



2010

ASD Tower Defense

Version 2.0 – En réseau

Intégration d'un mode multi-joueurs et amélioration du jeu sur différents points notamment l'architecture, l'interfaçage et l'ajout de ressources (images / son/ etc.)

Aurélien Da Campo
Lazhar Farjallah
Pierre-Dominique Putallaz
Romain Poulain

Vous avez l'esprit joueur ?
Recherchez les 7 créatures qui se cachent dans ce document.

GEN
Heig-vd - GEN
01/06/2010



Table des matières

1	Introduction	4
2	Analyse	4
2.1	Règles du jeu	4
2.1.1	Les créatures.....	4
2.1.2	Les tours.....	4
2.1.3	Les modes de jeu	4
2.1	Partage des responsabilités	5
2.2	Cas d'utilisation.....	6
2.2.1	Schéma global des acteurs	6
2.2.2	Etapes de lancement du jeu	7
2.2.3	Système de gestion de partie	9
2.2.4	Système de fin de partie.....	10
2.2.5	Système de chat.....	11
2.4	Serveur d'enregistrement.....	12
2.4.1	Acteurs principaux.....	12
2.4.2	Scénario 01 : Demande de la liste des parties (succès).....	13
2.4.3	Scénario 02 : Demande du nombre de parties (succès)	13
2.4.4	Scénario 03 : Enregistrement d'une partie (succès).....	14
2.4.5	Scénario 04 : Suppression d'une partie (succès)	14
2.4.6	Scénario 05 : Mise à jour des informations d'une partie	15
2.4.7	Scénario 06 : Fermeture de la connexion (succès).....	15
2.5	Serveur de Jeu.....	16
2.6	Serveur Web	16
3	Conception du projet.....	17
3.1	JSON	17
3.2	Protocole d'échange entre le client et le serveur.....	19
3.2.1	Serveur d'enregistrement.....	19
3.2.2	Serveur de jeu.....	19
3.3	Diagramme d'activité général.....	20
3.4	Architecture de l'application	21
3.4.1	Moteur du jeu	22
3.4.2	Intégration du pattern Observable / Observé.....	23
3.4.3	Les différentes classes Jeu	25
3.4.4	Réseau	26
3.5	Charte graphique	27
3.6	Sérialisation des scores et des terrains de jeu	30
4	Gestion de projet.....	31
4.1	Rôle des participants au sein du groupe.....	31
4.2	Plan d'itérations initial.....	32
4.2.1	Itération 1 – Serveur d'enregistrement + Interface graphique	32
4.2.2	Itération 2 – Serveur de Jeu + Architecture.....	33
4.2.3	Itération 3 –Intégration du serveur de jeu + Interface du Jeu en réseau.....	33
4.2.4	Itération 4 – Lifting de la GUI + Game Design + Amélioration Mode Solo	34
4.2.5	Itération 5 – Serveur Web (facultatif).....	34
4.3	Suivi du projet : itérations par itérations.....	35
4.3.1	Itération 1 – Serveur d'enregistrement + Interface graphique	35
4.3.2	Itération 2 – Serveur de Jeu + Architecture.....	36

4.3.3	Itération 3 – Intégration du serveur de jeu + Interface du Jeu en réseau.....	37
4.3.4	Itération 4 - Lifting de la GUI + Game Design + Amélioration Mode Solo	38
4.3.5	Itération 5 – Serveur Web (facultatif).....	39
4.4	Stratégie de tests	40
4.4.1	Condition des tests effectués	40
4.4.2	Stratégie des tests	40
4.4.3	Test n°01 : connexion client/serveur d'enregistrement avec échange de messages	41
4.4.4	Test n°02 : voir les parties inscrites sur le serveur d'enregistrement	42
4.4.5	Test n°03 : « Déenregistrer » une partie sur le serveur d'enregistrement.....	43
4.4.6	Test n° 04 : mettre à jour les informations d'une partie	44
4.4.7	Test n° 05 : le client peut se connecter et jouer une partie en réseau	45
4.4.8	Conclusion des tests	47
4.5	Stratégie d'intégration du code de chaque participant.....	48
5	Etat des lieux.....	48
5.1	Ce qui fonctionne (résultat des tests).....	48
5.2	Ce qu'il resterait à développer.....	50
5.2.1	Peaufinage et recontrôle de l'application réseau	50
5.2.2	Maillage	50
5.2.3	Traduction en plusieurs langues (principalement anglais).....	51
5.2.4	Plus de ressources	51
6	Auto-critique.....	52
7	Conclusion.....	52
8	Annexes	52

1 Introduction

Ce projet prend place durant notre 4e semestre au sein de la Haute Ecole d'Ingénierie et de Gestion du canton de Vaud (heig-vd). Ce cours de Génie logiciel (GEN) nous propose de mettre en pratique les notions théoriques acquises en créant une application de type Client/Serveur.

Nous avons tout de suite pensé à l'amélioration d'un jeu que nous avons créé durant nos cours d'Algorithmes et Structures de Données (ASD2) suivi durant notre 3e semestre. En effet, nous avons réalisé un jeu et ce nouveau projet est pour nous l'opportunité d'étendre ce logiciel en lui fournissant des fonctionnalités réseau.

Ce document vous présente le rapport du projet reposant sur une gestion de projet basée sur la méthode UP. Pour d'éventuelles questions, nous vous fournissons le rapport de la version 1.0 du projet réalisée durant notre cours ASD2 (2009-2010).

2 Analyse

2.1 Règles du jeu

Le principe du jeu est de survivre à diverses créatures dont la seule capacité est d'avancer le long du chemin le plus court pour rallier leur point de départ à la zone de fin. Lorsqu'une créature atteint la zone de fin, elle fait perdre une vie au joueur. Lorsque le joueur n'a plus de vie, il a perdu.

2.1.1 Les créatures

Toutes les créatures prennent le chemin le plus court depuis leur emplacement jusqu'à la zone de fin. Les créatures terrestres doivent contourner les murs et les tours. Les créatures volantes peuvent survoler les tours.

2.1.2 Les tours

Pour se défendre, le joueur peut acheter des tours qu'il place sur son plateau. Certaines tours font des dégâts (ciblés ou de zone), d'autres permettent seulement de ralentir les créatures. Certaines ne sont efficaces que sur un certain type de créature.

Les contraintes pour la construction d'une tour sont de ne pas la créer là où se trouve une créature à ce moment et il doit toujours y avoir un chemin entre la zone de départ et celle de fin.

2.1.3 Les modes de jeu

Le jeu, dans sa première version, était uniquement local et proposait donc uniquement un mode solo où le joueur survivait aux vagues de créatures lancées par l'ordinateur. L'intégration de fonctionnalités réseau permet son extension à un jeu multi-joueurs.

Une première analyse nous a permis de déterminer plusieurs types de parties dont voici quelques exemples :

- **Coopération** : plusieurs joueurs s'allient contre l'intelligence artificielle et jouent sur le même plateau.


