS MAX il faut utiliser l'outil *Autodesk FBX Converter*(Autodesk) pour exporter les fichiers max en format FBX.

Une fois le modèle en format FBX obtenu, il peut être utilisédans notre jeu. Pour utiliser les modèles, textures, audio...etc. XNA utilise un mécanisme appelé le Content Pipeline. Son rôle est de convertir ces modèles, textures et audio au moment de la construction du jeu dans un format spécifique qui leur permet d'être chargés à nouveau efficacement au moment de l'exécution.

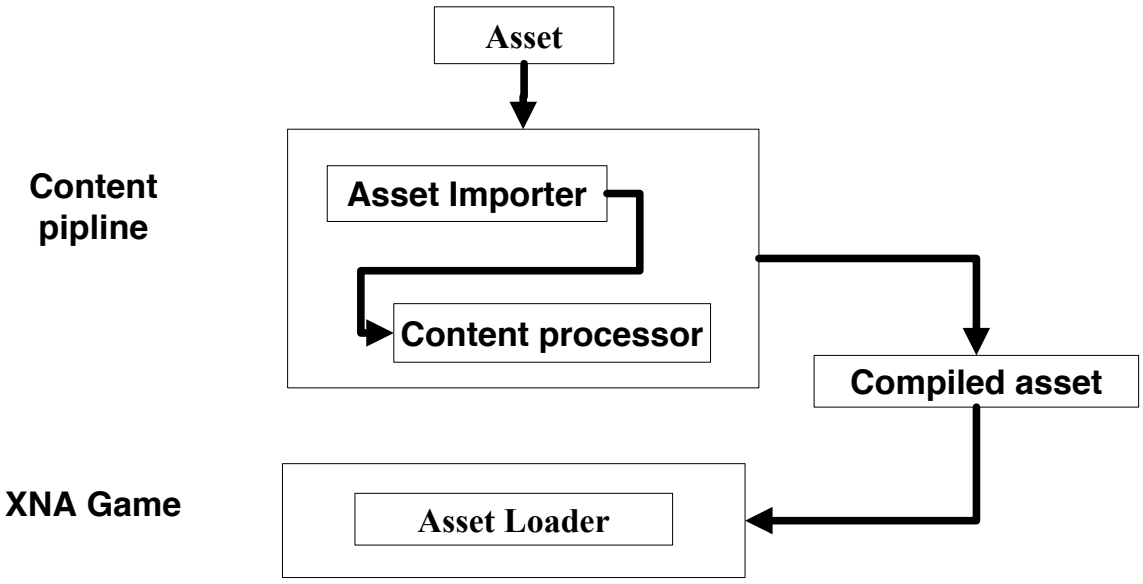


Figure III-8 Traitement d'un modèle par le Content Pipeline afin d'être chargé efficacement durant le jeu

III.4.4 Mise en œuvre de l'éditeur de niveau du jeu

Un jeu est généralement constitué d'un ensemble de niveaux, chaque niveau représente une étape de progression dans le jeu. Le fait de diviser le jeu en niveaux est très important car le passage d'un niveau à l'autre donne au joueur une sensation de progression et un envie de continuer le jeu afin d'atteindre le niveau suivant. D'un point de vue infographique, nous pouvons faire l'analogie entre un niveau et

une scène, alors il est important que chaque niveau soit différent de l'autre, et en conséquence, chaque scène doit être différente de l'autre. Les scènes peuvent être, à titre d'exemple, différents bâtiments dans une ville, ou encore différentes villes dans le monde …etc.

Le problème principal réside dans l'implémentation de ces niveaux. Imaginons un jeu qui contient 10 niveaux, chaque niveau se déroule dans une scène virtuelle contenant au moyen 100 objets virtuels. Comment peut-on implémenter ce jeu ? Il existe deux approches pour aborder ce problème.

La première approche est basée sur la programmation, cette approche est dite *hard coding* en anglais, l'idée est de spécifier manuellement les objets de la scène, leurs positions, rotations, mises à l'échelle, matériaux…etc. Ceci peut être effectué directement dans le code source du jeu ou dans des fichiers texte qui seront chargés par le jeu au moment de l'exécution. Il est claire que cette approche n'est pas pratique, car dans le cas de l'exemple cité précédemment, il faut spécifier au moyen 5 propriétés par objet ce qui implique au total 5*100*10 propriétés à spécifier pour le jeu entier.

La deuxième approche consiste à utiliser un outil dédié dit éditeur de niveau (game level editor), dans le cas des jeux 2D on applique des fois le terme (map editor). L'éditeur de niveau est un outil interactif, permettant de construire des niveaux, il permet par exemple de positionner les objets dans l'espace 3D, d'appliquer des transformations géométriques, des matériaux, de positionner les sources lumineuses, de définir des relations entre les objets (par exemple telle clé ouvre telle porte) ...etc.

L'avantage avec cette approche est la productivité, car le concepteur de niveau peut voir directement les résultats de ces actions sur l'éditeur de niveau sans avoir besoin d'exécuter le jeu

XNA ne possède pas un éditeur de niveau, pour cette raison nous avons cherché à contourner cette limite en utilisant l'environnement Unity 3D, ce dernier possède un éditeur de niveau avancé et simple d'utilisation puisqu'il représente un moteur de jeu assisté.

La combinaison entre Unity est XNA repose sur l'idée que les deux utilisent

utilisant le mécanisme de sérialisation qui permet de sauvegarder l'état d'un objet sous forme d'un fichier XML nous pouvons mettre en place un mécanisme qui permet d'exporter une scène créée par Unity 3D en un document XML, utilisable depuis un jeu XNA.

La démarche est la suivante :

- Implémenter pour chaque type d'objet dans la scène une classe C# mono est une classe C#.net, ces classes contenant les informations relatives à ce type d'objet.

- Implémenter un script C# mono qui permet de créer pour chaque objet de la scène une instance de la classe C# mono précédemment créée et il y'a copie les données de cet objet. Ensuite ce script doit sérialiser ces instances dans un document XML.

- Pour ce qui est de XNA et lors du chargement du jeu, les classes C#.NET créées lors de la première étape sont utilisées pour désérialiser le document XML ainsi que pour créer des instances qui sont à leur tour utilisées pour initialiser la scène XNA.

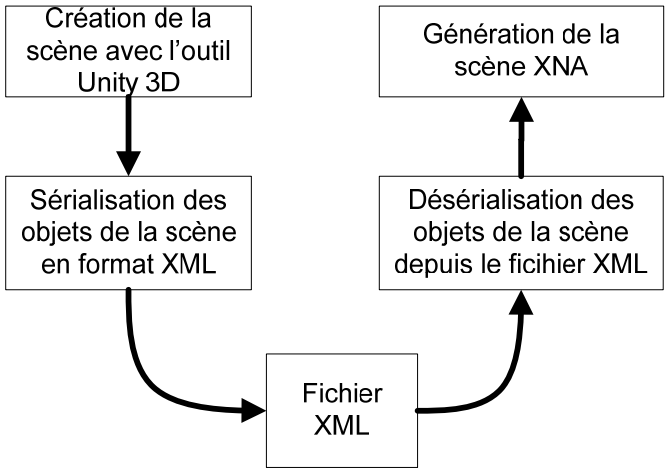


Figure III-9 Mécanisme d'édition des niveaux utilisé

L'extrait de code suivant représente un extrait d'un document XML d'une scène exportée depuis Unity 3D avec le mécanisme que nous avons développé.

```
<?xml version="1.0" encoding="utf-8"?>
<Scene id="myscene">
  <SceneModels>
    <MModel id="roadStraitNoZeb">...</MModel>
    <MModel id="houseRamp">
      <Transform>
        <Position>
          <X>11.6253042</X>
          <Y>-0.112857819</Y>
          <Z>12.868782</Z>
        </Position>
        <ObjectScale>
          <X>2</X>
          <Y>2.00000048</Y>
          <Z>2.00000048</Z>
        </ObjectScale>
        <ObjectRotation>
          <X>-270</X>
          <Y>0</Y>
          <Z>0</Z>
        </ObjectRotation>
      </Transform>
    </MModel>
    <MModel id="houseCorner">...</MModel>
    <MModel id="treePine">...</MModel>

  </SceneModels>
  <SceneLights>
    <MLight id="Directional light">
      <Transform>
        <Position>
          <X>-10564.94</X>
          <Y>-3230.643</Y>
```

Extrait de code III-1 Description d'une scène en format XML

III.4.5 Construction de l'espace virtuel

Pour aider l'utilisateur à apprendre la conduite dans des milieux urbains, nous avons choisi d'utiliser une ville virtuelle comme espace principal du jeu, l'objectif étant de pouvoir reproduire la majorité des situations de conduite en configuration réelle.

La première idée était de modéliser l'espace du jeu (ville virtuelle) avec la plateforme Unity 3D en utilisant quelques modèles de bâtiments et des segments de route, la tâche s'avérait plus difficile qu'elle le semblait. D'abord, la génération manuelle d'un réseau routier avec des segments est si délicate car plus la route est longe plus le facteur d'erreur de placement de ces segment augmente, d'un autre côté, Le nombre de bâtiments nécessaire pour créer une ville virtuel est vraiment énorme, même si on utilise un outil comme Unity 3D ou comme Blender, placer et gérer ces bâtiments reste un travail fastidieux.

Un autre problème avec cette méthode est l'uniformité de la ville obtenue, due au fait que la ville est obtenue en utilisant un nombre limité de modèles de bâtiments, la duplication de ces modèles rend la ville uniforme est non réaliste.

La Figure III-10 représente nos premières tentatives de modélisation d'une ville virtuelle avec l'apparition des problèmes et artéfacts cités ci-dessus.

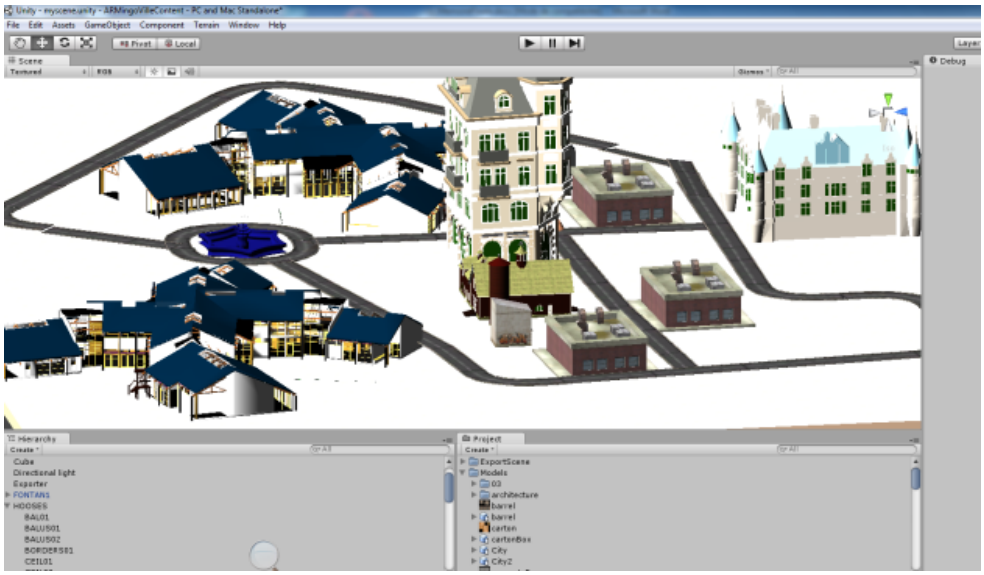


Figure III-10 Modélisation d'une ville virtuelle avec Unity 3D.

Pour faire face à ces problèmes, nous avons utilisé la modélisation procédurale pour générer la ville virtuelle, pour cette raison, nous avons utilisé l'outil CityEngine. Cet outil est basé sur la modélisation procédurale qui offre la possibilité de générer et de texturer des bâtiments 3D, des réseaux routiers, des espaces verts …etc.

La première étape dans le processus de modélisation d'une ville virtuelle est l'établissement d'un plan de la ville, la figure suivante représente une image satellitaire d'un coin de la ville de Biskra utilisé comme un plan de notre ville virtuelle.

Figure III-11 Plan d'une partie de la ville de Biskra

La deuxième étape consiste à définir le réseau routier de notre ville virtuelle en utilisant le plan établi précédemment comme référence. Le principe est de retracer les routes avec l'outil City Engine, ce dernier représente le réseau routier sous forme d'un graphe où les arrêtes sont des segments de route et les nœuds sont les croissements. Le rôle de l'utilisateur est de définir les segments et l'outil s'occupe de la génération du maillage, le principe de cette étape est représenté dans Figure III-12

Figure III-12 Construction du réseau routier avec l'outil City Engine

A la fin de ce processus nous obtenons le réseau routier de la ville (Figure III-13) sous forme d'un maillage 2D, l'étape suivante est de formuler des règles paramétrables qui seront appliquées pour la génération de la structure routière finale.

Figure III-13 Carte routière de la ville virtuelle

Les règles sont appliquées sur les segments pour modéliser la route. A titre d'exemple, selon les éléments connectés aux segments et les paramètres spécifiés par l'utilisateur, l'outil applique la texture définie par la règle, ajoute la décoration des côtés de la route… etc.

Après la construction du réseau routier, nous devons générer les bâtiments de notre ville virtuelle, les bâtiments sont regroupés dans des blocks où chaque bloc est délimité par une boucle de segments routiers (Figure III-14).

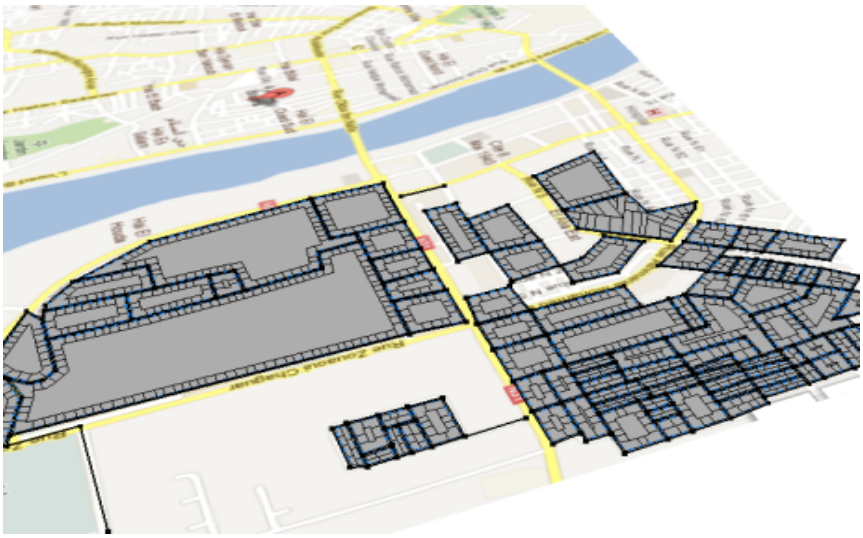


Figure III-14 Génération de blocs d'édifices.

Pour les blocs, nous pouvons spécifier des regèles de subdivision et de placement de bâtiments, l'intérêt de ces édifices est uniquement décoratif pour notre jeu. Autrement dit, il n'est pas utile de placer des édifices partout, le principe est de les placer uniquement sur les côtés de la route. Comme le montre la Figure III-14 nous avons paramétré la règle de génération des blocs de manière à ne subdiviser que les parties situées sur les côtés de la route. Le résultat de génération automatique des édifices est présenté dans la Figure III-15

Figure III-15 Génération d'édifices

A la fin du processus de modélisation de la ville virtuelle, le résultat est exporté sous forme d'un modèle 3D dans le format FBX.

III.4.6 Création du terrain du jeu

Pour générer le terrain sur lequel sera placée la ville virtuelle, nous avons utilisé la technique de HightMap pour modéliser le terrain. Le principe de la HightMap (en français carte de hauteur) est de stocker la hauteur de chaque point du terrain dans une image en niveaux de gris, chaque pixel dans cette image représente la hauteur d'un point du terrain, un pixel blanc correspond à un point très élevé tandis qu'un pixel noir représente un point avec une hauteur minimale (mer, fleuve… etc.). Il suffit donc de parcourir l'image pour créer les sommets 3D correspondant aux pixels, puis de les relier pour créer les triangles représentant le maillage du terrain.

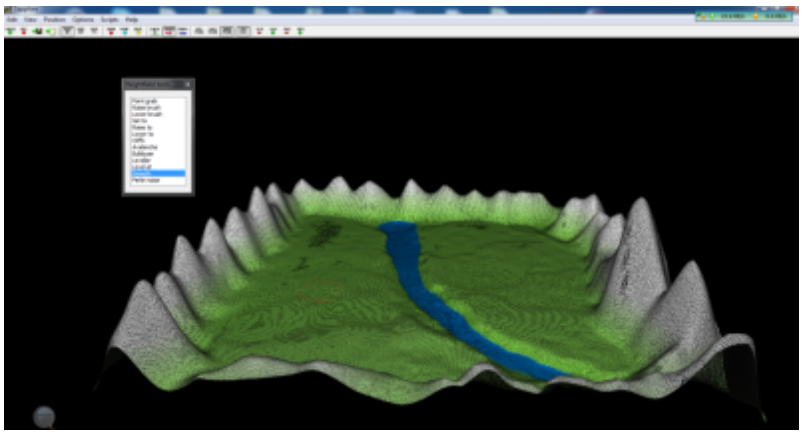


Figure III-16 Génération du HightMap avec l'outil L3DT

Pour créer le HightMap, nous avons utilisé l'outil l3DT dédié à la génération des terrains, l'outil offre plusieurs possibilités pour la création de terrain, génération aléatoire, procédurale ou interactive. Nous avons utilisé le mode interactif de cet outil pour modéliser le terrain de notre jeu. En commençant par un plan, il est possible d'utiliser un curseur 3D pour déformer ce plan et avoir le relief du terrain. La Figure III-16 représente l'étape de modélisation interactive du terrain.

Une fois la modélisation du terrain terminée, nous pouvons l'exporter sous forme d'un HightMap, l'outil permet également de générer la texture du terrain en utilisant quelques paramètres tels que le climat par exemple. Nous pouvons aussi générer ce qu'on appelle la carte d'eau, il s'agati d'une image en niveau de gris où les pixels noir représentent le sol et les pixels blanches représentent l'eau.

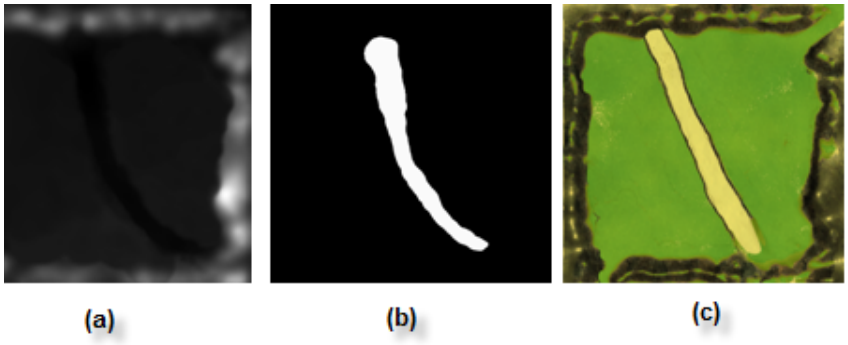


Figure III-17 Résultat de la modélisation du terrain (a) Carte de hauteur (b) carte de l'Eau, (c) Texture de terrain.

III.4.6.1 Chargement et rendu du terrain avec XNA

Pour faire le rendu de notre HightMap avec XNA, nous devons créer le maillage des triangles à partir de l'image en niveau de gris de la carte de hauteur. La première étape consiste à charger cette carte en l'important en tant que texture.

La technique que nous utilisons pour transformer la carte de hauteur en un maillage 3D consiste à créer le terrain sous forme d'une grille régulière constituée de cellules, chaque cellule comporte deux triangles et 4 sommets partagés (Figure III-18):

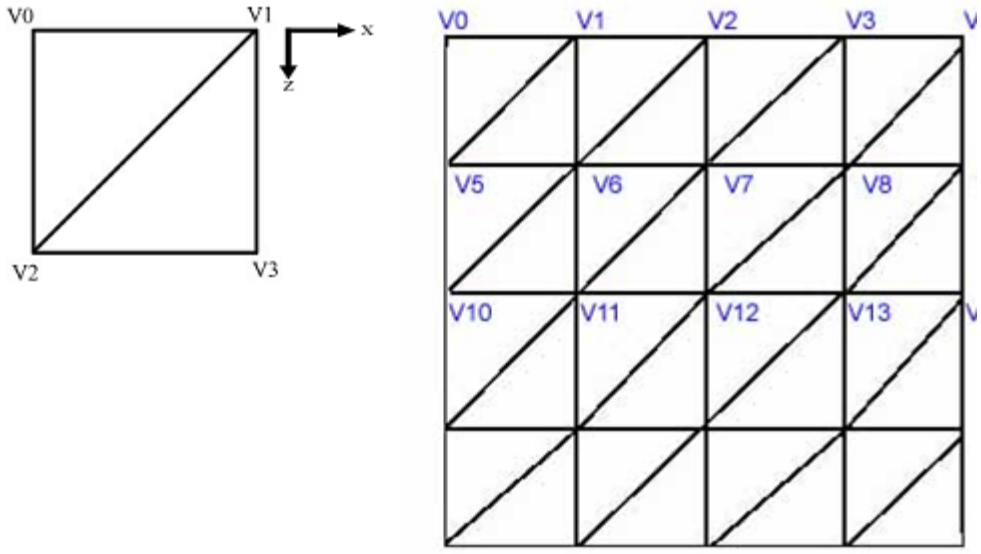


Figure III-18 Principe de création du maillage du terrain

Par exemple, pour un terrain de 4x4 cellules, nous aurons besoin de 4x4x2 triangles et de 5x5 sommets. De manière générale; on a :

- Nombre de triangles = nombre de cellules en X * nombre de cellules en Z * 2 ;
- Nombre de sommets = ((nombre de cellules en X))+1 *((nombre de cellules en Y) + 1) ;

Dans le cas d'un terrain généré à partir d'un HightMap de WxH pixels, nous avons :

- Nombre de sommets = H*W (chaque pixel représentera un sommet dans le terrain) ;
- Nombre de triangles = (H-1)*(W-1) *2.

