Université Paul Sabatier – Toulouse 3

Étude des *Serious Games* en Réalité Virtuelle Distribuée

MÉMOIRE

présenté et soutenu publiquement le 26 juin 2007

pour l'obtention du

Master Recherche

(Spécialité Image, Information, Hypermédia)

par

MURATET Mathieu

Composition du jury

Rapporteur : M. FARINAS Jérôme, Maître de Conférences, UPS, Toulouse

Encadrant : M. TORGUET Patrice, Maître de Conférences, UPS, Toulouse

Institut de Recherche en Informatique de Toulouse — UMR CNRS 5505

Remerciements

Je remercie Monsieur Luis Fariñas del Cerro, directeur de l'IRIT, pour m'avoir accueilli dans son laboratoire.

Je remercie Monsieur le Professeur Jean-Pierre Jessel, responsable de l'équipe VORTEX (Visual Objects from Reality to EXpression), pour m'avoir accepté dans son équipe.

Je remercie mon encadrant Patrice Torguet pour ses conseils, son aide et sa disponibilité, mais aussi Jean-Christophe Hoelt, Vincent Forest et Mathias Paulin pour leur aide et toute l'équipe VORTEX pour la sympathie et l'accueil de ses membres.

Á Céline...

Table des matières

Table des figures

Introduction

Dans les années 80, la révolution informatique a permis l'émergence de la réalité virtuelle dans laquelle des individus réalisent une expérience immersive et interactive dans un univers de synthèse. Cette discipline a connu son origine dans le domaine militaire en permettant de préparer des individus à des situations critiques. Dans le domaine civil, elle s'est développée avec les simulateurs de conduite. Les premiers systèmes permettaient d'immerger une seule personne à la fois dans un même monde virtuel. Par la suite, les innovations technologiques ont autorisé l'interaction de plusieurs utilisateurs distants. La réalité virtuelle distribuée est donc apparue avec de nouvelles perspectives et défis à surmonter.

Aujourd'hui, les jeux vidéo en réseau constituent des applications singulières de la réalité virtuelle distribuée. Les technologies des jeux vidéo, appliquées à des secteurs non divertissants, résultent d'un nouveau champ de recherche en informatique : le *serious game* ou jeu sérieux. Ces nouveaux jeux permettent la simulation, l'immersion et l'interaction dans des mondes virtuels en vue de former le joueur au domaine traité par le jeu. Le jeu vidéo est alors utilisé comme vecteur pour amener le joueur à résoudre des problèmes. Dans ce contexte d'apprentissage interactif, l'apprenant devient acteur de sa propre formation.

Le jeu sérieux peut être utilisé dans de nombreux secteurs d'activité comme en informatique, notamment en programmation. Notre objectif est d'aborder cette discipline en réalisant un prototype de jeu sérieux permettant l'apprentissage de la programmation par objet à travers un jeu de stratégie temps-réel.

Dans ce document, nous dresserons, tout d'abord, un état de l'art permettant de définir et de bien comprendre les objectifs de la réalité virtuelle, de la réalité virtuelle distribuée et des jeux sérieux. Nous présenterons, ensuite, les travaux de recherche réalisés au cours du stage.

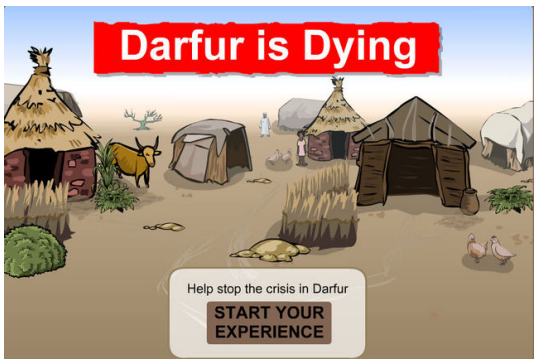

Fig. 1 – Exemple d'un jeu sérieux : *Darfur is Dying.*
Jeu destiné à faire prendre conscience au public des persécutions ayant lieu au Darfour, http ://www.darfurisdying.com/

1

État de l'art

1.1 Réalité virtuelle

La France connaît, depuis une dizaine d'années, un fort développement de la réalité virtuelle. Participant à son essor, un groupe de chercheurs français a rédigé le « Traité de la réalité virtuelle ». Ce document fait référence en la matière, son objectif est de rassembler un maximum d'informations sur les avancées du domaine [FMT06].

1.1.1 Présentation

La réalité virtuelle est une expérience immersive interactive où l'utilisateur perçoit un environnement de synthèse et agit physiquement avec les entités et éléments du monde virtuel à travers des techniques et outils informatiques. La finalité de la réalité virtuelle est de permettre, à une ou plusieurs personnes, une activité sensorimotrice, cognitive et/ou symbolique afin de simuler certains aspects du monde réel. Il ne s'agit donc pas de reproduire à l'identique un monde réel complet dans un monde virtuel. L'intérêt de la réalité virtuelle est de pouvoir s'intéresser à une partie du monde réel ou de modifier cette réalité. Elle permet de s'approcher au plus près des objectifs à atteindre définis lors de la conception de l'application.

La réalité virtuelle tient son origine dans les simulateurs militaires et civils (voir figure 1.1). Cependant, les simulateurs ne sont pas des systèmes de réalité virtuelle, proprement dits, car les postes de simulation ne sont pas simulés mais réels.

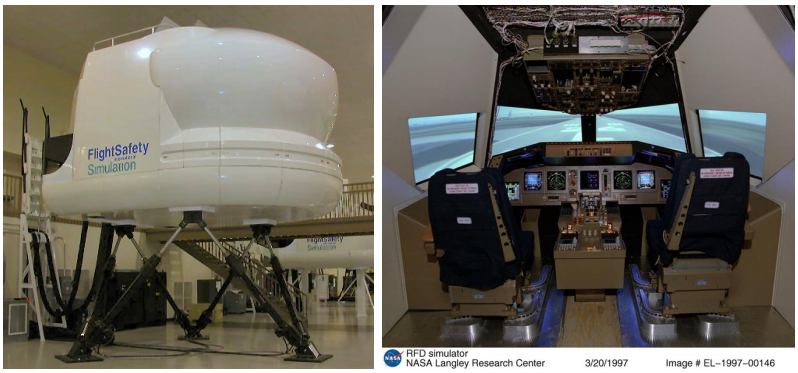

Fig. 1.1 – Simulateurs de vol.
À gauche, un simulateur de vol civil à mouvement « six axes ». À droite, Poste de pilotage expérimental d'un simulateur NASA.

Les périphériques jouent un rôle important dans la réalité virtuelle, car, ils permettent de donner l'illusion à l'utilisateur d'être réellement présent dans un monde virtuel. Grâce à des interfaces fournissant des canaux sensorimoteurs l'utilisateur pourra interagir avec des objets et des phénomènes simulés comme s'ils étaient réels. Les capacités et perspectives de la réalité virtuelle sont donc fortement dépendantes de la technologie. L'augmentation de la puissance des ordinateurs a permis de lever certaines contraintes, mais reste toujours un facteur limitant.

Actuellement, la réalité virtuelle exploite le domaine des sciences et des techniques ainsi que le domaine des sciences humaines et des sciences du vivant pour pouvoir répondre aux trois problématiques fortement imbriquées :

- modéliser l'activité humaine en environnements réels et virtuels ;
- modéliser et réaliser l'interface du sujet pour son immersion et son interaction dans un environnement virtuel ;
- modéliser et réaliser l'environnement virtuel.

La réalité virtuelle est donc un nouveau mode de travail et d'expérimentation multimodal[1] nécessitant des compétences dans les sciences de l'ingénieur et les sciences cognitives. Ce nouveau secteur d'activité ouvre des horizons prometteurs dans des domaines tels que la formation, la santé, l'urbanisme, la simulation scientifique ou l'industrie. À terme, la réalité virtuelle deviendra un secteur d'activité à part entière.

1.1.2 Immersion et interaction

Les deux mots-clés de la réalité virtuelle sont : l'interaction et l'immersion. Contrairement aux idées reçues, la réalité virtuelle dépasse la représentation d'environnements virtuels de plus en plus sophistiqués. Elle place, également, l'utilisateur en situation d'interaction avec le monde virtuel. Un dialogue doit donc s'établir entre l'homme et la machine, c'est la boucle « perception, cognition, action » (voir figure 1.2). L'utilisateur perçoit le monde virtuel au travers d'interfaces sensorielles. Les périphériques permettent à l'utilisateur de réaliser des activités qui sont transmises au calculateur. Celui-ci les interprète et modifie l'environnement si besoin, puis restitue les informations sensorielles à l'utilisateur.

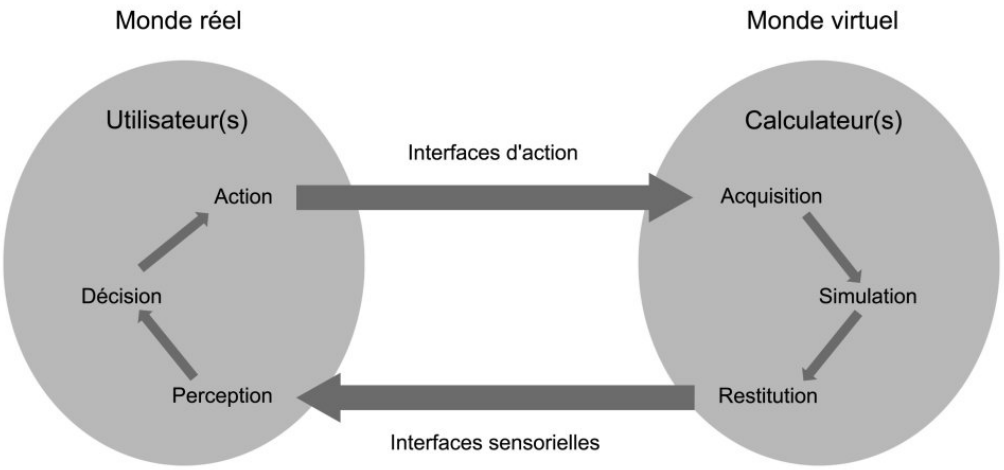

FIG. 1.2 – Boucle « perception, cognition, action » en communication avec le monde virtuel.

[1] multimodalité : qui fait intervenir plusieurs modes d'interaction (visuel, auditif, haptique)

Pour qu'un système de réalité virtuelle soit utilisable, il faut qu'il respecte un certain degré d'interactivité en fonction de l'application à réaliser. Deux contraintes apparaissent donc et doivent être gérées : la latence et les incohérences sensorimotrices. La latence est le décalage temporel entre une action réalisée par l'utilisateur et la perception des conséquences de cette action dans le monde virtuel. Les incohérences sensorimotrices sont les erreurs entre les actions et perceptions de l'utilisateur et respectivement les acquisitions et restitutions du calculateur.

Pour qu'un système de réalité virtuelle soit utilisé, il doit être anthropocentrique (tourné vers l'homme) et non technocentrique (dirigé vers l'ordinateur). En effet, l'utilisateur n'a pas à fournir d'effort pour s'adapter au système, mais le système doit être en adéquation avec les caractéristiques humaines. L'objectif de la réalité virtuelle n'est pas une simple immersion cognitive dans un monde artificiel, c'est une mise en situation pour réaliser une activité donnée. Lors de la conception d'un tel système, la question centrale est : quelles activités l'utilisateur devra-t-il exécuter ? Dans un environnement virtuel interactif, l'utilisateur va utiliser les automatismes acquis dans la vie réelle, pour organiser le virtuel selon un ensemble de règles spatio-temporelles et causales. Ce comportement naturel a été étudié par le psychologue Jean Piaget en définissant la notion de schèmes qui est l'organisation mentale des actions telles qu'elles se transfèrent ou se généralisent lors de la répétition de cette action en des circonstances analogiques. Donc, quelque soit l'application de réalité virtuelle, les actions des utilisateurs sont décomposables en plusieurs « Primitives Comportementales Virtuelles » (PCV) ou comportements basiques et peuvent être regroupées en quatre catégories : observer, se déplacer et agir dans le monde ainsi que communiquer avec autrui ou avec l'application.

L'immersion ne peut être réalisée qu'en présentant des informations et fournissant des méthodes d'interaction à l'utilisateur afin qu'il puisse se représenter le monde virtuel et y agir. Deux types de visualisation et d'interaction sont possibles : la représentation métaphorique et la représentation idiomatique.

- la représentation métaphorique : consiste à présenter les données par des objets du monde réel et les interactions proposées font référence à des actions réelles. Le principal atout de cette représentation est de placer l'utilisateur dans une situation familière qu'il reconnaît sans apprentissage. Cependant, la métaphore ne doit pas introduire de contresens dans l'observation, ou induire des limitations artificielles sur les actions réalisables par l'utilisateur ;
- la représentation idiomatique : consiste à présenter des objets et interactions non nécessairement issus du monde réel. Cette représentation est plus efficiente que la métaphorique car elle est pensée dans un souci d'efficacité, mais elle nécessite une formation.

1.1.3 Différents types de périphériques

Différents types de périphériques peuvent être utilisés pour réaliser l'immersion et l'interaction de l'utilisateur dans le monde virtuel. Le choix des dispositifs dépend de l'objectif à atteindre et des conditions de travail (revue de détail, revue de projet, nombre de personnes...).

Dispositifs d'affichage

Les dispositifs d'affichage vont permettre à l'utilisateur de visualiser le monde virtuel (voir figure 1.3).

- les casques de visualisation stéréoscopiques (ou HMD : *Head Mounted Display*) permettent une immersion totale avec six degrés de liberté, mais fournissent des couleurs et une définition de faible qualité ainsi qu'un champ de vision et une interaction réduite ;

- les fenêtres magiques comme le WindowVR contredisent l'idée selon laquelle il serait nécessaire d'utiliser des visiocasques pour faire de la réalité virtuelle en proposant une solution complète contenant écran de visualisation, traqueurs de position et commandes de navigation ;
- les murs d'images, les *powerwall* et les *reality center* permettent la visualisation d'objets en haute et très haute définition en proposant des présentations immersives pour des assemblées allant jusqu'à soixante personnes ;
- les écrans auto stéréoscopiques fournissent une visualisation d'image en 3D sans dispositifs supplémentaires ;
- les lunettes pour la vision stéréoscopique active/passive permettent une immersion partielle avec de bonnes couleurs et une définition d'image correcte, mais limitent les déplacements ;
- les *workbenchs* tolèrent une présentation en groupe restreint d'objets ou de scènes stéréoscopiques ne nécessitent pas un haut degré d'immersion ;
- les hémisphères (jusqu'à cinq personnes) et les dômes (jusqu'à trois cents personnes) rendent les images plus immersives de manière à occuper 180° du champ visuel de l'utilisateur ;
- les CAVE (*Cubic Automatic Virtual Environment*) permettent une immersion au travers un affichage rétro projeté sur les six faces d'une pièce cubique dans laquelle se situe l'utilisateur ;
- les dispositifs d'affichage mobile (PDA : *Personal Digital Assistant*, tablettes) fournissent des écrans couleurs mais une faible résolution et peu de puissance graphique.