- **Coopération-zone** : Idem que coopération mais chaque joueur possède une zone du plateau de jeu partagé et ne peut bâtir des tours que dans celle-ci.
- **Versus** : chaque joueur possède son propre plateau et joue seul contre tous les autres. Il gagne de l'argent périodiquement et en tuant des créatures. Pour augmenter son revenu périodique, il peut acheter des créatures qu'il envoie chez l'ennemi.
- **Domination** : un joueur est désigné pour se battre contre les autres. Le joueur seul a des caractéristiques et des bonus de meilleure qualité que les autres mais est handicapé par sa vitesse réduite d'exécution des opérations.

Nous avons uniquement retenu le mode versus car les délais ne permettent pas de les faire tous. Ce mode est inspiré d'une extension de Warcraft III (de Blizzard Entertainment) qui est la pionnière en matière de « Tower Defense ».

2.1 Partage des responsabilités

Nous vous présentons ici les personnes **responsables** des différentes implémentations du projet. Bien évidemment, chaque membre apportera sa contribution pour chacune des parties à mettre sur pied. C'est par contre aux responsables eux-mêmes de gérer les ressources humaines pour mener à bien les parties qui leur incombent.

Notons encore que les différentes parties présentées ici seront détaillées dans la suite du document.

	Aurélien	Lazhar	Pierre-Do.	Romain
Moteur et architecture du jeu (adaptation / extension)	✕			
Interfaces (Conception / Implémentation)	✕			
Serveur d'enregistrement (Protocole / Implémentation)		✕		
Client du jeu (Protocole / Implémentation)			✕	✕
Serveur du jeu (Protocole / Implémentation)			✕	✕
Amélioration du jeu (la version 2.0 doit apporter de nouveaux concepts)				

2.2 Cas d'utilisation

2.2.1 Schéma global des acteurs

Nous vous présentons tout d'abord un schéma global mettant en œuvre les différents acteurs de l'application.

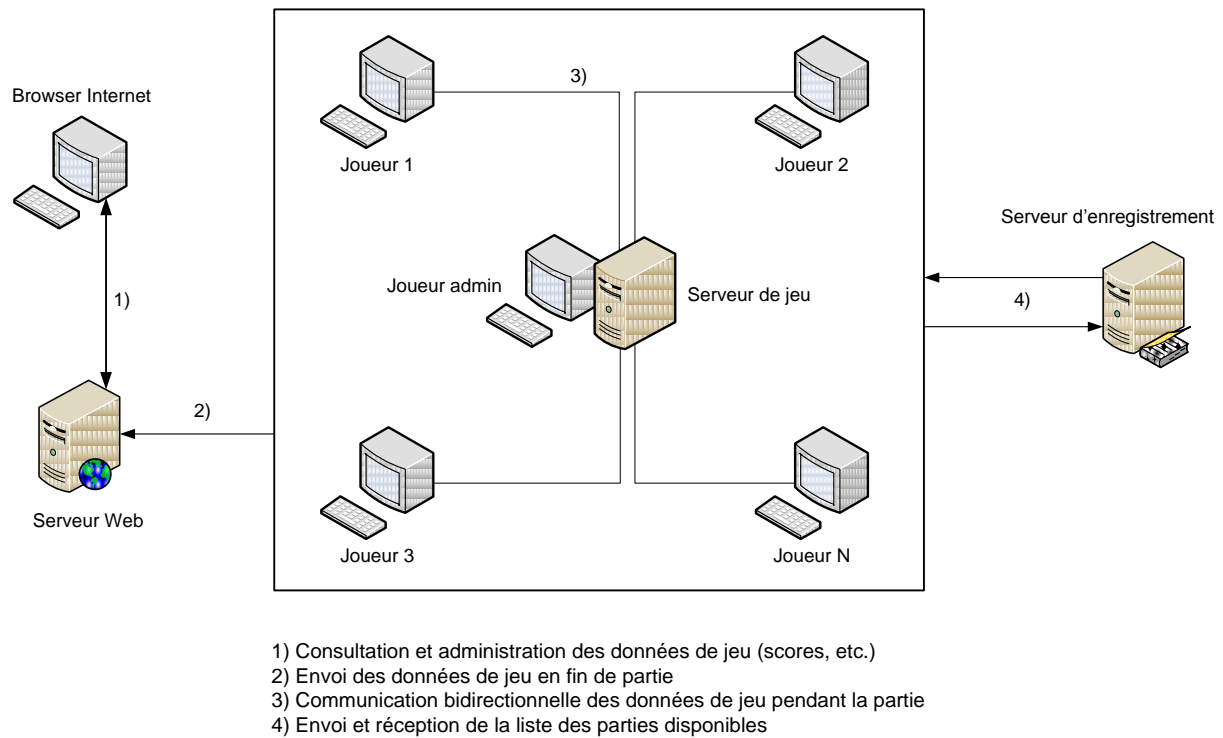


Figure 2.2.1 : Schéma global des acteurs

2.2.2 Etapes de lancement du jeu

Dans ce chapitre, nous vous présentons le cas d'utilisation du lancement de la terminaison d'une partie de jeu. Le scénario relatif est présenté à la suite de ce schéma.