En utilisant ces formules, nous pouvons calculer les sommets (vertices) à partir de la carte des hauteurs.

Pour chaque pixel P(x,y) de la carte des hauteurs, nous devons générer un sommet S(X,Y,Z) tel que :

- S.X = scale * (P.x -((w-1)/2)
- S.Z = scale * (P.y-((h-1)/2)
- S.Y = Hmax * HightMap.valeurDePixel(P)

Où :

- scale est le facteur d'échelle du terrain sur les dimensions X et Y ;
- Hmax est le facteur d'irrégularité du terrain
- HightMap.val eurDePixel(P) est une fonction qui donne la valeur du point P(x,y) de la carte des hauteurs.

Le code suivant permet de calculer tous les sommets du terrain et les placer dans un vertexBuffer.

```
99          int i=0;
100         // Create the terrain vertices.
101         for (int y = 0; y < heightfield.Height; y++)
102         {
103             for (int x = 0; x < heightfield.Width; x++)
104             {
105                 Vector3 position;
106
107                 // position the vertices so that the terrain  is centered
108                 // around x=0,z=0
109                 position.X = terrainScale * (x - ((heightfield.Width - 1) / 2.0f));
110                 position.Z = terrainScale * (y - ((heightfield.Height - 1) / 2.0f));
111                 position.Y = (heightfield.GetPixel(x, y) - 1) * terrainBumpiness;
112                 vertices[i++] = position;
113             }
114         }
115
```

Extrait de code III-2 Code d'initialisation des sommets du terrain

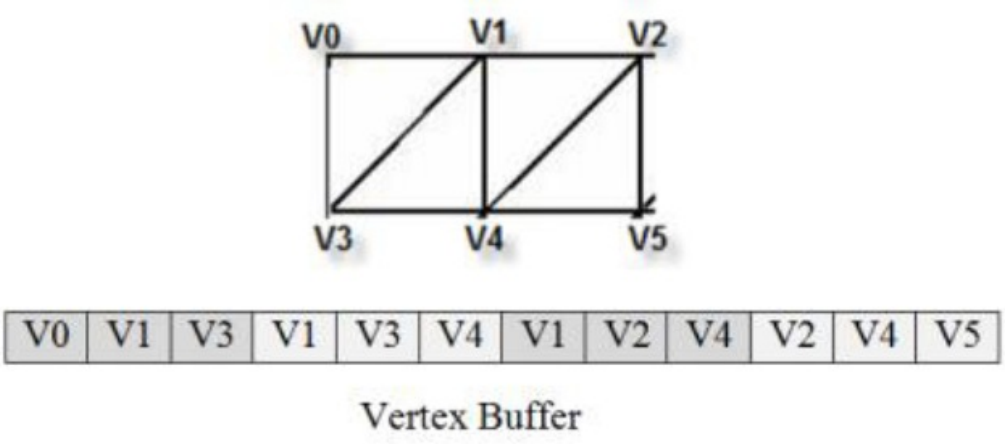


Figure III-19 problème de redondance des de donnés dans le vertexbuffer.

La figure III-19 montre que le stockage des données des sommets dans un vertexbuffer de la manière présentée auparavant souffre provoque une perte significative en mémoire liée à la redondance des données. Pour faire face à ce problème les moteurs graphique et les outils de modélisation utilisent le mécanisme d'index buffer. Ce mécanisme permet de réduire le nombre de sommets utilisés pour représenter l'objet, il s'agit d'un tableau d'indices où chaque trois indices représentent un triangle, ainsi, les sommets seront enregistrés une seul fois dans le vertex buffer et ils seront référencés d'autant qu'ils sont partagés dans le index buffer. La Figure III-20 représente un exemple permettant de démontrer l'utilité de ce mécanisme, dans cet exemple nous supposons qu'un sommet est représenté par une position, un vecteur normale et un pair de coordonnés de texture, en supposant également qu'un nombre réel sera codé en 32 bit nous pouvons estimer qu'un seul sommet occupera 32 octet en mémoire. Pour les indices la taille d'un indice dépend de la taille maximale de vertex buffer autorisé par la carte graphique, en générale les cartes graphique supportant des indices à 16 ou 32 bit. Dans le cas d'un indice à 16 bite chaque triangle aura besoin de 3 indices = 6 octets. En fessant des simples calculs, nous pouvons remarquer qu'avec un index buffer nous pouvons représenter 8 triangles partageant 6 sommets avec un espace mémoire de 216 octets au lieu de 384 octets si nous utilisons uniquement un vertex buffer.

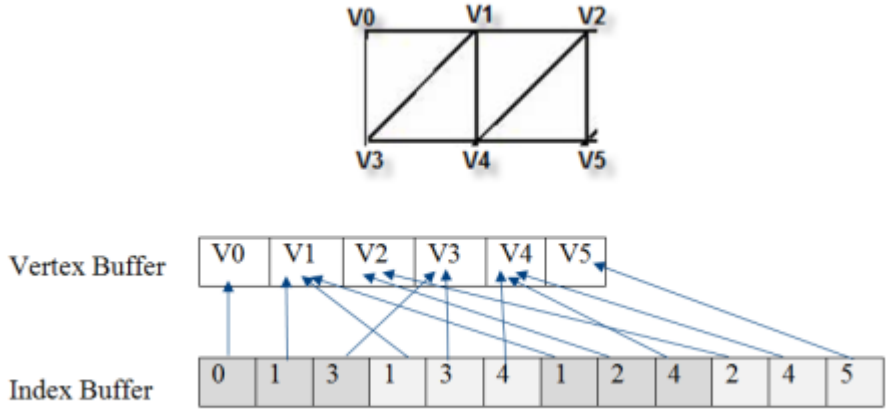


Figure III-20 Principe de fonctionnement de la technique de l'index buffer

En réalité, l'intérêt des index buffer ne se limite pas uniquement à la réduction de l'espace mémoire utilisé par la géométrie, mais ils contribuent à la réduction des performances nécessaires pour son traitement, sachant que la carte graphique applique les même calcules (éclairage, mappage texture, ...etc.) à chaque sommet de la géométrie à travers le vertex shader, l'utilisation d'un index buffer pour éliminer les sommets dupliqués permet d'augmenter massivement les performances.

Pour créer notre index buffer nous devons examiner le schéma de la Figure III-19 pour établir un mécanisme générique pour calculer les indices, les deux premier triangles sont constitué d'indices (0,1 ,3) et (1,3, 4), les deux deuxièmes sont constitués d'indices (0,1 ,3) et (1,3, 4). Nous pouvons créer les triangles deux à deux de la manière suivante :

Triangle 0 (0,0+1, 0+3)→ (i,i+1, i+nbsommetsX) (i=0)
Triangle 1 (0+1,0+3,0+3+1)→ (i+,i+nbsommetsX, i+nbsommetsX+1)(i=0)

Triangle 2 (0+1, 0+1+1, 0+1+3)→(i,i+1, i+nbsommetsX) (i=1)
Triangle 3 (0+1+1,0+1+3,0+1+3+1)→ (i+,i+nbsommetsX, i+nbsommetsX+1)(i=1)

Le code suivant permet d'initialiser l'index buffer.

```
32          public void buildIndexBuffer(int h,int w)
33          {
34              // w = nb de sommet en X = largeur de Hightmap
35              // w = nb de sommet en X = largeur de Hightmap
36              int nbcellsX = w - 1, nbcellsY = h - 1; // nbr de cellule =nb de sommets -1
37              int nbcells = nbcellsX * nbcellsY;
38              int nbTriangles = nbcells * 2;  // deux triangles par cellule
39
40              int index = 0;
41              int startVertex = 0;
42              int[] indexBuffer = new int[nbTriangles * 3]; // trois index par triangle
43              for (int cellY = 0; cellY < nbcellsY; cellY++)
44              {
45                  for (int cellX = 0; cellX < nbcellsX; cellX++)
46                  {
47                      // triangle 1 dans la cellule
48                      indexBuffer[index] = startVertex + 0;
49                      indexBuffer[index + 1] = startVertex + 1;
50                      indexBuffer[index + 2] = startVertex + w;
51
52                      index += 3;
53
54                      // triangle 2 dans la cellule
55                      indexBuffer[index] = startVertex + 1;
56                      indexBuffer[index + 1] = startVertex + w + 1;
57                      indexBuffer[index + 2] = startVertex + w;
58
59                      index += 3;
60
61                      startVertex++;
62                  }
63                  startVertex++;
64              }
65          }
```

Extrait de code III-3 code d'initialisation de l'index buffer du terrain à partir de la carte des hauteurs

Une fois nous avons l'index buffer de notre terrain, nous devons calculer les normales et les coordonnés de textures pour chaque sommet. Pour les coordonnés de textures il est très facile de les générer lors de la création de vertex buffer(, en effet, chaque sommet généré à partir d'un pixel (x,y) dans le HightMap aura les coordonnés de texture (u,v) suivants :

$$U = x/w$$
$$V = y/h$$

Où w et h sont les dimensions de HightMap

```
103             for (int y = 0; y < heightfield.Height; y++)
104             {
105                 for (int x = 0; x < heightfield.Width; x++)
106                 {
107                     Vector3 position;
108 
109                     // position the vertices so that the terrain  is centered
110                     // around x=0,z=0
111                     position.X = terrainScale * (x - ((heightfield.Width - 1) / 2.0f));
112                     position.Z = terrainScale * (y - ((heightfield.Height - 1) / 2.0f));
113                     position.Y = (heightfield.GetPixel(x, y) - 1) * terrainBumpiness;
114                     vertices[i] = position;
115                     float u = x / heightfield.Width;
116                     float v = y / heightfield.Height;
117                     Vector2 uv = new Vector2(u, v);
118                     texcord[i++] = uv;
119                 }
120             }
121 
```

Extrait de code III-4 Génération du vertex buffer

La génération des normales est un peu plus compliquée, en réalité l'un des inconvénients des index buffer est le fait que nous ne pouvons attribuer qu'une seule normale par sommet, dans le cas de plusieurs triangles qui partagent un sommet, la solution couramment utilisée consiste à attribuer à ce sommet la moyenne des normales des triangles qui le partagent. Nous pouvons calculer les normales de la manière suivante :

1) Mettre tous les normales à (0, 0, 0)
2) Pour chaque triangle :
    a) Calculer la normale de triangle en utilisant le produit vectorielle de deux vecteurs appartenant à ce triangle, typiquement V0-V1 et V2-V1 tel que V0,V1 et V2 sont les sommets constituant le triangle.
    b) Ajouter la normal calculée aux trois sommets
3) Diviser la normal de chaque sommet par le nombre de triangles qui le partagent.

Le code suivant permet de calculer la normal de chaque sommet du terrain.

```
{
    // initialiser tous les normals à (0,0,0)
    Vector3[] normals = new Vector3[nbvertices];
    for (int i = 0; i < normals.Length; i++)
    {
        normals[i] = Vector3.Zero;
        counter[i] = 0;
    }
    // calculer les normal de tous les triangles
    // et ajouter la normal au sommets;
    for (int i = 0; i < indexbuffer.Length; i += 3)
    {
        // obtenir les trois sommet formant le triangle
        Vector3 v0 = vertexbuffer[indexbuffer[i]];
        Vector3 v1 = vertexbuffer[indexbuffer[i + 1]];
        Vector3 v2 = vertexbuffer[indexbuffer[i + 2]];
        // calculer la normal de ce triangle
        Vector3 vec1 = v1 - v0;
        Vector3 vec2 = v2 - v1;
        Vector3 triangleNormal = Vector3.Cross(vec1, vec2);
        // ajouter la normal aux tois sommets
        normals[indexbuffer[i]] = normals[indexbuffer[i]] + triangleNormal;
        normals[indexbuffer[i + 1]] = normals[indexbuffer[i + 1]] + triangleNormal;
        normals[indexbuffer[i + 2]] = normals[indexbuffer[i + 2]] + triangleNormal;
        // mettee à jour le nombre de vecteurs normales ajoutés à chaque sommet
        counter[indexbuffer[i]]++;
        counter[indexbuffer[i + 1]]++;
        counter[indexbuffer[i] + 2]++;
    }
    // calculer la moyenne des normales
    for (int i = 0; i < normals.Length; i++)
    {
        if (counter[i] != 0)
            normals[i] = normals[i] / counter[i]; // counters[i] jamais égale à 0 en tt cas.
    }
```

Extrait de code III-5 calcul des normales du terrain

La représente le rendu du terrain à partir du HightMap

Figure III-21 Rendu du terrain à partir de la carte des hauteurs

La carte des hauteurs permet aussi d'aligner le réseau routier et le terrain, nous avons effectué cette étape afin que les routes et les édifices soient parfaitement alignés avec le terrain, le résulta du rendu du terrain et de la ville virtuelle est présenté dans la Figure III-22

Figure III-22 Rendu du terrain et de la ville virtuelle.

III.4.7 Programmation de la logique du jeu

La programmation de la logique de notre jeu se résume en quatre étapes élémentaires :

- Établissement et implémentation du modèle physique d'animation de véhicule : ce modèle sera utilisé par le véhicule conduit par le joueur et les autres véhicules gérés par le moteur game ;
- Établissement et implémentation des mécanismes de test de collisions ;
- Gestion de la caméra ;
- Implémenter une interface utilisateur du jeu.

III.4.7.1 Modèle physique de voiture

Le véhicule conduit par le joueur représente l'acteur principal du jeu, nous devons simuler correctement son mouvement de manière réaliste, ce réalisme est important, d'un côté, pour rendre le jeu plus attirant et motivant, et d'un autre côté, pour doter le véhicule du jeu d'un comportement réaliste permettant d'entraîner le

joueur à certains aspects de conduite qui ont une relation directe avec les lois de la physique, la distance minimale pour freiner, par exemple.

Pour simuler correctement le mouvement d'un véhicule, nous devons comprendre les lois de la dynamique qui contrôlent ce mouvement. En général, les mouvements d'intérêt sont le freinage, l'accélération et les virages. Les forces imposées sur le véhicule par le moteur, la gravité, et l'aérodynamique influencent son comportement. Dans ce qui suit, nous essayons d'étudier ces différentes forces afin de pouvoir simuler les mouvements du véhicule.

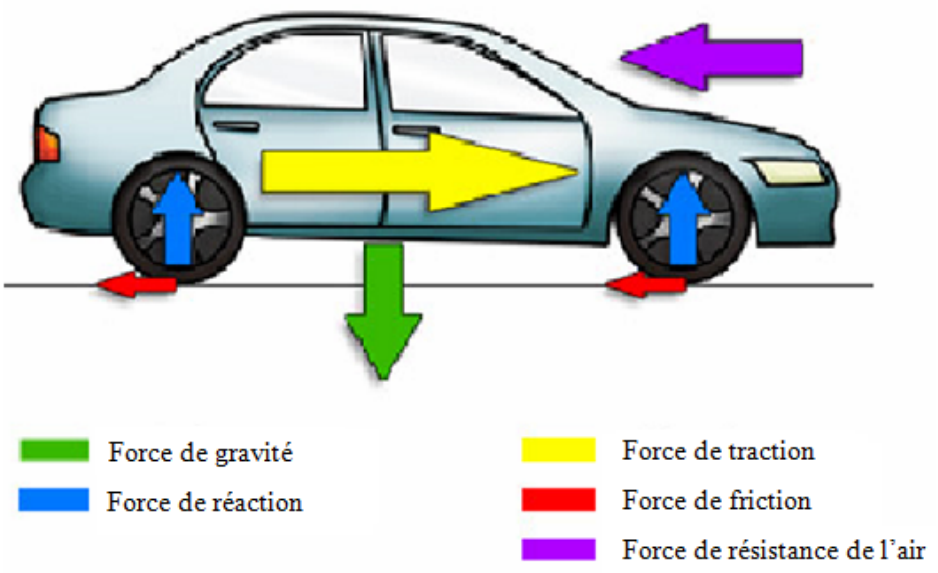


Figure III-23 les forces agissant sur le véhicule.

D'abord, nous allons considérer une voiture qui se déplace en ligne droite. La principale force qui influence le mouvement de la voiture est la force de traction ou de propulsion, à savoir la force délivrée par le moteur via les roues motrices. Nous allons considérer que la force de traction est l'équivalent en amplitude à la variable *Engineforce*, qui est contrôlée directement par l'utilisateur.

$$f_{traction} = D * EngineForce$$

Où D est un vecteur normalisé qui représente la direction du mouvement de la voiture.

Si la force de traction est la seule force qui contrôle la voiture, cette dernière va croître à des vitesses infinies. De toute évidence, ce n'est pas le cas dans la réalité. La voiture est soumise à des forces de résistance. La première et la plus importante

force est la force de résistance de l'air f_{drag}, cette force est proportionnelle au carré de la vitesse de la voiture. Lorsque la voiture roule vite cela devient la force de résistance la plus importante.

$$f_{drag} = -C_{drag} v|v|$$

Où C_{drag} est une constante et v est la vélocité et la notation $|v|$ désigne l'amplitude du vecteur v, l'amplitude du vecteur v est plus communément connue sous le nom de la vitesse.

$$|v| = \sqrt{{v_x}^2 + {v_y}^2}$$

$$f_{drag.x} = C_{drag} * v_x * |v|$$

$$f_{drag.y} = C_{drag} * v_y * |v|$$

Ensuite, il y a la résistance au roulement, cette force est provoquée par le frottement entre le caoutchouc et la surface de route. Nous allons approcher cette force de résistance avec une force qui est proportionnelle à la vitesse de voiture.

$$f_r = -C_r v$$

Où C_r est une constante et v est le vecteur vélocité.

A faible vitesse, la résistance au roulement représente la force de résistance principale, à des vitesses élevées la résistance de l'aire devient la force de résistance principale. A environ 100 km/h (30 m / s), elles sont égales [27]. Cela signifie que la constante C_r moyenne doit être environ 30 fois la valeur de C_{drag}.

La force totale longitudinale est la somme vectorielle de ces trois forces.

$$f_{long} = f_{traction} + f_{drag} + f_r$$

Lorsque le véhicule circule à une vitesse constante, les forces de traction et de résistance sont en équilibre, et f_{long} est égale à zéro.

L'accélération de la voiture (en mètres par seconde au carré) est déterminée par la force nette de la voiture (en Newton) et la masse de la voiture M (en kilogrammes) par le biais de la deuxième loi de Newton:

$$a = \frac{F}{M}$$

La vélocité de la voiture (en mètres par seconde) est déterminée par intégration de l'accélération dans le temps. Cela semble plus compliqué qu'il ne l'est, le plus souvent l'équation suivante est suffisance. Ceci est connu comme la méthode d'Euler pour l'intégration numérique.

$$v = v + dt * a$$

Où dt est l'incrément de temps en secondes entre deux appels successifs du moteur physique.

La position de la voiture est à son tour déterminée par l'intégration de la vitesse par rapport au temps:

$$p = p + dt * v$$

Avec ces trois forces, nous pouvons simuler l'accélération de la voiture de façon assez précise. Ensemble, elles déterminent également la vitesse maximale de la voiture pour une puissance donnée du moteur. Il n'est pas nécessaire de mettre une vitesse maximale dans le code, les équations forment une sorte de boucle de rétroaction négative. Si la force de traction est supérieure à toutes les autres forces, la voiture accélère. Cela signifie une augmentation de vélocité ce qui incite les forces de résistance à augmenter. La force nette diminue et l'accélération diminue en conséquence. À un certain point, les forces de résistance et la force du moteur s'annulent et la voiture a atteint sa vitesse maximale.

Lors du freinage, la force de traction est remplacée par une force de freinage qui est orientée dans la direction opposée. La force totale qui agit sur la voiture est alors la somme vectorielle de ces trois forces.

$$f_{long} = f_{break} + f_{drag} + f_r$$
$$f_{break} = -D * C_{break}$$

Dans ce cas, la force de freinage est supposée constante, il faut annuler cette force dès que la vitesse de la voiture devienne nulle sinon le véhicule finit par aller dans le sens inverse.