Fig. 1.3 – Dispositifs d'affichage.
De gauche à droite et de haut en bas : casque de visualisation stéréoscopique (de CYBERMIND), WindowVR (de Vistual Research Systems, Inc.), reality center (de Barco - SGI), écran auto stéréoscopique (de SynthaGram), lunette de vision stéréoscopique active (de CrystalEyes), workbench (du LSIIT : Laboratoire des Sciences de l'Image, de l'Informatique et de la Télédétection), hémisphère (d'elumens), CAVE (de Barco), PDA (d'Acer).

Dispositifs d'interaction

Deux types de périphériques sont disponibles. Les périphériques isométriques appartiennent à la première catégorie et opposent une résistance infinie à l'action de l'utilisateur. Ils sont immobiles ou a débattement limité et mesurent les mouvements en fonction des forces appliquées. Cependant, il y a une décorrélation entre le mouvement de la main et celui du pointeur. Le manque de retour d'effort peut gêner les tâches de sélection et l'utilisateur se fatigue du fait de la force nécessaire pour le manipuler. Les périphériques isotoniques appartiennent à la deuxième

catégorie. Ils ne possèdent aucune résistance et se déplacent librement avec la main.

Les dispositifs d'interaction vont permettre à l'utilisateur d'agir sur le monde virtuel (voir figure 1.4).

- les souris 3D (isométriques) fournissent six degrés de liberté ;
- les gants de données (isotoniques) permettent la manipulation d'objets tridimensionnels ainsi que la navigation et l'animation de scènes 3D. Un retour haptique (forces et/ou sensations) peut également être disponible (dispositifs portés à retour d'effort) ;
- les dispositifs tenus à retour d'effort (isométriques) différent des dispositifs portés sur l'espace dans lequel l'utilisateur évolue et sur les forces mises en jeu ;
- les capteurs de mouvements (isotoniques) magnétiques, optiques, mécaniques, acoustiques ou inertiels suivent les déplacements d'objets ou d'individus. Chaque technologie a ses avantages et inconvénients par exemple les capteurs magnétiques sont sensibles aux distorsions et interférences liées aux objets métalliques et les capteurs optiques sont limités par des problèmes d'occultation, de reflets ou de champ angulaire réduit. Il est, cependant, possible de combiner plusieurs technologies comme l'optique et l'inertiel en vue de faciliter la reconstruction d'une trajectoire suite à l'obstruction d'une cible par rapport aux caméras.

Fig. 1.4 – Dispositifs d'interaction.
De gauche à droite : souris 3D (de Hewlett-Packard), gant de donnée (de 5DT : Fifth Dimension Technologies), dispositif tenu à retour d'effort (de SensAble Technologies), capteurs de mouvement optiques.

Dans le domaine de la réalité virtuelle, un ou plusieurs utilisateurs sont immergés et interagissent avec un monde de synthèse. Un tel dispositif ne s'adresse qu'à des individus colocalisés et n'autorise pas la mise en relation de personnes distantes les unes des autres. La réalité virtuelle distribuée lève cette contrainte.

1.2 Réalité virtuelle distribuée

La réalité virtuelle distribuée a également été étudiée dans le « Traité de la réalité virtuelle ». La base de cette partie du rapport repose donc sur l'étude de J.P. Jessel et P. Torguet [JT06].

1.2.1 Présentation

La réalité virtuelle distribuée est une extension de la réalité virtuelle, car plusieurs personnes connectées en réseaux partagent virtuellement le même monde synthétique. La réalité virtuelle distribuée doit donc être capable de gérer toutes les fonctionnalités d'une application de réalité virtuelle classique (la gestion des périphériques spécialisés, l'application construite sur l'environnement virtuel, le rendu visuel, le rendu sonore, les interactions avec le monde virtuel...) ainsi que les communications réseaux permettant l'échange d'informations et le maintien de la co-

hérence du monde virtuel. Tous ces traitements doivent être effectués en un temps suffisamment court de façon à maintenir une interactivité convenable.

Les mondes synthétiques créés et gérés par un système de réalité virtuelle distribuée peuvent être virtuellement peuplés par de nombreux utilisateurs qui interagissent avec des mondes synthétiques au travers d'applications coopératives très réalistes et très intuitives. Les utilisateurs de ces environnements virtuels peuvent être colocalisés ou physiquement éloignés de plusieurs centaines de kilomètres voire même sur des continents différents.

1.2.2 Historique

La réalité virtuelle distribuée tient son origine dans les applications militaires, ludiques et démonstratives. Au niveau militaire SIMNET (*simulator networking*) puis DIS (*Distributed Interactive Simulation*) furent les premières applications fournissant un environnement réparti. Au niveau ludique et démonstratif, les MUD (*Multiple User Dimensions*), SGI Flight & Dogfight, Habitat, RB2 (*Reality Built for two*) et Doom ont ouvert la voie à la réalité virtuelle distribuée grand public.

- L'objectif de SIMNET (débuté en 1983 et opérationnel en 1990) était de fournir un monde virtuel permettant de simuler un champ de bataille où cent mille entités (simulateurs différents et délocalisés) pouvaient évoluer. L'application devait pouvoir fonctionner en temps réel, en répartition totale (pas de site central) et être peu coûteuse par rapport au coût d'une simulation grandeur nature. Le problème clé du projet était de relier, à travers un réseau, les différentes entités afin de créer un champ de bataille virtuel cohérent ;
- DIS (1993) est le successeur de SIMNET en intégrant plus de généricité. En effet, avec DIS, il n'était plus nécessaire d'utiliser un simulateur pour pouvoir faire partie intégrante de la simulation. Un simple ordinateur pourvu d'une carte réseau et pouvant gérer le protocole de communication de DIS pouvait prendre part à la simulation. La clé de l'architecture de ce système est la détermination du moment où un objet doit envoyer un message ;
- Les premiers mondes virtuels sont les MUD, généralement en mode texte, ils sont apparus au début des années 80 avec les premiers systèmes « en ligne »(*on-line*), les BBS (*Bulletin Board Systems*), pour permettre à leurs utilisateurs de discuter à distance ;
- *SGI Flight* (un simulateur de vol) fut programmé par Gary Tarolli en 1983. Les aspects répartis furent ajoutés petit à petit à partir de 1984. Au début de 1985, *Flight* fut amélioré en *Dogfight* en permettant aux joueurs de se tirer les uns sur les autres. Ils ont ainsi créé le premier monde virtuel en réseau interactif ;
- Suite aux MUD, les premiers environnements virtuels distribués sont apparus aux alentours de 1986. Le tout premier monde virtuel commercial « en ligne » fut Habitat (voir figure 1.5). Créé, en 1986, par *Lucasfilm Games* en collaboration avec *Quantum Computer Services* (qui est devenu depuis *America Online* (AOL)), cet environnement était représenté par des scènes animées en deux dimensions dans lesquelles interagissaient des utilisateurs connectés via des modems. Habitat a réussi à drainer, à l'époque, quinze mille utilisateurs sur une base totale de cent mille utilisateurs de ce type de service « en ligne ». Ces mondes virtuels 2D comme Habitat, communément appelés des *ChatWorlds* (littéralement : mondes où l'on bavarde), peuvent être considérés comme les ancêtres des jeux massivement multijoueurs (*Massively Multiplayer Online Game*) ou MMOG modernes comme *Ultima Online*, *Everquest* ou *World of Warcraft*. Le concept de MMOG est défini dans le paragraphe 1.3.6 ;
- RB2, de la société *VPL Research*, fut, dès 1990, le premier système, grand public de réalité virtuelle multiutilisateur, commercialisé permettant de définir des environnements virtuels

tridimensionnels sur lesquels il était possible d'interagir avec des outils de réalité virtuelle ;
– Le 10 décembre 1993, *Id Software* a sorti la version *shareware* de *Doom*. Ce jeu est probablement à l'origine de tous les jeux en réseau actuels. Il est estimé que la version *shareware* de *Doom* a été récupérée quinze millions de fois sur Internet et/ou passée de personne à personne.

FIG. 1.5 – Une scène typique d'Habitat (© 1986 *LucasArts Entertainment Company*).

1.2.3 Architectures

Pour que deux applications puissent communiquer à travers un réseau, il est nécessaire qu'elles « parlent la même langue », c'est-à-dire qu'elles utilisent la même suite de protocole de communication applicatif. Un protocole est l'ensemble des règles que deux applications vont utiliser pour communiquer. Ces règles définissent le format des paquets à envoyer, leur sémantique et le comportement à adopter en cas d'erreur. La suite de protocoles la plus utilisée actuellement est TCP/IP (*Transmission Control Protocol/Internet Protocol*). L'ensemble de tous les ordinateurs et de tous les réseaux mondiaux connectés, gérés par les protocoles TCP/IP, forme l'Internet.

Méthodes de communication

Les applications utilisant le protocole TCP/IP ont à leur disposition plusieurs méthodes de communication (voir figure 1.6) : le point à point, la diffusion totale (*broadcasting*) et la diffusion sur des groupes de communication (*multicasting*).
– la communication point à point consiste à envoyer un paquet à une application précise, identifiée par l'adresse de la machine sur laquelle elle est en exécution et son numéro de port. Pour ce type de communication, TCP/IP propose deux protocoles différents TCP et UDP (*User Datagram Protocol*). TCP est le protocole le plus fiable, il fonctionne en mode connecté et met en place des mécanismes de sécurité pour éviter les pertes et les mélanges de paquets. UDP, quant à lui, est non fiable mais plus rapide, il n'effectue aucun contrôle, les données peuvent donc être perdues, dupliquées ou arriver dans le désordre. De plus UDP fonctionne en mode non connecté (pour chaque paquet envoyé la destination doit être renseignée). La plupart des systèmes de réalité virtuelle distribuée utilisant

des communications point à point choisissent UDP et lui ajoutent des mécanismes, pour améliorer la fiabilité, en adéquation avec leurs besoins ;
- la diffusion totale consiste à envoyer systématiquement un message à tous les ordinateurs connectés au réseau à diffusion (comme par exemple les réseaux locaux Ethernet). Ceci est beaucoup moins coûteux en ressources réseau (bande passante : quantité de données que peut transporter le réseau en une seconde) que d'envoyer le même message à toutes les machines en utilisant le mode de communication précédent. En fait, la diffusion totale est rarement utilisée en raison de l'inondation du réseau complet par tous les messages. Seules des applications de réalité virtuelle distribuée fonctionnant sur des réseaux spécifiques utilisent cette méthode de communication (par exemple des organismes gouvernementaux comme l'armée américaine). UDP propose un service de ce type ;
- la diffusion sur des groupes de communication consiste à envoyer en une seule opération un message à plusieurs machines sur un réseau. Les machines (en fait leurs applications) qui souhaitent recevoir les messages d'une diffusion demandent leur inscription au groupe correspondant. Ainsi, seules les machines intéressées reçoivent le message (gros avantage par rapport à la diffusion totale). Il est aussi intéressant de noter que les machines peuvent s'inscrire et se retirer d'un groupe dynamiquement. UDP propose un service de ce type.

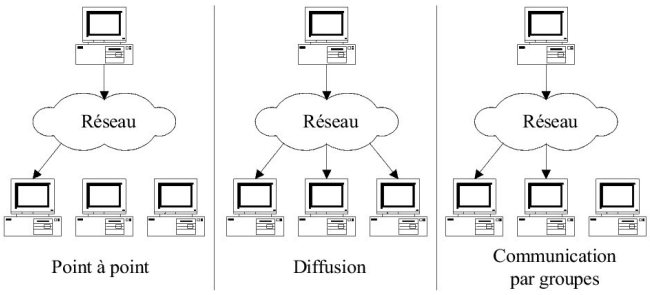

Fig. 1.6 – Différents modes de communication à travers un réseau.

Architectures réparties

L'architecture répartie détermine l'organisation physique des différentes machines ainsi que leur rôle. L'emplacement des données et leur gestion dépendent également de cette architecture. Le choix d'une méthode de communication est donc un point important dans la conception d'une application de réalité virtuelle distribuée. Cependant, le choix de l'architecture reste un point tout aussi crucial. Les deux grands principes s'articulent autour de la présence ou non d'un serveur, on parlera de systèmes centralisés et de systèmes sans serveur.

Systèmes centralisés : Ils s'articulent autour de deux entités : le serveur (une machine généralement très puissante en terme de capacités d'entrée-sortie qui fournit des services) et le client (une machine où s'exécute un programme client qui va communiquer avec le serveur pour récupérer des informations). Ces systèmes peuvent être découpés en trois sous groupes : l'architecture client-serveur centralisée, l'architecture client-serveur distribuée et l'architecture client-serveur distribuée avec plusieurs serveurs :
- architecture client-serveur centralisée (voir figure 1.7) : la base de données représentant le monde virtuel est entièrement gérée par un seul serveur. Lorsqu'un client se connecte à ce serveur, il exige des ressources (mémoire, CPU (*Central Processing Unit*), bande passante)

mais plus le nombre de clients augmente plus cette exigence sera difficile à satisfaire. Ce modèle d'architecture n'est évidemment pas très extensible et permet de gérer au mieux une cinquantaine d'utilisateurs en même temps dans le cas de communications purement textuelles (MUD par exemple). Ce type d'architecture est beaucoup utilisé par les jeux en réseaux ;

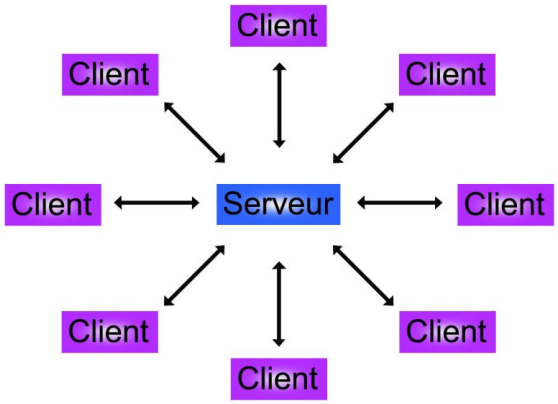

FIG. 1.7 – Architecture client-serveur centralisée.