2.2.2.1 Cas d'utilisation

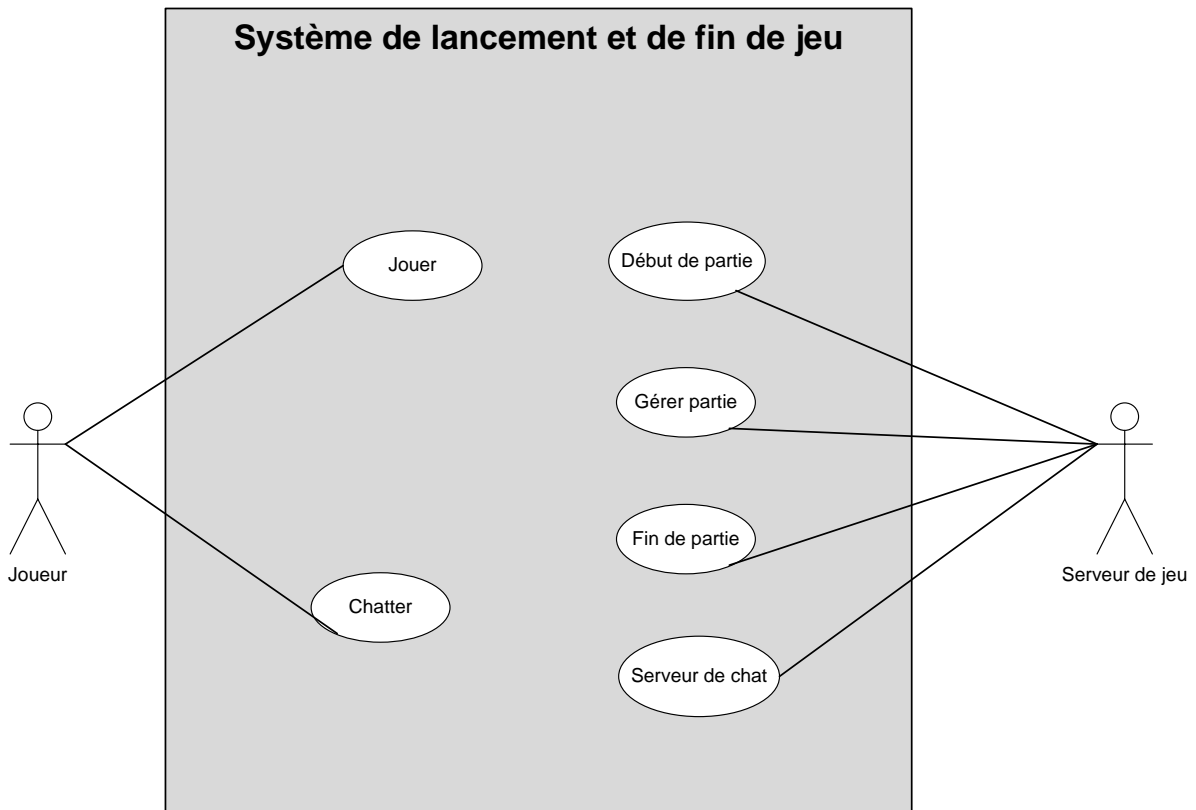


Figure 2.2.2 : Système de lancement et de fin de jeu

2.2.2.2 Acteurs principaux

- Joueur
- Serveur de jeu

2.2.2.3 Scénario principal (succès)

Joueur (Jouer)	Serveur de jeu (Début de partie)
<ol style="list-style-type: none">1. Le joueur choisit de rejoindre une partie ou en crée une2. Le joueur se connecte au serveur de jeu5. Un joueur peut demander de changer d'équipe7. Lorsque le joueur ayant créé la partie le décide, la partie commence	<ol style="list-style-type: none">3. Le serveur accepte la demande et la traite (création de la partie ou ajout du joueur à une partie)4. Le cas échéant, informe les autres joueurs de l'arrivée du dernier6. Le serveur informe les joueurs connectés du changement d'équipe effectué8. Le serveur valide le choix du joueur et informe les autres de ce changement9. Le serveur lance la partie à proprement dit

2.2.3 Système de gestion de partie

2.2.3.1 Acteurs principaux

- Joueur
- Serveur de jeu

2.2.3.2 Scénario principal

Joueur (Jouer)	Serveur de jeu (Gérer partie)
<ol style="list-style-type: none">1. Le joueur X veut effectuer une action (Pose de tour, envoi de créatures, etc....)4. La tour est affichée sur la carte de chaque joueur dans la partie du joueur X	<ol style="list-style-type: none">2. Le serveur contrôle que l'action puisse être effectuée3. Informe les joueurs que le joueur X a effectué une action en la spécifiant
Echecs	
	Erreur ! Source du renvoi introuvable. Le serveur refuse l'action
Un message d'erreur est affiché chez le joueur X	

2.2.4 Système de fin de partie

2.2.4.1 Acteurs principaux

- Joueur
- Serveur de jeu

2.2.4.2 Scénario principal

Joueur (Jouer)	Serveur de jeu (Fin de partie)
<ol style="list-style-type: none">1. Le joueur hôte met fin à la partie3. Le joueur quitte la partie	<ol style="list-style-type: none">2. Le serveur met fin à la partie pour chaque joueur et leur envoie le score de chaque joueur ainsi que leur état (nombre de vies etc....)

2.2.5 Système de chat

2.2.5.1 Acteurs principaux

- Joueur
- Serveur de jeu

2.2.5.2 Scénario principal (succès)

Joueur (Chatter)	Serveur de jeu (Serveur de chat)
<ol style="list-style-type: none">1. Ecrit un message4. Affiche le message retransmis par le serveur	<ol style="list-style-type: none">2. Reçoit le message3. Envoie le message à tous les joueurs de la partie

2.4 Serveur d'enregistrement

Voici le cas d'utilisation du serveur d'enregistrement avec un scénario présenté à la page suivante.

