HAUTE ECOLE de la Communauté française en HAINAUT

Catégorie TECHNIQUE
av V. Maistriau, 8a B-7000 MONS
. : +32[0]65 338154 - Fax : +32[0]65 313051
iel : secretariatisims@hecfh.be - www.isims.be

Travail de fin d'études présenté en vue de l'obtention du titre de Bachelier en Informatique et Systèmes

Réalisation d'un jeu de type « tower defence » en 3D

MONUN Emre

HAUTE ECOLE de la Communauté française en HAINAUT

Travail de fin d'études présenté en vue de l'obtention du titre de Bachelier en Informatique et Systèmes

Réalisation d'un jeu de type « tower defence » en 3D

MONUN Emre

Je remercie tous les enseignants de l'Institut Supérieur Industriel de Mons d'avoir partagé leurs savoirs et savoir-faire qui ont contribué à l'aboutissement de ce projet de fin d'études.

Table des matières

1. Introduction

Comme projet de vie professionnelle, je me vois bien devenir programmeur de jeux vidéo. Ce travail représente pour moi le point de départ de mon futur métier.

Il a pour but de mettre en place une application de jeux vidéo 3D.

La connexion sera assurée par le protocole de la couche réseau selon le modèle OSI.

Le langage utilisé pour la partie client est le C#. J'utiliserai le Framework XNA.

La partie serveur sera codée quant à elle en Java et disposera uniquement d'une interface console.

2. Tower Defence

Un “tower defence” est un jeu dont le but est de défendre une zone dans laquelle se déploient des vagues d’ennemis. La défense est assurée par des tours placées par le joueur, d’où le nom “Tower Defence”.

L’origine de ce jeu est presque inconnue, il est apparu sous forme de modification dans différents jeux possédant des fonctions d’édition de map.
Le jeu que j’ai créé est composé de 2 modes :

- Survival, une série de vagues est envoyée par intervalle, tant que le joueur dispose encore des vies.
  - Ce mode est jouable à 1 joueur, 2 joueurs ou 3 joueurs.
- Wave, un nombre de vagues déterminé par le joueur est envoyé par intervalle.
  - Ce mode est un affrontement de 1 contre 1 joueur, 2 contre 2 joueurs, ou 3 contre 3 joueurs.

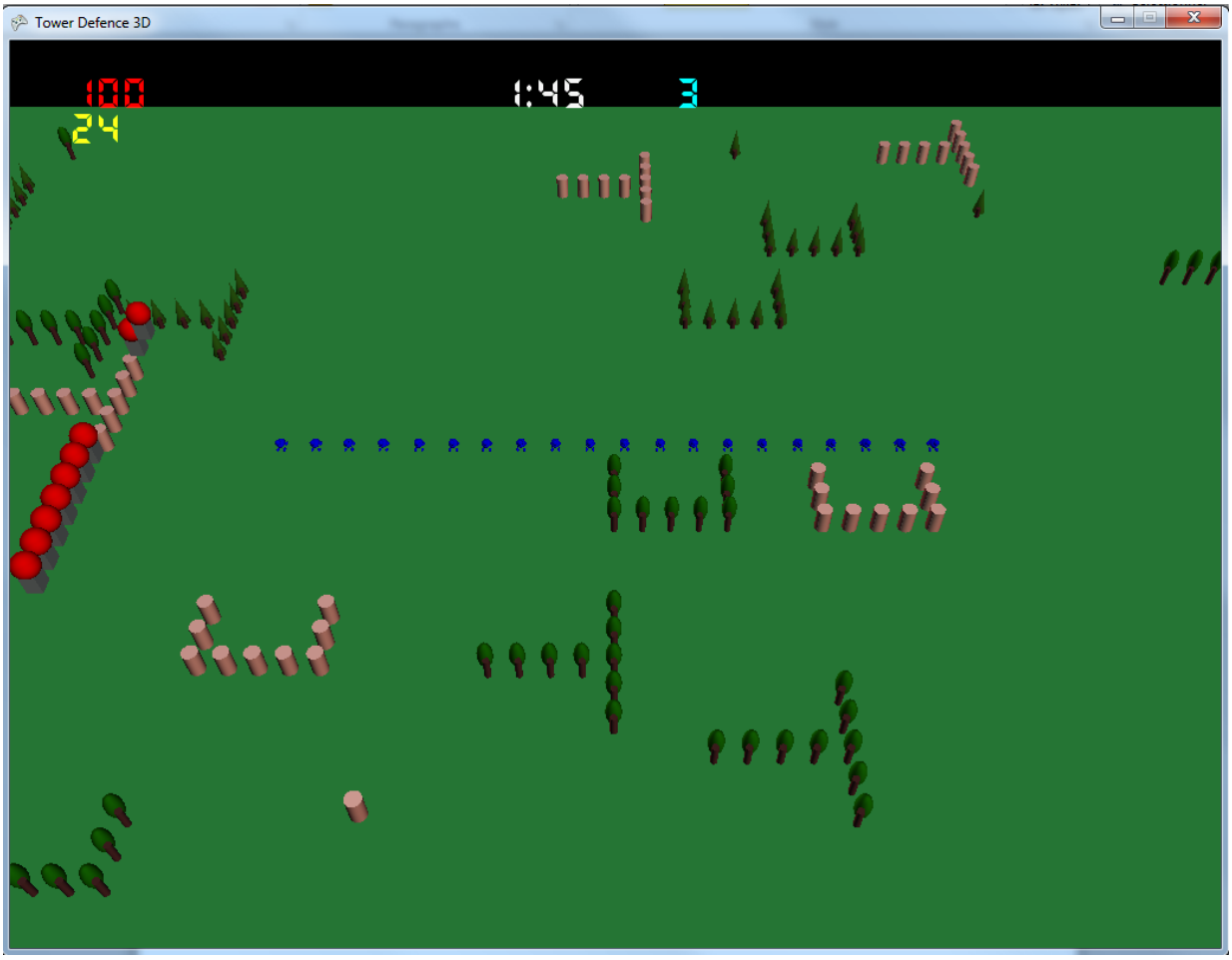

Figure 1 : Illustration Tower Defence réaliser.

La figure 1 montre la vague de monstres (représentés par une suite de petits objets à l’horizontale en bleu) et les tours prêtes à tuer les monstres (en rouge).

*Map : carte du jeu

3. Programmation application 3D

La mise en place d'une application 3D est possible en utilisant plusieurs méthodes.
Deux méthodes se démarquent particulièrement :

- Programmation en code natif ;
- Outils.

La sélection du langage peut dépendre aussi de la plate-forme cible : Windows, GNU/Linux, OS X, console, smartphone, ...
J'ai choisi un langage qui me permet d'exécuter l'application sur les systèmes d'exploitation Windows étant le plus répandu.

3.1 Programmation en code natif

La programmation en code natif requiert une connaissance avancée du langage. Néanmoins, celui-ci devra être capable d'intégrer les bibliothèques DirectX ou OpenGL, sans lesquelles il est difficile de communiquer avec la carte graphique.
Le langage de programmation le plus exploité pour la réalisation d'application 3D est le C++.
Pour certains langages, la programmation en code natif sollicite une connaissance approfondie des adresses de la carte graphique.

3.2 Outils

Il existe cependant des « outils » permettant de réaliser des applications 3D.
On entend parler de « programmeur/producteur », car tous ces "outils" ne nécessitent pas une connaissance poussée en programmation.
Ces "outils" permettent de diminuer la tâche du programmeur, en passant par des méthodes déjà existantes.
L'outil Framework XNA est celui sélectionné pour ce projet.

3.3 DirectX & OpenGL

DirectX et OpenGL sont des bibliothèques multimédias, permettant de faire la liaison entre l'application et l'interface de sortie.
OpenGL est multiplateforme, tandis que DirectX est uniquement utilisé sur des systèmes Microsoft.

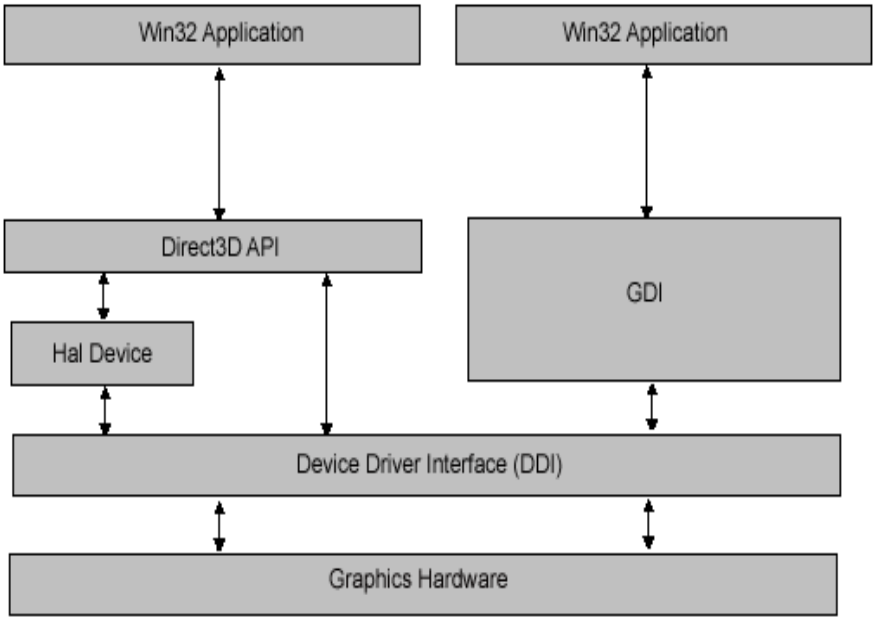

Figure 2: Architecture simplifiée des communications entre applications et interface graphique.

DirectX est la bibliothèque utilisée dans le cadre de ce projet.