- architecture client-serveur distribuée : la base de données représentant le monde virtuel est répartie sur les différents clients. Maintenant, seules les communications entre les clients sont gérées par le serveur. Cette amélioration réduit l'utilisation de la bande passante, mais le serveur reste un goulot d'étranglement ;
- architecture client-serveur distribuée avec plusieurs serveurs : Pour gérer ce problème plusieurs serveurs peuvent être mis en place et communiquer. Ces serveurs peuvent être organisés et gérer les clients de diverses façons : partitionnement basé sur la localisation physique des clients (chaque serveur doit avoir une image du monde virtuel dans son ensemble, ce qui est donc difficilement extensible) ou partitionnement basé sur la localisation virtuelle (plus facilement extensible car chaque serveur ne gère qu'une partie du monde virtuel). Cependant, les clients n'étant pas également répartis dans le monde, certains serveurs peuvent être facilement surchargés. Il devient donc nécessaire de mettre en place une gestion des clients en vue d'équilibrer la charge des différents serveurs. Le problème de distribution de charge peut être réalisé par des algorithmes statiques, dynamiques et adaptatifs où les algorithmes adaptatifs peuvent être considérés comme une classe spéciale des algorithmes dynamiques. Les algorithmes de distribution de charges dynamiques peuvent être encore classés en algorithmes de partage de charges et en algorithmes de mise en correspondance de charges. L'équilibrage des charges n'est pas la seule difficulté à surmonter dans ce type d'architecture, car ce modèle augmente la complexité pour maintenir une base de données cohérente et les messages des clients pouvant traverser plusieurs serveurs avant d'aboutir au destinataire, la latence augmente (latence : temps pris par un message pour passer d'un hôte à un autre en lui ajoutant le temps du traitement de l'information émise et reçue), ce qui diminue le niveau d'interactivité. Cependant, cette architecture a pour avantage de diminuer l'impact de l'ajout de nouveaux clients. Les MMOG sont des applications hautement interactives, la migration des clients est une tâche très lourde, car ils gèrent des milliers de personnes. Duong et al. [DZ03] se sont intéressés au problème

et ont mis en place cette architecture pour gérer un MMOG. On considère que tous les MMOG commerciaux utilisent ce type d'architecture.

Systèmes sans serveur : Ils sont organisés autour de l'architecture égal à égal (*Peer-to-Peer* ou P2P). Dans cette configuration, chaque ordinateur a le même rôle, il apporte de la mémoire, du temps de calcul et de la bande passante pour gérer les états des données partagées. La base de données représentant le monde virtuel est dupliquée sur les différentes machines qui communiquent les modifications locales à tous les autres ordinateurs.
Ce type d'architecture présente deux inconvénients majeurs :

- au niveau de l'extensibilité le nombre de messages de mise à jour augmente beaucoup avec le nombre de machines. Pour pallier ce type de problème, il est possible de diminuer la fréquence des messages de mise à jour grâce, par exemple, à la technique du *dead-reckoning* qui sera présentée dans le paragraphe 1.2.4 ;
- au niveau du maintien de la cohérence des copies de la base de données représentant le monde virtuel. De nombreux mécanismes, issus des systèmes d'exploitation distribués, existent pour gérer ce problème, du plus simple : à un instant donné, un seul ordinateur est autorisé à modifier la base de données et ce droit circule de machine en machine (grâce à un anneau à jeton logique par exemple) ; au plus complexe : toutes les machines procèdent à des modifications en précisant leur date et lorsqu'un conflit se produit la modification la plus ancienne est prise en compte et les autres sont annulées.

La méthode de communication étant indépendante de l'architecture choisie, il est tout à fait possible d'utiliser du point à point (voir figure 1.8), du *broadcasting* ou du *multicasting*. On notera tout de même que l'utilisation de nombreux groupes de communication peut augmenter l'extensibilité d'un système en diminuant le nombre de machines en communication directe à tout instant. Ce type d'utilisation des groupes de communication s'apparente à une architecture client-serveur distribuée avec plusieurs serveurs. L'avantage de ce modèle organisationnel, par rapport à une architecture à serveurs multiples, est de minimiser la latence due à la traversée du ou des serveurs.
Traditionnellement les MMOG sont gérés par une architecture client-serveur, car les serveurs

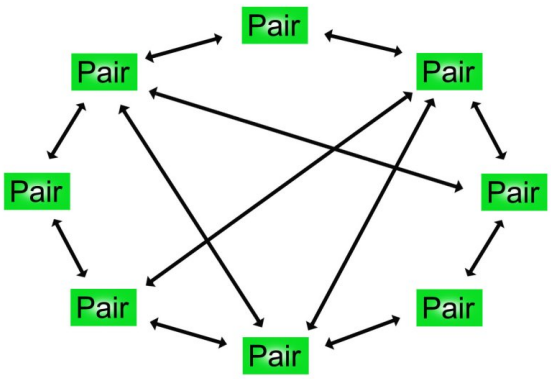

Fig. 1.8 – Architecture pair à pair distribuée en communication point à point.

peuvent gérer plus facilement les comptes des joueurs ainsi que les états du jeu. Cependant, cette architecture est moins flexible, plus onéreuse et limite l'extensibilité du nombre de joueurs.

Knutsson et al. [KLXH04] proposent une approche basée sur le P2P avec un système grandissant et rétrécissant dynamiquement avec le nombre de joueurs. Trois types de problèmes doivent être résolus pour rendre cette approche pleinement applicable : la performance, la disponibilité (perte d'un nœud et maintenance d'un grand nombre de redondance), et la sécurité (protéger les comptes et empêcher la triche). Avec cette architecture réseau et pour un même nombre de joueurs par régions, le nombre total de joueurs connectés n'a aucune conséquence sur les performances. En revanche, le système est plus sensible aux différences de concentration de joueurs, mais ce problème peut être résolu par le jeu en limitant le nombre de joueurs par région ou en modifiant dynamiquement la taille des régions en fonction de l'affluence des joueurs.

Synthèse : Chacune de ces deux architectures possède des avantages et des inconvénients, le choix n'est pas évident. Tout dépend de l'application et de ses contraintes en terme d'extensibilité, de fiabilité, de simplicité et d'interactivité.
Le modèle client-serveur est particulièrement recommandé pour des applications nécessitant un grand niveau de fiabilité. Les ressources étant centralisées, il est plus facile de gérer des ressources communes à tous les utilisateurs et d'éviter les problèmes de redondance et de contradiction. La sécurité est accrue, car le nombre de points d'entrée permettant l'accès aux données est moins important. En revanche, le coût est élevé en raison de la mise en place du matériel. De plus, le serveur est le maillon faible de l'architecture client-serveur, étant donné que toute l'application est architecturée autour de lui. La panne d'un serveur peut perturber, voir même bloquer, complètement l'application.
Le P2P, quant à lui, est plus flexible, plus extensible et beaucoup moins onéreux en terme de matériel, mais il est aussi plus difficile de garantir des performances, de la disponibilité et de la sécurité, car à tout moment un nœud responsable de données peut disparaître.
Il est cependant possible de concevoir des applications de réalité virtuelle mélangeant des architectures client-serveur et groupes de communication. Cela peut permettre de concilier les avantages des deux techniques en limitant leurs inconvénients.

1.2.4 Techniques d'optimisation

Les systèmes de réalité virtuelle distribuée, pour être utilisables, doivent garantir des performances suffisantes et satisfaire une bonne extensibilité. Ces deux contraintes dépendent fortement des performances du réseau entre les différentes machines. Ces performances sont liées à la bande passante, à la latence et à la gigue[2]. Lorsque la gigue devient élevée, le comportement des entités est alors non uniforme et l'utilisateur peut percevoir un manque de réalisme dans la simulation. Pour pallier ces différents problèmes les techniques qui ont été créées peuvent être classées en deux sous-ensembles : les méthodes de prédiction et les méthodes prenant en compte l'intérêt.

Méthodes de prédiction

Les méthodes de prédiction consistent à déterminer le comportement d'une entité en fonction de ses anciens messages de mise à jour. Il est donc nécessaire de mettre à jour l'entité uniquement si la prédiction est trop mauvaise par rapport à la situation réelle.

Le *dead-reckoning* est une technique implémentée dans SIMNET qui extrapole l'état d'un objet. Les objets sont dupliqués sur toutes les machines (ou simulateurs). L'une de ces machines contrôle réellement l'objet et les autres extrapolent, par exemple, la position grâce aux dernières

[2] `gigue : variation de latence entre plusieurs messages successifs`

informations qu'elles ont reçues (position et vitesse). C'est de la responsabilité des objets réellement contrôlés d'émettre des événements avant que les prévisions ne deviennent trop mauvaises. En fait, l'objet réellement contrôlé maintient, lui aussi, un modèle extrapolé le représentant et similaire à celui disponible sur les autres machines. Lorsque ce modèle devient trop éloigné de la réalité (d'après un seuil fixé pour l'objet), un événement contenant la mise à jour de cet objet est diffusé. Lors de la réception d'un message de mise à jour dû à un dépassement de seuil, le simulateur a deux possibilités : il peut modifier brusquement la position de l'objet ou il peut modifier petit à petit la position du véhicule jusqu'à la bonne position. La technique du *dead-reckoning* est très efficace pour limiter les échanges entre les machines.

Méthodes par zone d'intérêt

Le principe consiste à obtenir suffisamment d'éléments (en ce qui concerne la perception, la visibilité, les contraintes système...) afin d'assurer la cohérence du jeu sans avoir à traiter des informations inutiles. En fait, ces méthodes précisent les centres d'intérêt de chaque objet de façon à ne leur envoyer que ce qui est pertinent. On peut donc dire que le système filtre les messages selon l'intérêt des utilisateurs. Trois types de filtrage sont généralement utilisés :

- filtrage fonctionnel pour lequel l'entité peut choisir de communiquer avec un sous-ensemble d'entités en se basant sur leurs fonctionnalités ;
- filtrage spatial pour lequel l'entité interagit avec les entités qu'elle considère proches d'elle ;
- filtrage temporel qui part du principe suivant : certaines entités n'exigent pas des mises à jour en temps réel, elles recevront donc moins fréquemment d'informations.

La réalité virtuelle distribuée permet donc une expérience immersive à plusieurs utilisateurs pouvant être répartis sur tous les continents. Ces utilisateurs participent ensemble à l'évolution du monde virtuel et peuvent vouloir atteindre des objectifs communs ou concurrents. Les jeux vidéo, notamment ceux en réseaux, ont les mêmes caractéristiques que la réalité virtuelle distribuée et travaillent sur des problématiques similaires. De plus, le jeu vidéo peut être enrichi pédagogiquement pour réaliser des simulations et être utilisé dans des formations appliquées et théoriques. Un tel dispositif conduit aux jeux sérieux qui sont une utilisation pertinente de la réalité virtuelle distribuée.

1.3 Jeux sérieux

1.3.1 Définition

Selon Michael Zyda, un jeu sérieux est un combat mental, comportant des règles. Il est joué sur un ordinateur et utilise le divertissement pour atteindre des objectifs [Zyd05a]. Le jeu sérieux n'est cependant pas une application ludoéducative. En effet, une application ludoéducative est centrée sur la pédagogie et l'enseignement à diffuser, ensuite cet enseignement est « habillé » d'un pseudo jeu afin de le rendre plus accessible. Le problème posé par ce type d'approche est décrit par François Chobeaux [Cho06] de façon imagée : « Le jeu ne peut pas faire passer la pilule forcément amère de l'apprentissage ... car le sucre de l'enrobage est vite dissous par la salive de l'élève. ». Le jeu sérieux est avant tout un jeu vidéo qui, s'il est performant et amusant, pourra insuffler au joueur un ensemble d'informations sur le thème abordé. En effet, pour améliorer ses performances dans un jeu vidéo, le joueur doit apprendre. Si le jeu sérieux est motivant et pousse l'utilisateur à progresser, alors, il intégrera automatiquement une grande quantité

d'informations. Le jeu sérieux est donc un environnement d'apprentissage vidéo ludique qui ne s'adresse pas seulement aux enfants, mais à une cible beaucoup plus large.

1.3.2 Historique

Très tôt dans l'histoire des civilisations des activités culturelles sont apparues. Elles servaient à distraire le peuple (jeux olympiques grecs ou jeux du cirque romains). Quelques jeux font également leur apparition comme la « pettie » grecque ou le « latroncule » romain (voir figure 1.9). Ce sont des jeux de prise par encerclement, à connotation guerrière (ancêtres du jeu de dames). Ces jeux n'ont pas pour objectif premier l'éducation ou l'apprentissage de problèmes particuliers. Cependant, ce sont les jeux les plus anciens qui nous sont parvenus, basés sur un raisonnement intellectuel faisant, intervenir la réflexion, la planification et non le hasard.

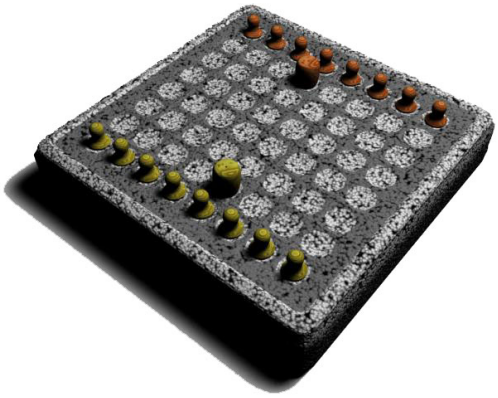

Fig. 1.9 – Un jeu de « latroncule ».

Plus récemment, on pourrait attribuer le rang de premier jeu sérieux à *Army Battlezone* (voir figure 1.10), un projet développé par Atari en 1980, conçu pour l'entraînement des militaires américains. Par la suite, des groupes variés aux États-Unis et au Royaume-Uni, ont utilisé l'éducation par le jeu[3] pour évoquer des problèmes sociaux ou de santé tels que la toxicomanie, la vaccination, les grossesses adolescentes, le SIDA et le cancer. Suivant les régions, le jeu sérieux n'est pas abordé de la même manière, par exemple aux États-Unis le divertissement est privilégié alors qu'en Europe l'aspect éducatif est mis en avant au détriment de l'amusement. La communauté des musées scientifiques utilise de plus en plus fréquemment l'éducation par le jeu. Les premiers exemples français sont le « Palais de la découverte » et la « Cité des sciences et de l'industrie » à Paris. Au fil des années, un nombre croissant de musées a utilisé l'éducation par le jeu tel que la « Cité de l'espace » à Toulouse, « Naüsicaa » à Boulogne-sur-Mer, « Vulcania » près de Clermont-Ferrand et « Bioscope » à Ungersheim en Alsace.