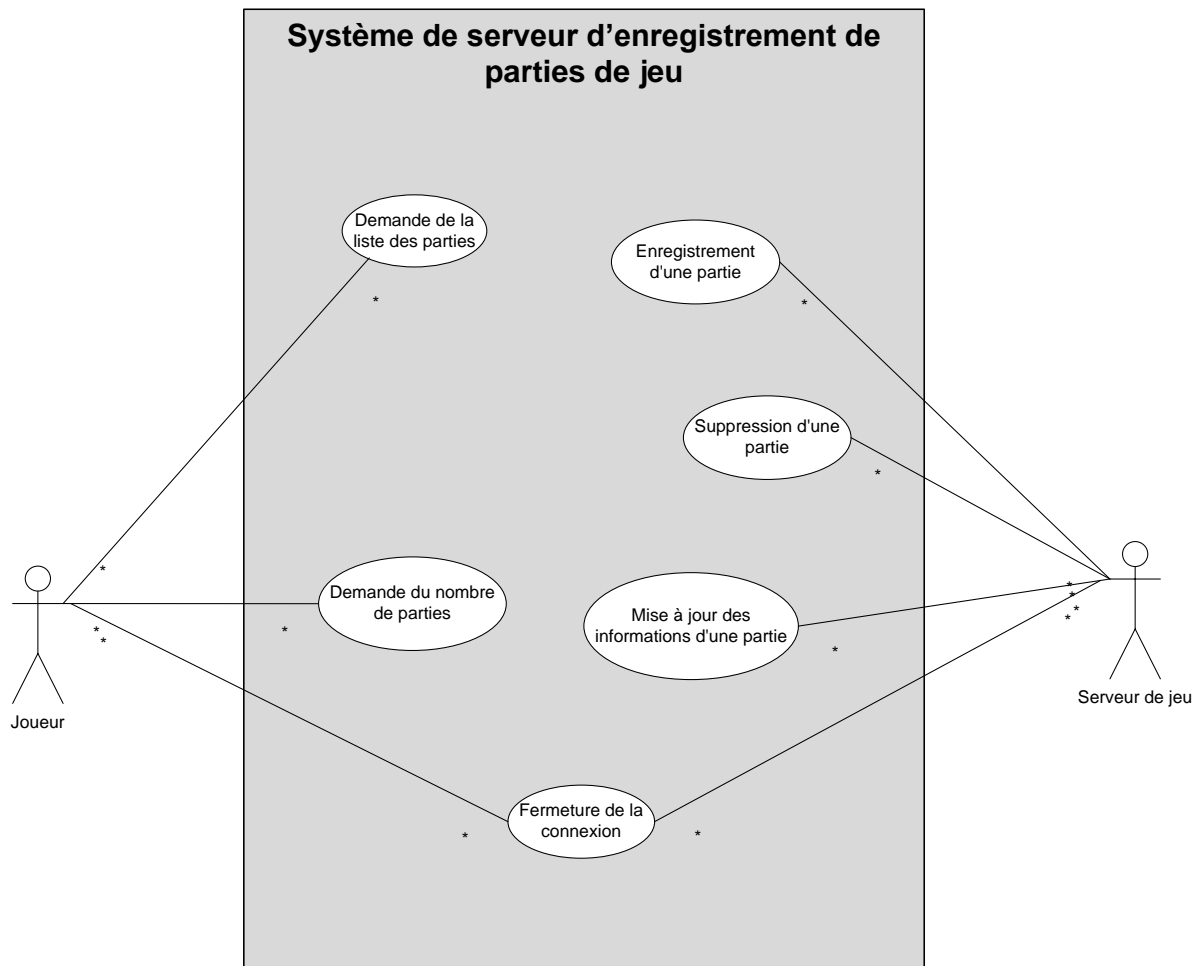


Figure 2.4.1 : Système de serveur d'enregistrement de parties de jeu

2.4.1 Acteurs principaux

- Joueur (d'une partie)
- Hébergeur (d'une partie)

2.4.2 Scénario 01 : Demande de la liste des parties (succès)

Joueur d'une partie	Hébergeur d'une partie
<ol style="list-style-type: none">le joueur demande la liste des parties (hébergeurs) disponiblesLe système fournit la liste des parties (hébergeurs) en attente de joueurs	<ol style="list-style-type: none">L'hébergeur crée une partieLe système enregistre la partie nouvellement créée dans sa base de donnéesL'hébergeur attend que des joueurs rejoignent la partie qu'il vient de créer

2.4.3 Scénario 02 : Demande du nombre de parties (succès)

Joueur d'une partie	Hébergeur d'une partie
<ol style="list-style-type: none">le joueur demande la liste des parties (hébergeurs) disponiblesLe système fournit la liste des parties (hébergeurs) en attente de joueursLe joueur choisit voit le nombre de parties créées dans la liste qui s'affiche	<ol style="list-style-type: none">L'hébergeur crée une partieLe système enregistre la partie nouvellement créée dans sa base de donnéesL'hébergeur attend que des joueurs rejoignent la partie qu'il vient de créer

2.4.4 Scénario 03 : Enregistrement d'une partie (succès)

Joueur d'une partie	Hébergeur d'une partie
	<ol style="list-style-type: none">1. L'hébergeur crée une partie2. Le système enregistre la partie nouvellement créée dans sa base de données3. L'hébergeur attend que des joueurs rejoignent la partie qu'il vient de créer

2.4.5 Scénario 04 : Suppression d'une partie (succès)

Joueur d'une partie	Hébergeur d'une partie
<ol style="list-style-type: none">1. Les joueurs se connectent à la partie2. La partie commence	<ol style="list-style-type: none">1. L'hébergeur crée une partie2. Le système enregistre la partie nouvellement créée dans sa base de données3. L'hébergeur attend que des joueurs rejoignent la partie qu'il vient de créer4. L'hébergeur signale au système que la partie est complète et qu'elle va commencer5. Le système efface l'enregistrement de sa base de données

2.4.6 Scénario 05 : Mise à jour des informations d'une partie

Joueur d'une partie	Hébergeur d'une partie
<p>1. le joueur demande la liste des parties (hébergeurs) disponibles</p> <p>2. Le système fournit la liste des parties (hébergeurs) en attente de joueurs</p> <p>3. Le joueur choisit une partie dans la liste qu'il vient de recevoir</p> <p>4. Le joueur se connecte à la partie</p>	<p>1. L'hébergeur crée une partie</p> <p>2. Le système enregistre la partie nouvellement créée dans sa base de données</p> <p>3. L'hébergeur attend que des joueurs rejoignent la partie qu'il vient de créer</p> <p>4. L'hébergeur signale au système qu'une place de moins est disponible.</p> <p>5. Le système met à jour l'enregistrement de la partie en diminuant le nombre de joueurs encore autorisés à jouer de 1.</p>

2.4.7 Scénario 06 : Fermeture de la connexion (succès)

Joueur d'une partie	Hébergeur d'une partie
<p>1. Le système ferme la connexion et libère les ressources réseau</p>	<p>1. L'hébergeur crée une partie</p> <p>2. Le système enregistre la partie nouvellement créée dans sa base de données</p> <p>3. L'hébergeur quitte le programme et arrête son exécution soudainement</p>

2.5 Serveur de Jeu

Voici le cas d'utilisation du serveur d'enregistrement. D'un point de vue de simplicité de réalisation et de compréhension, le scénario relatif sera **présenté directement dans le protocole lié**.