Transfert de charge

Dans le jargon automobile, le transfert de charge représente la variation des forces exercées sur les pneus et le revêtement. Le transfert de charge s'effectue durant les différentes manœuvres (freinage, virage, etc.) et est dû à des forces appliquées à la surface du pneu parallèlement au revêtement. Si une voiture accélère, le transfert de charge se fait sur l'essieu arrière, les pneus arrière sont donc plus "collés" à la route, il est donc intéressant, pour les voitures sportives de mettre les roues motrices à l'arrière, pour éviter le patinage. Si une voiture freine, le transfert de charge se porte sur l'essieu avant. Si le centre de gravité se situe à l'arrière alors durant le freinage il y aurait une bonne répartition des forces sur les pneus. Ainsi les voitures sportives à moteur arrière ont la réputation de mieux freiner que celle à moteur à l'avant.

L'effet de transfert de charge est important pour les jeux de voiture pour deux raisons. Tout d'abord l'effet visuel la voiture qui se penche visiblement vers l'arrière en réponse au freinage ou « s'accroupit ». Réciproquement ajoute beaucoup de réalisme au jeu et la simulation devient beaucoup plus réaliste.

Deuxièmement, la répartition de la charge affecte considérablement la force de traction maximale par roue,

C'est parce qu'il y a une limite de frottement pour une roue qui est proportionnelle à la charge sur cette roue:

$$f_{max} = \mu W$$

Où μ est le coefficient de friction du pneu. Pour les pneus de voitures ordinaires cela peut être de 1,0, pour les pneus de voiture de course il prend des valeur plus grandes jusqu'à 1,5.

Pour un véhicule stationnaire, la charge total du véhicule (W, qui est égale à M * g) est répartie sur les roues avant et arrière selon la distance de l'essieu arrière et avant à la CM (c et b respectivement):

$$W_r = \frac{b}{l} W$$

$$W_f = \frac{c}{l} W$$

Où b est la distance entre l'essieu avant et le CM, c la distance entre l'essieu arrière et le CM et L représente l'empattement

Si la voiture accélère ou décélère au taux a, la charge sur l'avant (W_f) et l'essieu arrière (W_r) peut être calculée comme suit:

$$W_f = \frac{c}{l}W - \frac{h}{l}Ma$$

$$W_r = \frac{b}{l}W + \frac{h}{l}Ma$$

Où h est la hauteur du centre de gravité, M est la masse du véhicule et a est l'accélération (négative dans le cas de la décélération).

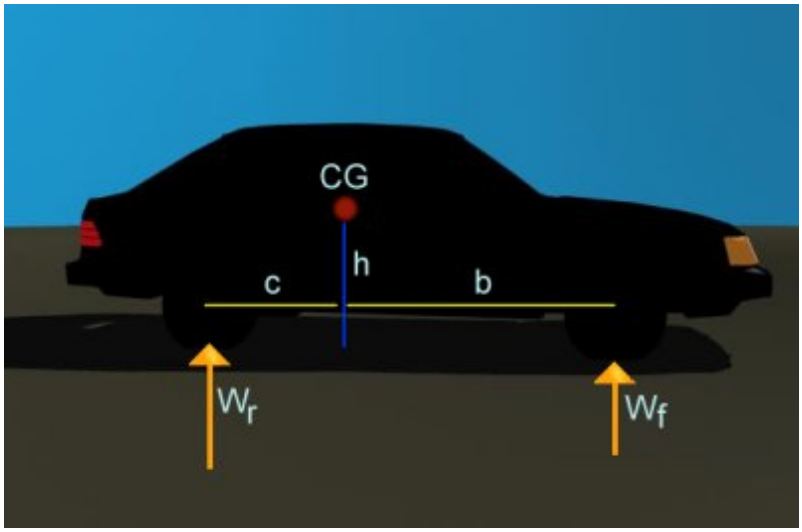


Figure III-24 Distribution de charge dans le cas d'un véhicule stationnaire

III.4.8 Implémentation du modèle physique

Pour avoir une simulation correcte du mouvement de la voiture, nous devons spécifier quelques propriétés physiques telles que le centre de masse, le poids et la puissance maximale du moteur.

La première propriété à définir est le poids de notre voiture, cette propriété joue un rôle important dans la simulation du comportement de la voiture, alors il est très important de choisir des valeurs qui sont proches de la réalité, une valeur entre 1000 et 2000 Kilogrammes est souvent acceptable. Pour notre modèle de voiture sport nous avons estimé son poids à 1300 Kilogrammes

Le centre de masse de la voiture n'est pas forcément le centre de modèle 3D de voiture, la position du centre de masse dépend de la position du moteur ainsi que d'autres facteurs. Cela peut varier d'un modèle de voiture à l'autre, pour le modèle de voiture utilisé dans notre jeu le centre de masse est placé avant le moteur et légèrement au-dessus du plancher de la voiture comme le montre la Figure III-26.

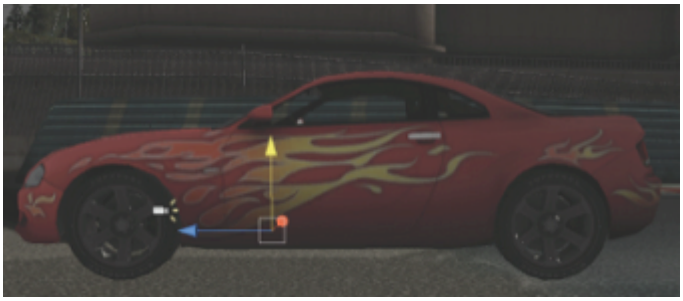


Figure III-25 Centre de masse de la voiture

En général, il est une mauvaise idée de positionner le centre de masse d'un côté ou de l'autre de l'axe des x de la voiture, ceci peut provoquer un comportement bizarre et inattendu de la voiture lors de sa direction, ainsi le centre de masse doit être toujours centré sur l'axe X de la voiture.

Le véhicule sera implémenté en utilisant trois classes :

- La classe *Véhicule* qui représente le véhicule ;
- La classe *Roue* qui représente une roue de véhicule ;
- La classe *Châssis* le châssis de véhicule.

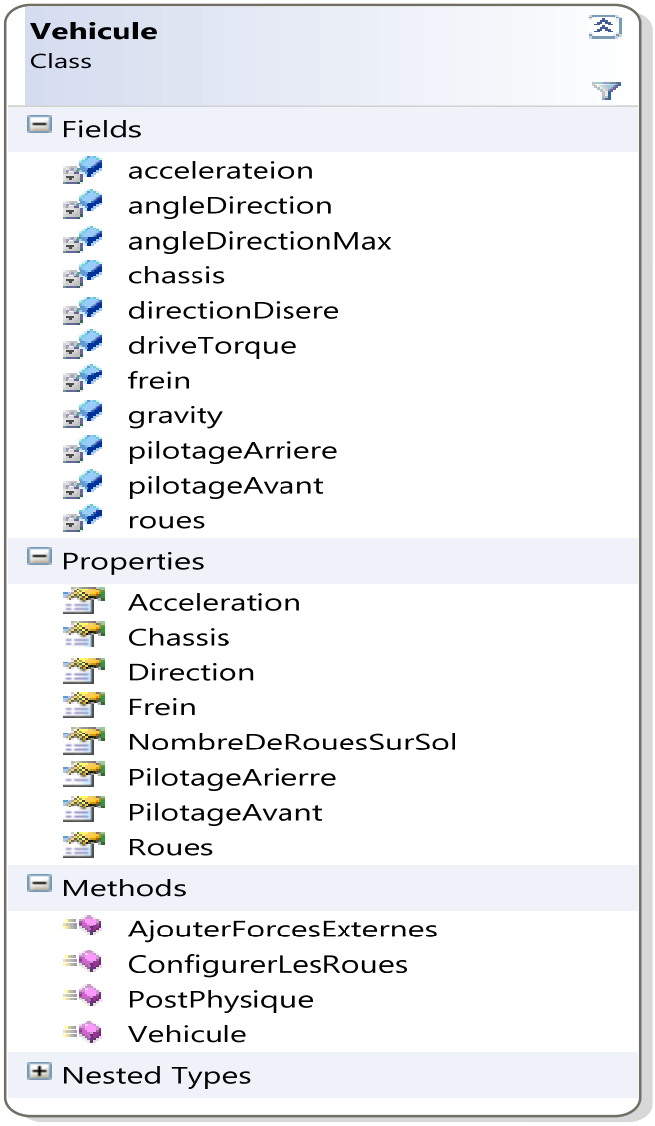


Figure III-26 Diagramme de classe de la classe *Véhicule*

On remarque qu'il existe trois types d'éléments dans ce diagramme de classe, les attributs (*fileds*), les propriétés (*properties*), et les méthodes (*methods*). Les langages orientés objets tels que java et C++ distinguent entre attributs et méthodes. Le langage C# ajoute la notion de propriétés à la notion de classe. Une propriété s'utilise comme un attribut et fonctionne comme une méthode.

Par exemple, la propriété `NombreDeRouesSurSol` qui détermine le nombre de roues actuellement en contact avec le sol peut être utilisée de la manière suivante :

x = *car*.`NombreDeRouesSurSol;`

```
10          public int NombreDeRouesSurSol
11          {
12              get
13              {
14                  int nb = 0;
15
16                  for (int i = 0; i < roues.Count; i++)
17                  {
18                      if (roues[i].SurSol)
19                          nb++;
20                  }
21                  return nb;
22
23              }
24          }
25
```

Extrait de code III-6 Déclaration d'une propriété en c#

Le véhicule sera contrôlé par l'ensemble de propriétés présent dans le diagramme de classe, la table suivante représente le rôle et les valeurs possibles de chaque propriété.

Propriété	Valeurs possibles	Rôle
Accélération	-1, 0 et 1	-1.0 accélérer vers l'arrière 0 pas d'accélération 1.0 accélérer vers l'avant
Direction	Entre -1, 0 et 1	-1.0 tourner à gauche ; 0 Même direction. 1.0 tourner à droite.
`Frein`	0 1	1 freiner le véhicule.
`PilotageAvant` `PilotageArriere`	Vrai /faux	Indique les roues motrices du véhicule.

Tableau III-6 rôles et valeurs des propriétés de la classe Vehicule.

Le code suivant représente l'association de contrôle du véhicule aux touches du clavier de la manière suivante :

- Touche flèche haut : accélérer le véhicule vers l'avant ;
- Touche flèche bas : décélérer le véhicule, en cas d'arrêt elle à l'effet d'accélérer le véhicule vers l'arrière.
- Touche flèche droite/gauche : tourner le véhicule respectivement vers la droite ou la gauche ;

```
10
11              if (keyState.IsKeyDown(Keys.Up) || keyState.IsKeyDown(Keys.Down))
12              {
13                  if (keyState.IsKeyDown(Keys.Up))
14                      carObject.Car.Acceleration = 1.0f;
15                  else
16                      carObject.Car.Acceleration = -1.0f;
17              }
18              else
19                  carObject.Car.Acceleration = 0.0f;
20
21              if (keyState.IsKeyDown(Keys.Left) ||
22                  keyState.IsKeyDown(Keys.Right))
23              {
24                  if (keyState.IsKeyDown(Keys.Left))
25                      carObject.Car.Direction = 1.0f;
26                  else
27                      carObject.Car.Direction = -1.0f;
28              }
29              else
30                  carObject.Car.Direction = 0.0f;
31
32              if (keyState.IsKeyDown(Keys.B))
33                  carObject.Car.Frein = 1.0f;
34              else
35                  carObject.Car.Frein = 0.0f;
36            }
37        }
```

Extrait de code III-7 Contrôle du véhicule avec les touches du clavier

En réalité, les propriétés sont associées à des variables locales de la classe Véhicule, ces variables sont utilisées durant la mise à jour de l'état du véhicule. Pour comprendre le mécanisme, nous prenons comme exemple le cas de la propriété *Direction*. Cette propriété est associée aux touches flèche droite et gauche du clavier et utilisée pour faire tourner le véhicule.

```
publicfloat Direction
         {
get {return directionDisere;}
set {directionDisere = value;}
 }
```

La proprité `Direction` joue le rôle d'une interface à une variable locale `directionDisere`, cette variable signifie que l'on veut tourner le véhicule dans une direction donnée. Le code permettant de faire tourner le véhicule est le suivant :

```
11          float dtderection = dt * 0.15;
12
13          direction += dtderection;
14
15          int inner, outer;
16          if (angleDirection > 0.0f)
17          {
18              inner = (int)WheelId.WheelFL;
19              outer = (int)WheelId.WheelFR;
20          }
21          else
22          {
23              inner = (int)WheelId.WheelFR;
24              outer = (int)WheelId.WheelFL;
25          }
26
27          float alpha = System.Math.Abs(angleDirectionMax * angleDirection);
28          float angleSgn = angleDirection > 0.0f ? 1.0f : -1.0f;
29          roues[inner].AngleDeDirection = (angleSgn * alpha);
30
31          float beta;
32
33          if (alpha == 0.0f)
34          {
35              beta = alpha;
36          }
37          else
38          {
39              Beta = alpha - alpha * 0.80;
40          }
41          roues[outer].AngleDeDirection = (angleSgn * beta);
42
43
```

Extrait de code III-8 Changement de la direction de véhicule

I'objectif n'est pas d'entrer dans le détail de chaque ligne de code mais plutôt de comprendre l'idée générale. A partir de la variable `directionDisere` qui indique que le joueur veut tourner le véhicule dans une direction déterminée par le signe de cette variable, nous devons calculer l'angle permettant de faire tourner les roues avant, en réalité les roues avant ne tournent pas avec le même angle, la roue intérieure (par rapport à la courbe de braquage) tourne plus que le roue extérieure.

La quantité de braquage représentée par la variable `direction` augmente progressivement tandis que la valeur de la variable `directionDisere` est différente de zéro. Selon la signe de la variable *steering,* nous déterminons quelle sera le roue intérieure et extérieure, nous calculons l'angle de rotation de chaque

roue et nous passons la valeur à l'objet *Wheel*. Il est évident que les roues ne vont pas tourner à 360 degré, donc, il faut fixer un angle de braquage maximal pour les roues, c'est le rôle de la variable `angleDirectionMax`.

Nous remarquons la présence d'une variable *dt* qui contrôle la quantité avec laquelle nous augmentons l'angle de braquage, il est très difficile d'expliquer le rôle de cette variable sans expliquer l'un des problèmes fondamentaux dans la programmation des jeux vidéo, nous parlons de la fameuse **boucle de jeu** ou gameloop.

Un jeu n'est autre qu'une boucle infinie qui exécute deux procédure *update()* et *render(),* dans le procédure update nous vérifions les entrées de l'utilisateur et nous mettons à jours les objets actifs dans le jeu (la voiture dans notre cas), dans le procédure *render()* nous affichons les objets selon le nouveau état du monde après la modification.

Ainsi, le test pour les touches du clavier et la mise à jour de l'angle de braquage des roues se font à l'intérieur de la procédure update. Le problème est le suivant : quel est la vitesse d'exécution de cette boucle, le temps d'exécution d'une itération de la boucle du jeu n'est pas prévisible, il dépend de plusieurs paramètres tels que la complexité des calculs dans la partie update, la complexité de la scène (temps de rendu), la puissance de calcul de la machine sur laquelle s'exécute le jeu,… etc.

Pour faire simple, le nombre d'itérations que la boucle du jeu effectue par seconde est variable d'une machine à l'autre, et il est variable même dans le jeu d'un monument à l'autre pour les raisons expliquées auparavant. En fait, le nombre d'itérations effectuées par seconde représente entre autre la vitesse du jeu, il est appelé le FPS (frames per second), il est utilisé par les développeurs pour mesurer les performances du jeu. A titre d'exemple, si on a deux algorithmes de rendu A et B qui sont appliqués sur la même scène, si l'algorithme A permet d'attendre 50 FPS et l'algorithme B permet d'attendre 60 FPS, le développeur peut choisir d'utiliser l'algorithme B car il donne les meilleures performances. Maintenant si le jeu s'exécute à 35 FPS sur une machine ordinaire, il peut s'exécuter à 65 FPS sur une machine plus puissante.

Bien qu'il soit possible de contrôler la vitesse d'exécution de la boucle du jeu à travers des mécanismes comme la suspension de processus ou l'utilisation des intervalles de temps, la quasi-totalité des jeux modernes sont conçus pour s'exécuter dans une boucle infinie à vitesse non contrôlée. Les raisons en sont simples : un jeu qui s'exécute avec un taux de 65 FPS aura une animation plus fluide est réaliste qu'un jeu qui s'exécute sur 35 FPS. Cette variabilité dans la vitesse d'exécution de la boucle du jeu n'a pas le même effet sur la mise à jour des objets dans le jeu. Par exemple, imaginons un objet qui bouge dans un jeu avec la formule :

Objet.position.x = Objet.position.x + 0.05 ;

Dans une machine A qui exécute le jeu à 40 FPS par exemple, l'objet parcourra 4 m après 2 seconde (40*0.05*2). Dans autre machine B plus puissante qui exécute le jeu à 65 FPS, l'objet parcourra 6.5 m (65*0.05*2)! Autrement dit, le problème est que la portion de mise à jour effectuée chaque seconde par le jeu varie selon le FPS du jeu par rapport à la machine sur laquelle il s'exécute.

La solution à ce problème consiste à lier le taux de mise à jour de l'état des objets à la fréquence de la boucle du jeu à laquelle s'exécute le jeu, c'est-à-dire, plus le jeu s'exécute vite plus on diminue la portion de mise à jour, plus la vitesse de boucle de jeu diminue plus on augmente la portion par laquelle on met à jour la scène.

Dt = 50/FPS

Objet.position.x = Objet.position.x + 0.05 *Dt;

Avec ce mécanisme l'objet fera toujours 5 m dans les deux machines

Machine A avec 40 FPS 40*50/40*0.05*2 =5m

Machine A avec 65 FPS 65*50/65*0.05*2=5m

En pratique, la valeur de FPS elle-même n'est pas constante, alors on utilise le temps d'exécution de l'itération précédente de la boucle du jeu.

En ce qui concerne notre exemple de braquage, nous pouvons maintenant comprendre facilement la signification de la ligne `dtderection = dt * 0.15;` où `dtderection` et la portion par laquelle nous augmentons l'angle de braquage et *dt* est taux d'augmentation de cette angle calculé à partir du temps d'exécution du dernier frame.

Afin d'augmenter le réalisme visuel de notre simulation, nous introduisons un autre facteur dans notre implémentation, il s'agit de la suspension de notre véhicule, l'objectif est de simuler la façon dont la voiture répond aux virages, à l'accélération, au freinage brusque… etc. Pour atteindre cet objectif, nous avons besoin d'avoir une sorte de modèle de la suspension de la voiture.

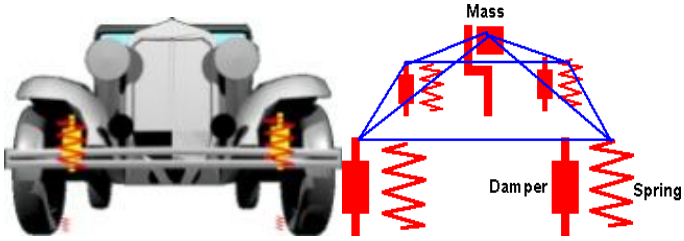


Figure III-27 modèle de suspension d'un véhicule

Un modèle simple de susponsion est schématisé dans la Figure III-27, dans lequel la susponsion est caractirisée par trois éléments : masse, ressort et amortisseur.

Le premier élément de la susponsion est le ressort de sussponsion (spring), il est défini par sa raideur, une voiture très lourde nécessite un ressort avec plus de raideur qu'une voiture légère. En fonction de la distribution de la masse du véhicule, les ressorts avant et arrière ne doivent pas avoir la même raideur. Donc les ressorts avent de notre véhicule doivnt avoir une valeur plus imporatnte que ceux de l'arrière.

Le deuxième élément de la susspontion est appelé amortisseur. Sans amortisseur, le ressort de sussponsion va s'étendre et se relache de manière incontrôlable ce qui résulte en une voiture bondisante. L'amortisseur absorbe l'énergie cinétique dégagée par le ressort et la transofrme en chaleur qui sera dissipée par un liquide hydraulique.

Les paramètres de sussponsion et les caractiristiques de friction des roues on été implémentés séparément dans la classe *Roue* dont le diagramme de classe est représenté dans la Figure III-28; C'est au niveu de cette classe qu'une bonne partie de l'implémentation du modèle physique est réalisée.

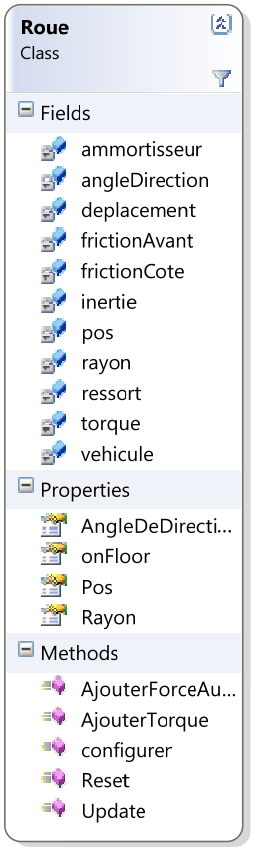


Figure III-28 Diagramme de classe de la classe *Roue*

L'étape suivante consiste à implémenter le comportement et le mouvement du véhicule selon le modèle expliqué auparavant. Il est plus ou moins facile d'implémenter directement le comportement du véhicule à partir des forces définies dans le modèle physique. Le plus difficile sera d'implémenter les réactions du véhicule par rapport à l'environnement, par exemple, choc entre deux véhicules, un véhicule est un objet…etc.