4. Framework XNA

Xbox Next-Generation Architecture (XNA) a été créé par Microsoft afin de gérer les applications dans la console de salon Xbox 360 et le lecteur multimédia Zune.
Elle utilise la bibliothèque .NET Framework et reconnait DirectX par défaut.
Avec la plateforme Mono-project, il est possible d'utiliser la bibliothèque OpenGL.
Le langage de programmation utilisé pour XNA est le C#.
Le premier bêta est rendu public gratuitement le 30 aout 2006. Par contre, pour rendre publique son application, il est nécessaire de s'inscrire sur le site "Microsoft Creation's Club" et de payer la somme de 99 $ par an.

4.1 Structure de fonctionnement

Les méthodes :
Une méthode est une fonction pouvant retourner une valeur ou non, qui traite des instructions définies.

La méthode LoadContent :
Permet de charger toutes les données multimédias (son, Mesh 3d, image) dans la mémoire.
La méthode UnloadContent
Permet de vider la mémoire.
La méthode Update :
Relue en boucle tant que le jeu est en fonction, elle permet de vérifier l'état des variables et ainsi effectuer des modifications.
La méthode Draw :
Comme la méthode Update, elle est relue en boucle, mais est utilisée uniquement pour dessiner des objets de jeu.

4.2 La variable GraphicsDevice

La variable GraphicsDevice est la variable associée à la carte graphique et ne peut être instanciée qu'une seule fois. (Instancier une classe désigne le chargement en mémoire de celle-ci).

5. Screen Manager

Un « Screen Manager » est une méthode de fonctionnement pour gérer le passage d'un écran à l'autre.
Ce passage d'un écran à un autre est dû à un changement d'état (ouvert, en veille, fermé).

Le rôle du « screen manager » est de donner accès à la variable GraphicsDevice et ainsi permettre la gestion des ressources de la carte graphique.

Il existe sur la toile différentes versions de « screen manager » sous forme d'API*.
Ceux-ci proposent des fonctions différentes selon leur programmeur.

J'ai cependant créé mon propre « screen manager » en m'inspirant de modèles existants, car les seuls « managers » intéressants sont payants et très chers.

API* : Application Programming Interface;Interface de programmation applicative

6. Graphique user interface

Il est intéressant d'avoir une interface graphique pour que les utilisateurs puissent utiliser l'application facilement.
Cette interface va leur permettre de choisir telle fonction ou autre.

XNA étant un environnement fonctionnant en Direct3D, il est difficile d'utiliser la bibliothèque Windows Form de Microsoft.

J'ai créé pour cela des objets « interface form » pour les importer dans mon application.

Il existe des API GUI* disponibles, mais ceux-ci sont en majorité incompatibles avec la version 4.0 de XNA.

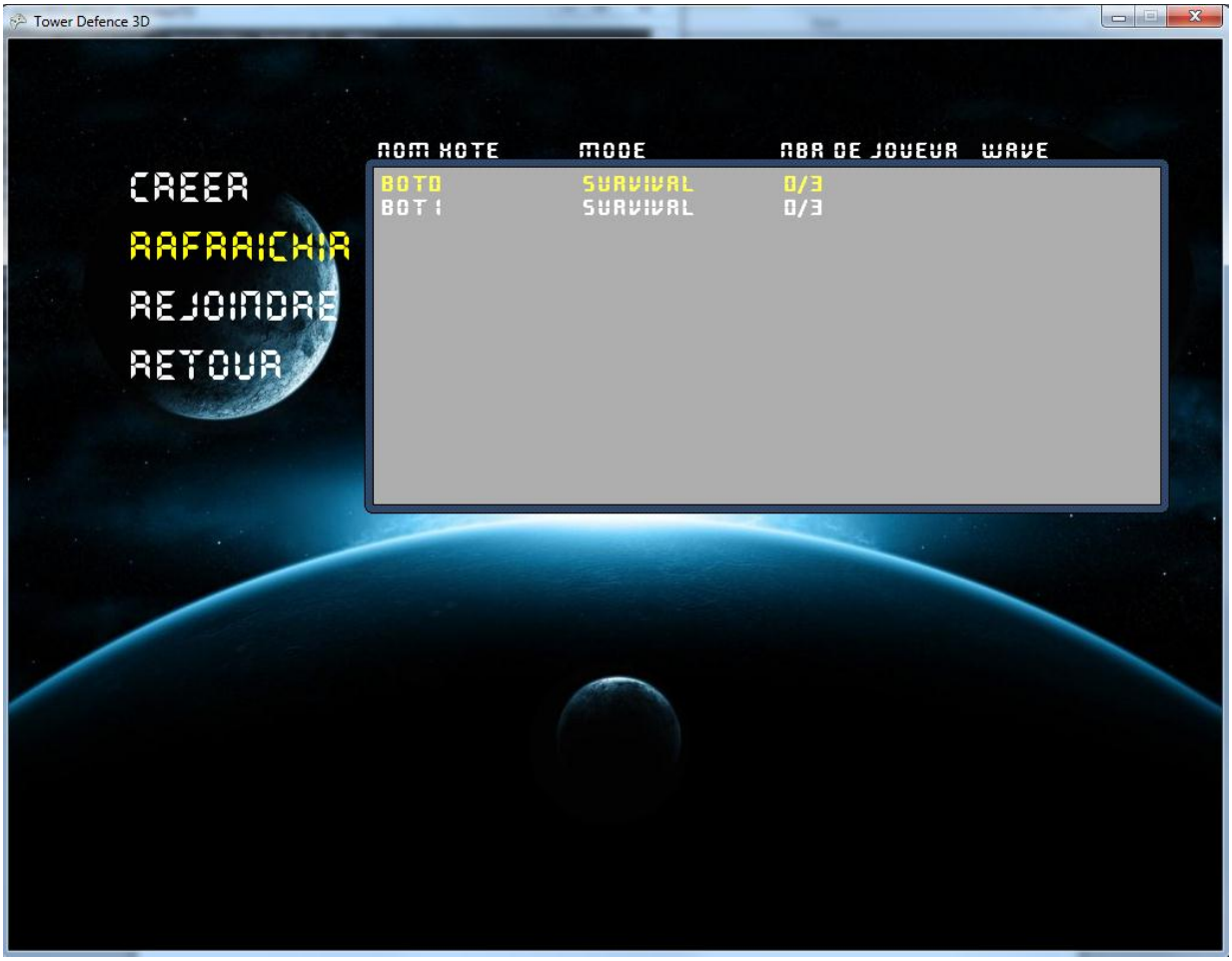

Figure 3 : illustration d'un GUI sur mon application.

GUI* : Graphique user interface ; interface graphique utilisateur.

7. Input manager

J'ai créé une classe qui me permet d'écouter (interpréter) les différentes interfaces d'entrée comme le clavier, la souris et la manette Xbox 360.

En général, l'écoute de ces interfaces retourne des variables d'état (touche appuyée : vrai, faux).

Cependant, chaque interface requiert une manière d'écoute différente.

7.1 Clavier

Pour le clavier, il faut créer une fonction permettant de vérifier chaque touche.

Toutefois, quelques soucis se posent au niveau de l'encodage : XNA ne reconnait que la table ASCII. Pour cela, les caractères spéciaux français ne sont pas reconnus et cela pose des problèmes.

J'ai donc trié les touches en fonction de la table ASCII.

7.2 Souris

Le pointeur de la souris est l'interface la plus utilisée dans un jeu « point and click ». C'est pourquoi il est très important de bien connaître la notion de position de la souris.

Le point 0,0 est situé dans le coin supérieur gauche de l'écran.

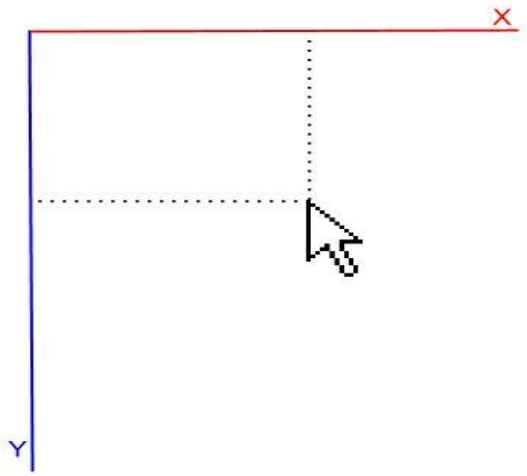

Figure 4: Position de la souris sur l'écran.

8. L'interaction avec objet

L'interaction d'un objet de l'environnement 3D avec le clavier est simple, il suffit de créer un événement lui disant de : « se déplacer à droite quand j'appuie vers la droite et ainsi de suite … ».

Par contre, l'interaction d'un objet avec la souris n'est pas chose facile.

Noter qu'on ne peut utiliser la position de la souris car elle se réfère par rapport à l'écran et non par rapport à la perspective de la camera.

Petit rappel, le monde 3D utilise 3 axes, X, Y et Z qui définit la profondeur.