Á l'heure actuelle, le jeu sérieux connaissant le plus grand succès est *America's Army* (voir figure 1.11) de Michael Zyda [Zyd05b]. Après une première évaluation défavorable, *America's Army* a démontré son efficacité comme simulateur d'entraînement militaire. Un soldat, jouant à ce jeu, peut apprendre et perfectionner des notions qui lui permettront de s'améliorer au cours d'exercices réels. En raison de son succès avéré, *America's Army* a été adapté aux services secrets qui nécessitent des entraînements spécifiques. Le succès de ce jeu confirme donc le potentiel des jeux sérieux et ouvre la voix vers les autres secteurs d'activités pouvant avoir recours à ce nouvel outil.

[3] `http ://www.socialimpactgames.com/`

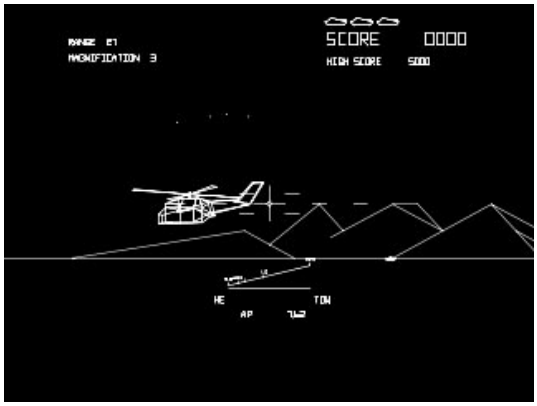

FIG. 1.10 – Premier jeu sérieux : *Army Battlezone*.

FIG. 1.11 – *America's Army* : le jeu sérieux le plus populaire.

1.3.3 Environnement et pédagogie

Pour qu'un jeu sérieux fonctionne, deux aspects doivent être étudiés avec attention : l'environnement dans lequel va évoluer le joueur (le jeu) et la pédagogie mise en place.

Environnement

L'environnement est la composante principale d'un jeu sérieux, car, s'il est bien réalisé, il permettra au joueur de s'investir, de progresser et donc d'apprendre à travers le jeu. Tous les points importants de l'environnement ont été étudiés par Johnson et al. [JVM05] :

- une des principales caractéristiques d'un bon jeu est le *gameplay* (*gameplay* : toutes les activités et stratégies employées par les concepteurs de jeux pour obtenir et garder le joueur engagé et motivé durant tout le jeu). Le *gameplay* ne résulte pas que du graphisme. Deux aspects du *gameplay* sont importants : engager le joueur à chaque instant et relier chaque action aux objectifs futurs ;
- un bon jeu fournit des retours (*feedback*) aux actions du joueur, ce qui lui permettra de chercher à améliorer ses performances. Ces retours sont très importants pour les jeux sérieux, car ils indiquent au joueur s'il réussit ou pas ;
- une interface simple, bien définie, qui supporte les interactions entre le joueur et le jeu (l'affordance) est gage de qualité. Par exemple, elle peut suggérer ou guider les actions de l'utilisateur. Ces ajouts d'informations ne correspondent pas à une scène réelle, mais sont

nécessaires pour maintenir une interaction fluide entre le joueur et le jeu (voir figure 1.12) ;

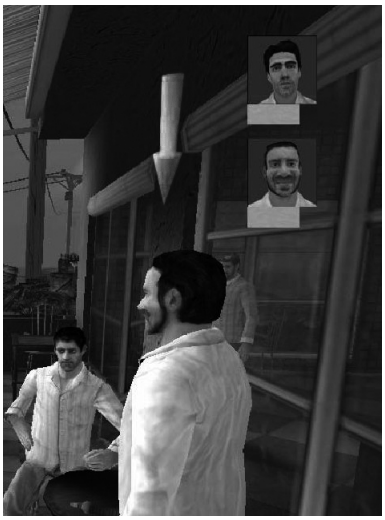

FIG. 1.12 – Exemple d'affordance dans un jeu vidéo.
La flèche au dessus du personnage aide le joueur dans la compréhension de la scène.

- le jeu ne doit pas présenter trop de difficultés. Il faut s'assurer que l'expérience du joueur correspond au niveau du défi à réaliser. S'il y a un trop grand décalage entre les capacités du joueur et la difficulté demandée, le jeu perdra de son intérêt ;
- il est souhaitable de proposer une version allégée du jeu réel où la complexité du *gameplay* est limitée. Ceci permet au joueur de développer ses compétences avant de rencontrer les défis du jeu complet ;
- dans les jeux sérieux modernes, l'utilisation du scénario est fondamentale pour maintenir l'intérêt du joueur et l'encourager à s'identifier au personnage. Il n'est pas nécessaire d'utiliser des périphériques de réalité virtuelle pour immerger un joueur ;
- enfin, un bon jeu doit être ludique. Cet aspect permet de maintenir l'intérêt et une attitude positive du joueur.

Pédagogie

Le jeu sérieux n'est pas seulement un logiciel, une histoire et une interface graphique. Il implique une pédagogie qui éduque et instruit. Cet ajout, sans subordonner l'histoire, rend le jeu sérieux. La dimension pédagogique est donc un point important des jeux sérieux.

Il est avéré que le jeu est fondamental pour l'enfant en ce qui concerne son développement sensorimoteur, cognitif, relationnel, social et affectif. De plus, selon Jean Piaget le jeu évolue avec l'âge :

- les jeux d'exercices (0 à 2 ans) : c'est le jeu de la période sensorimotrice. Le bébé effectue des exercices moteurs réflexes en fonction des informations perçues ;
- les jeux symboliques (2 à 8 ans) : exercent la capacité de l'enfant à représenter une réalité non actuelle c'est-à-dire l'imitation de l'adulte à travers le jeu (jeu de la poupée, de la dînette, du bricolage) ;
- les jeux de règles (6 à 15 ans) : ce type de jeux marque la socialisation de l'enfant. Ces jeux sont le reflet du code social (jeux comportant des règles).

Ces dimensions ludiques sont donc nécessaires pour la construction de l'enfant, mais contrairement aux jeux sérieux, ces types de jeux ne peuvent pas s'appliquer à une formation censée manipuler des concepts. Cependant, il est important de bien étudier comment inclure des jeux et des simulations dans des enseignements en prenant en compte le contexte, l'apprenant, la représentation interne du monde et le processus d'apprentissage (Freitas et al. [dFO06]).

FIG. 1.13 – Passerelle entre l'école et la vie quotidienne.
Mise en scène d'une étudiante confrontée à une difficulté. Grâce à son PDA (Personal Digital Assistant) qui lui sert de pont entre la vie présente et sa formation, elle peut résoudre le problème.

Klopfer et al. [KY05], quant à eux, proposent de nouvelles méthodes de travail à l'aide du jeu et des technologies portables (ordinateurs, téléphones ou PDA (*Personal Digital Assistant*) voir figure 1.13). Ces technologies, largement manipulées et maîtrisées par les jeunes, sont des opportunités qu'il faut apprendre à manipuler en vue de créer des passerelles entre le lieu d'apprentissage et la vie courante. Mettre en place des ponts et ainsi promouvoir le travail à travers le temps et l'espace à l'intérieur et à l'extérieur de la classe est un bon moyen pour que les apprenants puissent profiter de leurs enseignements à chaque instant de leur vie quotidienne. Ces auteurs précisent également que dans le domaine du jeu et de l'apprentissage (supporté par l'expérience du professeur), la recherche a montré que la construction de l'apprentissage par l'expérience aide à la résolution de problèmes.

Une autre vision intéressante du jeu, analysée par Juul [Juu06], laisse au joueur la possibilité de faire des choix. Des jeux comme Sims2 ou GTA sont très populaires, car ils proposent des objectifs où le joueur est libre dans ses prises de décision. Un élément intéressant de Sims2 pointe constamment les difficultés à surmonter par le joueur (fatigue, colère, saleté... voir figure 1.14). Donc, l'approche qui place le joueur dans une dynamique de résolution de problèmes est, semble-t-il, une motivation tout aussi grande que d'atteindre un succès définitif.

Un jeu peut également fonctionner même si aucun but n'est précisé a priori, le jeu n'est qu'un support pour l'imagination du joueur qui se fixe ses propres objectifs en respectant les contraintes techniques imposées par le jeu (Kapla voir figure 1.15[4], LEGO).

1.3.4 Grands domaines d'utilisation

Les jeux sérieux sont présents aujourd'hui dans plusieurs secteurs d'activité comme l'éducation, l'administration, la santé, la défense, les entreprises, la sécurité civile et les sciences. Suivant

[4] `http ://www.kapla.com/`

FIG. 1.14 – Jeu Sims2.
Pointe constamment les problèmes à résoudre par le joueur.

FIG. 1.15 – Exemple d'une construction en Kapla.

le public considéré, le type de jeu (présentation et contenu) va évoluer :

- pour le grand public, les jeux sérieux peuvent être utilisés pour la sensibilisation à des problèmes généraux de santé, de sécurité ou d'environnement. Ces jeux ont pour vocation de faire prendre conscience au joueur de difficultés ou problèmes généraux (voir figure 1) ;
- pour l'université ou au sein de l'entreprise, les jeux sérieux doivent pouvoir fournir un contenu plus complet et précis en fonction du niveau du joueur. L'objectif est la compréhension et l'apprentissage complets de problèmes particuliers pour permettre aux apprenants en fin de formation d'aborder et de résoudre des problèmes complexes ;
- pour des formations plus spécifiques comme le pilotage ou la chirurgie, des jeux sérieux à base d'immersion peuvent permettre des simulations physiquement réalistes avec des modèles mathématiques sous-jacents complets, en vue de préparer au mieux les personnes aux cas critiques.

Le jeu sérieux doit donc être conçu en fonction du secteur d'activité, du public et des moyens disponibles (matériels et financiers) pour sa mise en œuvre. Sue Blackman [Bla05] fait une synthèse sur l'industrie du jeu et ses applications au grand public. Le jeu vidéo devient aujourd'hui de plus en plus populaire ce qui entraîne de gros investissements. Les moyens mis en place permettent le développement d'outils et de bibliothèques facilitant et améliorant la conception des jeux vidéo. Les moteurs graphiques des jeux vidéo, de plus en plus perfectionnés, peuvent même être utilisés pour des application autre que le jeu car ils proposent des rendus temps réels et des gestions de la physique réaliste (voir figure 1.16). Des applications d'entraînement, de visualisa-

tion interactive et de simulation de situation utilisent largement les technologies des jeux vidéo. Pour ces raisons, les jeux aux bases sérieuses et amusantes joueront un rôle important dans un futur proche.

Fig. 1.16 – Utilisation des jeux sérieux dans l'industrie.
Pratt & Whitney, constructeur de moteurs d'avions, utilise les technologies des jeux vidéo pour créer une application de maintenance et de réparation pour leurs moteurs d'avion.

1.3.5 Apprentissage de la programmation

L'informatique est une vaste discipline possédant de nombreuses spécialités. L'apprentissage de la programmation en est une clé essentielle. La difficulté principale de la programmation est d'acquérir les compétences nécessaires afin de comprendre et d'analyser des problèmes. L'étape suivante consiste à exprimer ces résultats à l'aide d'un langage qui permettra à l'ordinateur de réaliser les tâches à effectuer. Cette dernière étape introduit des difficultés supplémentaires liées à la connaissance de la syntaxe du langage choisi.

Pour faciliter cet apprentissage, des logiciels ont été développés. Ces applications ont pour objectif d'aider les étudiants dans leur formation. Tous les logiciels suivants ne sont pas des jeux sérieux, mais ils proposent une façon originale d'aborder la programmation :

- le langage Logo (de Papert) fournit des bases qui permettent une approche constructive pour l'apprentissage dans lequel l'enfant teste personnellement différentes hypothèses par la simulation (Logo a sévèrement souffert d'un manque de puissance de calcul à l'époque de sa sortie). Il a ensuite été étendu par Klopfer et al. [KY05] pour évoluer en StarLogo TNG (*The Next Generation*). StarLogo TNG fournit un système de briques (voir figure 1.17) qui permet de se détacher de la syntaxe et de construire des programmes simples ;
- Scratch (de Resnick) est un nouvel environnement de programmation utilisable par les enfants pour créer leurs propres histoires animées, jeux vidéo ou créations interactives. L'utilisation de Scratch ne nécessite pas de connaissances en langage de programmation, car il utilise une interface graphique permettant l'assemblage de programme sous forme de blocs. L'objectif de Scratch est de laisser l'enfant créer tout en lui faisant apprendre les idées importantes et basiques de la programmation.

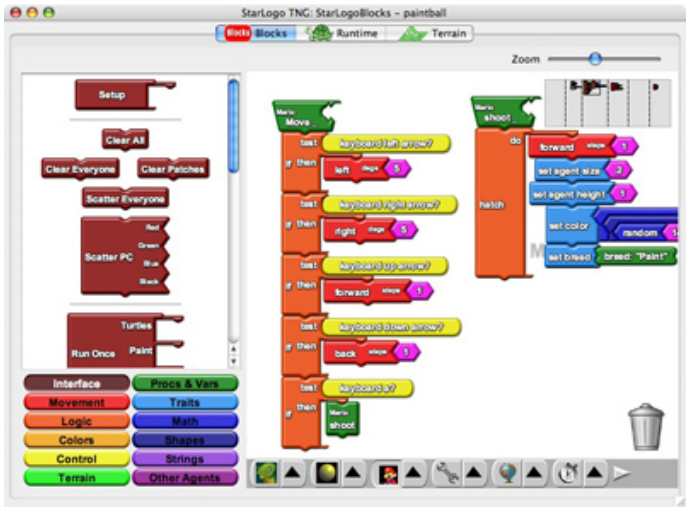

FIG. 1.17 – StarLogo.
Permet la construction de programmes simples à l'aide d'un système de blocs emboîtables.