En réalité, le mouvement et l'interaction entre les objets dans les jeux est géré par ce qu'on appelle le moteur physique, son rôle étant de calculer le mouvement des objets, la manière dont ils interagissent les uns avec les autres, la manière dont ils glissent sur le sol ou sur les murs, la manière dont ils rebondissent … etc. C'est lui aussi qui calculera la déformation des objets mous, des cheveux, poils, vêtements et autres rideaux.

Dans notre cas, par exemple, nous définissons les différentes forces agissant sur notre véhicule ainsi que ses propriétés physiques et le moteur de jeu prend en

charge le calcul du mouvement du véhicule. Cela est réalisé au niveau de la classe *Châssis,* cette dernière est composée de deux éléments Body et Skin.

Body est la représentation de notre véhicule dans le moteur physique, c'est à travers cette composante que nous spécifions au moteur physique l'ensemble de propriétés physiques de notre véhicule ainsi que les forces qu'ils le contrôlent.

Le skin représente une sorte d'enveloppe du véhicule, il sera utilisé pour le test de collision, les tests de collision seront l'objet de la prochaine section de ce chapitre.

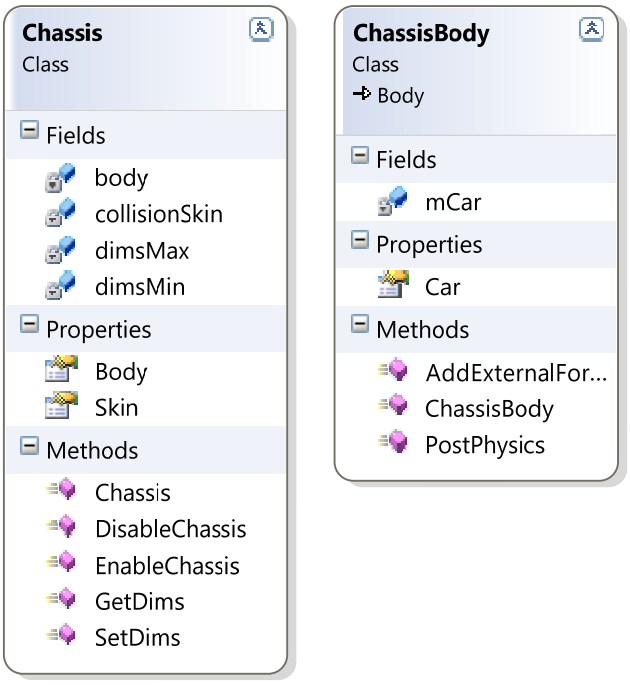


Figure III-29 Diagramme de classe de la classe Chassis

III.4.9 Test de collisions

Le test de collusion permet de détecter si deux objets s'interpénètrent, dans le cas d'un jeu vidéo, ce test est primordial car il est à la base de toute la logique du jeu. Par exemple, dans un jeu de stratégie militaire, si un missile est lancé, un test de collision est effectué entre ce dernier est les objets de la scène, si une collision est détectée avec un objet, ce dernier sera détruit.

Cet exemple est un cas typique de l'utilisation du test de collision dans les jeux, le problème est divisé en deux phases, détection de collision et réponse à la

relations positionnelles des objets dans la scène, la deuxième phase dite réponse à la collision est un problème de **dynamique** qui implique la prédiction du comportement des objets par rapport aux lois de la physique et les règles du jeu.

Le problème le plus sérieux avec la détection de collisions est le nombre de tests qui doivent être effectués à chaque itération de la boucle du jeu. Par exemple, si nous avons 'n' objets mobiles et 'm' objets statiques, le nombre de collisions potentielles à tester est obtenu de la manière suivante :

Chaque objet mobile pourrait entrer en collision avec un objet statique donc le nombre total de collisions à tester entre les objets statiques et mobiles est : n*m test de collision.

Pour les objets mobiles, le premier objet pourrait entrer en collision avec n-1 objets (puisque nous n'avons pas à vérifier si un objet est entré en collision avec lui-même) le second objet pourrait entrer en collision avec (n-2) des objets supplémentaires...etc. Ainsi, le nombre total de collisions potentielles entre les objets mobiles est : (n-1) * (n-2) * (n-3) ... 1 = n! / 2!*(n-2)!.

De la sorte, le nombre total de collisions à tester est : n! / 2!*(n-2)! + n*m.

Dans un jeu, nous avons besoin d'effectuer des tests de collision à chaque itération de la boucle du jeu, il est donc important que la détection de collision soit très efficace.

Partitionnement de l'espace du jeu

Comme nous l'avons déjà vu dans le paragraphe précédent, le nombre de tests de collisions à effectuer est proportionnel au nombre d'objets. Une façon de réduire le nombre de tests est de partitionner l'espace avec des octrees, kd-trees, des arbres BSP,…

Il suffit alors de tester des objets dans un espace donné (ou espaces adjacents éventuellement) en cas de collision.

Volumes englobant

Le test de collusion entre deux objets en utilisant leur géométrie est à la fois très compliqué et très coûteux, pour cette raison il est préférable d'utiliser des

formes géométriques qui permettent d'accomplir le test de collusion de manière très rapide.

Sphère englobante

Par exemple, le test de collusion entre deux sphères se réduit à la comparaison entre la distance séparant leurs deux centres et la somme de leurs rayons, si la distance entre les deux centres est inférieure à la somme des deux rayons, les deux sphères sont entrées en collusion. En plus de leur efficacité dans les tests de collusion, les sphères englobantes ont un autre avantage, lors de la rotation de l'objet englobé la sphère reste inchangée, en cas de mouvement il suffit d'ajouter le vecteur de déplacement à celui qui représente le centre de la sphère.

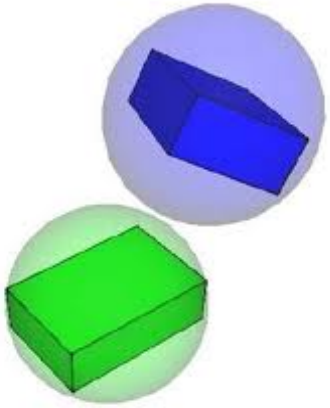


Figure III-30 Test de collision avec des sphères englobantes

Le problème majeur avec les sphères englobantes est qu'elles sont peu adaptées à des objets longs et étroits, dans ce cas, la sphère englobante aura beaucoup d'espace vide, ce qui pourrait entraîner des faux tests de collisions positives.

Boites englobantes

Nous pouvons également utiliser une boite englobante pour le test de collisions, l'avantage avec des boites englobantes est qu'elles sont mieux adaptées aux objets longs et étroits comme des bâtiments, voitures …etc.

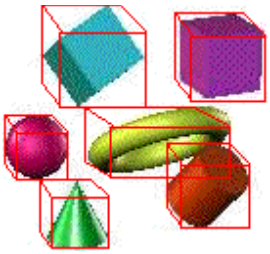


Figure III-31 Boites englobantes

Techniques mixtes

Pour certains objets qui ont une importance particulière dans le jeu, le test de collision avec des volumes englobants peut s'avérer insuffisant, la tendance est d'utiliser des tests de collisions hiérarchiques. Autrement dit, le test de collision se fait avec un volume qui englobe la totalité de l'objet, de sorte que si une collision est détectée par ce test, des tests supplémentaires sont effectués avec des volumes qui englobent des sous parties de l'objet.

La Figure III-32 représente les volumes englobant utilisés pour les tests de collision avec la voiture, à gauche la géométrie détaillée de la voiture englobée par une boite englobante, à droite une géométrie très proche de celle d'une voiture avec moins de polygones. Le principe est de tester la collision avec la boite englobante et si le test est positif, nous devons tester la collision avec la géométrie abrégée qui figure dans la partie droite de l'image.

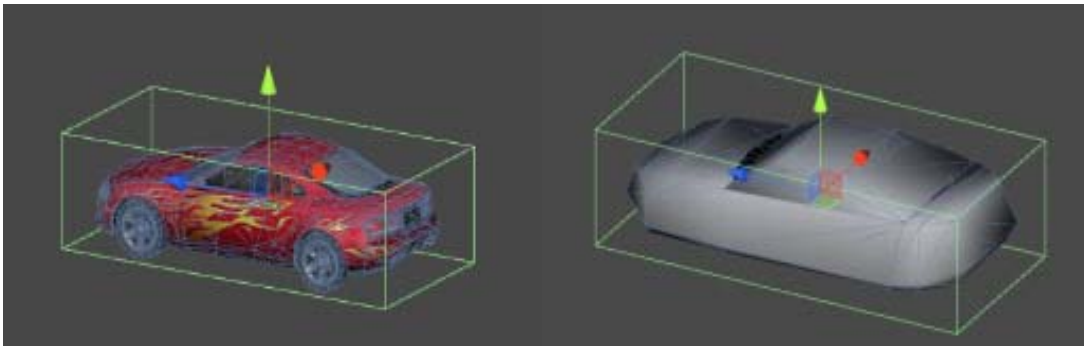


Figure III-32 Volumes englobant pour la voiture

Le mécanisme de test de collision hiérarchique est schématisé dans la Figure III-33

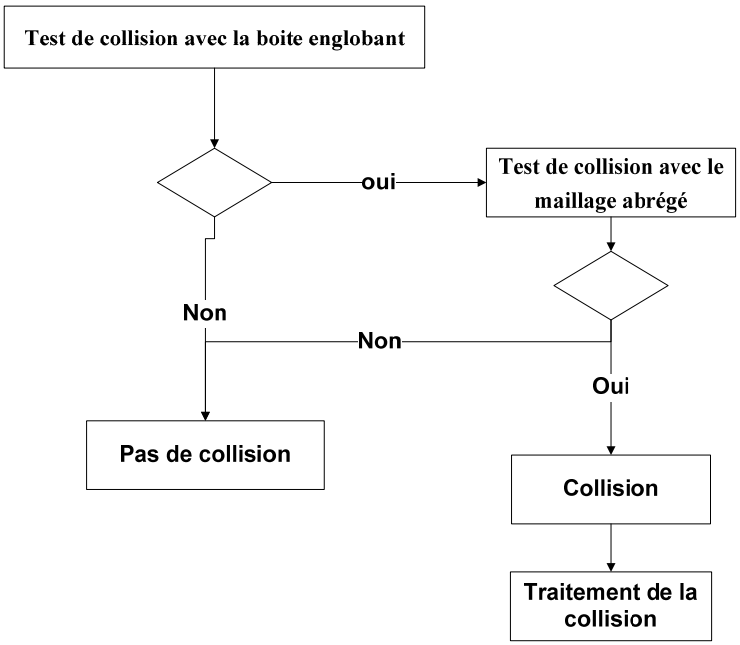


Figure III-33 Mécanisme de test de collision hiérarchique pour les voitures dans notre jeu

III.4.9.1 Test de collision avec l'environnement du jeu

Comme l'outil CityEngine n'est pas destiné principalement à la création des environnements pour les jeux vidéo, Les bâtiments générés avec cet outil posent un problème particulièrement concernent la détection de collision, d'un côté, l'outil ne génère pas une géométrie de collision pour les blocks de bâtiments, d'un autre côté, les bâtis sont exportés sous forme d'un seul maillage de polygones constitué de milliers de triangles, ce maillage n'est pas malheureusement structuré ce qui rend impossible de générer les volumes de collisions pour les blocks de bâtiments lors de lancement du jeu.

Pour résoudre ce problème, nous avons mobilisé (Figure III-34) un maillage de collision pour l'ensemble des édifices, ce maillage sera utilisé uniquement par le moteur physique, il sera placé dans la même position que les bâtiments mais il ne sera pas visible à l'utilisateur.

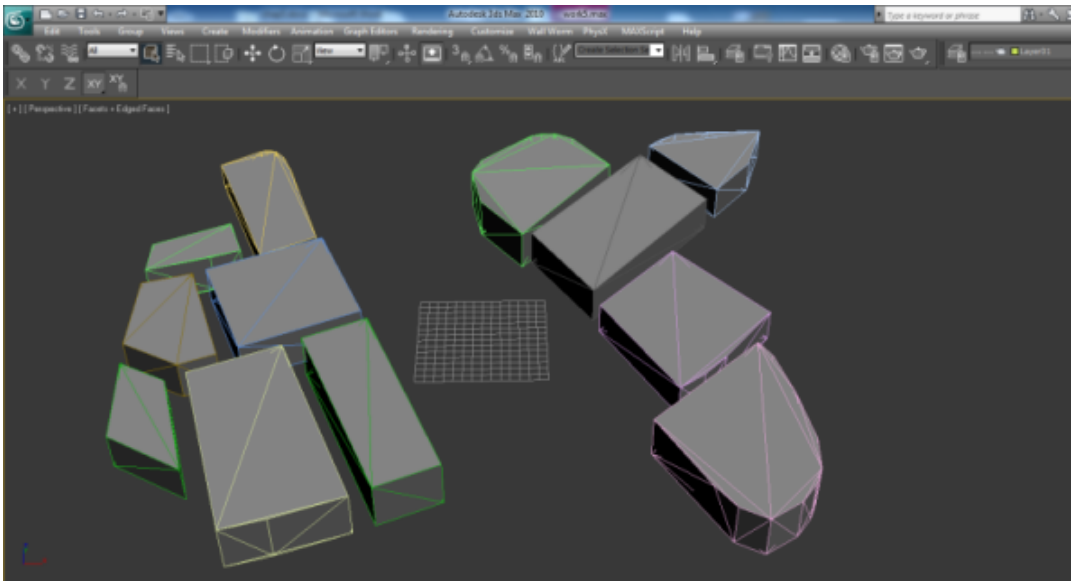


Figure III-34 Modélisation d'un maillage de collision pour les bâtiments

III.4.10 Gestion de la caméra

En mode réalité virtuel, nous avons besoin d'implémenter un type de caméra appelé caméra à la troisième personne (*third-person camera*) ou caméra de poursuite (*chase camera*), ce type de caméra est censé être placé derrière et au-dessus de caractère principale de jeu (dans notre cas derrière est légèrement au-dessus de véhicule), la position et l'orientation de la caméra doit être constamment mis à jour en fonction du mouvement de l'objet suivi par cette caméra.

La première étape consiste à choisir un point cible dans l'objet suivi, ce point va nous permettre de déterminer la rotation et la translation nécessaires pour placer la caméra en bonne position, dans le cas de notre jeu ce point est le l'origine de la géométrie de véhicule.

La deuxième étape consiste à définir une relation spatiale entre ce point et la caméra, par exemple la caméra sera placée trois mètres derrière le véhicule et à une altitude de 2 mètre sur l'horizon.

La meilleur notation qui permet de décrire cette relation spatiale entre la caméra et le centre de véhicule est celle de coordonnés sphérique, dans ce système de coordonnés, chaque point de l'espace est décrit par sa distance et sa rotation par rapport à un point particulier (l'origine). Une coordonnés sphérique est représenté pas trois composantes :

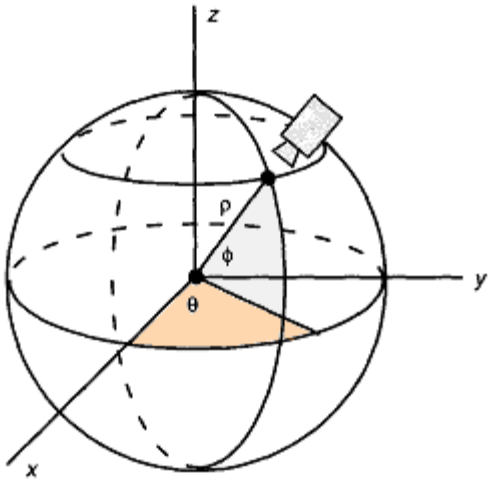


Figure III-35 Caméra à la troisième personne en coordonnés sphériques

- ρ est la distance du point P au centre O et donc ρ > 0;
- l'angle de rotation par rapport à l'axe Z (de 0 à 360 dégrées) ;
- φ l'angle de rotation du point p par rapport au plan X-Y (0 à 180 ou -90 à 90 degree).

En coordonnés sphériques, nous pouvons décrire l'offset idéel de la caméra par rapport au cible par un vecteur$(p, \emptyset, \varphi)$, ensuit, nous concertons ce vecteur en son équivalant cartésien pour obtenir la pose de la caméra.

$$x = \rho \cos\theta \sin\left(\phi + \frac{\pi}{2}\right)$$

$$y = \rho \sin\theta \sin\left(\phi + \frac{\pi}{2}\right)$$

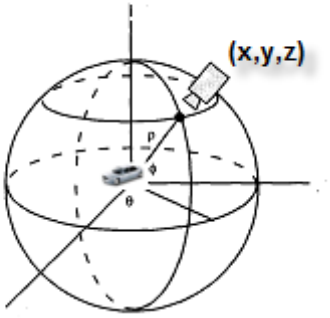
$$z = \rho \cos\left(\phi + \frac{\pi}{2}\right)$$

Il faut noter que dans la Figure III-35 ainsi que dans les équations précédentes l'axe Z est supposé être vers le haut, XNA utilise un système de coordonnées dont lequel l'axe Y pointe vers le haut est pas l'axe Z. nous avons pris cette différence en considération lors de l'implémentation des équitations de passage de coordonnés sphériques vers les coordonnés cartésiennes

```
316         public Vector3 sphiricalToCartisien(float p, float theta, float phi )
317         {
318             float x, y, z;
319             // x = p*cos(theta)*sin(phi+pi/2)
320             x = p * (float)Math.Cos(theta) * (float)Math.Sin(phi + MathHelper.PiOver2);
321             // z remplace y dans la formule
322             // z = p*sin(theta)*sin(phi+pi/2)
323             z = p * (float)Math.Sin(theta) * (float)Math.Sin(phi + MathHelper.PiOver2);
324             // y remplace z
325             // = p*cos(phi+pi/2)
326             y = p * (float)Math.Cos(phi + MathHelper.PiOver2);
327             return new Vector3(x, y, z);
328         }
```

Extrait de code III-9 Conversion des coordonnées sphériques en coordonnées cartésiennes

La coordonné p qui représente la distance de la caméra par rapport au véhicule sera fixé à 10 pour permettre au joueur de regarder le voiture et une parte de son environnement.



phi =45
P = 10
theta=atan2(y,x)

Durant l'exécution du jeu, et à chaque frame, la position de voiture est constamment récupéré depuis le moteur physique et donné au gestionnaire de la camera, ce dernier met à jour la camera de la manière suivante :

- Calculer l'angle $\emptyset$ en utilisant la formule $\emptyset = atan2(C.z - V.z, C.x - V.z)$ où V.x V.z sont les coordonnés de véhicule dans le plan X-Z, C.x, C.z sont les coordonnés de caméra dans ce plan. Atan2 est une variation de la fonction arc-tangente.atan2(y,x) est l'angle en radians entre la partie positive de l'axe des x d'un plan, et le point de ce plan aux coordonnées (x,y).

- Calculer l'offset de la caméra par rapport à la voiture en utilisant les coordonnés sphériques. $\emptyset$, p=10, $\varphi = 45^{o}$
- Placer et orienter la caméra

Le code suivant nous permet de mette à jours la caméra selon le mécanisme présenté ci-dessus.

```
278             // calculer l'angle theta
279             float theta = (float)Math.Atan2(cameraPosition.Z - TargetPosition.Z,
280                                             cameraPosition.X - TargetPosition.X);
281             float p = 10; // distance de la caméra par rapport à la voiture
282             float phi = -MathHelper.PiOver4; // caméra incliné à 45 degree
283             //position = position de voiture + offset
284             cameraPosition = TargetPosition + sphiricalToCartisien(p, theta, phi);
285             lookAt = TargetPosition; // point que le caméra doit regarder = le centre voiture
286             // créer la matrice view avec la pos de camera
287             view = Matrix.CreateLookAt(Position, LookAt, Vector3.Up);
288
```

Extrait de code III-10 Mise à jour de la position et l'orientation de la caméra

Après le teste de ce mécanisme, le mouvement de la caméra obtenu n'été pas satisfaisant, le mouvement été irrégulière et accompagné de vibrations, le problème est lié à la relation entre le mouvement de véhicule et le mouvement de la caméra, en effet, le véhicule effectue plusieurs mouvements et change la position et l'orientation pratiquement d'un frame à l'autre. Le fait que la position de caméra soit celle de voiture avec un offset rend le mouvement de caméra instable et irrégulier.

Pour faire face à ce problème il va falloir que la caméra soit dotée d'un mouvement séparé défini par une accélération et une vélocité propre à elle. La position calculée à partir de celle de voiture ne sera pas attribuée à la caméra mais elle sera considéré comme la prochaine destination de la caméra, ensuit la caméra va essayer d'attendre cette position en utilisant sa propre vélocité et accélération.