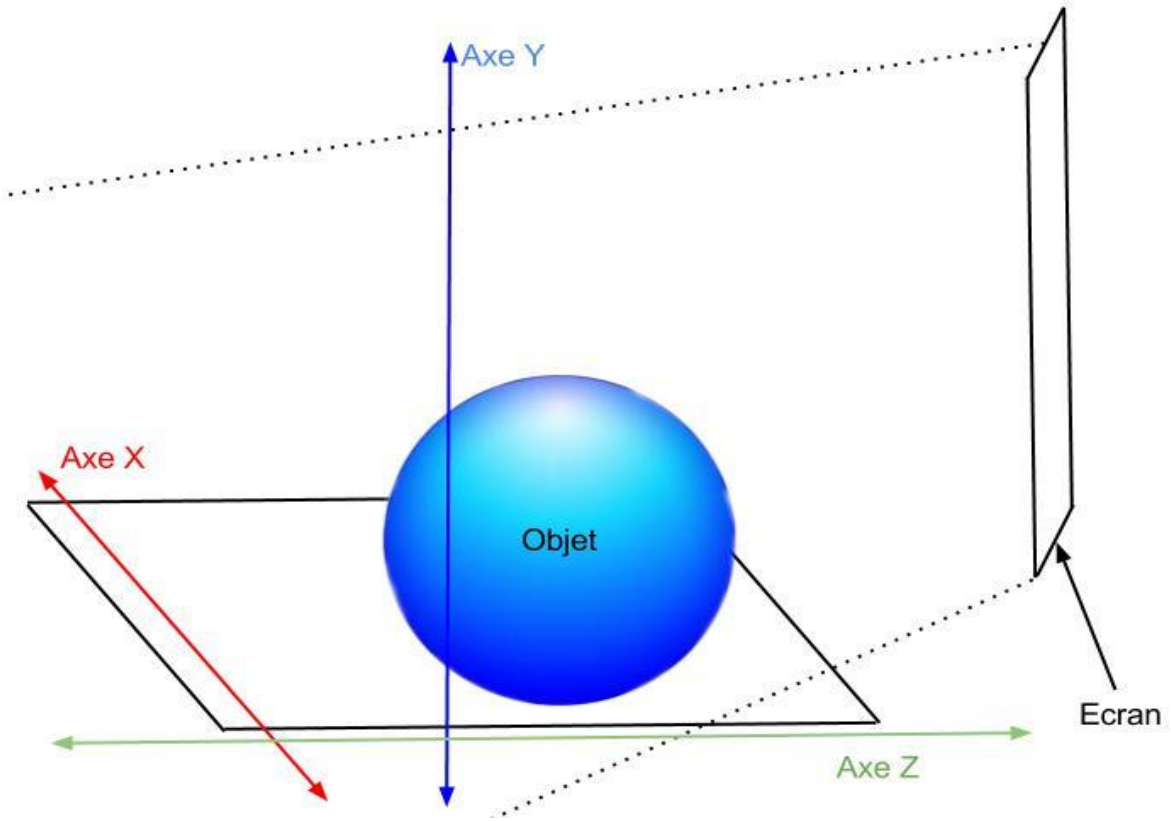

Figure 5 : Projection 3D

Pour retrouver la position par rapport à la projection, j'ai dû créer une méthode qui calcule la position de l'écran par rapport à l'objet et ainsi demander à l'objet de se déplacer au point récupéré.

9. Objet multimédia

9.1 Modélisation

Pour la réalisation d'un jeu vidéo 3D, il faut créer des modèles dits "Mesh".
J'ai créé les modèles grâce à Maya, un logiciel de création d'objets 3D.

Une fois créés, les modèles doivent être exportés au format X ou FBX.

9.2 Les images

Il est possible d'importer des images au format PNG ou JPEG pour réaliser soit un jeu en 2D soit une interface d'utilisateur.

9.3 Le son

La gestion du son est assurée par XACT, une librairie qui s'occupe particulièrement des effets sonores.
N'ayant aucune connaissance dans le domaine acoustique, je n'ai hélas pas ajouté de son dans mon application.
Le format MP3 et WMA sont pris en charge par XNA.

10. Les moteurs

Pour avoir une bonne organisation, il est important d'utiliser l'architecture Modèle-Vue-Contrôleur.
N'oublions pas que dans une application 3D, il y a beaucoup de flux de données qui circulent. Pour cela, les moteurs vont traiter ces informations pour mieux les dissocier, les moteurs sont en quelque sorte une architecture MVC avancée.
Chaque moteur possède des fonctions prédéfinies dans son domaine et permet de ainsi d'isoler une technique pour la rendre similaire à une action du même type ou de même famille.

10.1 Moteur graphique :

Le moteur graphique va gérer tous les objets 3D, les rendus, les effets, l'interaction visuelle entre les objets, l'ombre, la transparence, etc.

10.2 Moteur physique :

Le moteur physique va donner au monde virtuel un aspect plus réel, en y intégrant les lois de la physique, l'apesanteur, les collisions, l'énergie, …

10.3 Moteur réseau :

Le moteur réseau va gérer les communications entre les différents clients et le serveur.

10.4 Moteur audio:

Le moteur audio va gérer tous les effets sonores et musiques du jeu.

Les moteurs permettent de rassembler l'information dans une application.
À grande échelle, ils faciliteront la tâche du programmeur mais ne sont pas indispensables.

11. Pathfinding

Les objets qui demandent un déplacement logique pour aller d'un point A vers un point B ne peuvent le faire tout seul. Il faut passer pour cela par des algorithmes logiques appelés pathfinding ou recherche de chemin.
Le but principal d'un pathfinding est de retrouver la route la plus rapide et la plus fiable.
La fiabilité étant la route ayant la plus faible distance.

Il existe plusieurs algorithmes de déplacement, ceux-ci seront choisis selon l'environnement de jeu.
Dans mon cas, j'ai choisi un algorithme de déplacement pour un jeu de type "Point and click".

J'ai retenu 2 algorithmes :
- Algorithme de Dijkstra ;
- Algorithme A*.

11.1 Algorithme de Dijkstra:

Dijkstra est utilisé pour le protocole de routage dynamique IS-IS et peut-être utilisé pour un déplacement d'objet dans un jeu.
Celui-ci nécessite de connaître les points de déplacement voisin disponible, ce qui rend le déplacement statique.

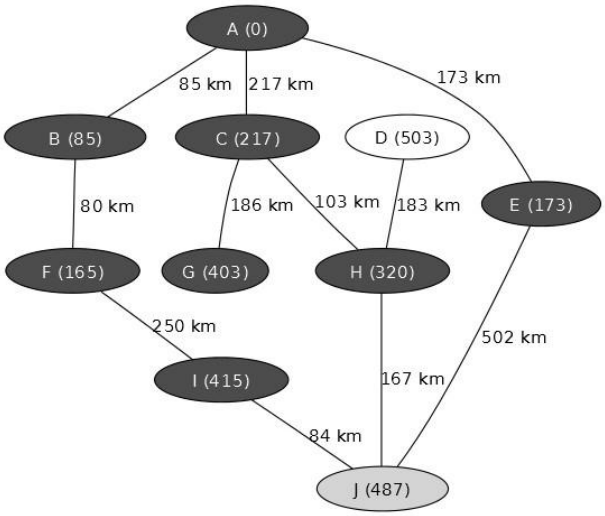

Figure 6: Résultat de recherche Dijkstra du point A vers le point J

11.2 Algorithme A* (A star ou A étoile)

La logique de cet algorithme est qu'il se déploie en étoile vers le point cible jusqu'à la rencontre du point de destination finale ou d'un éventuel obstacle.
J'ai utilisé A*. Cet algorithme est le plus utilisé du fait de son efficacité et de sa rapidité.

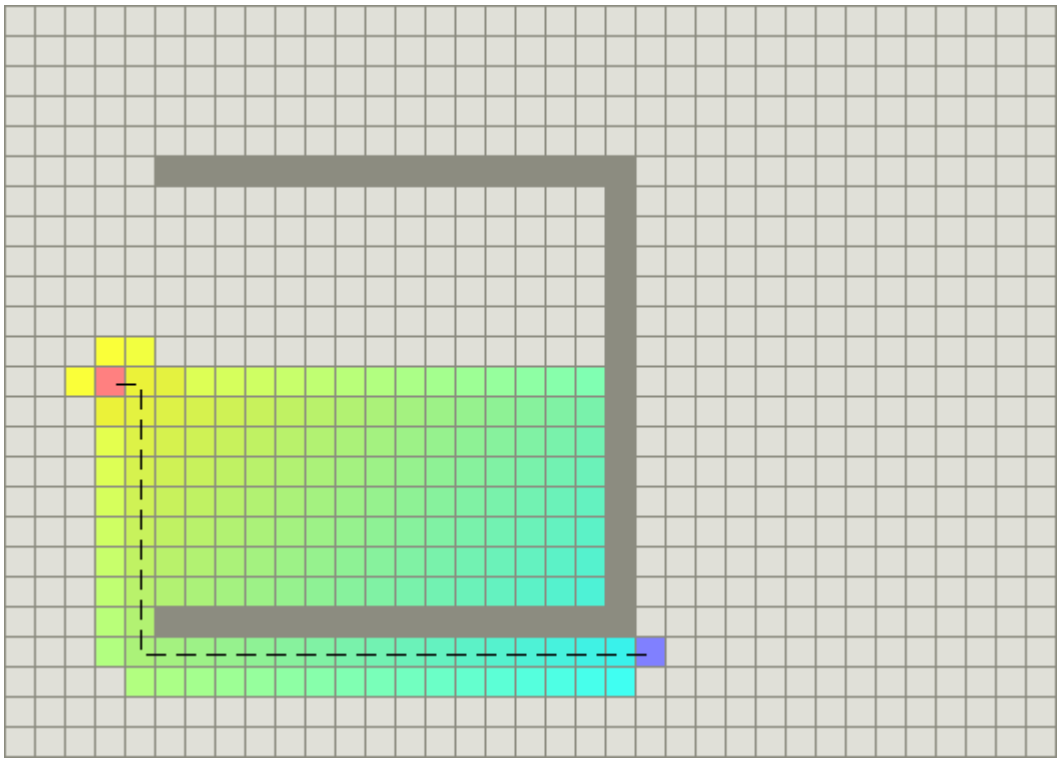

Figure 7 : Résultat de recherche par l'algorithme A*

12. Collision

Pour avoir un effet réel entre les objets du monde virtuel, je dois mettre en plus un système de détection de collision.
Lors de l'importation d'un modèle 3D dans notre monde virtuel, XNA crée une "Bounding sphere" ou "Sphère englobante" par défaut.
Cette sphère fera la coalition entre l'objet vu et l'objet en donnée.
Une collision sera définie si 2 "bounding sphere" sont adjacentes.
Le rayon de cette sphère est issu d'un calcul du point de vertex le plus loin du centre du modèle.

Figure 8: Représentation d'un "Bounding sphere" d'un objet

Le souci est que cette sphère générée par défaut est composée d'un très grand nombre de points obsolètes.
Ces points obsolètes vont provoquer une collision précoce des objets.