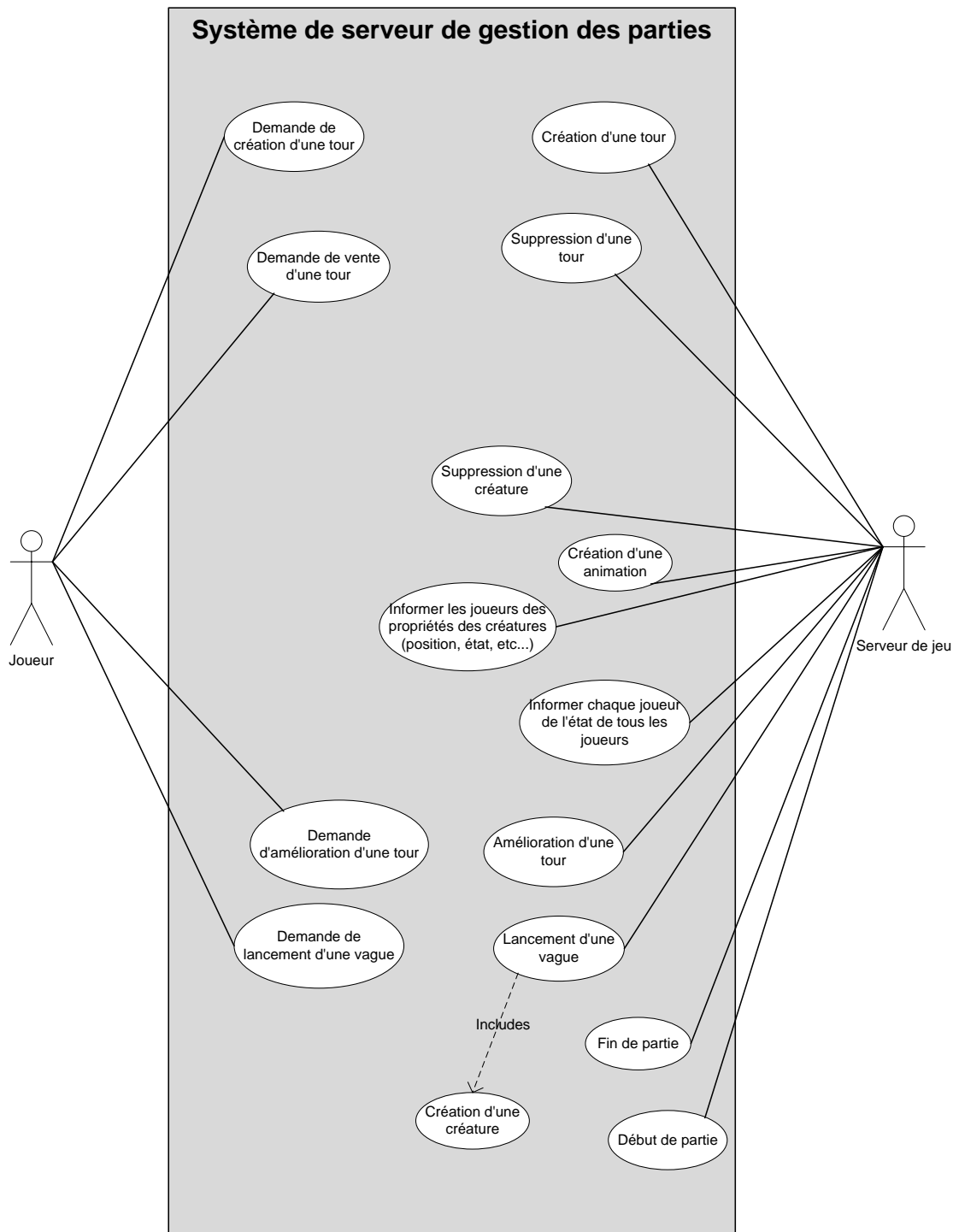


Figure 2.5.1 : Système de serveur de gestion des parties

2.6 Serveur Web

Cette étape étant optionnelle, nous n'avons pas réalisé les schémas relatifs au serveur web.

3 Conception du projet

Tous les messages de nos protocoles sont formatés en JSON. Dans le chapitre qui suit, nous vous présentons cette notation qui a pour grand avantage d'être extrêmes légère vis-à-vis d'une notation XML ou autre. Nous avons donc décidé d'utiliser comme format de message le JSON car il se prête très bien à cette situation et est très en vogue actuellement. Il s'agit principalement d'un format très standardisé que n'importe quelle entité peut comprendre, depuis n'importe quel langage.

3.1 JSON

« **JSON** (JavaScript Object Notation – Notation Objet issue de JavaScript) est un format léger d'échange de données. Il est facile à lire ou à écrire pour des humains. Il est aisément analysable ou « générable » par des machines. Il est basé sur un sous-ensemble du langage de programmation **JavaScript**.



JSON est un format texte complètement indépendant de tout langage, mais les conventions qu'il utilise seront familières à tout programmeur habitué aux langages descendant du C, comme par exemple : C++, C#, Java, JavaScript, Perl, Python et bien d'autres. Ces propriétés font de JSON un langage d'échange de données idéal. »

JSON se base sur **deux structures** :

- Une collection de couples nom/valeur. Divers langages la réifient par un objet, un enregistrement, une structure, un dictionnaire, une table de hachage, une liste typée ou un tableau associatif.
- Une liste de valeurs ordonnées. La plupart des langages la réifient par un tableau, un vecteur, une liste ou une suite.

Ces structures de données sont universelles. Pratiquement tous les langages de programmation modernes les proposent sous une forme ou une autre. Il est raisonnable qu'un format de données interchangeable avec des langages de programmation se base aussi sur ces structures.

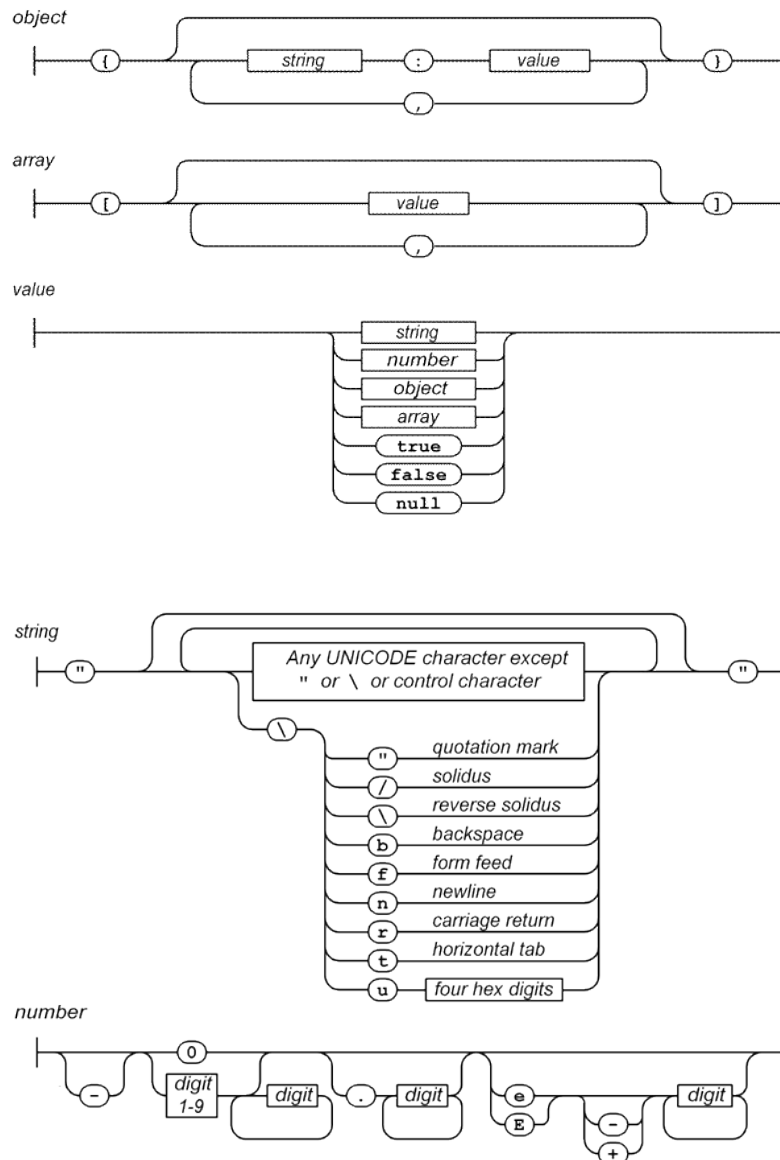
En JSON, elles prennent les formes suivantes :

Un objet, qui est un **ensemble de couples nom/valeur non ordonnés**. Un objet commence par { (accolade gauche) et se termine par } (accolade droite). Chaque nom est suivi de : (deux-points) et les couples nom/valeur sont séparés par , (virgule).