Pour éliminer les mouvements indésirables nous ajoutons au mécanisme précédent un système masse-ressort (Figure III-36), ce dernier va rendre le mouvement de la caméra plus lice et régulier.

$$F_R = -kx$$

$$F_A = -cx$$

$$F = F_R + F_A = -kx - cv$$

Où :

k : Raideur de ressort

c : Coefficient d'amortissement

v : Vélocité

x : L'étirement du ressort

Dans notre problème, x représente la distance entre la position actuelle de la caméra et la position idéale calculée en fonction de la position de voiture. L'accélération générée par le ressort va conduire la caméra doucement vers sa position idéale. Le rôle de l'amortisseur et d'amortir le mouvement quand la caméra atteint sa destination.

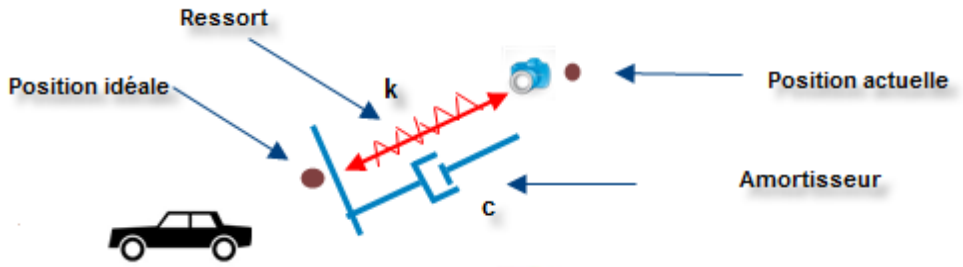


Figure III-36 Système mass-ressort-amorti utilisé pour contrôler le mouvement de la caméra.

```
292         private void UpdateCamera2(GameTime gameTime)
293         {
294             float theta = (float)Math.Atan2(cameraPosition.Z - TargetPosition.Z,
295                                             cameraPosition.X - TargetPosition.X);
296             float p = 10;
297             float phi = -MathHelper.PiOver4;
298             idealPosition = TargetPosition + sphiricalToCartisien(p, theta, phi);
299             float epsilon = (float)gameTime.ElapsedGameTime.TotalSeconds;
300             // calculer l'étirement de ressort
301             Vector3 stretch = cameraPosition - idealPosition;
302             // calculer la force de rappel du ressort Fr=-k*x
303             Vector3 fr = -k * stretch; // stretch et l'itérément = x dans la formule
304             // calculer la force de d'amortissement fa=-c*v
305             Vector3 fd = - c * velocity;
306             // calculer la somme des forces.
307             Vector3 force = fr+fd;
308             // accélération = force/masse
309             Vector3 acceleration = force / mass; // F=a*m alors a= F/m
310             // appliquer l'acélération par intégration numirique d'euler
311             velocity += acceleration * epsilon;
312             // appliquer la Vélocité par intégration numirique d'euler
313             cameraPosition += velocity * epsilon;
314             // créer la matrice view avec la pos de camera
315             view = Matrix.CreateLookAt(cameraPosition, TargetPosition, Vector3.Up);
316         }
```

Extrait de code III-11 système masse-ressort pour le contrôle de la caméra

III.5 Interface utilisateur du jeu

L'interaction avec l'utilisateur est implémentée aujourd'hui avec le modèle WIMP qui signifie (*Window, icon, menu and pointingdevice*). Dans le domaine des jeux vidéo, les interfaces utilisateur ne sont pas développées selon le même modèle. En pratique, chaque jeu est développé avec une interface unique est distinguée, au contraire d'autres types de logiciels qui cherchent à utiliser les mêmes éléments d'interface pour simplifier l'utilisation et la phase de prise en main du logiciel par l'utilisateur.

Cela peut être justifié par le fait que, pour le joueur et le développeur, un jeu est une expérience qui doit être unique est ne doit par ressembler aux autres. Nous pouvons prendre comme exemple les jeux de foot comme *FIFA* et *Pro évolution* qui sortent chaque année avec une nouvelle version et une nouvelle interface graphique.

D'un autre côté, il est injuste de comparer l'interface graphique d'un jeu vidéo à celle d'un logiciel, le rôle de cette interface est totalement différent. Dans un logiciel, c'est cette interface qui nous permet de travailler avec le logiciel et d'exploiter ses fonctionnalités, par contre dans un jeu, cette interface à un rôle complémentaire, elle sert à changer des options, afficher les scores, démarrer ou fermer le jeu…etc.

Les interfaces dans les jeux vidéo sont développées en utilisant une technique appelée HUD (*Head-up display*) traduit en français par le terme L'affichage tête haute. L'origine du nom provient de la technologie utilisée dans l'aviation militaire où le pilote étant en mesure de garder la tête haute et un système lui superpose les informations dans le champ de vision.

Dans un jeu, l'interface HUD est l'ensemble d'éléments affichés à l'écran ayant pour but de renseigner le joueur sur son statut, la seule limite qu'on a dans un jeu est que nous nous disposons que d'un espace d'affichage limité souvent sur les côtés.

Les informations affichées varient d'un type de jeu à l'autre. Pour notre serious game, les informations à afficher au joueur sont les suivantes :

- Temps limite de la mission ;
- Temps écoulé depuis le début de la mission ;
- Les tops scores et le nom des missions ;
- Vitesse selon laquelle la voiture roule ;
- Santé du véhicule ;
- Une carte optionnelle du trajet indiquant le point de départ, le point d'arrivée et la position du joueur.

La première étape dans la création d'une interface HUD d'un jeu vidéo consiste à utiliser un éditeur d'image avancé comme Adobe Photoshop ou GIMP (open source) pour créer les éléments d'interface. La Figure III-37 représente la création des différents éléments de l'interface avec l'outil GIMP.

Figure III-37 Création des éléments de l'interface HUD avec l'outil GIMP

Pour le temps imparti de la mission, le temps écoulé, les scores et le nom de mission ces informations peuvent tout simplement être affichées sous forme de

texte. En pratique, il est recommandé d'utiliser un fond semi transparent derrière le texte pour augmenter sa lisibilité, car si le texte est directement affiché au-dessus du rendu du jeu, il peut devenir invisible, c'est le cas d'un texte blanc affiché sur l'un des coins supérieurs de l'écran et une caméra pointée sur un ciel nuageux par exemple.

Un autre élément dans l'interface de notre serious game est l'indicateur de vitesse ou compteur de vitesse, il permet au joueur de maîtriser la vitesse de son véhicule par rapport aux limites imposées par la signalisation ou le code de la route.

Il est constitué d'un fond semi transparent sur lequel sont inscrites les différentes vitesses du véhicule, les vitesses supérieure à 120 km/h son marquées en rouge. Il est constitué également d'un espace sur lequel sera affichée la vitesse en format numérique.

Tous les éléments de l'interface sont regroupés sur une seule image png, il est important de noter au préalable les coordonnés de chaque élément de l'interface afin de l'utiliser dans l'étape suivante. L'image qui sera utilisée par le jeu est représentée dans la Figure III-38.

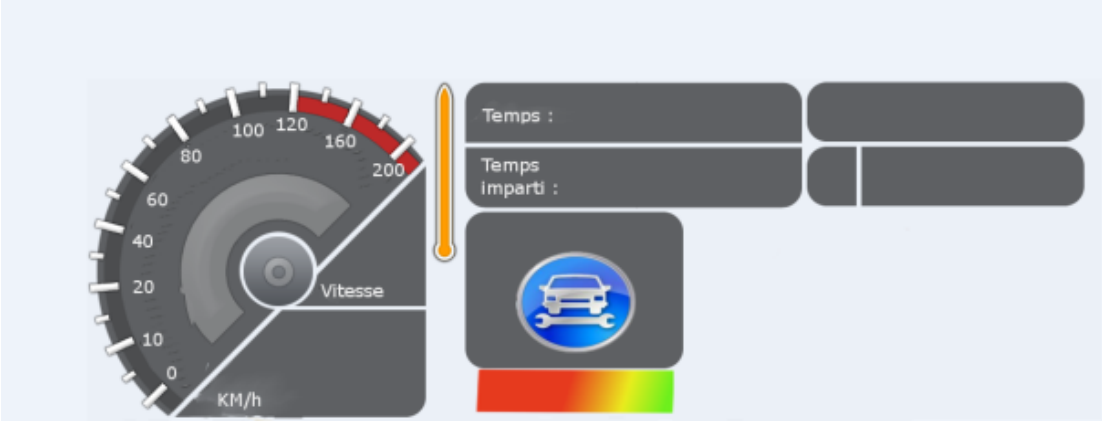


Figure III-38 Image de l'interface HUD utilisé dans le jeu

L'étape suivante consiste à positionner les éléments d'interface sur l'écran. En pratique, l'image contenant les éléments d'interface est chargée dans le jeu sous forme de texture. Ensuite, chaque élément d'interface est copié depuis cette texture sous forme d'une surface rectangulaire est placé dans la bonne position.

Pour le compteur de vitesse, nous devons calculer l'angle de rotation de l'image d'aiguille en fonction de la vitesse actuelle, avec la formule suivante :

AngleAiguille = (vitesse/200)/180.

Pour le niveau de santé de véhicule, nous avons dessiné un rectangle dont la couleur change du rouge au vert graduellement. Le niveau de santé du véhicule est utilisé pour déterminer la zone à copier.

RectangleSanté.X = (santé/100) * rectenglecouleur.X

La Figure III-39 montre un exemple d'apparition finale de l'interface HUD dans le jeu avec des valeurs arbitraires pour les informations.

Figure III-39 Interface HUD du jeu

Nous avons aussi besoin d'un menu pour notre jeu, ce dernier est implémenté de la même manière que les autres éléments de l'interface. Il est constitué de boutons, chaque bouton est d'une forme carrée où une icône indiquant la fonction du bouton est dessinée à l'intérieur, le bouton actif est marqué par un contour jaune indiquant qu'il est sélectionné.

Figure III-40 Texture utilisée pour le menu du jeu

Si le jeu passe au mode menu, une image est affichée comme fond d'écran, les boutons sont positionnés les uns à côté des autres selon un ordre bien déterminé, le pointeur de la souris sera également activé. Si le joueur pointe sur un bouton, il sera légèrement agrandi et entouré avec un contour (Figure III-41).

Figure III-41 Menu principal du jeu

III.6 Conclusion

Dans ce chapitre nous avons présenté les différentes phases d'implémentation d'un jeu vidéo à vocation utilitaire, nous avons essayé de mettre l'accent sur les différentes étapes du processus de mise en œuvre. Nous avons commencé par le choix et la conception d'un serious game d'aide à l'apprentissage et le respect de code du la route. Ensuite, nous avons entamé l'implémentation par le choix d'outils d'implémentation convenables à la réalisation de notre proposition. Le reste de ce chapitre a été consacré aux différents aspects de l'implémentation d'un

Dans le chapitre suivant, nous reprenons ce travail en essayant d'utiliser la réalité augmentée mobile en tant que plateforme principale de notre serious game.

Chapitre IV Mise en œuvre d'un Serious Game en RA mobile pour l'apprentissage des règles de circulation routières

IV.1 IV.1 Introduction

Dans le chapitre précédent nous avons présenté la conception et l'implémentation d'un serious game d'aide à l'apprentissage et le respect des règles de circulation routière. Dans ce chapitre nous reprenons le travail avec comme objectif la réalisation d'une version en réalité augmentée de notre serious game.

IV.2 Proposition d'une architecture pour la mise en ouvre du serious game en mode réalité augmentée.

L'implémentation d'une application de réalité augmentée peut utiliser différentes configurations logicielles et matérielles, nous discutons ci-après les différentes configurations possibles pour mettre en ouvre notre serious game en mode réalité augmentée.

IV.2.1 Configuration bureau

Dans cette configuration, l'utilisateur est sensé utiliser un ordinateur de bureau doté d'une caméra. Du point de vue technique, l'utilisation d'un ordinateur de bureau permet de bénéficier de performances très élevées (vitesse de calcul, mémoire RAM, accélération graphique, …etc.). Cette configuration offre une liberté de choix sur le plan logiciel, car sur un ordinateur ordinaire, le développeur peut utiliser le système d'exploitation de son choix (linux, Windows, Mac OS,…etc.), le langage de programmation et l'API graphique qui lui convient …etc.

Le problème majeur de cette configuration est le dispositif d'affichage, l'utilisation d'un écran ordinaire n'est pas un choix favorable pour une application de RA, surtout dans le cas d'une application multi-utilisateur, l'utilisation d'un Visio-projecteur de RA ou des lunettes de RA peut rendre cette configuration plus adaptée et plus immersive.

IV.2.2 Configuration mobile

Comme nous l'avons présenté dans notre état de l'art sur la réalité augmentée, il existe diverses plateformes matérielles pour implémenter des

applications de réalité augmentée : lunettes semi transparentes, vidéo casques, projecteurs spatiales… etc. Ces plateformes sont coûteuses et, dans des cas inaccessibles pour le grand public et même des fois pour les chercheurs. Avec la révolution du monde des Smartphones ces derniers sont devenus une plateforme prometteuse pour les applications de réalité augmentée. D'un côté, ils sont à la portée de tout le monde, et d'un autre côté, ils sont dotés de tous ce qui est nécessaire pour implémenter des applications de réalité augmentée (caméra, écran tactile, connectivité, boussole, GPS…etc.).

Pour les raisons ci-dessus, nous avons choisi d'utiliser les Smartphones comme plateforme pour la version RA de notre serious game. Nous avons ensuite étudié les différentes alternatives matérielles et logicielles pour créer notre prototype.

La première alternative été la solution native, dans ce cas le système est entièrement implémenté sur le Smartphone. Cette implémentation dépend entièrement du Smartphone et du système d'exploitation qu'il utilise. Le tableau suivant représente une liste non exhaustive des Smartphones, leurs systèmes d'exploitation et les environnements de développements utilisés :

Smartphones	**Système**	**Environnement de développement**
Nokia et SonyEricson	Symbian 60	C++ pour symbian et Carbidec++
Nokia	Symbian 3	Qt et C++
IPhone, IPad…etc.	IOS 2, 3, 4, 5	Objective C sous Mac OS X
HTC, Samsung, LG, Google	Android	Java
Samsung	Bada	Java
HTC, Nokia,	Windows phone 7	.NET et XNA

Tableau IV-1 Liste des systèmes d'exploitation des Smartphones.

Il est remarquable que le marché des Smartphones soit actuellement en plein essor et constante augmentation, cela a conduit à une concurrence acharnée entre les constructeurs. Cette concurrence se joue sur l'aspect matériel avec la constante augmentation des capacités de ces dispositifs : CPU, mémoire, résolution d'écran…etc. Mais cette concurrence se joue aussi sur l'aspect système

limite son utilisation aux Smartphones qu'il propose. Cela implique une difficulté majeure dans le développement des applications pour des Smartphones, car, comme le montre le tableau, chaque système d'exploitation impose des outils de développements différents aux autres systèmes d'exploitation.

A ce problème s'ajoute d'autres beaucoup plus importants. D'abord, la création des jeux 3D sur les Smartphones est encore plus compliquée que leur création sur les autres plateformes. Cette complexité est liée d'un côté, aux capacités graphiques limitées des Smartphones et d'un autre côté au manque en terme d'outils dédiés à la création des jeux vidéo 3D. Cela n'empêche pas de signaler que certains Smartphones commencent dernièrement à intégrer des puces graphiques (GPU).Sur le plan logiciel, l'apparition de OpenGL ES est aussi encourageante.

IV.2.3 Configuration mobile client-serveur

Pour tirer partie des performances élevés des ordinateur de bureau (CPU, mémoire, GPU) d'un part, et profiter de la mobilité et des capteurs intégrés (GPS, Boussole, accéléromètre) dans les Smartphones d'autre part, nous proposons une architecture hybride en client-serveur. Dans cette architecture, le Smartphone joue le rôle de client tandis qu'un ordinateur joue le rôle d'un serveur de calcul. La communication entre le client et le serveur se fait en temps réel à travers un réseau local sans-fil. Le Smartphone envoi la vidéo captée par la caméra intégrée accompagnée des données des capteurs (position GPS, accélération, orientation) et des commandes de l'utilisateur. De son côté, le serveur effectue la tâche de suivi et identifie les objets reconnus dans la vidéo reçue. La position des objets, les données des capteurs et les commandes de l'utilisateur sont utilisées par le moteur de jeu pour faire avancer la boucle du jeu. Le serveur effectue le rendu des objets virtuels 3D (sur une fenêtre de la même taille et la même résolution que l'écran du Smartphone) et envoi le résultat (image 2D) au Smartphone. A son tour, le Smartphone superpose l'image 2D reçue sur la vidéo du monde réel. La Figure IV-1 présente l'architecture globale du système proposé.

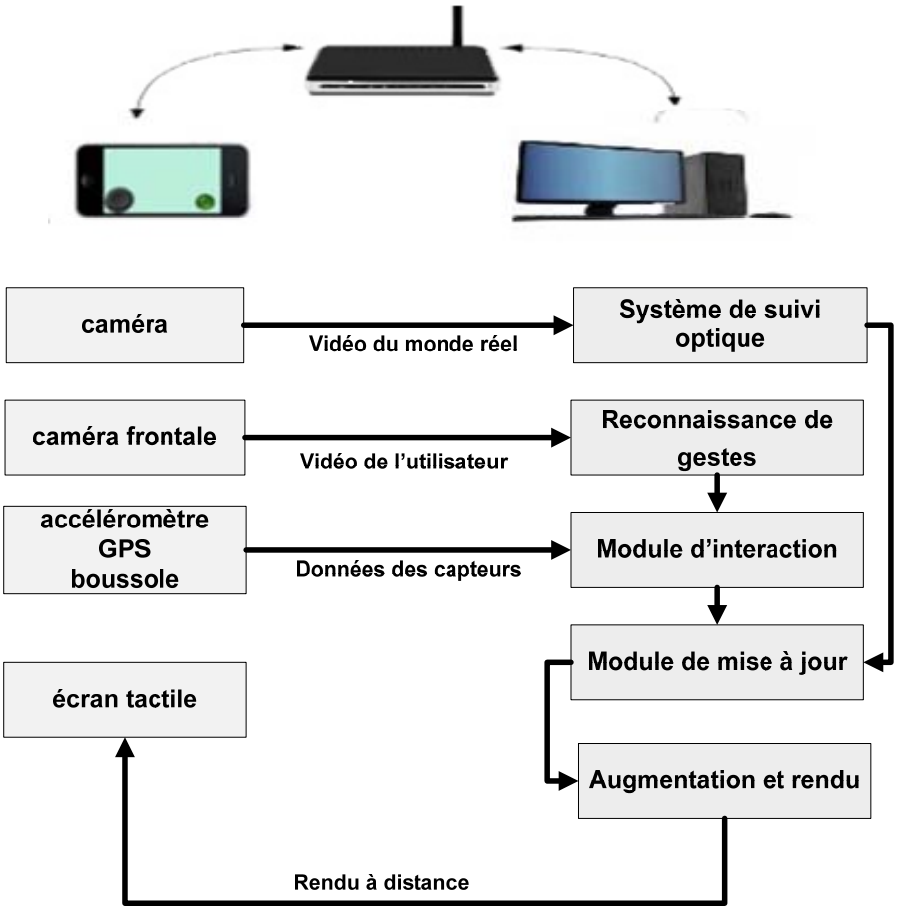


Figure IV-1 Architecture client-serveur de notre serious game en mode RA

Pour implémenter cette architecture, nous avons introduit des modifications au niveau de notre précédente implémentation du serious game. La modification concerne principalement la boucle du jeu. La nouvelle boucle du jeu est présentée dans la Figure IV-2

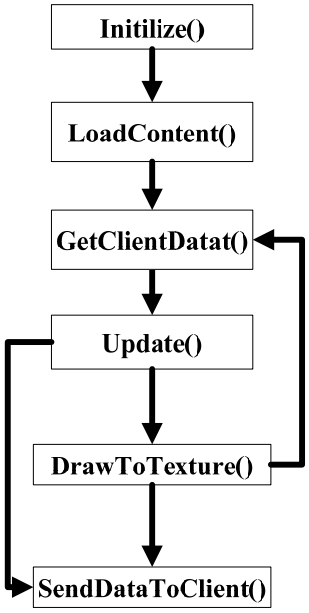


Figure IV-2 Boucle de jeu client serveur

La méthode GetClientData(), permet de recevoir les informations transmises par le client, ces informations contiennent :

- Les Images de la caméra avant et de la caméra frontale ;
- Les valeurs de l'accéléromètre ;
- La valeur du capteur boussole ;
- Les coordonnées GPS ;
- Les boutons appuyés ;
- La zone de l'écran touchée dans le cas d'un écran tactile.

Les images des caméras doivent être traitées par le module de suivi. Le résultat est passé ensuite au module de mise à jour avec les autres données (Accéléromètre, boussole, etc.….).

La méthode *update* doit mettre à jour la scène selon les données reçues.

La méthode drawToTexture(), utilise un mécanisme appelé RenderTarget, ce mécanisme permet d'effectuer le rendu de la scène sur une texture au lieu de le faire sur l'écran. Ce mécanisme est souvent utilisé pour implémenter l'effet miroir par exemple, des bilboards ou même pour effectuer des post-traitements sur le résultat de rendu avant d'être affiché sur l'écran (rendu déféré).