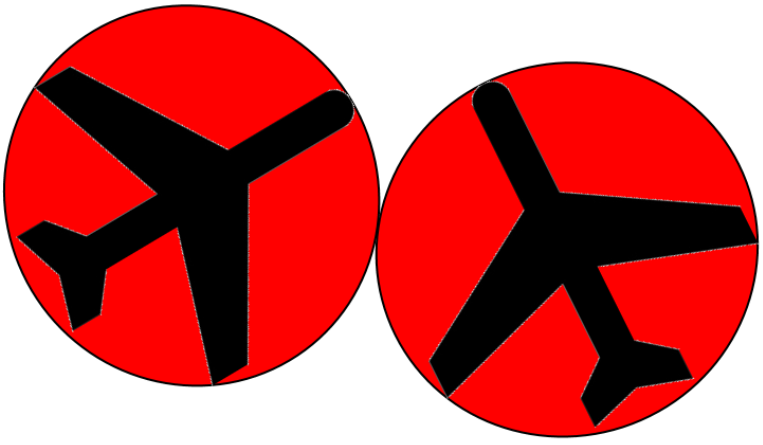

Figure 9: Collision précoce

Pour remédier à ce problème, il faut créer plusieurs "bounding sphere" de sorte à délimiter les endroits de collision possible.

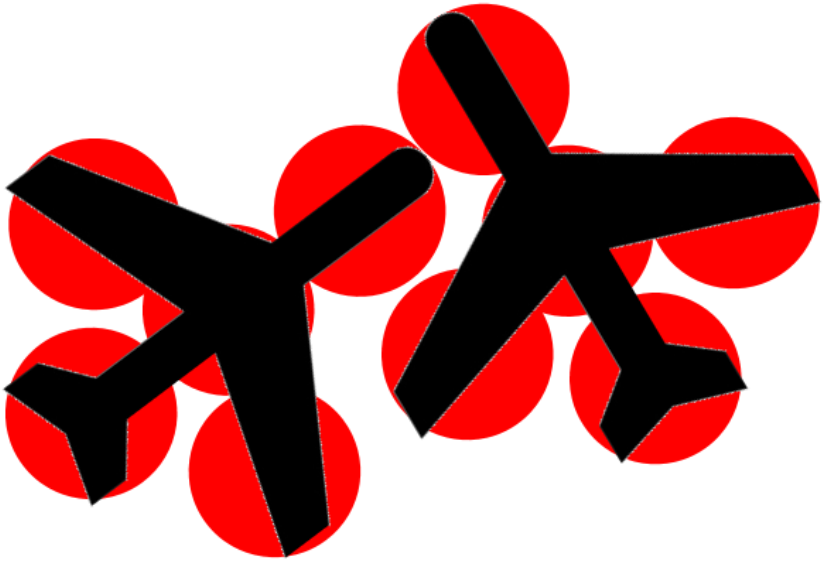

Figure 10: Collision avec des “Bounding Sphere” créées

Il est préférable d’utiliser un moteur physique pour générer les différents “Bounding Sphere”, sans lequel il est indispensable de savoir faire des calculs matriciels.

13. Client - Serveur

Pour assurer une communication entre le client et le serveur, je dois utiliser une bibliothèque commune entre le langage C# et Java.
Les sockets sont la bibliothèque commune qui permet d'établir cette communication, à condition que les messages envoyés soient encodés identiquement.
Le C# a comme encodage par défaut l'ASCII, tandis que Java ne le détermine pas.

J'ai utilisé l'Unicode comme encodage entre les sockets* Java et C#.

Les sockets* permettent une communication par le protocole TCP et UDP.

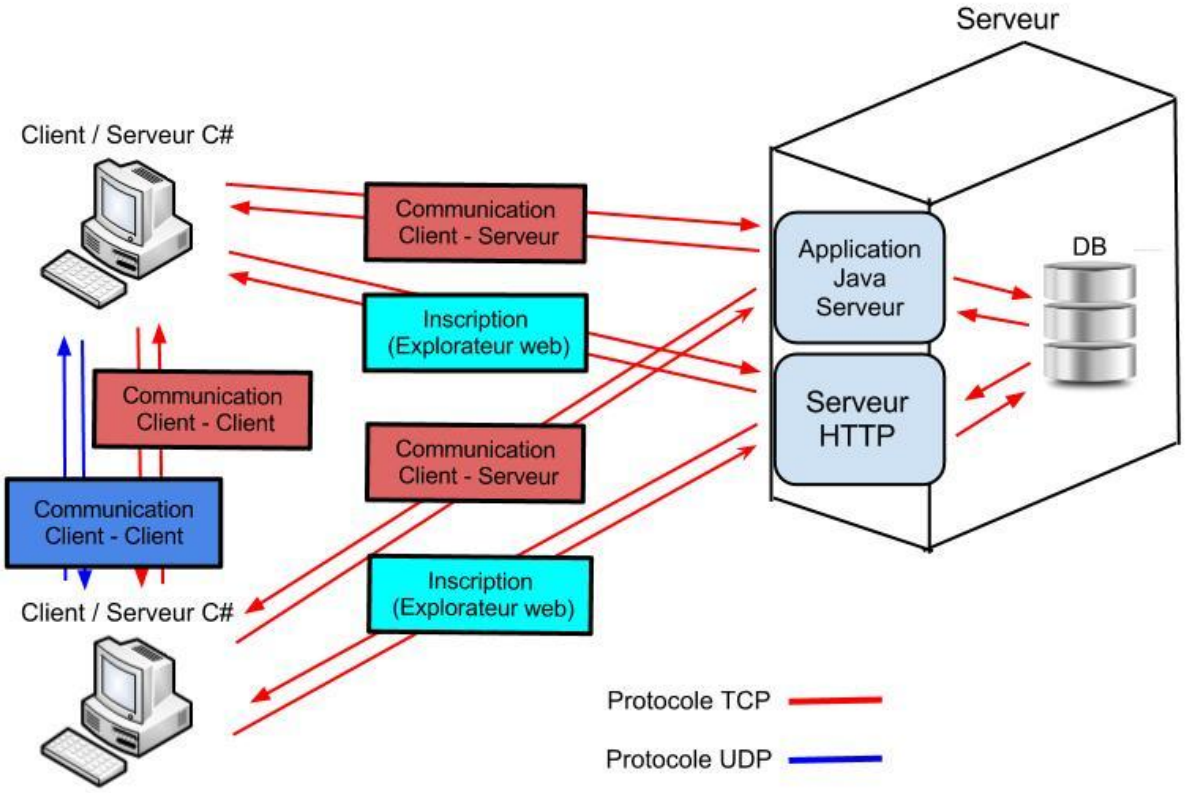

Figure 11: Architecture Client - Serveur application Tower Defence 3D

Socket**"Interface permettant la communication entre un système unique et un réseau utilisant le protocole TCP/IP"*

13.1 Serveur

Le serveur est composé d'un service MySQL pour le stockage de données des utilisateurs, d'un serveur HTTP pour l'inscription et de l'application serveur.
La base de données MySQL est accessible uniquement en local par le serveur HTTP et l'application, ainsi je protège son contenu.

Il est préférable que l'application serveur soit multiplateforme. De ce fait, j'ai choisi de coder le serveur en Java.
Si j'avais utilisé le C# pour l'application serveur, celui-ci aurait la nécessité d'être hébergé sur une plateforme qui prend en charge les bibliothèques .NET.
La communication entre le client et le serveur nécessite une fiabilité, donc j'ai utilisé le protocole TCP.

Le rôle du serveur est de :

- Faire officie d'intermédiaire entre les clients, aucune partie ne sera gérée par l'application serveur.
- Réaliser l'authentification des clients connectés grâce à la base de données MySQL.
- Recenser toutes les parties et d'informer les parties existantes à tous les clients.

13.2 Client / Serveur

Les clients devront créer un compte depuis le site web et télécharger l'application client.
Les clients authentifiés peuvent créer une partie, l'application client va informer le serveur intermédiaire qu'une partie est créée.
Cependant, il est nécessaire au client voulant créer une partie d'ouvrir le port 27000 en UDP pour pouvoir accueillir d'autres joueurs.

Pourquoi devrait-on créer plusieurs serveurs ?
Les applications « Client / Serveur » ne sont présentes ici que pour assurer un hébergement de partie.
Explication :
Quand le joueur 1 va créer une partie, il va en quelque sorte créer/ouvrir la partie « Host » et ainsi permettre aux autres joueurs et à lui-même de lui rejoindre, cette partie « Host » va gérer tout le contenu du jeu et assurer que tous les joueurs ayant joint la partie puissent avoir les informations communes.

Dans un jeu vidéo, il est important que la communication soit rapide, pour cela nous devons créer une connexion UDP et non pas TCP (voir plus pour la raison du choix de l'UDP au détriment de TCP).