- le jeu *Robocode*[5] consiste à programmer l'intelligence artificielle de petits robots qui combattent jusqu'à ce qu'il n'en reste plus qu'un. Ce jeu créé et distribué gratuitement par IBM est facile d'utilisation. Un simple robot peut être développé en quelques minutes, cependant les plus sophistiqués ont nécessité plusieurs mois de travail faisant intervenir les statistiques ou les réseaux de neurones. *Robocode* est donc une manière d'aborder ou d'approfondir la programmation Java de façon amusante ;
- la *RoboCup*[6] (voir figure 1.18) est une initiative éducative de recherche internationale. Son but est de regrouper la recherche en intelligence artificielle et en robotique. Cette manifestation intègre et examine une gamme étendue de technologies à travers trois compétitions : la *RoboCupSoccer*, la *RoboCupRescue* et la *RoboCupJunior*. La *RoboCupSoccer* consiste à confronter des robots réels ou simulés dans des matchs de foot. Différentes ligues et modes de jeu ont été imaginés, en vue de permettre la participation d'un maximum de disciplines scientifiques, pour traiter de problèmes sur les environnements dynamiques et la manipulation d'informations incomplètes ou bruitées. La *RoboCupRescue* est une nouvelle compétition mise en place pour apporter plus de diversité. Elle est complémentaire de la *RoboCupSoccer* en proposant des défis dans la logistique et la planification à long terme pour la collaboration d'agents hétérogènes et d'équipes d'agents. Enfin, pour les jeunes, la *RoboCupJunior* est une compétition fortement orientée sur l'éducation et destinée à fournir une introduction excitante aux caractéristiques de la robotique ;
- Cleogo [CB98] est un environnement de travail qui permet à plusieurs personnes de développer simultanément des programmes à l'aide de trois métaphores de programmation : un langage de manipulation directe, un langage iconique et un langage à base de texte standard. Avec Cleogo, les utilisateurs communiquent en vue de s'organiser pour résoudre un problème avec cohérence. Le constat d'une collaboration entre les différents types de programmation est très attendue ;
- Alice2 [KCC+02] est un environnement de programmation destiné à l'enseignement de la programmation à travers un monde virtuel 3D. Un système de programmation par *drag*

[5] `http ://robocode.sourceforge.net/`
[6] `http ://www.robocup.org/Intro.htm`

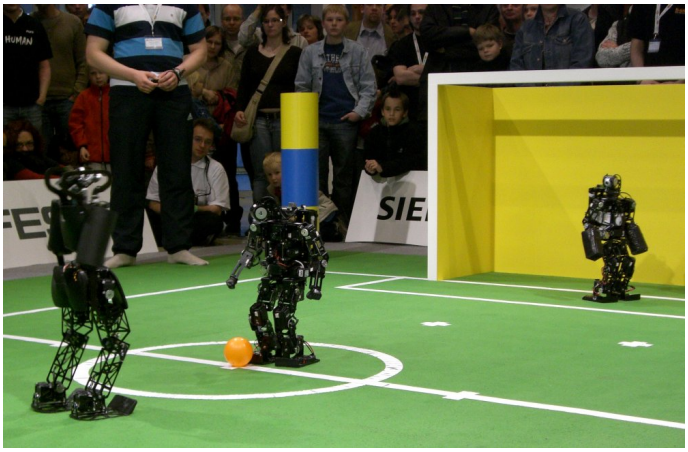

FIG. 1.18 – *RoboCup* en Allemagne (2007).

and drop a été développé. L'utilisateur peut ainsi expérimenter la logique et la structure de programmation sans avoir à se soucier de la syntaxe. Contrairement à d'autres interfaces de programmation sans saisie, il est possible de construire des programmes en Java ou C++ et d'utiliser une forme simplifiée de programmation parallèle. Une expérience informelle semble indiquer qu'une telle interface permet aux novices d'apprendre la logique et les structures de contrôle de la programmation sans être perturbés par la saisie de texte.

1.3.6 Intérêt du massivement multijoueur

Avant d'étudier l'intérêt du massivement multijoueur pour les jeux sérieux, il convient d'expliquer ce terme avec l'aide de Steinkuehler [Ste04]. Un jeu massivement multijoueur en ligne (*Massively Multiplayer Online Game*) ou MMOG est un jeu vidéo de haut niveau en 2D ou 3D joué en ligne. Traditionnellement, un jeu est dit MMOG s'il satisfait les deux conditions suivantes :

- l'accessibilité en ligne et a un très grand nombre de joueurs ;
- la persistance du jeu qu'il y ait des joueurs connectés ou pas.

Différents types de MMOG cohabitent, les MMOFPS (*Massively Multiplayer Online First Person Shooter*), les MMORTS (*Massively Multiplayer Online Real-Time Strategy*) et les MMORPG (*Massively Multiplayer Online Role Playing Game*) qui sont les plus populaires (dont *World of WarCraft* voir figure 1.19). Pour ce dernier type de MMO, les joueurs peuvent communiquer et interagir avec l'environnement grâce à un « avatar » (avatar : entité représentant un joueur dans un jeu vidéo). Des narrateurs structurent le monde pour créer des objectifs et assurer la pérennité du jeu, cependant le joueur est totalement libre de ses actions dans la limite des possibilités du jeu. Ce type de jeu domine actuellement le monde du divertissement grâce à une large diffusion et une popularité grandissante sans distinction de groupe, d'ethnie, de classe économique, mais avec un risque d'addiction pour ceux qui s'y branchent. Quelques applications ont pour but d'utiliser le principe des MMOG à des fins de formation. Les MMOG étant des applications récentes peu d'analyses ont été menées. Il conviendra, donc, dans l'avenir de les analyser plus précisément de façon à avoir une meilleure réflexion sur leurs utilisations.

Un MMOG utilise la plupart des technologies de la réalité virtuelle distribuée et apporte au jeu sérieux la communication, l'interaction, l'échange d'informations avec d'autres joueurs dans un but de reconnaissance au sein d'un groupe et d'apprentissage collectif [DM04]. Tous ces

FIG. 1.19 – Jeu : *World of WarCraft.*
Une grande quantité de joueurs évolue dans un monde persistant.

aspects ne sont pas présents dans un jeu individuel. Pour qu'il y ait communication et socialisation il faut qu'il y ait des temps d'arrêt pour que les joueurs puissent échanger. Les concepteurs doivent donc mettre en place des moyens pour permettre et inciter ces communications.

1.3.7 Quelques résultats

Le sommet annuel des jeux sérieux aux États-Unis[7] et son équivalent en Europe[8] servent de lieu de rassemblement dans ce domaine. Ces initiatives permettent de regrouper les dernières innovations, de diffuser les nouvelles perspectives des jeux sérieux et de présenter les résultats obtenus, afin de légitimer l'utilisation des jeux sérieux. Voici les résultats présentés lors de la conférence de Washington DC aux États-Unis qui nous paraissent les plus intéressants :

- l'expérience conduite avec Alice2 [KS06] dans un lycée et une université a permis aux promotions l'utilisant d'obtenir les meilleurs résultats. En effet, la promotion ayant utilisé Alice2 obtient un B et 88% de ses étudiants ont accédé à la deuxième année. La promotion n'ayant pas utilisé Alice2 obtient un C et 47% des étudiants sont passés au niveau supérieur (voir figure 1.20) ;
- Rick Blunt [Blu06] a analysé trois jeux (*Industry Giant II*, *Zapitalism* et *Virtual U*) et leurs impacts sur les résultats obtenus par les étudiants suivant des formations économiques. Il a étudié cette influence en fonction du sexe, de l'origine ethnique et de l'âge des participants. Il en conclut que dans tous les cas les jeux sérieux apportent un gain significatif à la formation (voir figure 1.21). On notera une exception intéressante pour les personnes de 41 à 50 ans (voir figure 1.22) qui obtiennent de moins bons résultats. Une explication

[7] www.seriousgamessummit.com
[8] www.sgseurope.com/

	CS1 Grade	Take CS2?
No Alice 2 Class Prior to CS1	C	47%
Alice 2 Class Prior to CS1	B	88%

FIG. 1.20 – Alice2 améliore les résultats obtenus en cours de formation.

plausible serait que ces personnes n'aient pu être sensibilisées aux jeux vidéo dès leur plus jeune âge.

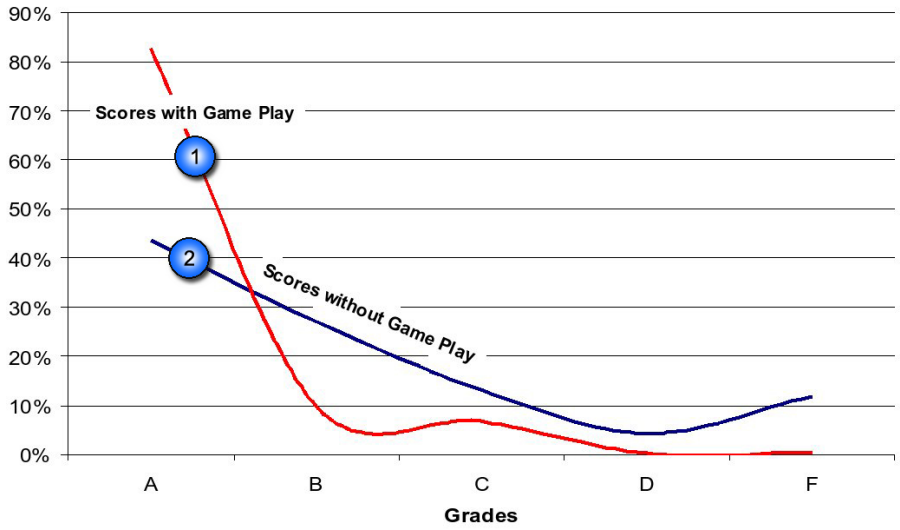

FIG. 1.21 – Utilité du jeu *Zapitalism*.
Avec le jeu (courbe 1) la quasi-totalité des personnes (83%) obtiennent la note maximale A. Sans l'utilisation du jeu (courbe 2), il y a davantage de mauvais résultats (D et F).

1.4 Synthèse de l'état de l'art

Le jeu sérieux permet donc une immersion et une interaction avec un monde virtuel afin de servir de support à une formation où l'aspect ludique permet de motiver le joueur et le maintient dans une dynamique d'apprentissage. De plus son utilisabilité dans la plupart des secteurs d'activités lui donne un avantage certain sur son devenir. Enrichi par la technologie des MMOG, le jeu sérieux peut prendre une place importante et s'imposer comme un complément aux méthodes de formation classique.

Ce nouveau domaine de recherche, tirant ces origines de la réalité virtuelle distribuée, n'en est qu'à ces débuts. Les principaux travaux devant être menés sur les jeux sérieux pour permettre leur essor et leur développement doivent porter, selon Michael Zyda [Zyd05b], sur l'infrastructure, la conception de jeux cognitifs et l'immersion.

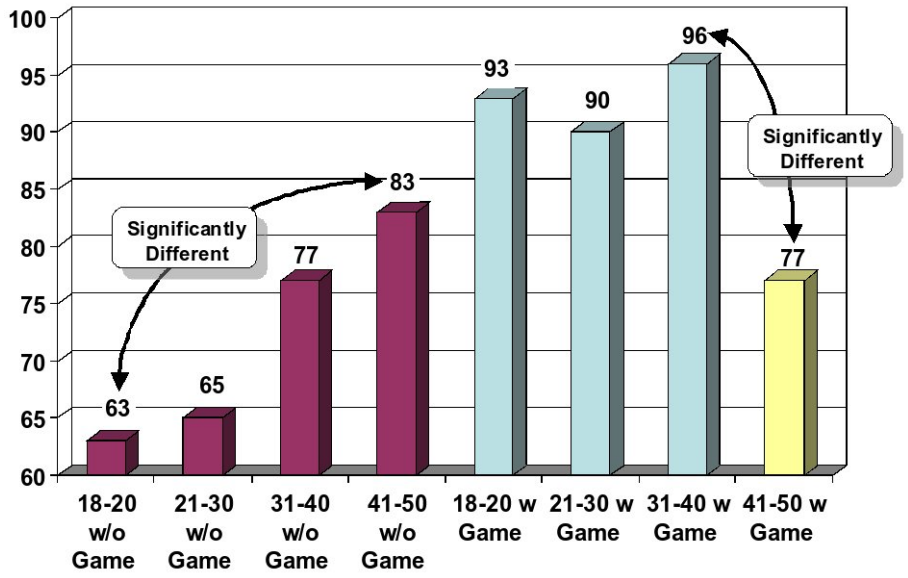

FIG. 1.22 – Utilité du jeu *Virtual U* en fonction de l'âge des participants.
Les scores sont moins bon sans utiliser le jeu (les quatres premiers bâtons) qu'avec le jeu (les trois bâtons suivants) à l'exception de la tranche d'âge 41-50 ans (le dernier bâton).

- l'infrastructure consiste à développer les MMO, les moteurs de jeux et leurs outils, le *streaming* (*streaming* : technique consistant à transmettre des données à un utilisateur et à les lui rendre disponible au fur et à mesure de leurs arrivées pour ainsi lui éviter l'attente d'un téléchargement complet), les consoles de jeux nouvelles générations, le « sans fil » et les technologies mobiles ;
- la conception de jeux cognitifs doit s'appuyer sur la modélisation et la simulation des émotions et des comportements humains. Elle doit également analyser et innover sur des styles et genres de jeux nouveaux. Enfin, l'intégration d'une pédagogie dans l'histoire des jeux sérieux reste un point important à approfondir ;
- l'immersion doit être améliorée à travers le graphisme, le son, l'haptique, mais aussi en utilisant les émotions et l'état du joueur.

De nombreux travaux restent donc à mener dans le domaine des jeux sérieux. Notre contribution s'attachera à améliorer un moteur de jeu afin de lui fournir les capacités d'intégration de programmes informatiques de façon interactive et intuitive. Ce moteur pourra être, par la suite, utilisé comme outil, en vue de créer un nouveau type de jeu sérieux innovant, permettant l'apprentissage de la programmation orientée objet.

2

Création d'un prototype de jeu sérieux pour la programmation orientée objet

La mise au point d'un jeu sérieux pour la programmation orientée objets (POO) nécessite, dans un premier temps, de préciser cette notion. Nous rappellerons donc, en référence à Bertrand Meyer [Mey90], les concepts fondamentaux de ce paradigme de programmation[9]. Ensuite, nous expliquerons le choix du jeu ORTS utilisé comme support pour la suite des travaux. Puis, nous présenterons les principales difficultés du projet et les solutions apportées. Enfin, nous décrirons plus en détail l'intégration de notre application dans ORTS et son fonctionnement à travers un exemple.

2.1 Concepts de programmation orientée objet

L'émergence de la POO vient de la réalisation des contraintes d'un programme, posées lors de la conception du logiciel, telles que la validité, la robustesse, l'extensibilité, la réutilisabilité et la compatibilité. L'élément fondamental est « l'objet ». C'est une structure de données représentant un concept. L'objet possède des attributs (données) qui décrivent sa composition interne et des méthodes qui déterminent son comportement, c'est-à-dire les actions que l'objet est à même de réaliser. En opposition à une structure de données quelconque, il les regroupe - les encapsule - en son sein. L'objet est donc la donnée manipulée par le programme. Il est possible que plusieurs objets différents, mais représentant le même concept (ayant le même type) cohabitent dans un programme. La notion de « classe » intervient donc pour modéliser un ensemble d'objets. En réalité, un objet est une instanciation d'une classe. Il existe, également, des langages objets basés sur le concept de prototype/instance (comme JavaScript) au lieu du concept de classe/instance (comme C++ et Java). Dans ce cas, tout objet peut servir de prototype pour créer de nouveaux objets qui pourront à leur tour être spécialisés (ajout/modification/suppression d'attributs et/ou de méthodes).

La programmation orientée objet repose sur trois principes fondamentaux : l'encapsulation, l'héritage et le polymorphisme.