Un tableau est une **collection de valeurs ordonnées**. Un tableau commence par [(crochet gauche) et se termine par] (crochet droit). Les valeurs sont séparées par , (virgule).

Une valeur peut être soit une **chaîne de caractères entre guillemets**, soit un **nombre**, soit **true** ou **false** ou **null**, soit un **objet** soit un **tableau**. Ces structures peuvent être imbriquées.

Mis à part quelques détails d'encodage, voilà qui décrit le langage dans son intégralité.



Une **chaîne de caractères** est très proche de ses équivalents en C ou en Java. Un **nombre** est très proche de ceux qu'on peut rencontrer en C ou en Java, sauf que les formats octal et hexadécimal ne sont pas utilisés. De l'espace blanc est autorisé entre tous lexèmes.

Source : <http://www.json.org/>

3.2 Protocole d'échange entre le client et le serveur

Vous trouverez en annexe le protocole JSON d'échange entre le client et le serveur des principaux messages de communications.

A chaque message sera associé un scénario d'utilisation correspondant selon le cahier des charges définis dans les premières étapes du projet. Nous avons utilisé ce protocole pour nous permettre de nous répartir les tâches entre l'implémentation du client et du serveur. En effet chacun avait ce document comme référence de la syntaxe et de la sémantique de chaque message. Il s'agit donc du lien entre les clients considérés comme des vues et le serveur considéré comme le modèle et le contrôleur.

3.2.1 Serveur d'enregistrement

Le protocole du serveur d'enregistrement décrit comment communiquent les deux acteurs l'utilisant, à savoir l'hébergeur d'une partie ainsi que le joueur d'une partie. Il est principalement question de l'enregistrement d'une nouvelle partie ainsi que la récupération de la liste des parties disponibles.

Nous présentons en annexe la description des acteurs ainsi que le scénario de succès associés. Nous présentons de plus un schéma de communication entre les deux parties prenantes du système.

3.2.2 Serveur de jeu

Le protocole du serveur de jeu décrit les échanges entre le client et le serveur concernant les interactions de jeu. Le client va envoyer des changements d'états ainsi que des demandes de création d'objet, demandes auxquelles le serveur va donner une confirmation en cas de réussite ou une erreur en cas d'échec.

Le serveur quant à lui va envoyer à tous les clients l'état courant de l'ensemble des objets, qu'ils aient été créés par le client cible ou d'autres clients.

Le schéma de protocole ainsi que les scénarios d'utilisations associés sont présentés en annexe « Serveur de jeu, dialogue client/serveur JSON ». Une checklist ainsi qu'un bref descriptif des services est également inclus dans le document.

3.3 Diagramme d'activité général

Ce diagramme illustre un cheminement logique du programme entre les différents cas : partie solo, initier une partie réseau ou rejoindre une partie réseau.

On remarque que le serveur n'est créé que chez le joueur qui initie une partie réseau, un joueur voulant rejoindre une partie réseau devant se connecter en tant que client à ce serveur. Un client peut donc assumer soit le rôle de client seul, soit le rôle de client/serveur. En cas de partie solo, il n'y a pas de serveur créé, le client joue uniquement sur les ressources de sa machine.