13.3 Socket TCP

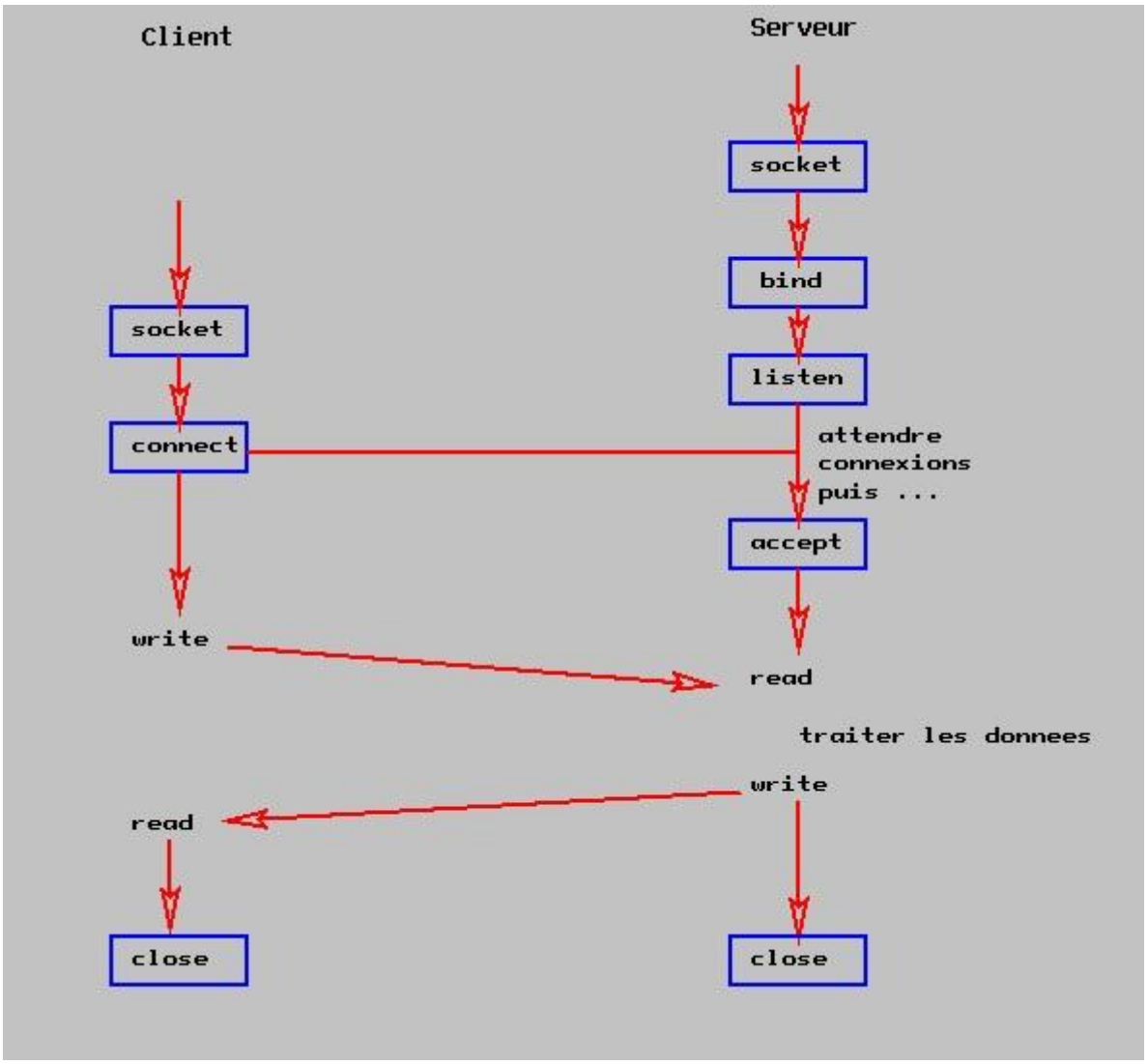

Figure 12: Schéma de communication Client - Serveur TCP

Sur ce schéma, nous pouvons voir que le serveur ouvre la connexion et attend une communication du client. Dès que le serveur détecte une connexion de la part du client, il accepte celle-ci et l'ajoute dans une instance.
Chaque client est distingué par une instance de connexion et le message ainsi traité est associé à cette instance.

13.4 Authentification Client - serveur

Voici le processus d'authentification des clients sur le serveur:

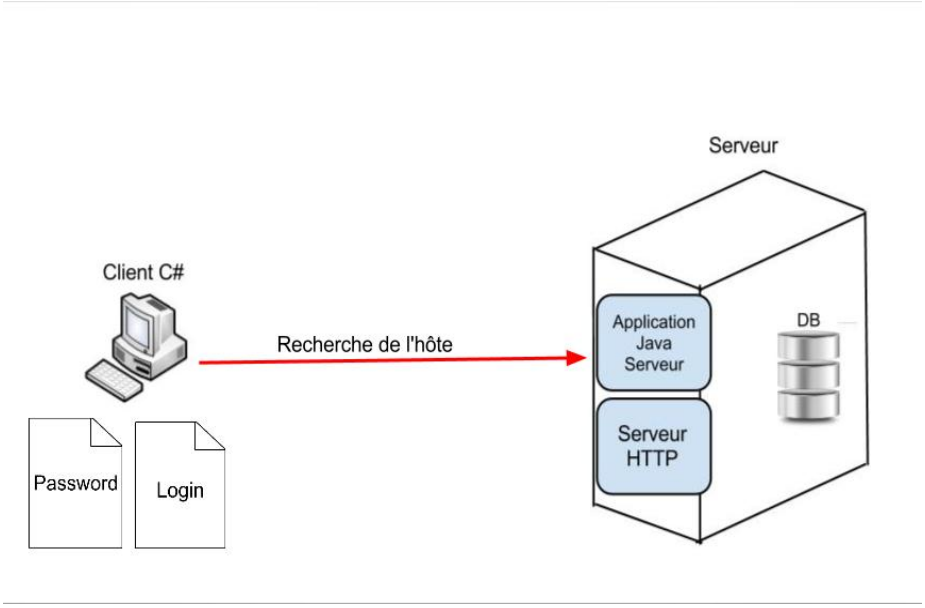

Figure 13 : Recherche du serveur

Le client entre son login (nom d'utilisateur) et mot et passe. Puis, il initie la connexion.

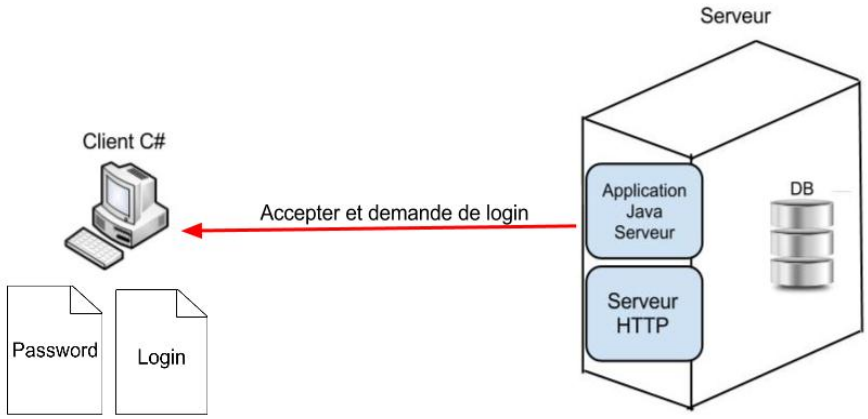

Figure 14: Connexion acceptée et demande de login

Le serveur accepte la connexion et démarre le processus d'authentification par une "demande de login".

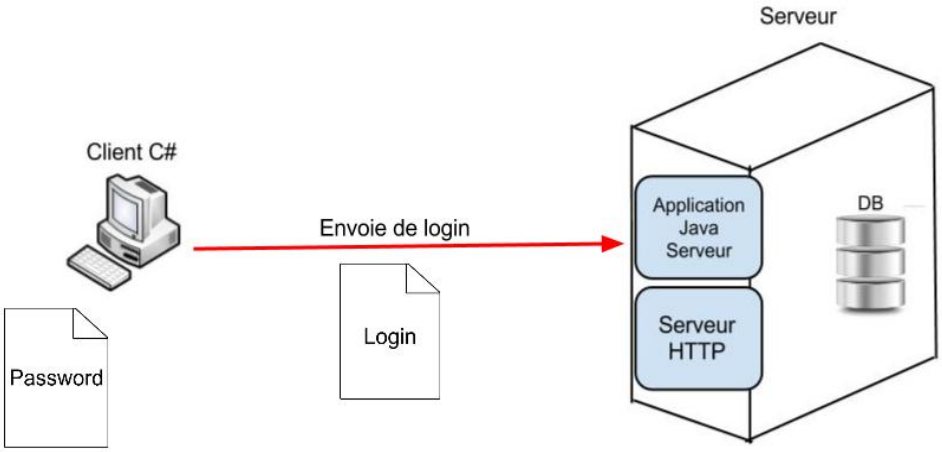

Figure 15: Envoi de login

Le client reçoit le message “demande de login” et répond en lui envoyant son login.

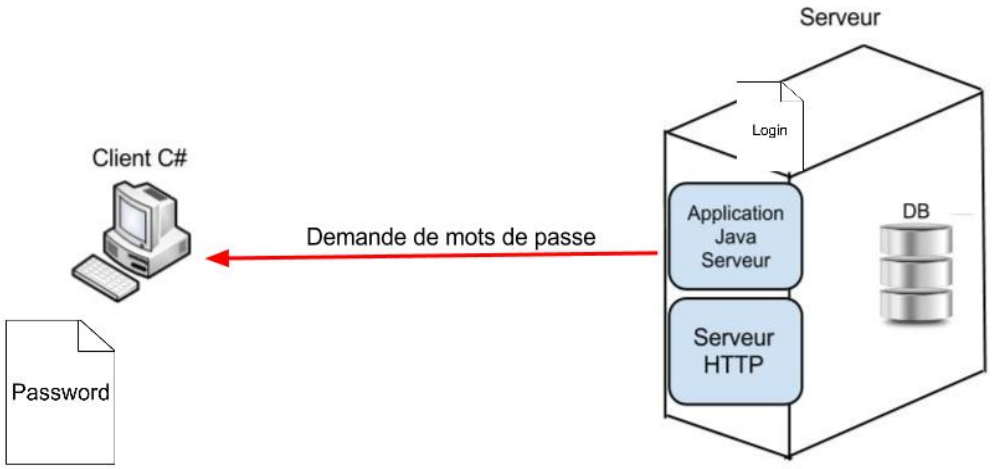

Figure 16: Demande de mot de passe

Le serveur reçoit le login, le stocke en mémoire et fait une demande de mot de passe.

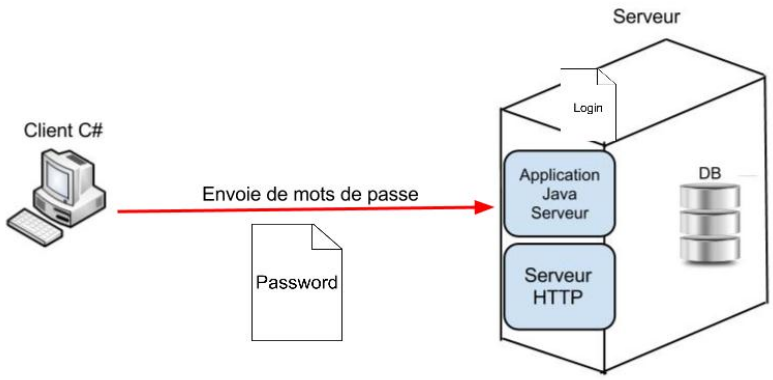

Figure 17: Envoi du mot de passe

Le client reçoit le message “demande de mot de passe”, et répond en lui envoyant le mot de passe.