Nous avons utilisé ce mécanisme pour rediriger l'affichage de notre jeu vers l'écran de téléphone mobile. L'idée consiste à faire le rendu sur une texture (Image 2D) et à l'envoyer au client sur le téléphone mobile.

Le problème que nous avons rencontré à ce stade est celui du format de l'image envoyée. Une première approche consiste à envoyer l'image sous forme d'un tableau de pixels sans compression. Supposons que nous travaillons sur une résolution 340x260, la taille de l'image serra 340*260*3 = 259Ko, si le taux de rafraîchissement minimal pour avoir une animation fluide est 15 image/seconde, alors on aura besoin de débit de Transfert de 3Mo/s.

La seconde approche est d'encoder l'image dans le format JPEG avec un taux de compression de 60% qui nous permet d'avoir des images de 17ko de taille avec une résolution de 640x360 ce qui est une résolution suffisante pour la plupart des téléphones de nos jours. Cette approche favorise la bande passante mais il faut noter que le décodage d'image JPEG peut prendre du temps sur quelques modèles de téléphone ce qui peut compromettre la contrainte temps réel.

Une amélioration consiste à utiliser la différence entre frames où les pixels qui n'ont pas changé de couleurs par rapport à l'image précédente prennent des valeurs transparentes. Cette technique permet de réduire la taille des images envoyées, de cette manière une image est envoyée chaque seconde, et les différences sont envoyées pendant cette seconde

IV.3 Protocole de communication

La communication entre le client et le serveur se fait au-dessus du protocole TCP, ce dernier permet d'échanger des données binaires entre le client et le serveur. Le protocole TCP est appelé protocole de transport, il permet d'acheminer les données sans tenir compte ni de leur type ni de leur signification. Pour mettre en place un système client serveur, il est nécessaire d'utiliser un protocole de niveau application, ce protocole est responsable du dialogue entre le client et le serveur. Un exemple très connu de cette architecture est celui d'un serveur web et un navigateur. Pour assurer qu'un site web soit consultable depuis n'importe quel ordinateur, tous les navigateurs et les serveurs web utilisent un seul protocole de niveau application appelé protocole HTTP. Pour notre cas, nous devons implémenter un protocole spécifique à notre système.

Pour concevoir un tel protocole, nous devons prendre en considération les points suivent :

- Le récepteur n'a aucun mécanisme qui lui permet de séparer un message du message suivant ;
- Il se peut qu'un seul message soit reçu par le récepteur à plusieurs reprises.

Dans ce cas, il existe trois approches :

Utiliser des messages de taille fixe, cette approche n'est bénéfique que dans les cas où les données à envoyer ont la même taille. Si la taille des données est inférieure à celui choisie pour le message, il faut utiliser de technique de bourrage (Padding). D'un autre côté, si la taille des données est supérieure à celle du message choisi, il faut fragmenter les données et les envoyer à plusieurs reprises ;

Utiliser un marqueur pour délimiter les messages, le problème est que ce marqueur ne doit pas figurer dans les données envoyées, par exemple, utiliser des messages encodés en Base64 et des caractères d'échappement comme '\n ' ;

Un mécanisme plus souple consiste à précéder le message par sa taille, de cette manière le récepteur peut anticiper la fin du message actuel et le début du message suivant. Ce mécanisme permet d'envoyer des données de taille variable et il n'impose aucun codage particulier de ces données.

Vu que le client et le serveur échangent plusieurs messages de différents types, il faut que chaque message contienne un indicateur spécifiant le type de message afin que le récepteur puisse le décoder correctement. La figure suivante représente le format d'un message dans notre protocole de communication.

Figure IV-3 Format de message de protocole de communication.

Les trois premiers octets représentent la taille de la partie 'données' dans ce message, l'octet suivant représente le type de ce message, le reste des octets sont des données.

La partie 'données' peut être à son tour encodée selon le type du message à plusieurs parties, par exemple les images provenant de caméra du téléphone mobile peuvent être de deux types, soit une image complète, soit la différence par rapport à la dernière image envoyée. La figure suivante représente ce principe de sous-codage.

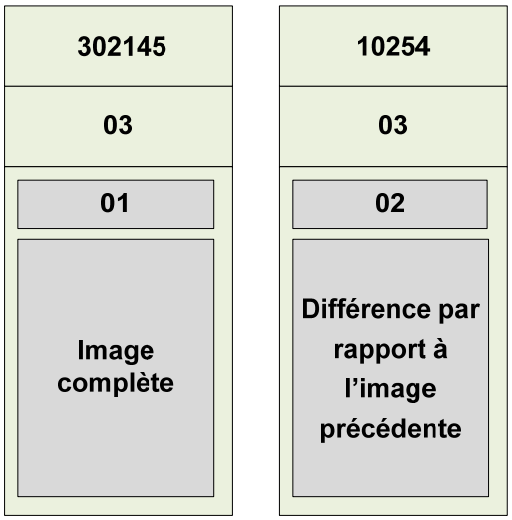


Figure IV-4 principe de sous-codage

IV.4 Plateforme de développement pour la version RA

Pour la réalisation de la version RA de notre serious game selon l'architecture client-serveur proposée, nous utilisons le résultat de notre travail dans la version RV présentée dans le chapitre précèdent. Autrement dit, le jeu implémenté précédemment va constituer le noyau de notre application serveur. Nous ajoutons à ce noyau un module réseau qui va prendre en charge la communication avec le Smartphone à travers une connexion TCP/IP sur un réseau sans-fil. Nous ajoutons également, un module de suivi optique qui va prendre en charge le traitement de la vidéo envoyée en temps réel au serveur. Nous adaptons aussi le module d'interaction dans notre ancienne implémentation pour tirer parti des différentes possibilités d'interaction qui s'ajoutent aux Smartphones, notamment, les capteurs et l'écran tactile.

Nous avons choisis d'utiliser l'IPhone pendant nos travaux, néanmoins, notre implémentation est indépendante de ce choix, autrement dit, les fonctionnalités que nous avons utilisées sont partagées par une large gamme de Smartphones.

Pour le côté client, nous devons créer une application IPhone qui va prendre en charge la communication avec le serveur. Au niveau de cette application, nous devons implémenter le protocole de communication spécifié auparavant, accéder et envoyer la vidéo de la caméra du Smartphone, accéder et envoyer les données des différents capteurs, et finalement recevoir et afficher le résultat de rendu envoyé par le serveur.

Le tableau suivant représente l'ensemble des outils utilisés pour implémenter ces composantes.

Outil / plateforme	Description	Licence
Alvar 1.5	Bibliothèque de suivi optique de marqueurs visuels	Développé par VTT Technical Research Centre of Finland Version de base gratuite Version pro payante
Xcode	Plateforme de développement d'applications pour Mac os, iPhone et iPad	Proposé gratuitement par Apple

Tableau IV-2 Liste des outils utilisés dans la version RA.

IV.5 Développement des applications pour iPhone

IOS est une variante du système d'exploitation Mac OS X destiné aux dispositifs mobiles iPhone, iPod touch et iPad. La programmation sur IOS se fait en Objective-C, un langage qui nécessite l'utilisation de Xcode, environnement de développement pour Mac OS X. La programmation passe donc obligatoirement par un ordinateur Mac équipé d'un processeur Intel et d'une version récente du système d'exploitation Mac (10.5 ou ultérieur).

Pour pouvoir tester les applications sur un iPhone ou un iPad le développeur doit acheter la licence de développeur Apple au prix de 99$ par ans, cette dernière permet également au développeur de soumettre ses applications dans l'AppStore.

IV.5.1 L'environnement Xcode

Xcode est l'Environnement de Développement Intégré par défaut sur Mac OS X. Il permet l'écriture, la gestion et la compilation de projets de développement, écrits notamment en Objective-C. L'iPhone SDK y ajoute les bibliothèques de développement pour iOS. Il est donc possible pour le développeur de créer des projets d'applications pour ce système. Pour tester l'application, deux possibilités existent : le développeur peut brancher un iPhone ou iPod Touch à son ordinateur Mac, puis y lancer l'application comme test à condition d'adhérer au programme des développeurs d'Apple, ou lancer l'application en test dans iPhone Simulator.

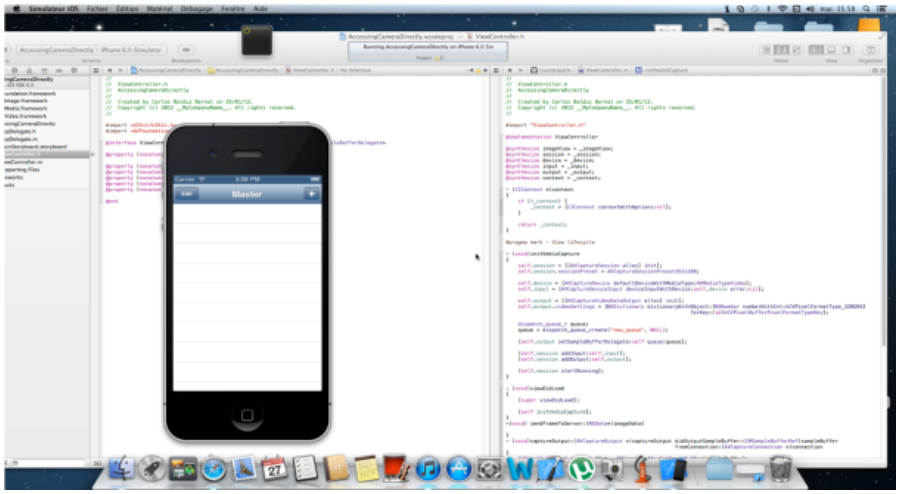


Figure IV-5 l'environnement XCode et l'émulateur iPhone.

IV.5.2 Le langage Objective-c

Le langage Objective-C est un langage de programmation orienté objet réflexif. C'est une extension du C ANSI, comme le C++, mais qui se distingue de ce dernier par sa distribution dynamique des messages, son typage faible ou fort, son typage dynamique et son chargement dynamique. Contrairement au C++, il ne permet pas l'héritage multiple mais il existe toutefois des moyens de combiner les avantages du C++ et de l'Objective-C. Ce langage est principalement utilisé dans les systèmes d'exploitation d'Apple : Mac OS X et son dérivé iOS,

En Objective-C, tout est objet tout comme en Smalltalk dont il s'inspire fortement. C'est donc un langage fortement orienté objet. L'héritage simple induit un arbre d'héritage avec une racine : la classe NSObject. C'est à partir d'elle que vont dériver toutes les classes. Par exemple, un objet de classe NSString, ou NSArray,

dérive de la classe NSObject (indirectement). NSMutableArray dérive de la classe NSArray, qui est donc sa superclasse.

On distingue deux types de fichiers sources en objective-c, les fichiers avec l'extension .h et les fichiers avec l'extension .m. les fichiers .h sont les fichiers d'entête au niveau de ces fichiers les variables de classe sont déclarées, les signatures des méthodes etc. les méthodes déclarées dans le fichier .h sont implémentés dans le fichier .m correspondant.

IV.6 Implémentation du module réseau

Le module réseau prend en charge la communication entre le Smartphone et le serveur. En réalité, ce module est constitué de deux partie : la partie serveur et la partie client. La partie serveur est programmée avec le langage C# sous la plateforme .NET alors que la partie client est programmée avec le langage Objectiv-C sous la plateforme Mac OS.

La différence dans le langage de programmation et dans l'architecture des deux communicants impose une attention particulière dans l'encodage et le décodage des données, car les deux communicants peuvent utiliser une représentation différente des données (Big-endian ou little-endian)[3], nombre de bits pour représenter un entier (entier en 16,32 ou 64 bit)…etc.

IV.6.1 Implémentation du module réseau côté serveur

La première étape consiste à implémenter le protocole de communication qui est implémenté sous forme d'un ensemble de constantes de type byte (codé sur 8 bits) où chaque constant représente une opération ou un type de données. L'extrait de code suivant représente une démonstration de ce principe :

[3]Il existe deux architectures différentes pour gérer le stockage des donnés en mémoire. Elles sont appelées BigEndian et LittleEndian et font référence à l'ordre dans lequel sont stockés les octets

```
14  static string header = "C7F9B1";
15  static int dataLengthLenght = 8;
16  public static int HeaderLenght = 6;
17  public static int CMDLenght = 4;
18  public static int DataLenghtLength = 8;
19  public static int DeviceNameLength = 24;
20
21  public const byte NoCommande = 0011;
22  public const byte StartGame = 0012;
23  public const byte StopGame = 0013;
24  public const byte PauseGame = 0014;
25  public const byte ResumeGame = 0015;
26  public const byte Disconnect = 0016;
27  public const byte Connect = 0017;
28  public const byte GameFrame = 0018;
29  public const byte PhoneConfig = 0019;
30  public const byte NotComplited = 0100;
31  public const byte SENSORDATA = 25;
32
```

Extrait de code IV-1 protocole de communication en C#

Au démarrage du serveur, nous lançons un Thread qui a pour tâche l'attente de connexions des clients sur un port de communication particulier : en cas de connexion d'un client, le serveur lance deux Threads, l'un pour l'envoi des données vers le client et l'autre pour la réception. Ensuite, le serveur se remet à l'écoute sur le même port pour attendre la connexion d'un nouveau client.

L'utilisation de deux Thread pour gérer la communication entre le serveur et le client a été imposée par le fait que les données à échanger sont de taille importante, autrement dit, le serveur peut recevoir les données de la caméra et de capteurs depuis le Smartphone pendant qu'il lui envoi le résultat du rendu du frame précédent. Le principe de fonctionnement du serveur est représenté dans la Figure IV-6.

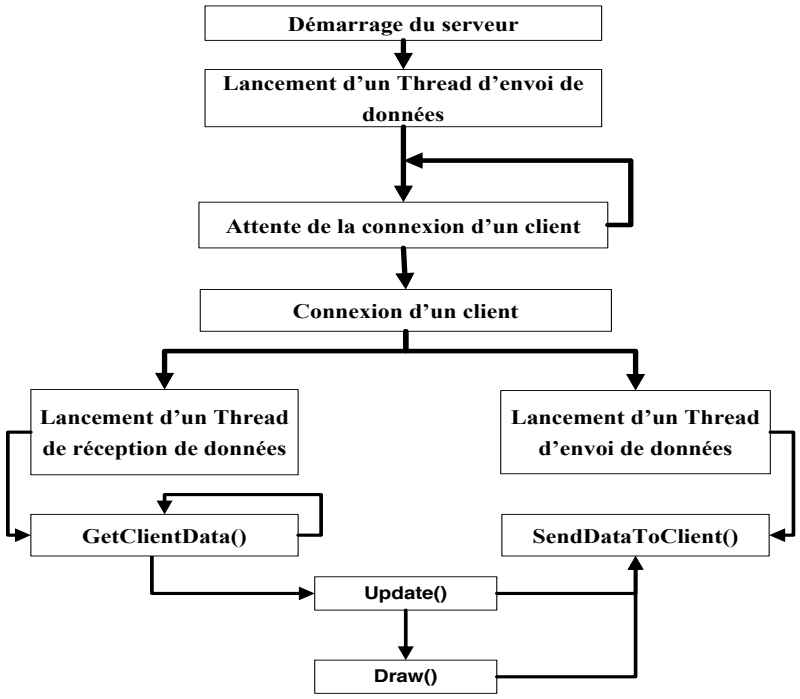


Figure IV-6 Principe de fonctionnement du serveur

L'implémentation de ce principe est représentée dans l'extrait du code suivant :

```
11      class Server
12      {
13          string host;  long dealy = 60;
14          int port = 37000;  bool started = false;
15          bool hasclient = false;
16          private TcpListener serverlistner = null;
17          private Protocol protocol = new Protocol();
18          TcpClient myclient = null;
19          Thread sendingThread = null;  Thread recivingThread = null;
20          Thread ConnectionThread = null;
21          public bool Start()
22          {
23              try
24              {
25                  serverlistner = new TcpListener(IPAddress.Any, port);
26                  serverlistner.Start();
27                  ConnectionThread = new Thread(new ThreadStart(AcceptClient));
28                  ConnectionThread.Start();
29                  started = true;
30                  return true;
31
32              }
33              catch (Exception ex)
34              {
35                  return false;
36              }
37          }
38          long time;
39          private void AcceptClient()
40          {
41
42              myclient = serverlistner.AcceptTcpClient();
43              hasclient = true;
44              sendingThread = new Thread(new ThreadStart(sendingThreadCode));
45              recivingThread = new Thread(new ThreadStart(reciptionThreadCode));
46              sendingThread.Start();
47              recivingThread.Start();
48          }
```

Extrait de code IV-2 code du serveur en C#

Le diagramme de classe du module réseau de notre serious game est présenté par la Figure IV-7, il regroupe les classes suivantes :

- La classe Server gestion de la connexion avec les clients ;
- Les classes Codage et BmpWriter responsables de l'encodage et du décodage de l'image ;
- La classe Protocol qui regroupe tous les constantes liées au protocole de communication ;
- La classe Packet et ses classes dérivées qui représentent les différents types de messages échangés entre le client et le serveur.

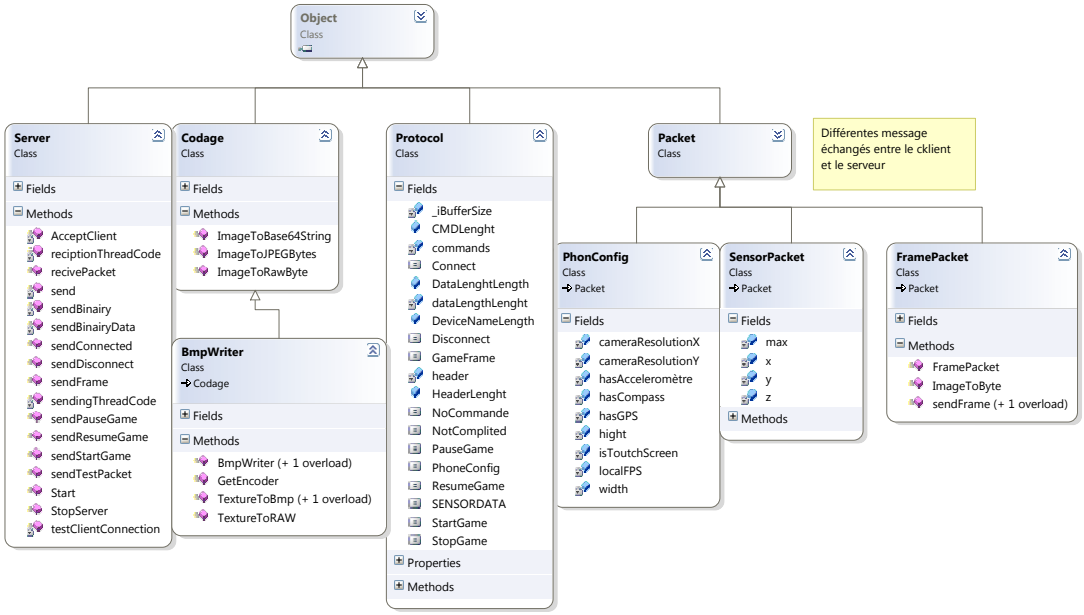


Figure IV-7 Diagramme de classe de module réseau

IV.6.2 Implémentation du module réseau côté client

Avec pratiquement le même principe, le module réseau est implémenté côté client. Le principe de fonctionnement de module réseau de l'application client est présenté dans la Figure IV-8.