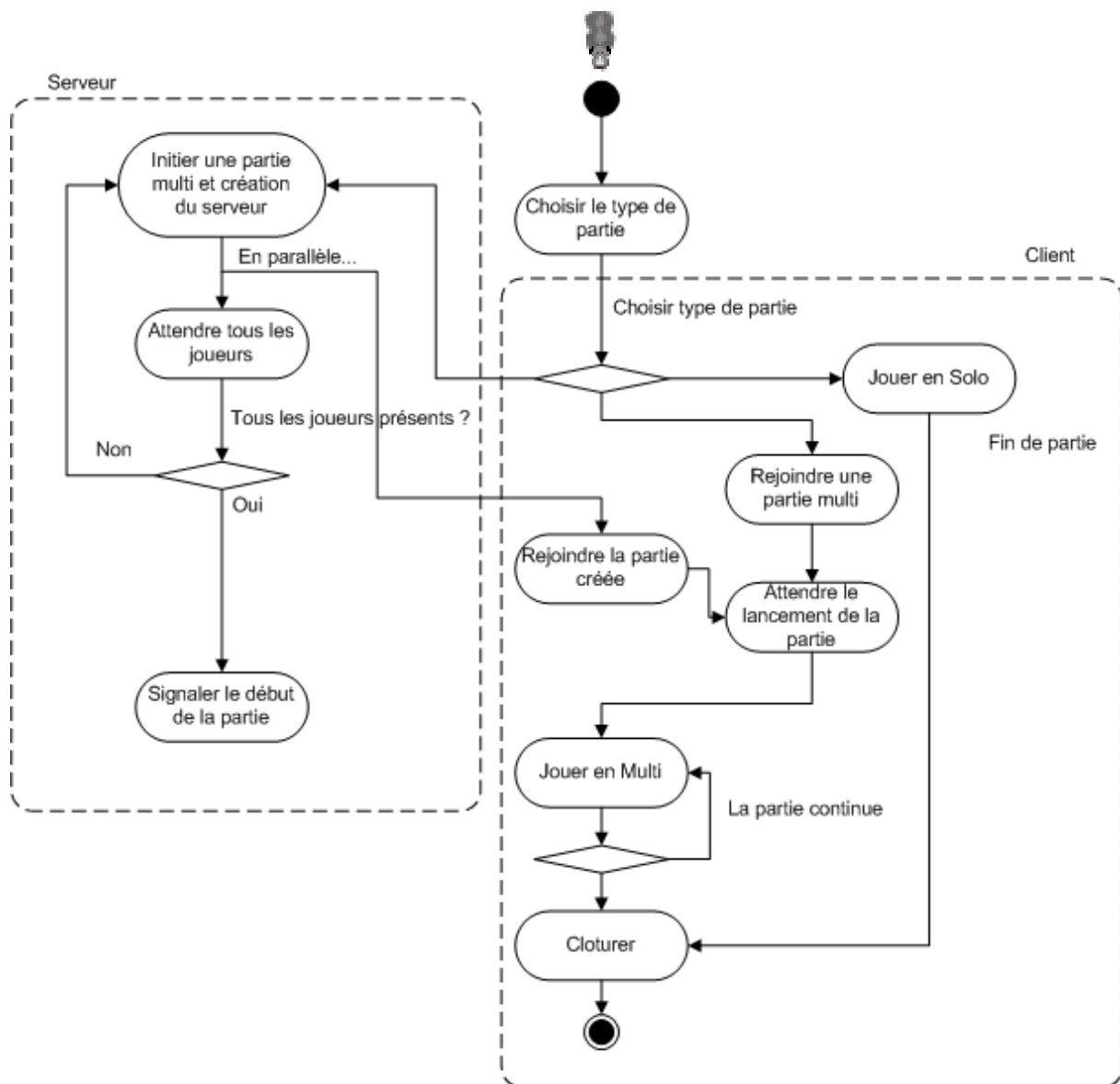


Figure 3.3.1 : Diagramme d'activité général

3.4 Architecture de l'application

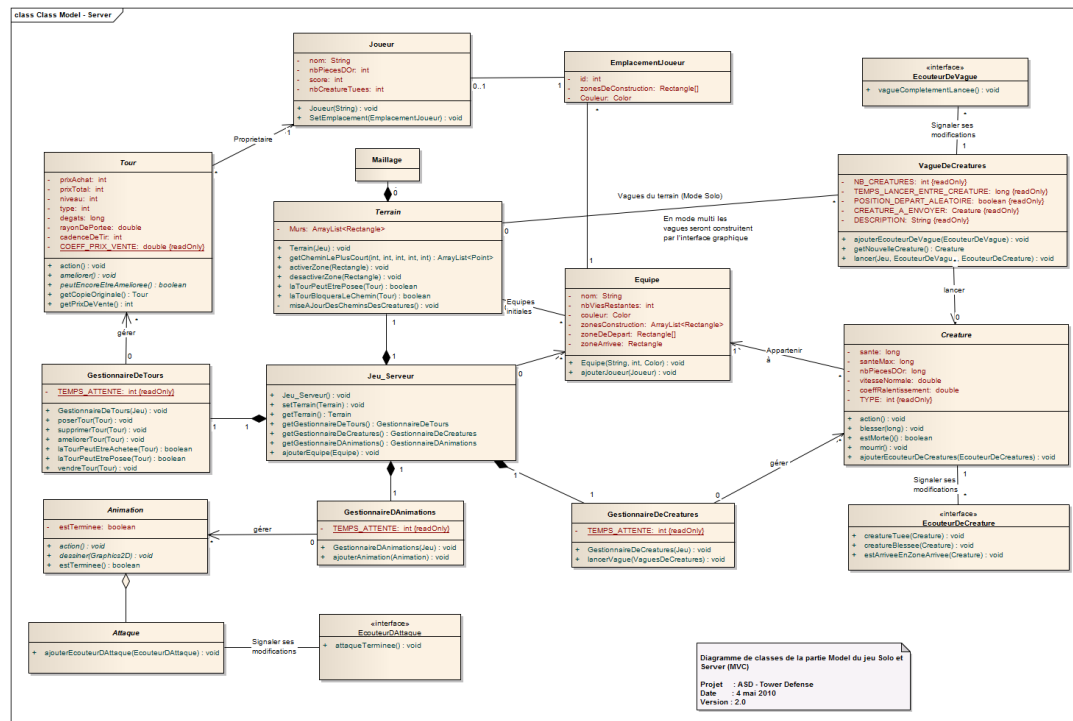
L'architecture d'ASD - Tower Defense a subi d'énormes changements dans le but d'intégrer correctement la notion de partie réseau. En effet, à la place de créer une nouvelle architecture pour le jeu multi-joueurs, nous avons décidé d'étendre le moteur du jeu solo de la version 1.0 pour incorporer les différents éléments constituant cette deuxième version. Pour être franc, cette tâche ne fût pas une mince affaire. En effet, nous avons dû intégrer différentes notions et ajuster toute la partie solo pour qu'elle s'adapte aux nouveaux changements. Les nouvelles notions intégrées sont listées ci-dessous :

- La gestion des équipes
- La gestion des joueurs
- La gestion des emplacements des joueurs sur le terrain multi-joueurs
- La gestion du lancement de vagues par les joueurs
- L'adaptation des terrains pour l'intégration des nouveaux éléments notamment la sérialisation des terrains pour instanciation dynamique (choix de la carte de jeu)
- L'adaptation de la gestion des tours (intégration des propriétaires des tours)
- L'adaptation de la gestion des créatures (intégration des propriétaires et cible)
- L'adaptation des vues en fonction des changements du model

Les responsables de l'architecture sont très satisfaits de cette nouvelle architecture. Elle modélise maintenant un monde cohérent et particulièrement en terme de maintenabilité. En effet l'architecture étant aujourd'hui très propre, les futures améliorations s'intégreront facilement dans ce modèle. Il était très profitable pour nous de revoir cette partie d'une part pour l'intégration des fonctionnalités réseaux mais aussi pour l'avenir du projet. Notons que la plupart des membres du projet portent un intérêt particulier à continuation de ce jeu.

3.4.1 Moteur du jeu

Voici le diagramme de classes représentant le moteur du jeu. Il correspond aux parties Model & Contrôleur dans une architecture MVC. Celui-ci est utilisé presque tel-quels pour le mode solo et le serveur de jeu. Le client d'une partie multi-joueurs, quant à lui, met également en place une structure similaire mais celle-ci est tout de même quelque peu épurée car ce dernier ne gère pas le jeu, il affiche simplement les états de jeu que le serveur lui fournit. Le client utilise tout de même les gestionnaires mais uniquement pour calculer certains éléments du jeu afin d'éviter que le serveur doive envoyer en tout temps l'état du jeu.



La classe **Jeu** est la classe maîtresse de ce model. Elle contient et fourni trois gestionnaires et le terrain.

Chaque **gestionnaire** encapsule et gère toutes les entités d'un type d'élément du terrain (**Tour**, **Créature**, **Animation**). Une tâche sera créée spécialement pour la gestion de ses éléments.

Une **tour** appartient à un joueur. Un **joueur** construit des tours dans un **emplacement** et celui-ci appartient à une **équipe**.

C'est le **terrain** qui fournit les **équipes** initiales et les **vague de créatures** du mode solo. Le terrain contient le **maillage** pour les déplacements des créatures. Ce dernier est détaillé dans la version 1.0 du jeu. (c.f. rapport ASD-TD v1 0).

Une **créature** à été lancée par une équipe via une **vague de créatures**. En mode versus les vagues de créatures seront construites avec l'interface graphique.

Les **animations** sont des éléments divers pouvant être affiché sur le terrain. Elles sont par exemple créées par la mort d'une créature pour indiquer le gain d'argent. Les **attaques** sont des animations créées par les tours (flèche, boulet, etc.). C'est elles qui blessent les **créatures**.

La **fenêtre de jeu** implémentera les diverses interfaces de se model pour se tenir au courant des modifications du model.

Figure 3.4.1 : Schéma du moteur du jeu

Ce schéma de plus grande taille est fourni en annexe.

3.4.2 Intégration du pattern Observable / Observé

La notion d'observer est un élément fondamental dans notre projet, sans celui-ci, il aurait été très difficile de réaliser une architecture dite MVC propre et respectueuse des principes de bonne pratique. L'architecture présentée dans la Figure 3.4.1 met en place un certains nombres d'écouteurs (de Listener qui est terme couramment utiliser par la librairie Swing). Toutes les implémentations de ce modèle ont été réalisées à notre manière. En effet, il été bien plus simple pour nous de fournir directement des interfaces permettant de définir nos observateurs et de mettre en place une structure simple pour chaque classe observée.