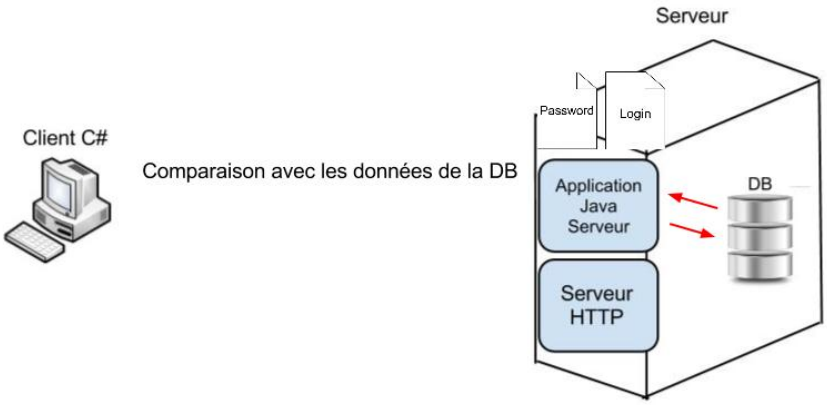

Figure 18: Comparaison avec les données de la DB

Une fois le login et mot de passe récupérés, le serveur effectue une vérification avec la base de données.

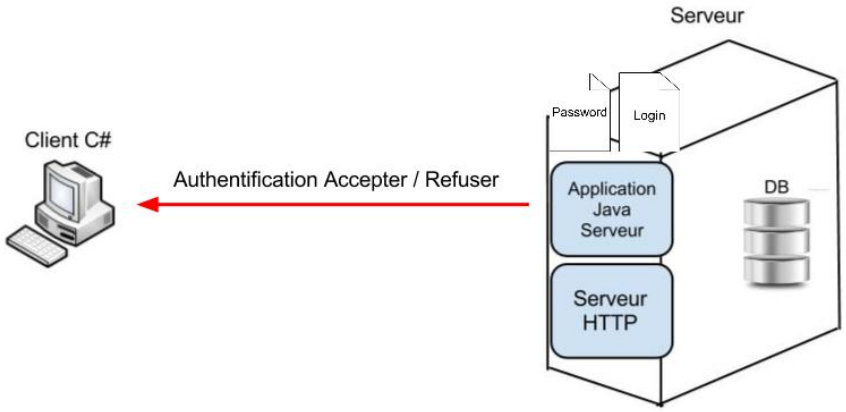

Figure 19: Connexion acceptée ou refusée

Si l'authentification s'est correctement déroulée, le serveur renvoie un message « connexion acceptée » et il ferme la connexion.

13.5 Socket UDP

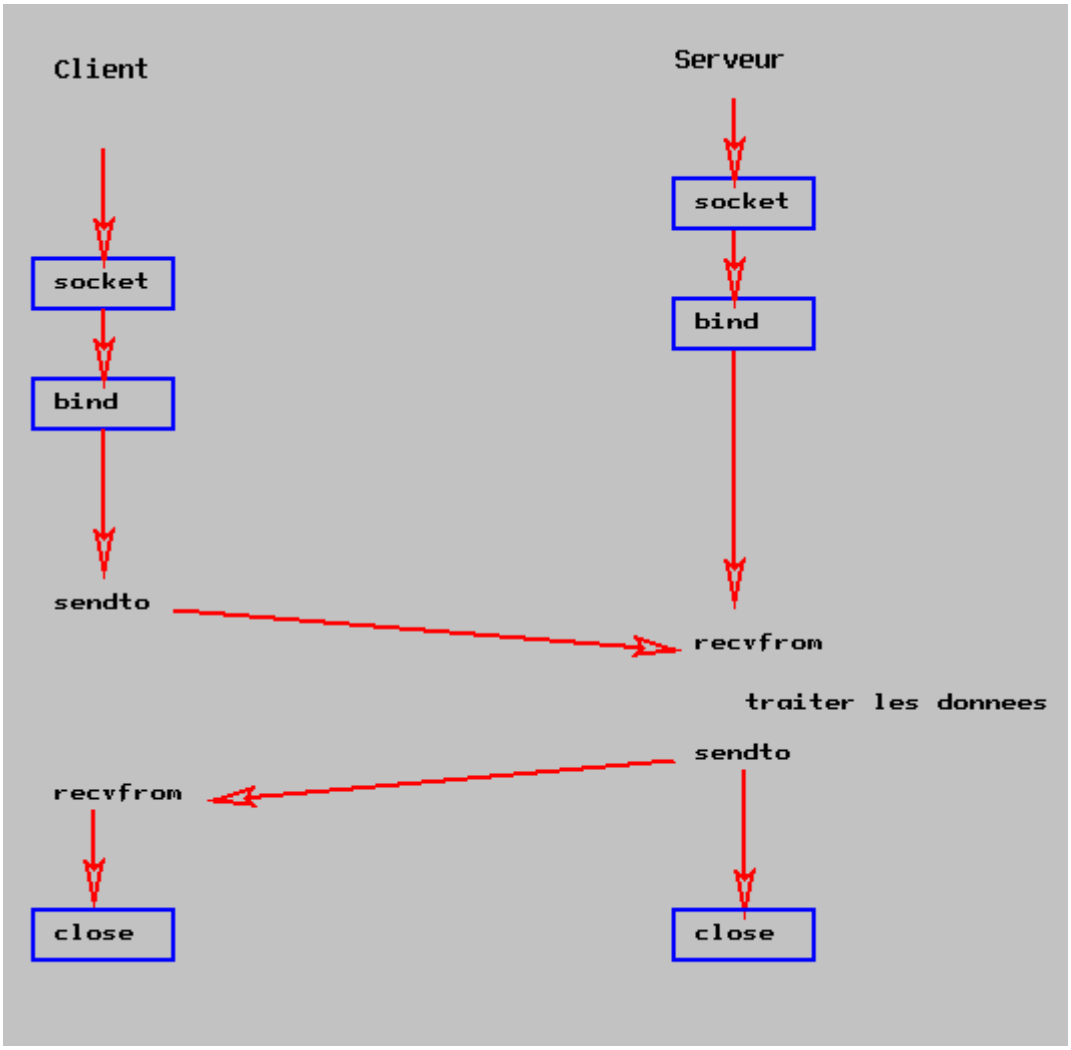

Figure 20: Schéma communication Client - Serveur UDP

La connexion pour les sockets UDP est destinée à un flux important de données. UDP est le protocole le plus répandu dans la réalisation d'une communication entre applications jeux vidéo.

Le protocole UDP ne nécessite pas d'authentification, ce qui veut dire qu'une fois exécuté, tout le monde connaissant l'adresse IP et le port du serveur UDP peut entrer en communication.

Ce protocole pose un souci à ce niveau : on ne peut accepter les paquets venant de n'importe qui, cela peut causer des erreurs de communication.

Pour remédier à ce problème, j'ai mis au point un système d'authentification par le biais du serveur intermédiaire.

13.6 Authentification Client - Client

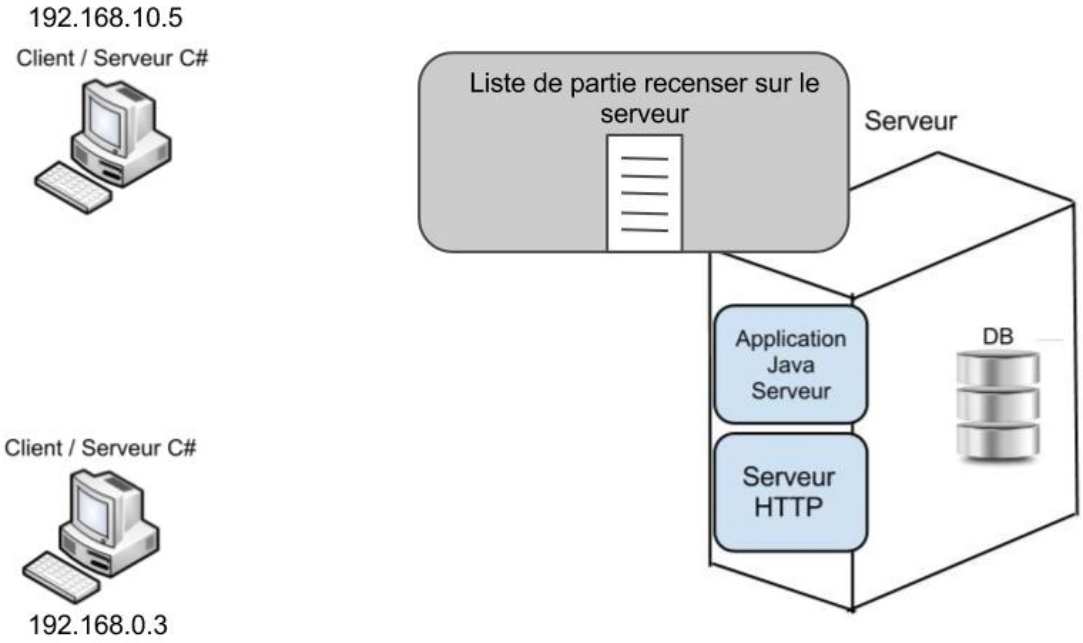

Figure 21: Illustration de toutes les parties recensées sur le serveur intermédiaire

Les joueurs qui créent une partie sont listés sur le serveur, permettant ainsi aux autres joueurs de rejoindre leur partie.