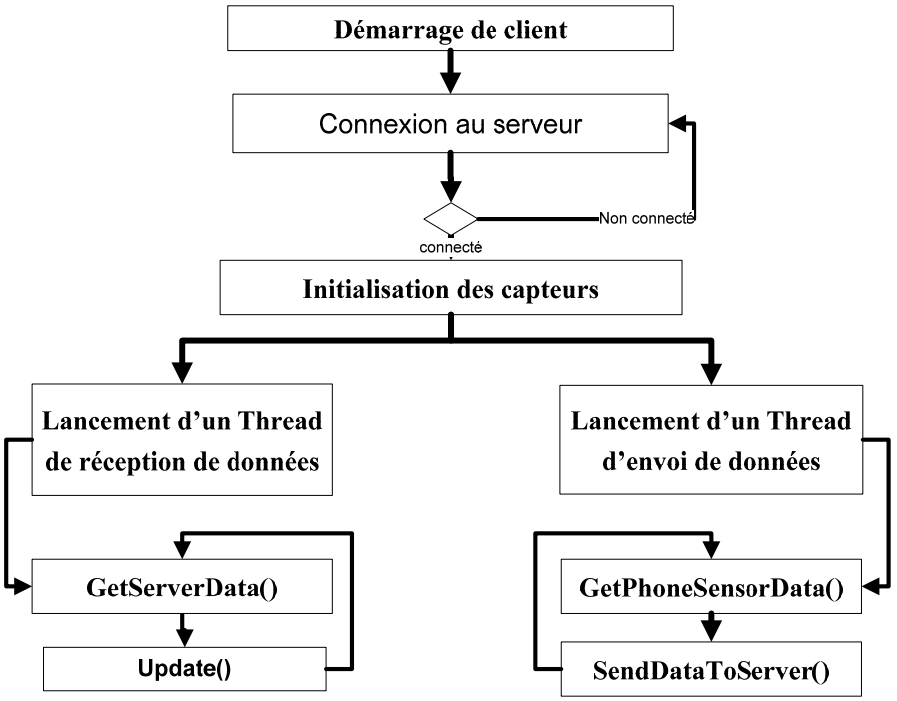


Figure IV-8 Principe de fonctionnement de l'application client

Le système IOS de l'iPhone utilise un noyau Unix, pour créer un socket sous le système IOS, il faut utiliser la fonction C CFStreamCreatPairWithSocketToHost, cette fonction prend comme paramètre l'adresse du serveur ainsi que le port de communication, cette fonction initialise deux objets de type CFReadStreamRef et CFWriteStreamRef, ces derniers sont converti en NSInputStream et NSOutputStream, NSInputStream permet de recevoir de données depuis le serveur, NSOutputStream : permet d'envoyer des données au serveur. Le code suivant permet d'établir une connexion entre l'iPhone et notre serveur implémenté précédemment.

```
-(void)initNetworkCommunication {

    CFReadStreamRef readStream;
    CFWriteStreamRef writeStream;
    host =@"192.168.1.6";
    CFStreamCreatePairWithSocketToHost(NULL, (CFStringRef)host, 37000, &readStream, &writeStream);
    inputStream = (NSInputStream *)readStream;
    outputStream = (NSOutputStream *)writeStream;
    inputStream.delegate = self;
    outputStream.delegate= self;
    [inputStream scheduleInRunLoop:[NSRunLoop currentRunLoop] forMode:NSDefaultRunLoopMode];
    [outputStream scheduleInRunLoop:[NSRunLoop currentRunLoop] forMode:NSDefaultRunLoopMode];

    [inputStream open];
    [outputStream open];


}
```

Extrait de code IV-3 initialisation d'une connexion TCP en objective-C

La connexion au serveur ne prend lieu qu'après l'appelle de la méthode open sur les deux objets NSinputStream et NSOutputStram.

IV.7 Implémentation du module de suivi optique

La réalité augmentée est basée essentiellement sur un système de suivi qui permet de localiser les objets dans le monde réel. Les différents systèmes de suivi utilisés en RA ont été présentés dans le premier chapitre de ce mémoire, nous rappelons qu'il existe deux familles l'une basée sur l'utilisation des capteurs et l'autre basée sur le suivi optique. Dans le cadre de notre travail, nous utilisons principalement le suivi optique des marqueurs visuels avec la contribution des capteurs présent dans les Smartphones modernes.

IV.7.1 Principe de suivi de marqueurs visuels

L'un des premiers systèmes de suivi de marqueurs visuels développés est le système ARToolKit, il s'agit d'une librairie de fonctions (en langage C/C++) pour les applications de réalité augmentée.

Développé au Human Interface Technology Laboratory de l'Université de Washington et au MIC Research Labs au Japon, ARToolKit, [28] utilise les techniques de vision par ordinateur pour calculer la position réelle de la caméra et son orientation relative à des marqueurs 2D (marker card), associés à des objets virtuels. La documentation et les sources d'ARToolKit nous permettent de

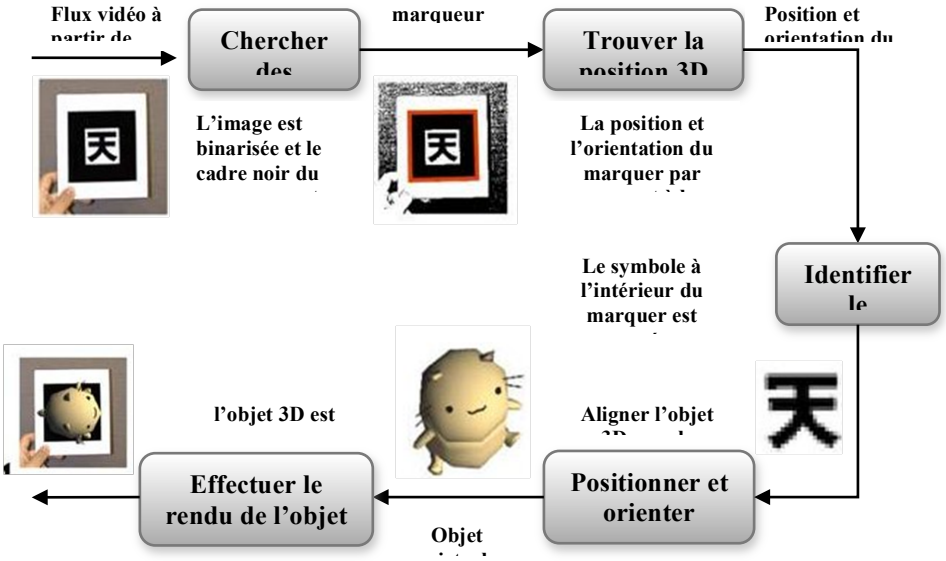


Figure IV-9 Principe de fonctionnement global du suivi de marqueurs 2D

Dans un premier temps, l'image de la caméra est binarisée (image en noir et blanc) selon un seuil d'éclairage. La seconde étape consiste à rechercher tous les carrés 2D présents dans cette nouvelle image, puis pour chacun d'entre eux, de capturer le marqueur (ou motif) qu'il contient et de le comparer à ceux connus par la machine. Si un marqueur est reconnu, alors ARToolKit utilise la taille réelle du carré et l'orientation du motif qu'il contient pour calculer la position réelle de la caméra vidéo par rapport au marqueur. Une matrice 3*4 est alors complétée, servant à positionner une caméra virtuelle qui permet d'afficher l'objet virtuel de façon très précise sur le marqueur.

Cette méthode de suivi des marqueurs 2D dans une image possède deux avantages. D'une part, basée sur la détection de carrés dans l'image, cette méthode est très robuste et demande peu de ressources machine, offrant un fonctionnement temps réel très satisfaisant. D'autre part, elle est prévue pour détecter plusieurs marqueurs simultanément dans une image toujours en temps réel.

Contrairement au principe du suivi 3D, les occultations ne sont pas prises en compte, si un marqueur n'est pas entièrement visible, alors il ne sera pas détecté.

IV.7.2 Modèle et calibration de caméra

La calibration de la caméra est la première étape à réaliser en RA, il s'agit de modéliser le processus de formation des images et de trouver la relation entre les

coordonnées spatiales d'un point de l'espace avec le point associé dans l'image prise par la caméra.

Il existe plusieurs modèles de caméras, le plus simple est le modèle du sténopé ou modèle pin-hole (en anglais). Il s'agit d'un modèle idéal couramment utilisé en infographie et en vision par ordinateur pour exprimer le principe de fonctionnement de la caméra. Il ne tient pas compte de certains effets optiques (tels que les distorsions non linéaires) qui sont souvent des propriétés des caméras réelles.

Il s'agit d'une modélisation simple et linéaire du processus de formation des images au sein de la caméra. Ce modèle suppose que le système optique de la caméra, c'est-à-dire sa lentille respecte les conditions de Gauss. Si on utilise la notation matricielle des coordonnées homogènes, il est possible de décrire de manière simple ce processus. Il suffit d'exprimer les relations de passage du repère monde au repère caméra, d'exprimer la projection du repère caméra dans le plan image et d'appliquer la transformation affine qui conduit aux coordonnées de l'image. La relation est la suivante pour un point M de coordonnées (X,Y,Z,1) dans l'espace (repère monde) et dont l'image est de coordonnées (su , sv, s) dans le plan image :

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Les paramètres employés dans ce modèle sont usuellement divisés en deux catégories : les paramètres intrinsèques qui sont internes à la caméra, et les paramètres extrinsèques qui peuvent varier suivant la position de la caméra dans l'espace de travail.

Parmi les paramètres intrinsèques nous comptons :

- f_x,f_y : correspondent à la distance focale exprimée en largeurs et en hauteurs de pixels, sont les facteurs d'agrandissement de l'image multipliée par la longueur focale ;
- c_x : Coordonnées de la projection du centre optique de la caméra sur le plan image.

Les paramètres extrinsèques sont :

- R : matrice de rotation permettant de passer du repère lié à l'espace de travail au repère lié à la caméra,
- T : translation permettant de passer du repère lié à l'espace de travail au repère lié à la caméra.

Au total, cela fait 16 paramètres à estimer, ces paramètres sont regroupés dans deux matrices: celle des paramètres intrinsèques et celle des paramètres extrinsèques.

En réalité, ce sont uniquement 10 paramètres à estimer car, la matrice de rotation R contient 9 éléments, mais la relation qui la définit comme matrice de rotation R RT = 1 réduit le nombre d'éléments indépendants à trois (les trois angles polaires).

La matrice des paramètres intrinsèques ne dépend pas de la scène et une fois estimée elle peut être réutilisée tant que la longueur focale ne change pas (dans le cas d'une caméra à zoom optique par exemple). Donc, l'objectif de la phase de calibration est d'estimer la matrice des paramètres intrinsèques, ce qui limite la tâche de module de suivi en uniquement l'estimation des six paramètres extrinsèques, autrement dit, la position et l'orientation de la caméra.

Les Lentilles réelles présentent souvent une certaine distorsion, essentiellement une distorsion radiale et une légère distorsion tangentielle. Les coefficients de distorsion ne dépendent pas de la scène visualisée et ils appartiennent donc également aux paramètres intrinsèques de la caméra. Aussi, ils restent les mêmes quelque soit la résolution de l'image capturée. Autrement dit, si, par exemple, une caméra a été calibrée sur des images de résolution 320x240, ces paramètres peuvent être utilisés avec des images de résolution 640x480 de la même caméra sauf que f_x,f_y , c_x et c_y doivent être mises à l'échelle d'une façon appropriée.

Le modèle détaillé ci-dessus nous permet de résumer les trois fonctions principales d'un système de RA :

La calibration qui consiste à estimer les paramètres intrinsèques de la caméra à partir de plusieurs captures d'un modèle de calibration connus (i.e. chaque vue est décrite par plusieurs correspondances des points 3D et leur projection 2D).

Calcul de la position et de l'orientation de la caméra (paramètres extrinsèques) étant donnée les paramètres intrinsèques, en utilisant quelques points 3D et leurs projections.

Augmentation :qui consiste à projeter des points 3D de synthèse sur le plan image étant donnés les paramètres intrinsèques et extrinsèques de la caméra utilisée.

Pour calibrer la caméra utilisée dans notre travail, nous utilisons un programme de calibrage fourni avec la bibliothèque de suivi Alvar. La calibration utilise l'image d'un échiquier (Chessboard) dont le nombre de carrés et la taille de chaque carré sont connus. La caméra doit être orientée vers l'image de l'échiquier en veillant à ce que tous les carrés soient visibles.

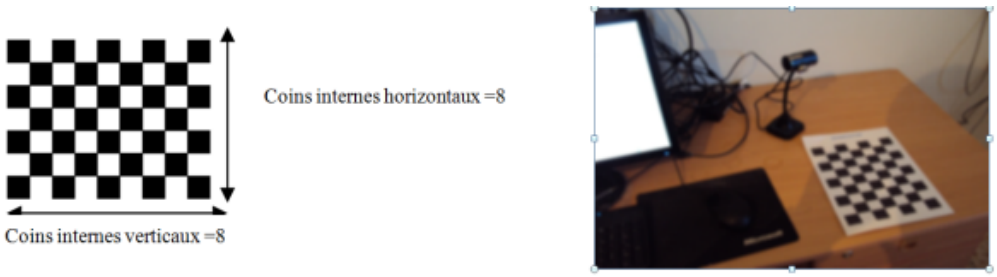


Figure IV-10 Calibrage de la caméra

L'algorithme de calibrage utilise comme paramètres le nombre d'images à exploiter ainsi que le nombre de coins internes horizontaux et le nombre de coins internes verticaux. Par exemple; dans l'image que nous utilisons, le nombre de coins internes horizontaux est 6 et le nombre de coins internes verticaux est 8 Figure IV-11.

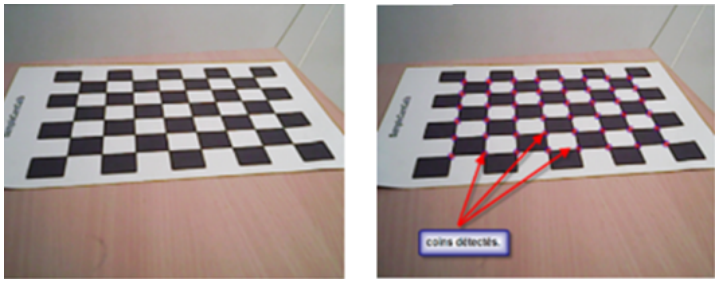


Figure IV-11 coins détectés par le programme de calibrage

L'outil de calibrage enregistre les résultats du calibrage de la caméra dans un fichier XML (Extrait de code IV-5), ce fichier sera ensuite utilisé par le système de

```
<?xml version="1.0"?>
<opencv_storage>
<intrinsic_matrix type_id="opencv-matrix">
  <rows>3</rows>
  <cols>3</cols>
  <dt>d</dt>
  <data>
    898.9146211949588400 0. 435.5830886274187000 0.
    1.4578945667473629e+003 176.1313715396504400 0. 0. 1.</data></intrinsic_matrix>
<distortion type_id="opencv-matrix">
  <rows>4</rows>
  <cols>1</cols>
  <dt>d</dt>
  <data>
    -0.0146357944074727 0.4092208503311628 7.9487568144368491e-004
    8.9161986038182039e-003</data></distortion>
<width>640</width>
<height>480</height>
</opencv_storage>
```

Extrait de code IV-4 données de calibrage sous format XML

IV.7.3 Suivi optique des marqueurs visuels avec la plateforme ALvar

Une fois les paramètres de la caméra identifiés, nous pouvons désormais utiliser la bibliothèque de suivi pour détecter les marqueurs visuels. Pour initialiser le module de suivi, il faut spécifier la source de la vidéo, la résolution de l'image et le fichier de calibrage pour la caméra utilisée. Le module de suivi traite le flux vidéo de la caméra et cherche s'il existe des marqueurs visibles. Ainsi, pour chaque marqueur visible, une matrice de transformation représentant sa position, son orientation et la mise à l'échelle est estimée. La Figure IV-12 représente un échantillon de marqueurs utilisés.

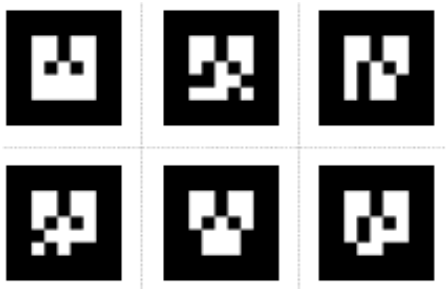


Figure IV-12 Exemple de marqueurs utilisés

Chaque marqueur est distingué par un identificateur numérique unique, cet identificateur est utilisé par le programmeur pour interroger le système de suivi sur la visibilité est la position du marqueur associé

Figure IV-13 Test du module de suivi visuel (a) scène réelle (b) scène augmentée

IV.7.4 Mise en correspondance Monde réel/Monde virtuel

Dans tous les jeux 3D classiques, la scène et les acteurs du jeu sont positionnés par rapport à un repère global dit repère du monde. Dans un jeu de réalité augmentée, les objets virtuels sont positionnés par rapport au monde réel, ceci dépend de la manière par laquelle ce monde réel est interprété. Dans notre cas, le monde réel est interprété à travers ces marqueurs visuels que nous plaçons dans ce monde. Le système de suivi est capable uniquement d'estimer la position 3D de ces marqueurs. En conséquence, la seule solution fiable qui permet de positionner les objets virtuels dans l'espace réel est de les placer par rapport aux marqueurs. Si nous voulons placer un vase virtuel sur une table réelle, il faut placer physiquement un marqueur sur la table et placer le vase sur le marqueur. De cette manière, l'image finale va donner l'impression que le vase est placé sur la table.

D'un point de vue pratique, le système de suivi optique estime la position, l'orientation et la mise à l'échelle du marqueur visuel, cela est représenté mathématiquement par ce qu'on appelle en infographie la matrice de transformation. Pour superposer un objet virtuel avec le marqueur il suffit d'utiliser la matrice de transformation du marqueur pour positionner cet objet. Le pseudo code suivant représente ce principe.

```
16          MarkerNode marker = new MarkerNode();
17          if (marker.MarkerFound)
18          {
19              // obtenir la matrice de transformation de marqueur
20              Matrix markerTransform = marker.WorldTransformation;
21              // créer la matrice de transformation de l'objet par rapport au repère fixe
22              Matrix objectTransform = Matrix.Identity;
23              object Transform = objectTransform *
24              Matrix.CreateTranslation(10, 35, 22);
25              objectTransform = objectTransform *
26                                  Matrix.CreateRotationX(MathHelper.PiOver2);
27              // créer un cube texturé
28              BoxNode box = new BoxNode("cube");
29             // box.Model = newTexturedBox(32.4f);
30              // créer la matrice de transformation finale par rapport au marquer
31              Matrix finalTransform = markerTransform * objectTransform;
32              // positioner le cube par rapport au marquer
33              box.WorldTransformation = finalTransform;
34          }
35
```

Extrait de code IV-5 placement d'un objet virtuel à l'aide de la matrice de transformation du marqeur

Bien que la majorité des prototypes de réalité augmentée que nous avons pu voir dans les démonstrations utilisent un nombre limité d'objets virtuels, et en conséquence, ils attachent chaque objet virtuel à un marqueur, au moment du rendu, le remplacement de la matrice de transformation de l'objet s'avère moins pratique dans le cas d'une application où le nombre d'objets virtuels est important. Nous avons implémenté un mécanisme qui permet de superposer un nombre non limité d'objets virtuel en utilisant uniquement deux marqueurs. Le principe est d'utiliser le marqueur en tant que repère pour l'ensemble des objets virtuels. Les transformations de l'objet virtuel sont exprimées d'abord selon un repère fixe, puis elles sont traduites en fonction de la position actuelle du marqueur. Ceci se traduit mathématiquement par la multiplication de la matrice de transformation de l'objet par la matrice de transformation du marqueur.

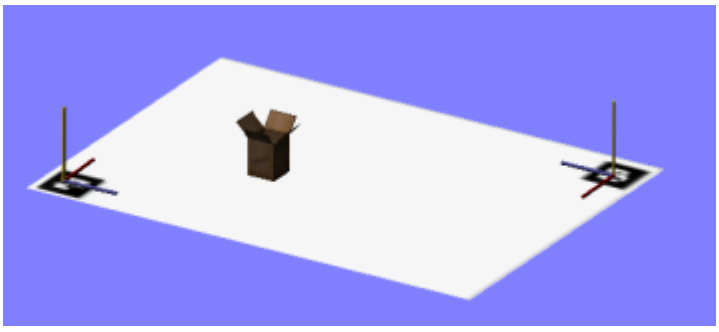


Figure IV-14 Principe de positionnement d'objets en mode RA

La Figure IV-14 présente cette approche, nous avons utilisé deux marqueurs placés au niveau de l'angle bas-gauche et l'angle haut-droit, l'objectif est qu'au moins l'un des deux marqueurs reste visible pendant le mouvement de la caméra.

Pour simplifier la gestion de la scène et améliorer cette approche, nous avons utilisé la technique du Graphe de scène, pour lequel nous avons trois types de nœuds, les nœuds Marquer, Transformation et Objet.

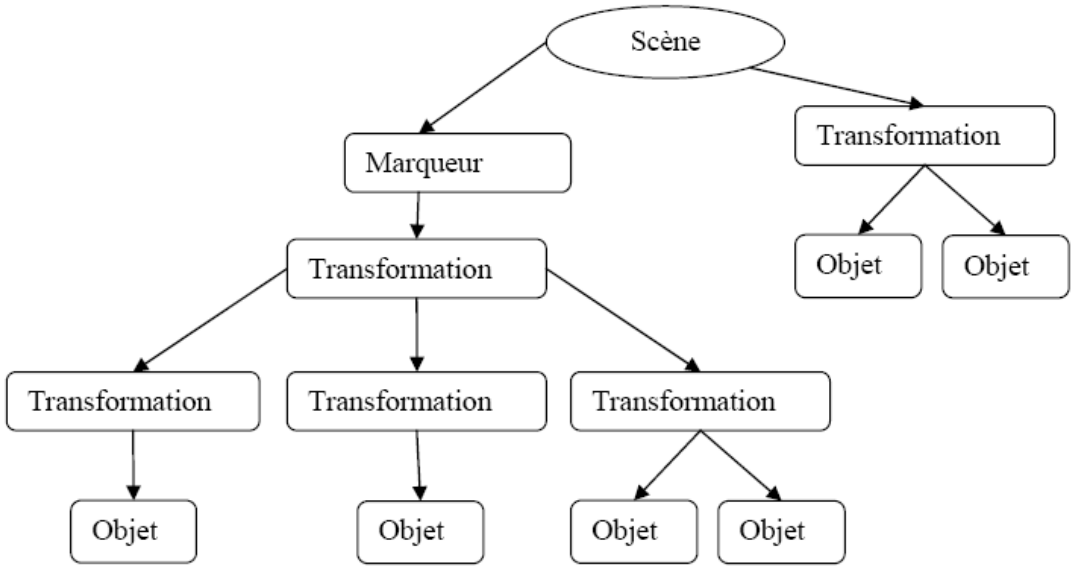


Figure IV-15 Scène graphe utilisé pour la gestion de la scène

Au moment du chargement de la scène depuis le fichier XML générée depuis l'éditeur de niveau, chaque objet de la scène et associé à un nœud de type transformation, ce nœud représente la transformation de l'objet par rapport à la scène virtuelle chargée depuis l'éditeur de scène. Ensuite, tous les nœuds transformation associés aux objets sont attachés à un seul nœud transformation. A son tour, ce dernier est attaché au nœud marqueur.