– encapsulation : c'est l'idée de regrouper les données et les méthodes au sein d'une structure

[9] `paradigme de programmation : manière d'analyser un problème dans le but de concevoir un programme informatique.`

(l'objet). Ce concept permet de garantir l'intégrité et de contrôler la visibilité des données présentes dans un objet. L'implémentation de celui-ci est cachée à l'utilisateur. Dans tous les cas, les membres d'un objet ne sont accessibles qu'à travers eux ;
- héritage : c'est un mécanisme qui permet de définir une classe à partir d'une autre. La classe, ainsi définie, hérite de tous les éléments visibles de la (ou des) classe(s) parente(s) (attributs, méthodes). La nouvelle classe peut être enrichie de nouveaux éléments et modifier le traitement des méthodes héritées. L'héritage est donc un mécanisme intéressant pour réutiliser du code, mais peut également dégrader l'architecture et la maintenance du programme en cas de mauvaise utilisation ;
- polymorphisme : le nom de polymorphisme vient du grec et signifie « qui peut prendre plusieurs formes ». Le polymorphisme est la capacité d'un objet à prendre plusieurs formes, c'est-à-dire à être manipulé en fonction de son type, mais aussi en fonction d'autres types compatibles avec le sien. Ce mécanisme permet, par exemple, de manipuler plusieurs objets de classes différentes mais dérivant d'une même classe de façon identique. En effet, l'héritage permet de redéfinir des méthodes en conservant leur nom et leur signature. Le programme retrouvera dynamiquement la méthode à appeler en fonction du type d'objet sur lequel elle est appliquée et de sa signature.

2.2 Choix d'un moteur de jeu

2.2.1 Précisions sur l'objectif

Notre objectif est de réaliser un prototype permettant d'apprendre la programmation par objet à travers un jeu de stratégie temps-réel. Ce type de jeu a été choisi car il est adapté à l'introduction de nombreux concepts de la programmation par objet. En effet, il est facile d'établir un parallèle entre la notion de classe et le genre d'une unité de combat et ainsi mettre en évidence les concepts fondamentaux de la programmation par objet :
- l'encapsulation sera mise en évidence par les attributs et les méthodes d'une classe qui seront comparables aux caractéristiques et aux comportements d'une unité de combat ;
- pour l'héritage : des unités peuvent hériter des caractéristiques et des comportements d'autres unités et se spécialiser en fonction de leurs spécificités (par exemple, les médecins hériteront des particularités des unités pédestres pour se déplacer au même titre que des soldats, mais posséderont, en plus, la capacité de soigner les autres unités) ;
- concernant le polymorphisme, il permet de gérer uniformément des unités de nature différentes (par exemple, donner le même ordre de déplacement à un tank et à un médecin. Chacun d'eux évoluera à sa manière et en fonction de ses caractéristiques pour atteindre l'objectif).

Dans ce type de jeu, le joueur donne des ordres à ses unités pour qu'elles réalisent des actions (attaquer, défendre...). À l'heure actuelle, ces ordres sont donnés à la souris en cliquant sur une carte. Le but est d'inciter le joueur à programmer ses ordres. Le développement complet d'un jeu de stratégie temps-réel est une tâche trop importante pour la durée d'un stage Master2 recherche. En effet, la conception d'un jeu vidéo mobilise du temps et une logistique importante où différentes équipes de plusieurs personnes collaborent pour concevoir l'univers du jeu, le moteur graphique, les modèles des unités, l'intelligence artificielle, la communication réseau... Nous avons donc recherché un jeu pouvant servir de point de départ, correspondant à nos attentes et susceptible de supporter nos modifications.

Un tel jeu doit nécessairement être ouvert afin d'avoir accès au code. Nous pourrons, ainsi, l'étudier en vue d'y apporter nos modifications. Nous n'avons pas pu travailler avec de grands

classiques tels que *Warcraft III*[10], *Age Of Empires III*[11] ou bien d'autres jeux car ils sont propriétaires et leurs codes ne sont pas disponibles. Quelques projets, cependant, sont en développement avec la volonté d'une diffusion de l'information. Nous avons étudié les deux plus récents orientés multijoueurs et fournissant une interface 3D : *Total Annihilation Spring*[12] (*TA Spring*) et *Open Real-Time Strategy*[13] (ORTS).

2.2.2 *TA Spring* (*Total Annihilation Spring*)

TA Spring (figure 2.1) est un jeu de stratégie temps-réel 3D orienté multijoueur et inspiré du jeu *Total Annihilation*. Ce jeu est toujours en développement mais propose des versions stables et jouables. Il est, graphiquement, le plus abouti avec notamment la présence d'un moteur physique. Le jeu a été conçu dans une optique de modularité en utilisant un système de *mods* qui permet d'utiliser le moteur du jeu pour réaliser sa propre version. Le jeu s'appuie également sur un langage de script « Lua » qui lui permet de définir les missions et les objectifs à atteindre. Cependant, la création de *mods* reste une tâche complexe.

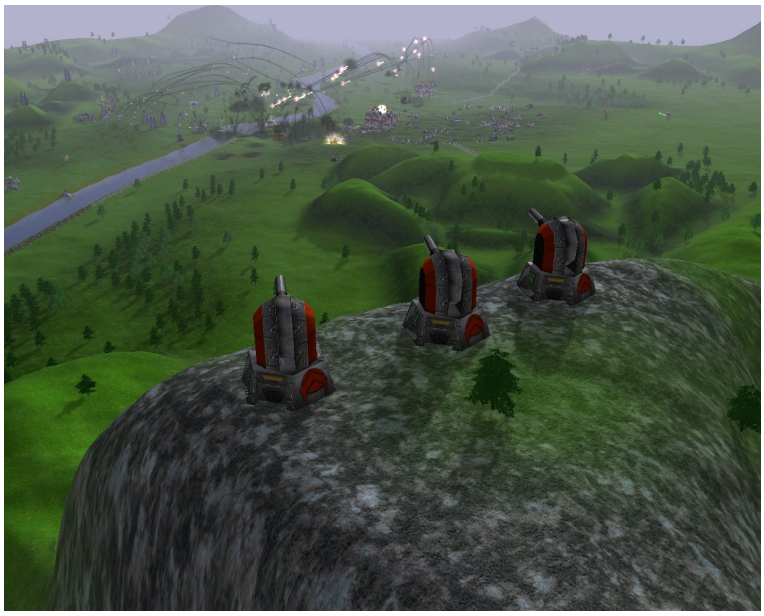

FIG. 2.1 – Visualisation du jeu *TA Spring*.

2.2.3 ORTS (*Open Real-Time Strategy*)

ORTS (figure 2.2) est également un jeu de stratégie temps-réel 3D multijoueur. Ce jeu est développé par Michael Buro [Bur02] et Timothy Furtak [BF05] du département informatique de l'université des sciences d'Alberta (Canada). L'objectif de ce projet est de fournir un environnement de programmation en vue d'étudier des problèmes liés à l'intelligence artificielle (IA). Ce jeu est donc conçu pour permettre à l'utilisateur de programmer et d'intégrer facilement une

[10] `http ://www.blizzard.fr/war3/`
[11] `http ://www.ageofempires3.com/`
[12] `http ://spring.clan-sy.com/`
[13] `http ://www.cs.ualberta.ca/ mburo/orts/`

IA. ORTS est donc bien adapté et correspond mieux à nos attentes. C'est pour cette raison que nous avons choisi ORTS comme point de départ et base de travail de notre projet.

Fig. 2.2 – Visualisation du jeu ORTS.

2.3 Problèmes à résoudre

Bien que ORTS soit prévu pour être extensible, il ne résout pas l'ensemble des problèmes qui nous sont posés, à savoir, intégrer du code dans un programme en fonctionnement et cacher la complexité de l'application au joueur.

2.3.1 Intégrer du code dans un programme en fonctionnement

Un de nos objectifs est de permettre au joueur de saisir du code, de l'intégrer au jeu pour qu'il puisse être exécuté. De cette manière le joueur pourra observer l'exécution de son code à travers le comportement des unités dans le jeu vidéo. Pour l'instant, à chaque modification du code, il est nécessaire d'interrompre l'exécution du programme, de le recompiler et de le relancer pour que les modifications prennent effet. C'est le principe des langages compilés tel que C++ qui est utilisé par ORTS. À ce stade, le premier problème à résoudre est de savoir comment intégrer le code du joueur dans le jeu sans avoir à arrêter ou recompiler tout le jeu. Cette amélioration permettra une plus grande interactivité car le joueur pourra modifier, compiler et intégrer son code sans stopper le jeu et ainsi garder l'avancement et la cohérence de la partie.

2.3.2 Cacher la complexité de l'application au joueur

Le prototype réalisé concerne les joueurs débutants en programmation orientée objet. Leur principale préoccupation est de comprendre les concepts et les principes de cette discipline. Il convient donc, de les aider en leur dissimulant toutes difficultés supplémentaires liées à la complexité du moteur du jeu de façon à ce qu'ils puissent se concentrer au maximum sur leurs objectifs.

2.4 Solutions proposées

Pour répondre aux problèmes posés, nous avons choisi d'utiliser une bibliothèque dynamique et de concevoir une interface pour le développement du code.

2.4.1 Bibliothèque dynamique

Principes

La sémantique du terme bibliothèque dynamique résume bien son utilité. La bibliothèque fournit des fonctions qui peuvent être appelées et exécutées par le programme qui la consulte. Quant à la notion de dynamique, elle indique que la bibliothèque pourra être chargée, utilisée et éliminée pendant l'exécution du programme.

Utilisation

Dans notre application, la bibliothèque contient le code saisi par le joueur et définit le comportement de ses unités. Le jeu utilise donc cette bibliothèque pour déterminer les actions à réaliser. À chaque modification de la bibliothèque, la nouvelle IA est rechargée et, si aucune bibliothèque n'est accessible, alors aucun traitement automatique ne sera exécuté. Grâce à ce principe, le code contenant le comportement des unités est complètement indépendant du jeu. Le joueur peut donc maintenant modifier son code et le recompiler sous forme d'une bibliothèque pour qu'il soit automatiquement intégré au jeu en fonctionnement.

En règle générale, la bibliothèque se suffit à elle même. Pourtant, dans notre application, elle est censée accéder à la boite à outils d'ORTS afin de manipuler les données. Il a donc fallu recompiler ORTS pour permettre à la bibliothèque d'accéder aux éléments du jeu.

Toutefois, lorsque la bibliothèque est chargée, le jeu reste tributaire du temps d'exécution de l'IA. Pour comprendre ce problème, il convient d'étudier un peu plus en détail le fonctionnement d'ORTS. ORTS est un jeu multijoueur basé sur une architecture client-serveur. Le serveur est responsable de la simulation des actions des unités et doit déterminer les connaissances de chaque joueur concernant l'état du monde courant. A chaque cycle, le joueur réalise des actions qui sont envoyées au serveur puis traitées et le résultat est renvoyé au client. Pour maintenir la fluidité du jeu, le client doit maintenir un rythme minimum de huit cycles par seconde, le nombre de cycles étant directement lié au temps passé dans l'exécution de l'IA. Ainsi, il devient impossible de concevoir des IA trop exigeantes en temps de calcul, car les performances du jeu risquent de chuter fortement. L'exécution de la bibliothèque est donc réalisée dans un *thread* (ou processus léger) pour permettre au client de rester actif et apte à réagir aux actions de l'utilisateur ou du serveur. Dorénavant, des IA complexes peuvent être mises en place sans influencer les performances du jeu.

La programmation parallèle introduit des difficultés supplémentaires au niveau de la réalisation. En effet, ce type de programmation demande la mise en place de mécanismes entre les différents processus pour permettre d'assurer la synchronisation et la cohérence des données partagées.

Enfin, une dernière amélioration a été ajoutée pour gérer les erreurs de l'utilisateur. En effet, comme le joueur est un débutant, dans le domaine de la programmation par objet, il est fortement probable qu'il se trompe. Ces erreurs peuvent causer des pannes système ou lever des exceptions. Une fois la bibliothèque chargée par le jeu, si une erreur est générée, elle entraîne une interruption de toute l'application et donc l'arrêt complet du jeu. Cependant, il serait intéressant que les erreurs déclenchées dans la bibliothèque n'aient aucune influence sur le fonctionnement du jeu.

Or, nous avons précisé au paragraphe précédent que la bibliothèque était exécutée dans un *thread*. Cette particularité a été utilisée en mettant en place une gestion des signaux systèmes. Ainsi, lors d'une erreur de programmation seul le *thread* exécutant la bibliothèque est interrompu. Une information est alors donnée à l'utilisateur pour lui indiquer le type d'erreur ayant stoppé l'IA. De cette manière, l'exécution générale du jeu n'est plus tributaire des mauvais fonctionnements de l'IA. Toutefois, si une erreur est générée hors de la bibliothèque cela entraînera l'arrêt complet de l'application.

L'utilisation de la bibliothèque dynamique possède un avantage supplémentaire. Elle dissimule au joueur la complexité du jeu vidéo. Comme nous l'avons précisé précédemment, le joueur doit saisir le code correspondant à l'IA de ses unités. En règle générale, vouloir modifier une partie d'un programme consiste, au préalable, à analyser la structure, l'organisation et le fonctionnement de l'application. Compte tenu de la complexité de cette tâche, les étudiants devront en être dispensés pour se concentrer sur la conception de leurs IA et apprendre les concepts de la programmation orientée objet. La bibliothèque dynamique aide le joueur en extrayant l'IA du jeu. De cette manière, l'utilisateur n'a pas conscience des difficultés liées à l'intégration de son code dans ORTS.

La bibliothèque dynamique concerve le déroulement et la cohérence du jeu en donnant la possibilité au joueur de modifier son code de façon interactive. Elle l'aide dans son travail en lui cachant la complexité du jeu. Cependant, l'utilisation de l'application reste fastidieuse, en raison de l'architecture du jeu et de son système d'arborescence de fichiers. Pour parfaire l'accompagnement du joueur, il convient de rendre l'utilisation du logiciel plus intuitive. Pour cette raison, nous avons conçu une interface appelée le « centre de développement ».

2.4.2 Centre de développement

Utilisation

Le centre de développement est la composante du jeu qui facilite la conception et la manipulation des réalisations du joueur. Il est lancé automatiquement en début de partie pour gérer les différents projets à travers un ensemble de menus et donner l'illusion d'être intégré au jeu. Cependant, le centre de développement est autonome et n'est pas nécessaire au fonctionnement d'ORTS.