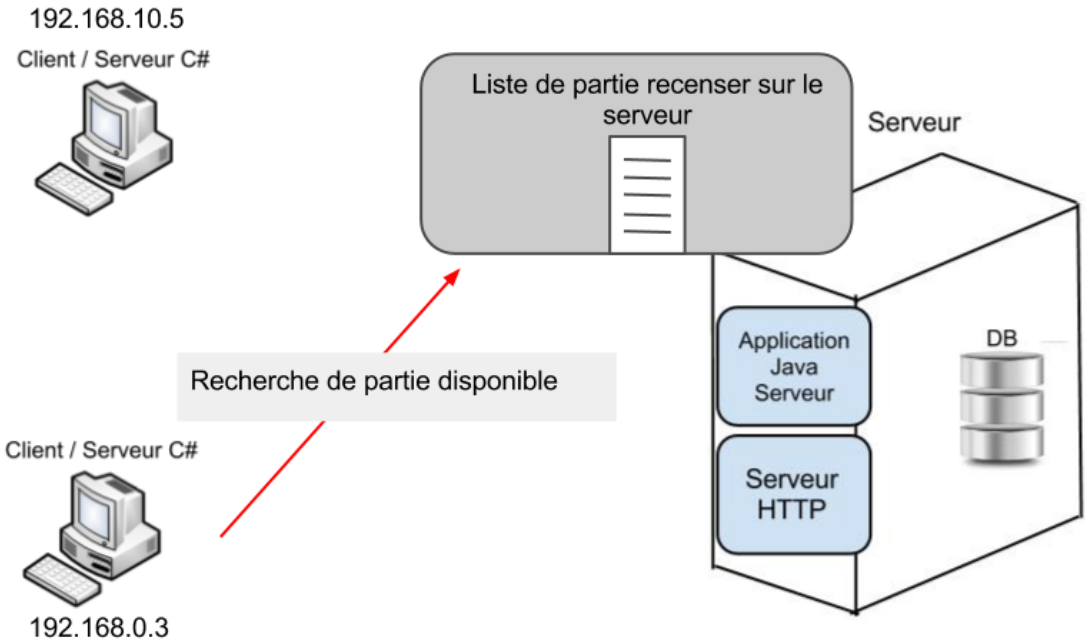

Figure 22: Recherche de parties disponibles dans la liste

Le serveur transmet la liste complète des parties disponibles. Dans cette liste, on pourra retrouver les informations du type de jeu et de l'hôte.

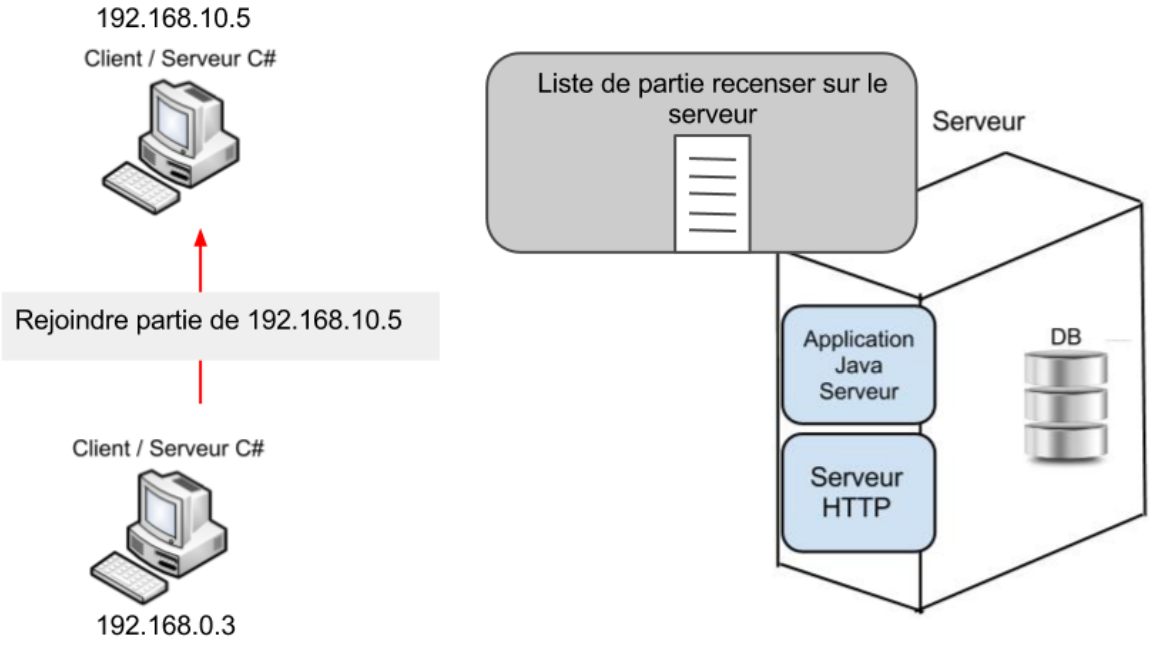

Figure 23: Tentative de rejoindre une partie.

Le joueur voulant rejoindre une partie en choisit une dans la liste et entame une tentative de communication avec l'hôte.

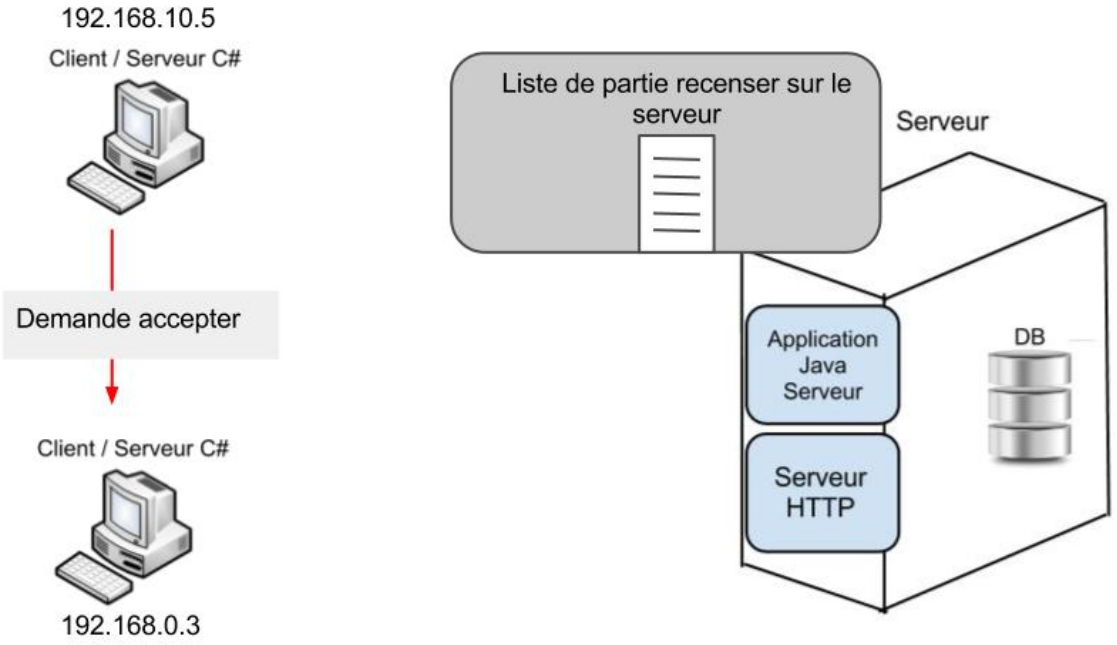

Figure 24: L'hôte accepte le client.

Apres vérification de la situation et de la disponibilité dans la partie, l'hôte décide si oui ou non le joueur pourra rejoindre la partie.

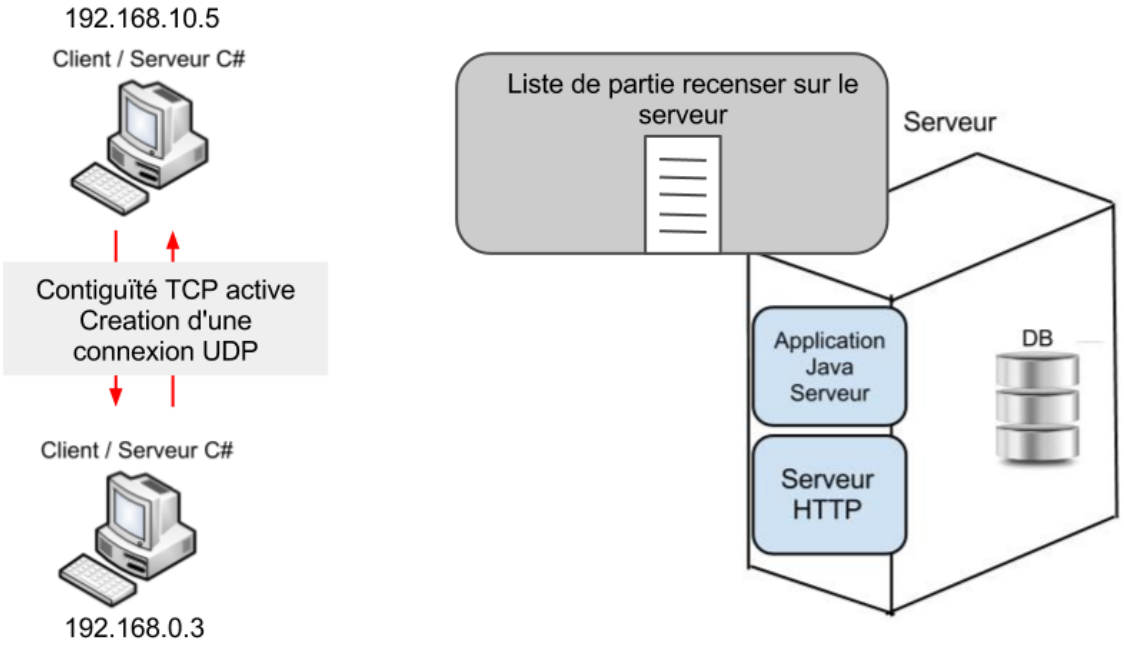

Figure 25: Contiguïté TCP active.