IV.8 Utilisation des capteurs

Les Smartphones d'aujourd'hui viennent avec une panoplie de capteurs tels que l'accéléromètre, la boussole, le système de positionnement GPS, le microphone, la caméra…etc. Ensemble, ces capteurs permettent de développer toute une nouvelle

différents capteurs disponibles dans le Smartphone utilisé pour la réalisation du présent travail.

Figure IV-16 Exemple de capteurs disponibles dans l'IPhone

Dans ce travail nous avons étudié les applications potentielles de ces capteurs dans le cadre d'une application de réalité augmentée, nous proposons les applications suivantes :

L'accéléromètre, ce capteur mesure l'orientation exacte du téléphone mobile par rapport à une position initiale, les valeurs de l'accéléromètre sont généralement utilisées pour contrôler des objets virtuels, dans ce travail nous étudions la possibilité d'utiliser ces valeurs pour assister le module de suivi visuel car ces valeurs représentent l'orientation de la caméra ce qui permet d'améliorer la précision de module de suivi.

La caméra, la caméra principale est utilisée pour filmer le monde réel pour envoyer la vidéo au module de suivi. Les Smartphones disposent d'une deuxième caméra mise en place pour les appels vidéo sur les réseaux 3G, le fait que cette caméra soit toujours en face de l'utilisateur permet de l'utiliser comme une autre source d'interaction basée sur la reconnaissance de gestes.

Le microphone, ce capteur peut également être utilisé comme moyen d'interaction potentiel, cette possibilité peut être réalisée grâce à un module de reconnaissance vocale.

Le capteur de lumière ambiante, ce capteur est utilisé principalement pour paramétrer l'appareil photo du Smartphone en fonction des conditions lumineuses de

éclairage en réalité augmentée, un problème que nous n'abordons pas dans le cadre de ce travail

Dans la partie suivante nous montrons comment récupérer les valeurs de l'accéléromètre en utilisant le SDK de la plateforme IOS. Ainsi, pour utiliser l'accéléromètre, il faut suivre les étapes suivantes ;

- Créer un objet de type UIAccelerometer;
- Implémenter le protocole *UIAccelerometerDelegate* ;
- Récupérer les valeurs de trois axes X,Y et Z;

Le code suivant permet de récupérer les valeurs de l'accéléromètre et les afficher dans une interface utilisateur simplifiée.

```
- (void)viewDidLoad
{

    self.accelerometer = [UIAccelerometer sharedAccelerometer];
    self.accelerometer.updateInterval = .1;
    self.accelerometer.delegate = self;

    [super viewDidLoad];
}

- (void)accelerometer:(UIAccelerometer *)accelerometer didAccelerate:(UIAcceleration *)acceleration {
    self.labelX.text = [NSString stringWithFormat:@"%@%f", @"X: ", acceleration.x];
    self.labelY.text = [NSString stringWithFormat:@"%@%f", @"Y: ", acceleration.y];
    self.labelZ.text = [NSString stringWithFormat:@"%@%f", @"Z: ", acceleration.z];

    self.progressX.progress = ABS(acceleration.x);
    self.progressY.progress = ABS(acceleration.y);
    self.progressZ.progress = ABS(acceleration.z);

}
```

Extrait de code IV-6 accès à l'accéléromètre del'IPhone

Ainsi, les valeurs de l'accéléromètre sont mises à jour à chaque fois qu'un mouvement est détecté, Apple recommande de faire la moyenne des valeurs consécutives de l'accéléromètre de l'IPhone avant de les utiliser. Nous utilisons un tampon dont la taille varie entre 4 et 10, les lectures successives sur l'accéléromètre sont enregistrées dans ce tampon, ensuite la moyenne de ces lectures est calculée et les valeurs X,Y et Z sont envoyées au serveur.

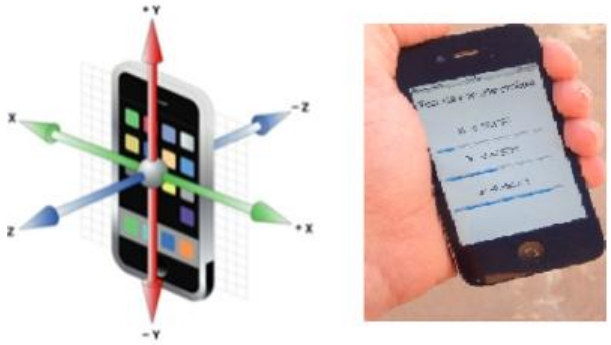


Figure IV-17 Teste de l'accéléromètre

IV.9 Utilisation de la caméra

Depuis la version 4 de l'IOS, il est désormais possible d'accéder à la caméra de l'iPhone, il est donc possible d'envoyer la vidéo de la caméra au serveur en temps réel pour que ce dernier puisse appliquer le suivi visuel, détecte les marques et passe le résultat au module de mise à jour du jeu. Ce dernier met à jour les objets virtuels, le module de rendu est ensuit invoqué et le résultat du rendu est renvoyé au Smartphone. Il faut noter que le streaming temps réel de la vidéo de la caméra n'est pas pris en charge par défaut par les Smartphones, l'idée consiste alors à lancer la vidéo de la caméra et à prendre des captures (Snapshots) dans un intervalle de temps réguliers, ces captures sont des images de format JPEG, ils sont envoyées en temps réel au serveur pour former un flux vidéo MJPEG.

Le code suivant permet d'initialiser la capture vidéo de la caméra de l'iPhone:

```
- (void)initVedioCapture
{
    self.session = [[AVCaptureSession alloc] init];
    self.session.sessionPreset = AVCaptureSessionPreset352x288;

    self.device = [AVCaptureDevice defaultDeviceWithMediaType:AVMediaTypeVideo];
    self.input = [AVCaptureDeviceInput deviceInputWithDevice:self.device error:nil];

    self.output = [[AVCaptureVideoDataOutput alloc] init];
    self.output.videoSettings = [NSDictionary dictionaryWithObject:[NSNumber numberWithInt:kCVPixelFormatType_32BGRA]
                                                            forKey:(id)kCVPixelBufferPixelFormatTypeKey];

    dispatch_queue_t queue;
    queue = dispatch_queue_create("new_queue", NULL);

    [self.output setSampleBufferDelegate:self queue:queue];

    [self.session addInput:self.input];
    [self.session addOutput:self.output];

    [self.session startRunning];
}
```

Extrait de code IV-7 Initialisation de la caméra dans la plateforme IOS

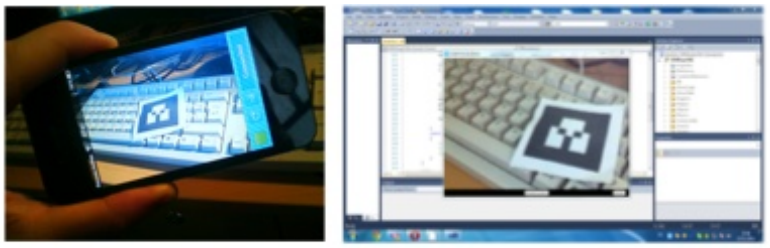
Après l'initialisation de la caméra, nous lançons un thread dont le rôle est de prendre une image chaque N millisecondes et l'envoyer au serveur. La résolution de l'image et le nombre d'images envoyées par seconde dépendent de la performance globale de l'application. Ces paramètres seront discutés ultérieurement dans ce chapitre. Le pseudo code suivant représente la capture et l'envoi d'une image de 352x288 chaque 62 milliseconde.

```
- (void)captureOutput:(AVCaptureOutput *)captureOutput didOutputSampleBuffer:(CMSample
                                                    fromConnection:(AVCaptureConnec
{
    CVPixelBufferRef pixel_buffer = CMSampleBufferGetImageBuffer(sampleBuffer);
    CIImage *ciImage = [CIImage imageWithCVPixelBuffer:pixel_buffer];
    UIImage * frame = [UIImage imageWithCIImage:ciImage];
    CGFloat compressionLevel = 0.5;
    NSData * imageData = UIImageJPEGRepresentation(frame, compressionLevel);
    [self sendFrameToServer:imageData];

}
```

Extrait de code IV-8 Capture et envoi d'une image au serveur

La figure suivante représente la vidéo de l'IPhone reçu en temps réel par le serveur.



(a) (b)

Figure IV-18 (a) Vidéo capturée par le Smartphone (b) vidéo reçue par le serveur.

IV.10 Rendu à distance sur l'écran du Smartphone

Notre application cliente est capable de transférer l'image de la caméra du Smartphone au serveur accompagnée de différentes donnés des capteurs, le serveur utilise l'image de la caméra pour réaliser le suivi visuel des marqueurs visuels. Les donnés des capteurs peuvent être utilisées par le serveur pour contrôler le véhicule ainsi que pour assister le suivi visuel.

Le rendu de finale du jeu est effectué sur une texture en utilisant le RenderTarget de la plateforme XNA, cette texture est ensuite encodée en image JPEG avec un taux de compression que nous pouvons choisir, l'image JPEG est ensuit transmit au client pour qu'elle soit afficher sur l'écran du Smartphone. La

Figure IV-19 Rendu à distance sur l'écran du Smartphone

IV.11 Interface utilisateur de l'application mobile

L'interface utilisateur de l'application IPhone que nous avons développée se compose de :

- Un menue principale donnant accès aux différents modes du jeu ainsi qu'au page de configuration.
- Une page de configuration permettant de modifier les paramètres du jeu;
- Une page d'informations sur l'application;
- Une fenêtre permettant d'exécuter le jeu en mode RV;
- Une fenêtre permettant d'exécuter le jeu en mode RA.

Le menu principal de l'application est constitué de quatre boutons, chacune de ces boutons permet de naviguer vers une interface différente. La Figure IV-20 montre l'apparence finale de ces interfaces sur l'IPhone.

Figure IV-20 Menu principal de l'application client.

Le contrôle de véhicule dans l'application client se fait à l'aide de l'écran tactile et l'accéléromètre de l'IPhone, les gestes de l'utilisateur sont instantanément interprétés et transmit au serveur. Nous utilisons un joypad et des boutons virtuels pour contrôler le jeu.

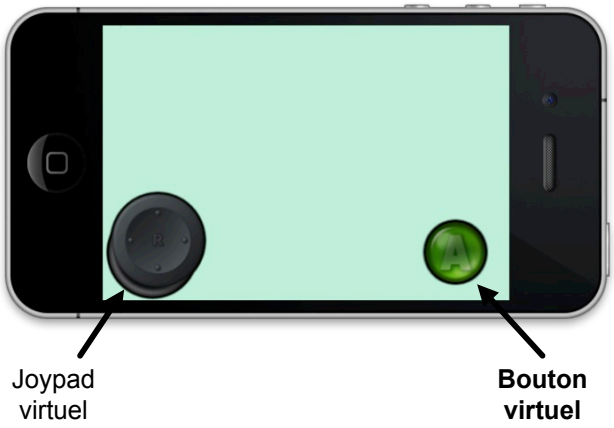


Figure IV-21 joypad et bouton virtuel utilisés pour contrôler le jeu

IV.12Conclusion

Dans ce chapitre, nous avons présenté le processus d'implémentation d'un serious game en RA mobile. Nous avons présenté une architecture exploitant une architecture client-serveur. Le client est un Smartphone qui sera utilisé par le joueur pour interagir avec le jeu en utilisant la réalité augmentée comme interface. Le serveur reçoit la vidéo de la caméra du Smartphone et les valeurs de son accéléromètre à travers une liaison sans-fil et les analyse en temps réel. Ensuite le

Conclusion et perspectives

Conclusion et perspectives

Dans ce mémoire, nous avons abordé la problématique de développement des applications de réalité augmentée et particulièrement les jeux vidéo. Nous avons choisi de s'orienter vers le domaine des serious game, un domaine de recherche émergeant et en plein essor. Les contributions que nous relevons de ce travail sont :

- La proposition d'un serious game d'aide à l'apprentissage et le respect des règles de circulation routières;
- La réalisation d'une version en réalité virtuelle du serious game proposé;
- La réalisation d'une version en réalité augmentée mobile du même serious game;
- Proposition d'une architecture en client-serveur permettant de combiner les performances d'un ordinateur de bureau et les possibilités offertes par les Smartphones modernes.

Le serious game que nous avons proposé traite un sujet très important dans note vie quotidienne, celui de la sécurité routière. Notre contribution principale est la mise en relation entre le scénario du jeu et le contenu sérieux de telle manière que le message pédagogique puis être transmit de façon complètement implicite.

L'existence du serious game en version RV permet de toucher une large audience, car le jeu ne demande qu'un ordinateur de bureau pour être utilisé. D'un autre côté, l'existence de cette version nous permet de focaliser l'étude uniquement sur les aspects liés aux serious game.

Malgré que ses capacités qui ne cessent d'augmenter, un Smartphone ne peut pas assurer tout type de fonctionnalité de manière autonome. L'utilisation d'une architecture en client serveur telle que nous l'avons proposée permet de réaliser une large gamme d'applications telle que :

- Des jeux de haute qualité visuelle sur des Smartphones;
- Des jeux multi joueur et des applications collaboratives;
- Accéder à des larges bases de donnés.

Perspectives envisageables pour ce travail

Conclusion et perspectives

Le développement des jeux vidéo est un travail d'équipe par nature, le fait que nous avons assuré la tâche de réalisation individuellement nous a obligé à focaliser uniquement sur quelques aspects du processus de développement au détriment de quelques critères de qualité. Notre serious game peut être perfectionné de la manière suivante :

- Ajouter plus de niveaux et missions dans le jeu;
- Améliorer la qualité et le réalisme visuel, en utilisant des modèles 3D de bonne qualité et en profitant des possibilités offertes par les GPU modernes;
- Ajouter un aspect multi-jour au game play du jeu.

Sur le plan réalité augmentée mobile, les Smartphones de dernière génération sont déjà dotés d'une puce graphique capable d'assurer le rendu du jeu. Nous pensons qu'il est possible de rendre l'architecture proposée (client serveur) beaucoup plus équilibrée, en implémentant le rendu du jeu partiellement ou complètement sur le Smartphone.

Il est également possible de porter l'une des bibliothèques de réalité augmentée open source telle que ARtoolkit sur le système iOS en bénéficiant de l'accélération matériel de la puce graphique de l'IPhone.

Bibliographie

1 R. Azuma, Y. Baillot, R. Behringer, S. Feiner, S. Julier & B. MacIntyre, Recent Advances in Augmented Reality, IEEE Computer Graphics and Applications, Vol. 21(n°6), 2001: pp. 34-47.

2 R. Azuma, A survey of augmented reality, Presence: Teleoperators and Virtual Environments, Vol. 6(n°4), 1997: pp. 355-385.

3 M. Schneider, B. Schwald, H. Seibert & T. Weller, Medarpa, a medical augmented reality system for minimal-invasive interventions, Studies in health technology and informatics, n° 94, 2003: pp. 312-314.

4 S. Feiner, B. Macintyre & D. Seligmann, Knowledge-based augmented reality, Communications of the ACM - Special issue on computer augmented environments, Vol. 36(n°7), 1993: pp. 53-62.

5 R. Wichert, Collaborative Gaming in a Mobile Augmented Reality Environment. In: in EUROGRAPHICS - Ibero-American Symposium in Computer Graphics - SIACG 2002, Proceedings Guimarães, Portugal, 2002: pp. 31-37.

6 B. Thomas, B. Close, J. Donoghue, J. Squires, P. Bondi & W. Piekarski, First Person Indoor/Outdoor Augmented Reality Application: ARQuake. Personal and Ubiquitous Computing, Vol. 6(n°1), 2002: pp. 75-86.

7 D. Wagner, T. Pintaric, F. Ledermann & D. Schmalstieg, Towards massively multi-user augmented reality on handheld devices, In: Proceedings of the Third international conference on Pervasive Computing, Berlin, Heidelberg, 2005: pp. 208-219.

8 M. Fjeld, F. Voorhorst, M. Bichsel & H. Krueger, Navigation Methods for an Augmented Reality System, In: Extended Abstracts of ACM Conference on Human Factors in Computing Systems CHI'00, 2000, pp. 8-9.

9 N. Streitz, T. Prante, C. Müller-tomfelde, P. Tandler & C. Magerkurth, Roomware: the second generation. In: CHI '02 extended abstracts on Human factors in computing systems, New York, USA, 2002: pp. 506-507.

Bibliographie

10 J. Rekimoto & K.Nagao, The world through the computer: computer augmented interaction with real world environments, In: Proceedings of the 8th annual ACM symposium on User interface and software technology, New York, USA, 1995: pp. 29-36.

11 O. Bimber & R. Raskar, Spatial Augmented Reality: Merging Real and Virtual Worlds, Natick, MA, USA, A. K. Peters Ltd Editions, 2005.

12 J. Rolland, L. Davis & Y. Baillot, A survey of tracking technologies for virtual environments, Fundamentals of Wearable Computers and Augmented Reality (Chap. 3), 2001: pp. 67-112.

13 D. Wagner, T. Langlotz & D. Schmalstieg, Robust and unobtrusive marker tracking on mobile phones. In: ISMAR 2008. 7th IEEE/ACM International Symposium on Mixed and Augmented Reality , 2008: pp. 121-124.

14 A. Comport, E. Marchand, M. Pressigout & F. Chaumette, Real-time markerless tracking for augmented reality: the virtual visual servoing framework. IEEE Transactions on Visualization and Computer Graphics, Vol. 12(n°4) , 2006: pp. 615-628.

15 S. You, U. Neumann & R.Azuma, Hybrid Inertial and Vision Tracking for Augmented Reality Registration, IEEE Virtual Reality Conference VR '99, n°1, 1999: pp. 260-267.

16 RL. Holloway, Registration Error Analysis for Augmented Reality, Presence, Vol. 6(n°4) , 1997: pp. 413-432.

17 J. P. Rolland & H. Fuchs, Optical Versus Video See-Through Head-Mounted Displays, In: Medical Visualization, Presence: Teleoperators and Virtual Environments, 2000: pp. 287-309.

18 E. Viirre, H. Pryor, S. Nagnata & T.A. Furness, The virtual retinal display: a new technology for virtual reality and augmented vision in medicine, Studies in health technology and informatics, n° 50, 1998: pp. 252-257.

Bibliographie

19 M. Mohring, C. Lessig & O. Bimber, Video See-Through AR on Consumer Cell-Phones, In: Proceedings of the 3rd IEEE/ACM International Symposium on Mixed and Augmented Reality, Washington DC, USA, 2004: pp. 252-253.

20 M. Zyda, From Visual Simulation to Virtual Reality to Games, IEEE Journal Computer, Vol. 38, Issue 9, 2005: pp. 25-32.

21 E. Amato, Vers une instrumentalisation communicationnelle des jeux vidéo : quelles formes de séduction idéologique ou publicitaire ? Dans I MEIMARIS M., GOUSCOS D. (dir.), Enjeux et Usages des Technologies de l'Information et de la Communication, tome 2, [Actes du Colloque international EUTIC 2007 du 7-10 novembre 2007, Athènes], Gutenberg publications, 2007: pp. 306-315.

22 J. Alvarez, Du jeu vidéo au serious game: Approches culturelle, pragmatique et formelle, Thèse de Doctorat de l'université de Toulouse, 2007.

23 A. Tricot & A. Rufino, Modalités et scénario d'interaction dans des environnements informatisés d'apprentissage, Revue des Sciences de l'éducation, numéro thématique, XXV (1) , n° 1, 1999: pp. 105-129.

24 M. Zyda, A. Mayberry, C. Wardynski, R. Shilling & M. Davis, The MOVES institute's America's army operations game. In: Proceedings of the 2003 symposium on Interactive 3D graphics, New York, NY, USA, 2003: pp. 219-220.

25 T. Tami & P. Margie, Les jeunes et la sécurité routière. Rapport de l'organisation mondiale de la Santé, Genève, 2007.

26 C. Games, Référentiel de compétences 2010 - Les métiers de production du Jeu Vidéo, 2010. [*w*ww.lesmetiers.net/]

27 T. Zuvich, Vehicle Dynamics for Racing Games, http://www.gdconf.com/2000/library/homepage.htm

28 A. Clarck, Serious Games, University Press of America, 1970.

29 Serious Games, 2006: http://seriousgames.canalblog.com/

30 V. Argyriou, M. Sevaslidou & S. Zafeiriou, Virtual University as a Role Playing Game, IEEE Education Engineering (EDUCON), Madrid, SPAIN, 2010.

31 World Food Program: , United Nations: http://www.wfp.org/

Bibliographie