Description

Le centre de développement est composé de trois parties (voir figure 2.3) :

- 1 : la barre de menus permet de gérer les projets, les fichiers de chaque projet et l'utilisation de l'IA.
  Le menu « Projet » donne la possibilité de créer, d'ouvrir, de fermer ou de supprimer un projet. Un seul projet peut être ouvert à un instant donné. Dans ce cas, son nom est indiqué dans le titre de la fenêtre (par exemple « chasseur » sur la figure 2.3). Lors de la création d'un projet, deux fichiers sont automatiquement ajoutés au projet (Ia.H et Ia.C). Ils permettent de générer la classe « Ia » qui sert d'interface avec ORTS. L'utilisateur doit alors compléter la méthode « calculeActions » en utilisant ces propres classes pour que son code soit pris en compte automatiquement.
  Le menu « Fichier » autorise la création, l'ouverture, la sauvegarde, la fermeture et la suppression d'un fichier. Les options de ce menu sont actives si un projet est ouvert et gèrent uniquement les fichiers du projet en cours.

Le menu « IA » a pour rôle de lancer ou de stopper l'IA correspondant au projet en cours. Comme pour les fichiers, ces actions sont possibles uniquement si un projet est ouvert ;
- 2 : les zones de saisie sont organisées sous forme d'onglets et présentent le contenu des fichiers ouverts du projet en cours. C'est ici que le joueur peut saisir son code pour améliorer son IA. ;
- 3 : la zone d'information permet de rendre compte des résultats du lancement de l'IA. S'il y a lieu, les erreurs de compilation sont affichées dans ce champ pour donner la possibilité au joueur de corriger ses fautes de syntaxe.

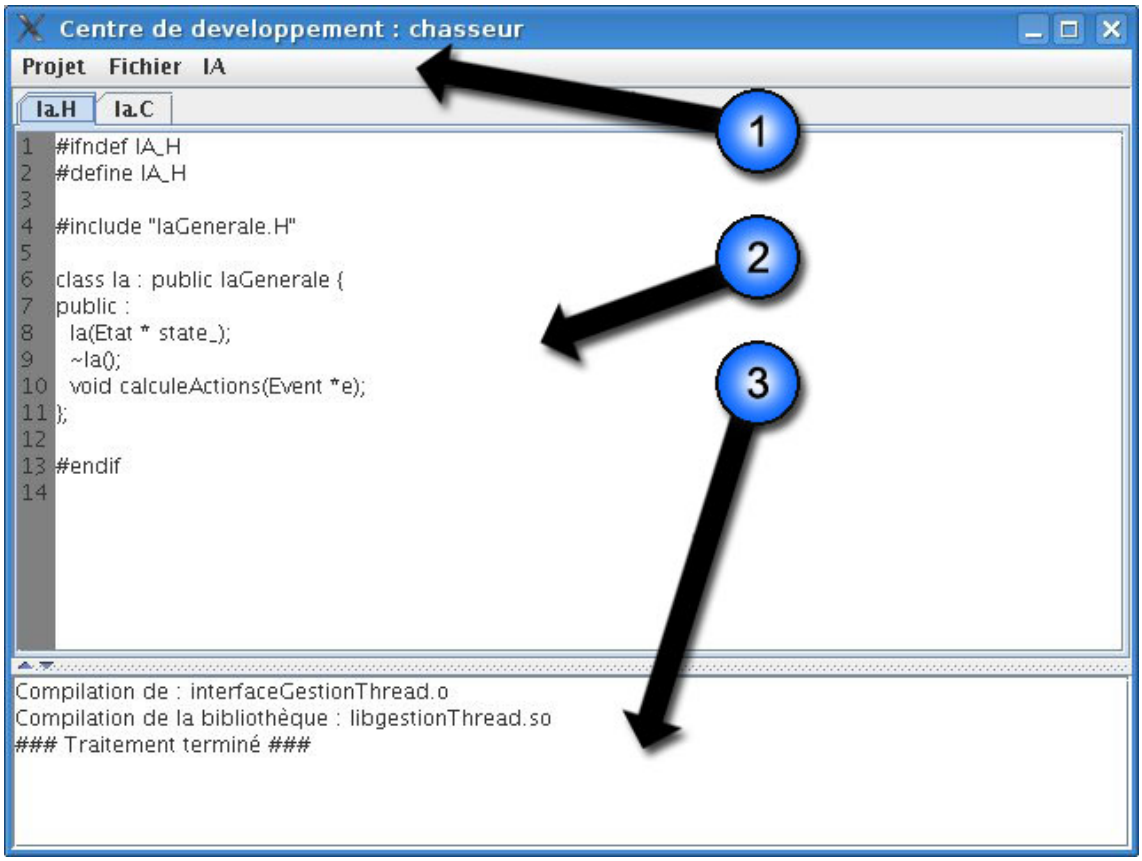

Fig. 2.3 – Différentes parties du centre de développement.
1 - La barre de menus. 2 - Les zones de saisie. 3 - La zone d'information.

2.5 Vue générale

2.5.1 Architecture de l'application

Notre application est composée de trois entités (voir figure 2.4) : ORTS, la bibliothèque dynamique et le centre de développement. Pour rendre notre logiciel fonctionnel, nous avons dû modifier ORTS en y ajoutant une structure représentant l'état du jeu (« Etat »), un outil de synchronisation (« Moniteur ») et une classe chargée de gérer la bibliothèque (« ChargeurIA »). Nous détaillerons leur utilité dans la suite de ce paragraphe. La bibliothèque dynamique est composée d'une interface permettant son utilisation (« interfaceGestionThread ») et d'un *thread*

(pour plus de détails sur les raisons de l'utilisation du *thread*, voir paragraphe 2.4.1) qui exécute le code du joueur.

« ChargeurIA » a pour but de gérer la bibliothèque dynamique, il l'utilise si elle existe et peut la recharger si elle a été modifiée. Il pilote également le *thread* à travers « interfaceGestionThread » pour maintenir la même version entre la bibliothèque et le *thread*.

« Moniteur » donne les moyens au *thread* et à « ChargeurIA » de se synchroniser de façon à respecter l'intégrité et la cohérence du déroulement du jeu.

Le code du joueur, exécuté dans le *thread*, peut accéder aux informations du jeu à travers « Etat ». Cette structure sert de point d'entrée sur les données d'ORTS.

Enfin, le centre de développement permet au joueur de modifier le code exécuté dans le *thread* et de (re)générer la bibliothèque.

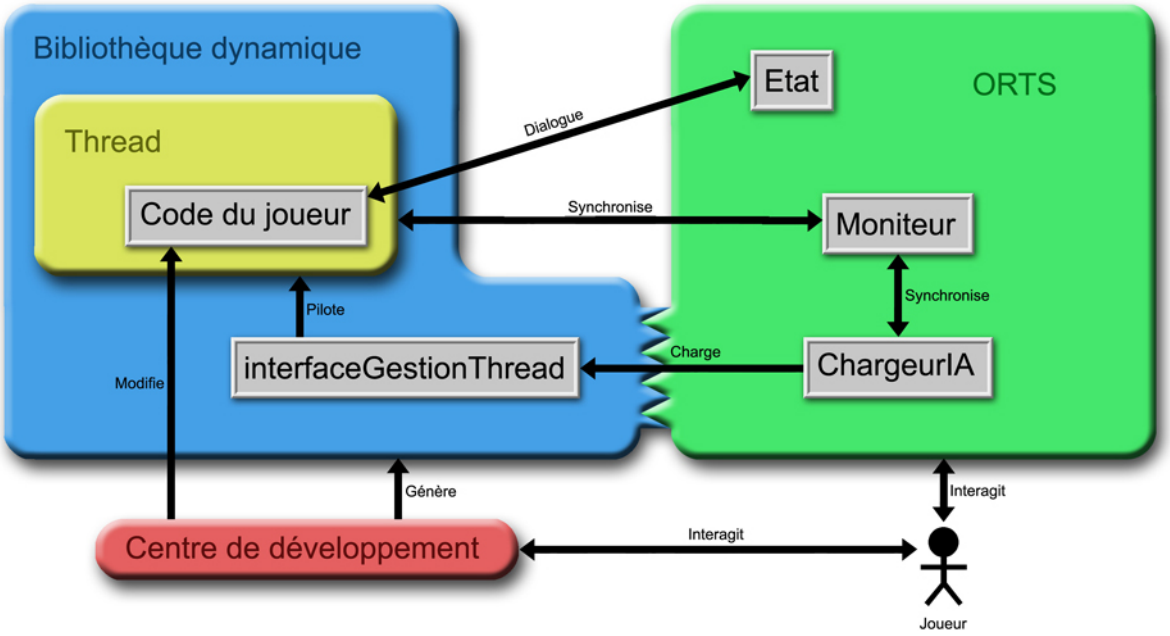

FIG. 2.4 – Architecture de notre application.

2.5.2 Utilisation

Afin de gérer les différents modules d'ORTS, des options peuvent être renseignées lors du lancement de l'application (elles peuvent être combinées) :

- orts_programming : exécute le jeu ;
- orts_programming -nogfx : exécute le jeu sans l'affichage graphique ;
- orts_programming -nocdd : exécute le jeu sans le centre de développement ;
- orts_programming -justcdd : exécute le centre de développement sans lancer le jeu.

2.5.3 Illustration

Pour illustrer le fonctionnement de notre application, nous avons créé un programme permettant à des soldats de suivre leurs adversaires (voir figure 2.5 pour l'illustration et annexe 1 pour le code). Lorsque le jeu démarre, toutes les unités se déplacent aléatoirement. L'utilisateur

modifie alors le code, pour permettre aux soldats de choisir une cible à suivre, puis il lance la génération de l'IA. Lorsque la compilation se termine avec succès, le code est pris en compte et les soldats contrôlés suivent les adversaires qui passent à leur portée.

2.6 Synthèse des travaux et perspectives

La première phase de notre travail a consisté à prospecter différents jeux afin de les étudier et de sélectionner le plus adapté à notre problématique. Cette recherche nous a permis de nous concentrer, plus rapidement, sur les objectifs à atteindre.
Nous avons donc amélioré le moteur d'ORTS pour lui permettre d'intégrer du code pendant son exécution. ORTS est devenu un outil ludique et interactif permettant la conception de programmes orientés objet. Cependant, le jeu sérieux, à proprement dit, n'existe pas encore. Il reste à mettre en place la partie scénario qui poussera le joueur à s'investir dans le jeu pour apprendre la programmation orientée objet.
ORTS est un jeu multijoueur, cependant, il serait intéressant de le porter vers un système MMOG de façon à pouvoir profiter pleinement des avantages de la réalité virtuelle distribuée. Il serait alors possible à plusieurs utilisateurs de partager leurs expériences dans un monde virtuel persistant dans un souci de formation collective. De plus le domaine des MMORTS a peu été développé, laissant des perspectives de recherches ouvertes et passionnantes.

Le centre de développement permet une utilisation plus intuitive du logiciel, pourtant il pourrait être amélioré de façon à faciliter l'interaction avec l'environnement virtuel. Des fonctionnalités sont à ajouter comme la possibilité d'annuler la dernière action, de colorer le code pour identifier plus facilement la syntaxe. Il serait également intéressant de mettre en place un système optionnel de saisie par *drag and drop* en utilisant des briques comme dans Alice2 ou StarLogo TNG. Ceci aiderait les débutants en les soulageant de la syntaxe du C++. Enfin, le centre de développement est aujourd'hui séparé d'ORTS, il serait intéressant de les fusionner afin d'obtenir une seule entité représentant le jeu dans sa globalité.

Toutes ces optimisations permettraient à ORTS de devenir un jeu sérieux massivement multijoueur au sens littéral du terme. Il pourrait ainsi servir d'initiation à la programmation orientée objet tout en permettant des perspectives de jeu dont la seule limite serait l'imagination des joueurs.

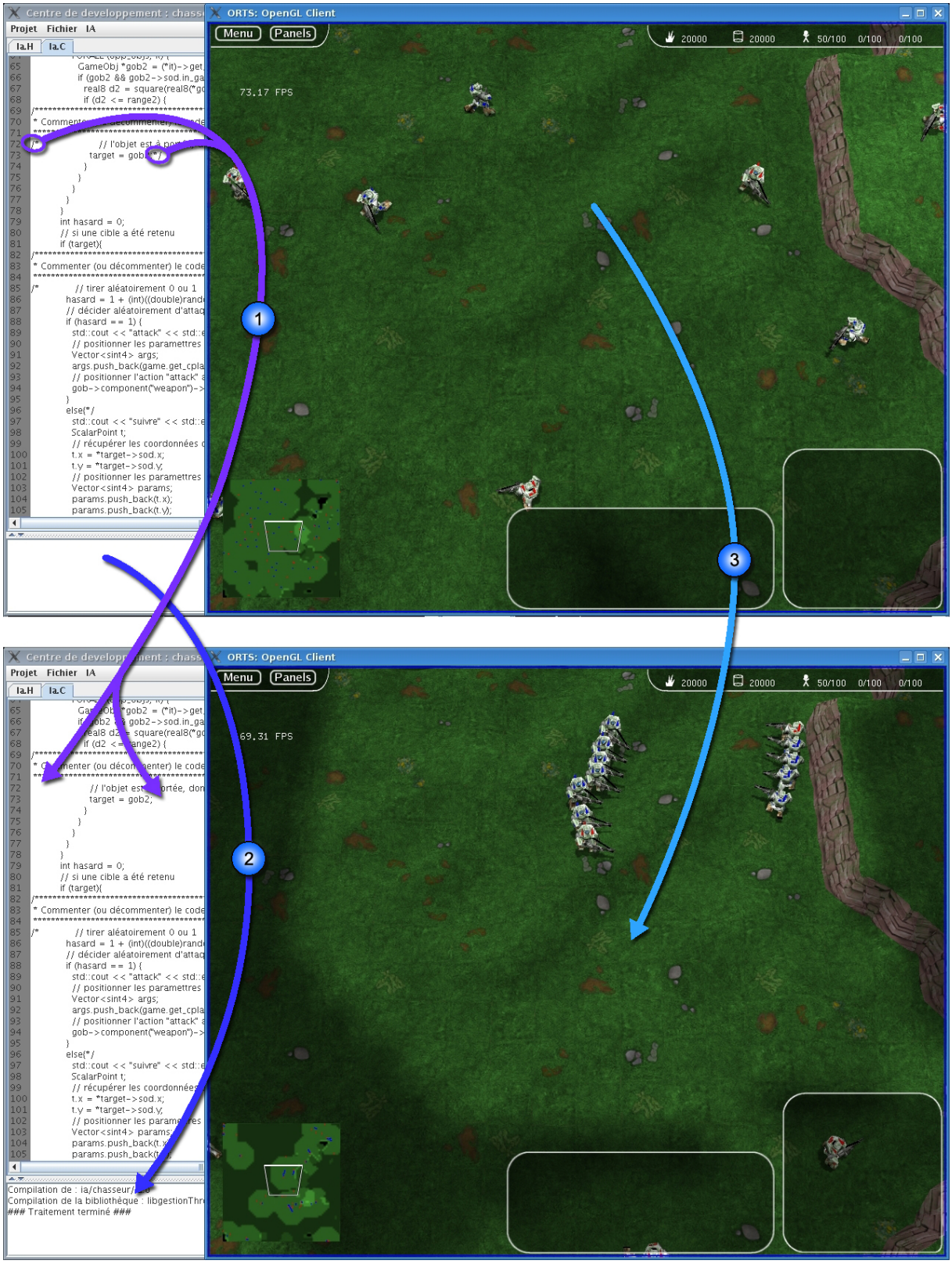

FIG. 2.5 – Exemple d'utilisation de notre application.
1 - Modification du code. 2 - Compilation réussie. 3 - Le code est pris en compte et permet aux unités bleues de suivre des unités rouges passant à leur portée.