Une fois le nouveau joueur accepté, l'hôte crée une connexion UDP pour assurer un transfert de données non fiable mais rapide.

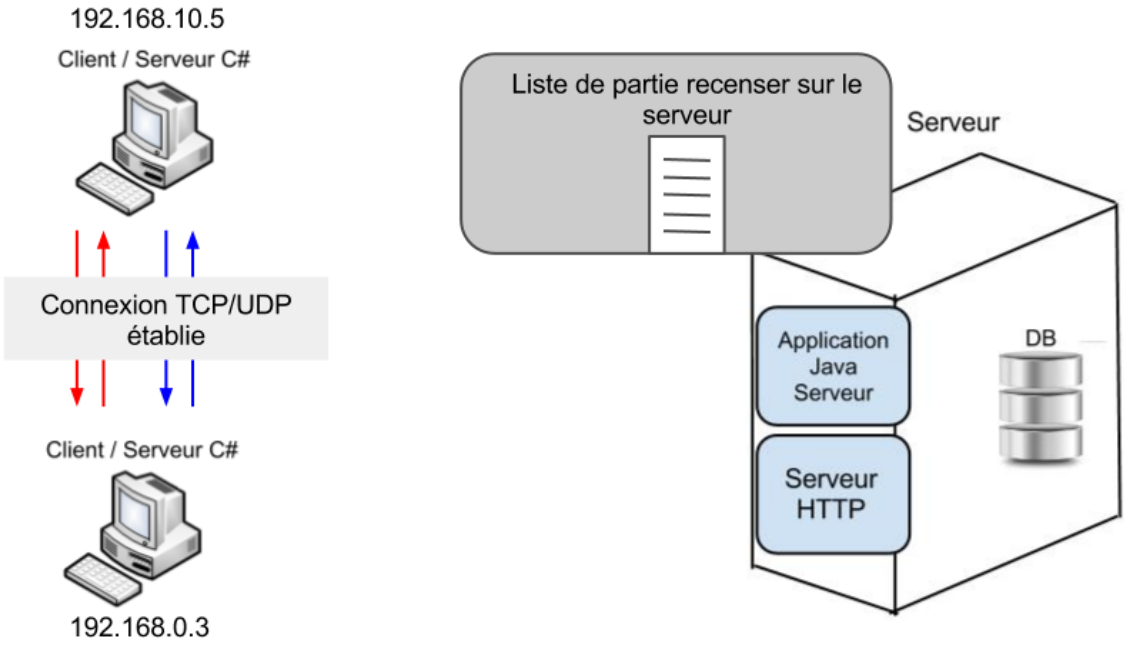

Figure 26 : Contiguïté complète

Établissement d'une contiguïté entre les clients et échange de données.

14. Protocole UPNP

Dans mon cas de figure, l'hôte de la partie devient potentiellement un serveur et comme dans tous les serveurs, leurs services requièrent une connexion qui passe par un port valide.

Pour cela, je dois passer par UPNP « Universal Plug And Play », qui est un service disponible sur la majorité des routeurs grand public.
Ce service permet à une application dans le réseau local d'ouvrir un port temporaire qui permet d'effectuer une communication.

Le protocole UPNP est fréquemment exploité par diffèrents logiciels, permettant à des utilisateurs non expérimentés d'ouvrir un port valide et fonctionnel.

Ce protocole peut être utilisé par différentes applications malveillantes, raison pour laquelle il est désactivé par défaut sur les routeurs.

15. Site d'inscription

J'ai réalisé un site permettant à l'utilisateur de créer un compte.

15.1 Technique et outils pour réaliser le site :

Code : PHP, xHTML, JQuery, CSS,

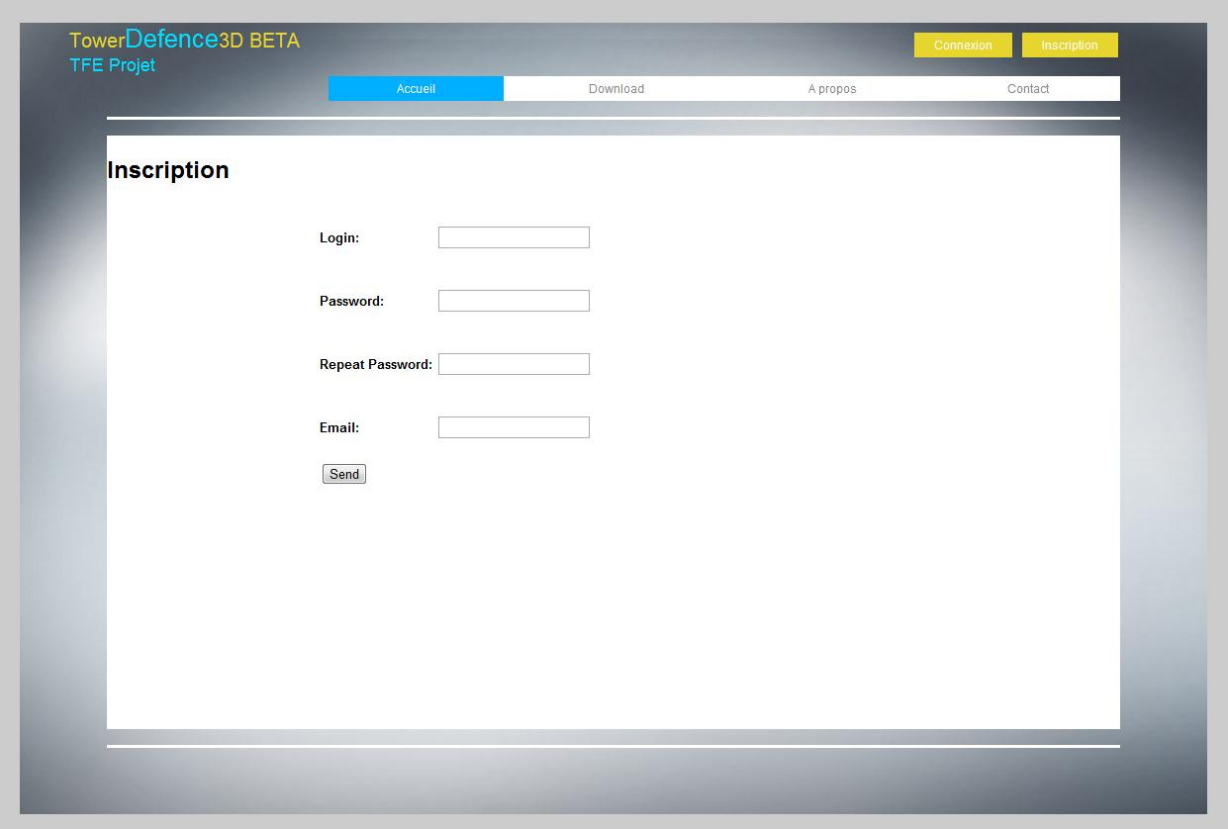

Figure 27 : interface web.

16. Clés de registre

Les données sauvegardées en local sont stockées dans le registre de Windows. J'ai créé une classe permettant de créer/modifier les clés de registre.

Le registre de Windows est une base de données locale permettant de stocker toute sorte de clés contenant des valeurs.

Cela me permet de récupèrer des variables sans devoir vérifier chaque fois si la valeur est correcte ou non.

Le chemin d'accès au registre est
« Ordinateur\HKEY_CURRENT_USER\Software\Angoraslan\TowerDefence3D ».

Les clés de registre sont créées au premier lancement de l'application.

17. Utilisation de l’application

Avant de jouer, l’utilisateur doit :

- Créer un compte à partir du site web ;
- Télécharger l’application sous forme d’installeur précompilé par Visual Studio ;
- Installer les bibliothèques requises :
    - Windows installeur 3.1
    - Windows Framework .NET 4.0
    - Windows Framework XNA redistribuable
- Installer l’application à partir du fichier Setup.exe

Enfin, l’utilisateur n’a plus qu’à lancer l’application à partir du fichier Tower Defence 3d.exe dans le menu démarrer de Windows.

18. Difficultés rencontrées

Le problème majeur du projet est le sujet en lui-même qui est très vaste. En effet, la notion 3 dimensions m'étant totalement inconnue, j'ai dû énormément me documenter sur le sujet et consulter l'aide en ligne de Microsoft notamment.

De plus, la gestion des flux client / serveur n'a pas été évidente car chaque action nécessite un message qui doit être traité très rapidement.

19. Conclusion

Tout au long de ce travail, je vous ai présenté une des méthodes de réalisation d'un jeu vidéo. J'estime à l'heure actuelle que les jeux vidéo font partie des produits les plus consommés dans le monde numérique, d'où la raison de mon choix.

Pour qu'un jeu vidéo puisse être apprécié des consommateurs, ce dernier doit répondre à différents critères:

- Un rendu graphique attrayant et réel. De plus en plus de jeux vidéo sur le marché utilisent une technologie de pointe de sorte à obtenir un rendu proche de la réalité.
- Une jouabilité aisée : le contrôle des personnes, ou l'interaction avec les objets du monde virtuel.
- Des effets spéciaux attirants.
- Un scénario exemplaire.

Bien entendu, la réalisation d'une application à grande envergure nécessite un travail d'équipe, une équipe pour chaque partie.

J'espère plus tard pouvoir faire partie d'une telle équipe et travailler au sein d'une maison d'édition de jeux vidéo.

20. Bibliographie

20.1 Ouvrages

Professional XNA Game Programming: For Xbox 360 and Windows, *Benjamin Nitschke*

20.1 Sites web

http://msdn.microsoft.com/en-us/library/bb200104.aspx, bibliothèque *Microsoft XNA*

http://msdn.microsoft.com/library/kx37x362, bibliothèque *Microsoft C#*

http://www.xnaresources.com, *guide de développement application avec Framework XNA*

http://www.spi.ens.fr/beig/systeme/sockets.html, les *sockets explication.*

http://www.phstudios.com/, *guide de création un tower defence 2D*

http://msdn.microsoft.com/fr-fr/library/h5e7chcf.aspx *Guide de création de registre de Windows*