Conclusion

L'étude approfondie de l'état de l'art met en évidence un continuum logique entre la réalité virtuelle, la réalité virtuelle distribuée et les jeux sérieux couplés aux technologies MMO. Les jeux sérieux MMO sont alors posés comme une application pertinente de la réalité virtuelle distribuée. Ils simulent des environnements virtuels interactifs et immergent de nombreuses personnes interagissant sur ce monde synthétique.

Nous avons également défini le jeu sérieux et étudié ses concepts fondamentaux, à savoir l'importance de la dimension ludique et éducative. Ces nouveaux logiciels se placent en complément des méthodes de formation classique. Ils proposent d'aborder, de façon originale, des sujets pouvant couvrir la majorité des secteurs d'activités.

Enfin, nous avons créé un prototype de jeu sérieux permettant la manipulation de la programmation orientée objet à travers un jeu de stratégie temps-réel. Ce prototype est un jeu de stratégie classique, fournissant la fonctionnalité supplémentaire d'ajouter du code contrôlant les unités du jeu. Ce logiciel permet, contrairement à d'autres applications permettant l'apprentissage de la programmation, de manipuler interactivement un langage complet et très largement utilisé : le C++. Les programmes, ainsi conçus, sont automatiquement intégrés au jeu, sans interruption de la partie. Ce prototype permet donc aux joueurs novices ou confirmés de se divertir tout en développant des programmes informatiques.

Annexe

1 Exemple de codage d'une IA

Le code suivant est contenu dans la méthode « calculeActions » de la classe « Ia » et permet aux unités militaires du joueur de se déplacer aléatoirement sur le terrain. Si le joueur décommente les parties indiquées dans le code, alors, ses unités pourront déterminer une cible et donc suivre ou attaquer les adversaires passant à leur portée. On notera que la longueur et/ou la complexité du code pourraient être améliorées en écrivant une interface simplifiée d'ORTS.

```
void Ia : :calculeActions(Event *e){
  // vérifier que le message vienne bien du GameStateModule
  if (e->get_who() == GameStateModule : :FROM) {
    // si c'est un message de STOP : sortir
    if (e->get_what() == GameStateModule : :STOP_MSG) exit (0) ;
    // si c'est un message d'erreur : sortir
    if (e->get_what() == GameStateModule : :READ_ERROR_MSG) exit (10) ;
    // si c'est une nouvelle vue du jeu, alors calculer le traitement
    if (e->get_what() == GameStateModule : :VIEW_MSG) {
      // récupèrer le jeu
      const Game &game = state->gsm->get_game() ;
      // récupérer l'identifiant du joueur
      const sint4 cid = game.get_client_player() ;
      // récupérer la carte
      const Map<GameTile> &map = game.get_map() ;
      // récupérer la liste des objets appartenant au joueur
      const Game : :ObjCont &objs = game.get_objs(cid) ;
      // récupérer les dimensions de la carte
      const sint4 points_x = map.get_width() * game.get_tile_points() ;
      const sint4 points_y = map.get_height() * game.get_tile_points() ;
      // initialisation du générateur de nombres pseudo-aléatoires
      srand (time(NULL)) ;
      // pour chaque objet appartenant au joueur
      FORALL (objs, it) {
        // récupérer l'objet
        GameObj *gob = (*it)->get_GameObj() ;
        // vérifier qu'il soit dans la partie et vivant
        if (gob && gob->sod.in_game && *gob->sod.x >= 0 && !gob->is_dead()) {
          // récupérer l'arme de l'objet en cours
```

```
            ScriptObj *weapon = gob->component("weapon");
            if (!weapon) continue;
            // récupérer le nombre de joueurs
            const sint4 n = game.get_player_num();
            GameObj *target = 0;
            // s'il y en a au moins un autre que nous alors rechercher une cible
            if (n > 1) {
               // récupérer la portée maximale au sol de l'arme
               sintptr *prange = weapon->get_int_ptr("max_ground_range");
               if (!prange) continue;
               const sint4 range2 = *prange * *prange;
               // pour chaque joueur
               FORS (i, n) {
                  if (i == cid) continue;
                  // récupérer les objets de ce joueur
                  const Game : :ObjCont &opp_objs = game.get_objs(i);
                  // pour chacun de ses objets vérifier s'il y en a un à la portée
                  FORALL (opp_objs, it) {
                     GameObj *gob2 = (*it)->get_GameObj();
                     if (gob2 && gob2->sod.in_game && *gob2->sod.x >= 0 && !gob2->is_dead()) {
                        real8 d2 = square(real8(*gob->sod.x - *gob2->sod.x)) +
                                    square(real8(*gob->sod.y - *gob2->sod.y));
                        if (d2 <= range2) {
/*****************************************************************************
 Commenter (ou décommenter) le code suivant pour que l'objet ne retienne pas (ou retienne) la cible
 *****************************************************************************/
/*                      // l'objet est à portée, donc on le sauvegarde
                        target = gob2;*/
                        }
                     }
                  }
               }
            }
            int hasard = 0;
            // si une cible a été retenue
            if (target){
/*****************************************************************************
 Commenter (ou décommenter) le code suivant pour que l'objet n'attaque pas (ou attaque) la cible
 *****************************************************************************/
/*          // tirer aléatoirement 0 ou 1
            hasard = 1 + (int)((double)rand() / ((double)RAND_MAX + 1) * 3);
            // décider aléatoirement d'attaquer ou de suivre la cible
            if (hasard == 1) {
               std : :cout « "attack" « std : :endl;
               // positionner les paramètres de l'attaque
               Vector<sint4> args;
               args.push_back(game.get_cplayer_info().get_id(target));
               // positionner l'action "attack" avec ses paramètres pour l'arme en question
```

```
                gob->component("weapon")->set_action("attack", args);
            }
            else{ */
                std::cout « "suivre" « std::endl;
                ScalarPoint t;
                // récupérer les coordonnées de la cible à suivre
                t.x = *target->sod.x;
                t.y = *target->sod.y;
                // positionner les paramètres du déplacement
                Vector<sint4> params;
                params.push_back(t.x);
                params.push_back(t.y);
                params.push_back(*gob->sod.max_speed);
                // positionner l'action "move" avec ses paramètres pour l'objet en question
                gob->set_action("move", params);
//              }
        }
        else {
            // aucune cible n'est définie, donc on marche aléatoirement
            std::cout « "chercher" « std::endl;
            // si l'objet est arrêté alors définir une nouvelle direction aléatoire
            if (*gob->sod.speed == 0) {
                ScalarPoint t;
                // tire un nombre entre 0 et 100
                hasard = (int)((double)rand() / ((double)RAND_MAX + 1) * 100);
                if (hasard < 33) {
                    // cible aléatoire sur la limite du terrain (x ou y inchangé)
                    hasard = (int)((double)rand() / ((double)RAND_MAX + 1) * 100);
                    switch (hasard & 3) {
                        case 0 : t.x = *gob->sod.x; t.y = 0; break;
                        case 1 : t.x = *gob->sod.x; t.y = points_y-1; break;
                        case 2 : t.x = 0; t.y = *gob->sod.y; break;
                        case 3 : t.x = points_x-1; t.y = *gob->sod.y; break;
                    }
                }
                else {
                    // cible aléatoire sur la limite du terrain
                    hasard = (int)((double)rand() / ((double)RAND_MAX + 1) * 100);
                    switch (hasard & 3) {
                        case 0 :
                            t.x = 0;
                            t.y = (int)((double)rand() / ((double)RAND_MAX + 1) * points_y);
                            break;
                        case 1 :
                            t.x = points_x - 1;
                            t.y = (int)((double)rand() / ((double)RAND_MAX + 1) * points_y);
                            break;
                        case 2 :
```

```
                    t.x = (int)((double)rand() / ((double)RAND_MAX + 1) * points_x) ;
                    t.y = 0 ;
                    break ;
                case 3 :
                    t.x = (int)((double)rand() / ((double)RAND_MAX + 1) * points_x) ;
                    t.y = points_y - 1 ;
                    break ;
            }
        }
        // positionner les paramètres du déplacement
        Vector<sint4> params ;
        params.push_back(t.x) ;
        params.push_back(t.y) ;
        params.push_back(*gob->sod.max_speed) ;
        // positionner l'action "move" avec ses paramètres pour l'objet en question
        gob->set_action("move", params) ;
    }
  }
 }
  }
   }
    }
}
```

Bibliographie

[BF05] M. Buro and T. Furtak. On the development of a free rts game engine, March 2005. GameOn'NA Conference.

[Bla05] S. Blackman. Serious games...and less ! *SIGGRAPH Comput. Graph.*, 39(1) :12–16, 2005.

[Blu06] R. Blunt. Game-Based Learning Works : Teaching Business Courses With Video Games. In *Serious Games Summit D.C.*, Washington, DC, USA, oct 2006.

[Bur02] M. Buro. Orts : A hack-free rts game environment. In *Computers and Games*, pages 280–291, 2002.

[CB98] A. Cockburn and A. Bryant. Cleogo : Collaborative and Multi-Metaphor Programming for Kids. In *APCHI '98 : Proceedings of the Third Asian Pacific Computer and Human Interaction*, page 189, Washington, DC, USA, 1998. IEEE Computer Society.

[Cho06] F. Chobeaux. Jeu et éducation nouvelle, 2006. Vers l'éducation nouvelle. http ://www.cemea.asso.fr/IMG/VENtextejeux.pdf.

[dFO06] S. de Freitas and M. Oliver. How can exploratory learning with games and simulations within the curriculum be most effectively evaluated ? *Comput. Educ.*, 46(3) :249–264, 2006.

[DM04] N. Ducheneaut and R. J. Moore. The social side of gaming : a study of interaction patterns in a massively multiplayer online game. In *CSCW '04 : Proceedings of the 2004 ACM conference on Computer supported cooperative work*, pages 360–369, New York, NY, USA, 2004. ACM Press.

[DZ03] T. N. B. Duong and S. Zhou. A Dynamic Load Sharing Algorithm for Massively Multiplayer Online Games. In *Networks, 2003. ICON2003. The 11th IEEE International Conference on*, pages 131–136, 2003.

[FMT06] P. Fuchs, G. Moreau, and J. Tisseau. *Traité de la réalité virtuelle, Tome III. Chapitre 1 - Introduction.*, chapter 1. Presses de l'École des Mines de Paris, 2006.

[JT06] J.P. Jessel and P Torguet. *Traité de la réalité virtuelle, Tome III. Chapitre 16 - La réalité virtuelle distribuée.*, chapter 16. Presses de l'École des Mines de Paris, 2006.

[Juu06] J. Juul. A New Kind Of Game, 2006. GDC - Serious Games Summit Keynote - http ://www.gamasutra.com/features/20060321/carless_01.shtml.

[JVM05] W. Lewis Johnson, H. Vilhjalmsson, and S. Marsella. Serious Games for Language Learning : How Much Game, How Much AI ? In *12th International Conference on Artificial Intelligence in Education (AIED)*, Amsterdam, The Netherlands, july 2005.

[KCC+02] C. Kelleher, D. Cosgrove, D. Culyba, C. Forlines, J. Pratt, and R. Pausch. Alice2 : Programming without Syntax Errors, 2002. UIST.

[KLXH04] B. Knutsson, H. Lu, W. Xu, and B. Hopkins. Peer-to-peer support for massively multiplayer games, March 2004. IEEE Infocom.

[KS06] C. Kelleher and S. Seabolt. Carnegie Mellon's Alice 3.0 Plus EA's Sims : A Unique Industry/Academia Collaboration. In *Serious Games Summit D.C.*, Washington, DC, USA, oct 2006.

[KY05] E. Klopfer and S. Yoon. Developing games and simulations for today and tomorrow's tech savvy youth. *Tech Trends*, 49(3) :33–41, 2005.

[Mey90] B. Meyer. *Conception et programmation par objet.* InterEditions, 1990.

[Ste04] C. A. Steinkuehler. Learning in massively multiplayer online games. In *ICLS '04 : Proceedings of the 6th international conference on Learning sciences*, pages 521–528. International Society of the Learning Sciences, 2004.

[Zyd05a] M. Zyda. Creating a science of games. In *Simulation Interoperability Workshop*, Orlando, Florida, sep 2005.

[Zyd05b] Michael Zyda. From visual simulation to virtual reality to games. *Computer*, 38(9) :25–32, 2005.

Résumé

Dans ce document, nous présentons le lien entre réalité virtuelle, réalité virtuelle distribuée et jeu sérieux massivement multijoueur « en ligne ». Nous proposons, également, un prototype de jeu sérieux permettant l'apprentissage de la programmation orientée objet à travers un jeu de stratégie temps-réel. Cet outil permet au joueur de contrôler ses unités militaires par programmation. Le code, ainsi saisi, est intégré dynamiquement dans l'application en cours d'exécution en respectant la cohérence et la fluidité du jeu. De plus, un effort a été mené pour clarifier et simplifier l'utilisation de notre logiciel afin d'aider le joueur durant la partie.

Mots-clés: Réalité virtuelle, réalité virtuelle distribuée, jeu sérieux, jeu massivement multijoueur « en ligne », jeu de stratégie temps-réel, programmation orientée objet, intégration dynamique de code.

Abstract

In this document, we aimed to show the link between virtual reality, distributed virtual reality and massively multiplayer online serious games. We have also designed and implemented a serious game prototype allowing object-oriented programming training through a real-time strategy game. This tool allows the player to control his military units using programming. Once the code written, it is dynamically integrated into the running application while assuring the game data consistency and fluidity. Besides, we aimed to clarify and simplify our software usability as best as possible in order to help the player during a game run.

Keywords: Virtual reality, distributed virtual reality, serious game, massively multiplayer online game, real-time strategy game, object-oriented programming, dynamic integration of code.