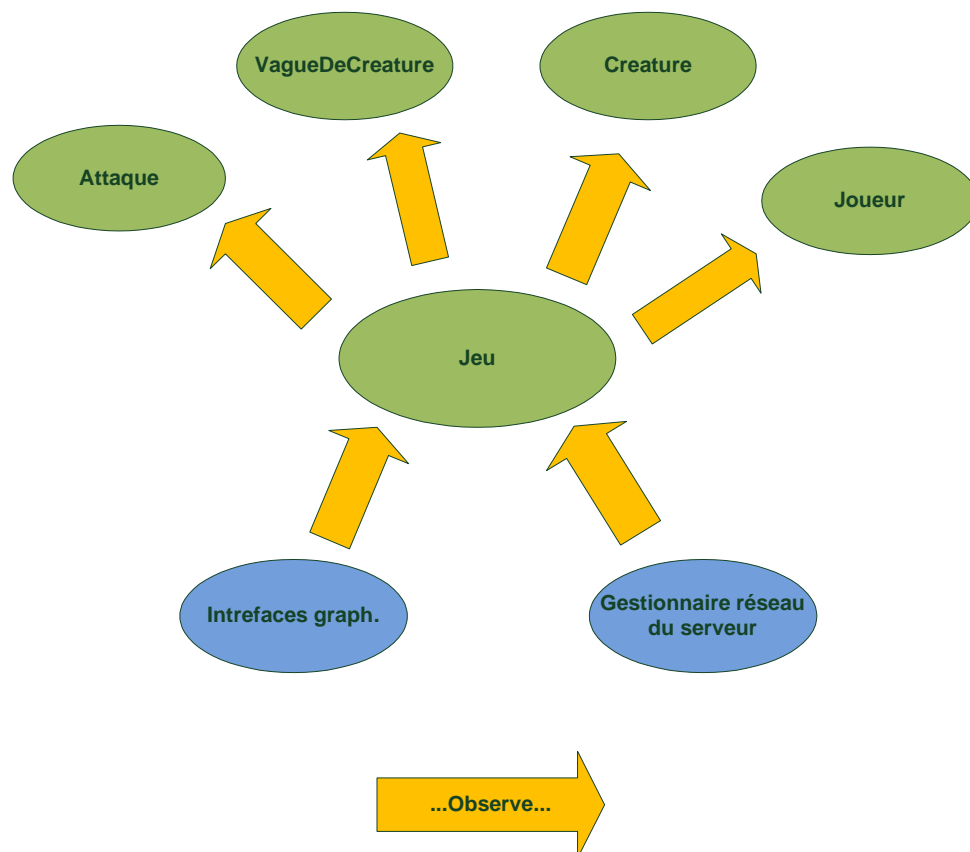


Figure 3.4.2 : Qui observe quoi ?

La figure 3.4.2 permet d'illustrer les différentes classes observatrices de l'application. Nous pouvons remarquer ici la présence du jeu qui en fait est une sorte de contrôleur qui écoute les principaux éléments du modèle. S'il vient à recevoir une notification, il effectue les opérations qu'il lui semble nécessaires et transmet ensuite (s'il considère qu'il doit) à toutes les entités qui l'observent. Il n'y a pas que des notifications transférées, le jeu indique également ses propres notifications. Les deux observateurs principaux du jeu sont les interfaces graphiques et le gestionnaire réseaux du serveur. Les interfaces graphiques sont notifiées afin de d'informer l'utilisateur final des changements de l'état du jeu. Le gestionnaire réseaux est également notifié de ces changements afin de transmettre aux clients connectés les divers modifications qui ont eut lieu. Ce dernier tri les notifications reçues par le jeu et n'envoie que le strict minimum afin de ne pas trop surcharger le réseau.

Voici la liste des interfaces réalisées avec leurs méthodes de notification :

Interface EcouteurDAttaque (observée : Attaque , observateur : Jeu)	
attaqueTerminee(Tour attaquant, Creature cible)	Informe l'écouteur de la fin d'une attaque
Interface EcouteurDeCreature (observée : Creature , observateur : Jeu)	
creatureBlessee(Creature creature),	Informe l'écouteur qu'une créature à été blessée
creatureTuee(Creature creature, Joueur tueur)	Informe l'écouteur de la mort d'une créature
creatureArriveeEnZoneArrivee(Creature creature)	Informe l'écouteur l'arrivée d'une créature
Interface EcouteurDeVague (observée : VagueDeCreatures , observateur : Jeu)	
vagueEntierementLancee(VagueDeCreatures vagueDeCreatures)	Informe l'écouteur du lancement de vague termine
Interface EcouteurDeJoueur (observée : Joueur , observateur : Jeu)	
joueurMisAJour(Joueur joueur)	Informe l'écouteur qu'un joueur a été modifié
Interface EcouteurDeJeu (observée : Jeu , observateur : Les interfaces et les gestionnaires réseaux)	
partieInitialisee()	Informe l'écouteur que la partie à été initialisée
partieDemarree()	Informe l'écouteur que la partie a démarrée
partieTerminee(ResultatJeu resultatJeu)	Informe l'écouteur que la partie est termine
etoileGagnee()	Informe l'écouteur que le joueur a gagné une étoile
joueurAjoute(Joueur joueur)	Informe l'écouteur qu'un joueur a rejoint la partie
joueurMisAJour(Joueur joueur)	Informe l'écouteur qu'un joueur a été mis à jour
equipeAPerdue(Equipe equipe)	Informe l'écouteur qu'une equipe a perdue
tourPosee(Tour tour)	Informe l'écouteur qu'une tour à été posée
tourVendue(Tour tour)	Informe l'écouteur qu'une tour à été vendue
tourAmelioree(Tour tour)	Informe l'écouteur qu'une tour à été améliorée
vagueEntierementLancee(VagueDeCreatures vague)	Informe l'écouteur du lancement de vague termine
creatureAjoutee(Creature creature)	Informe l'écouteur qu'une créature à été blessée
creatureBlessee(Creature creature),	Informe l'écouteur qu'une créature à été blessée
creatureTuee(Creature creature, Joueur tueur)	Informe l'écouteur de la mort d'une créature
creatureArriveeEnZoneArrivee(Creature creature)	Informe l'écouteur l'arrivée d'une créature
animationAjoutee(Animation animation)	Informe l'écouteur qu'une animation à été ajoutée
animationTerminee(Animation animation)	Informe l'écouteur qu'une animation à été terminée

3.4.3 Les différentes classes Jeu

Le schéma de la **Erreur ! Source du renvoi introuvable.** illustre les différentes classes qui dérivent du moteur du jeu. Chacune de ces classes redéfinissent une partie des méthodes du moteur afin d'adapter son fonctionnement aux spécifications propre du mode de jeu.

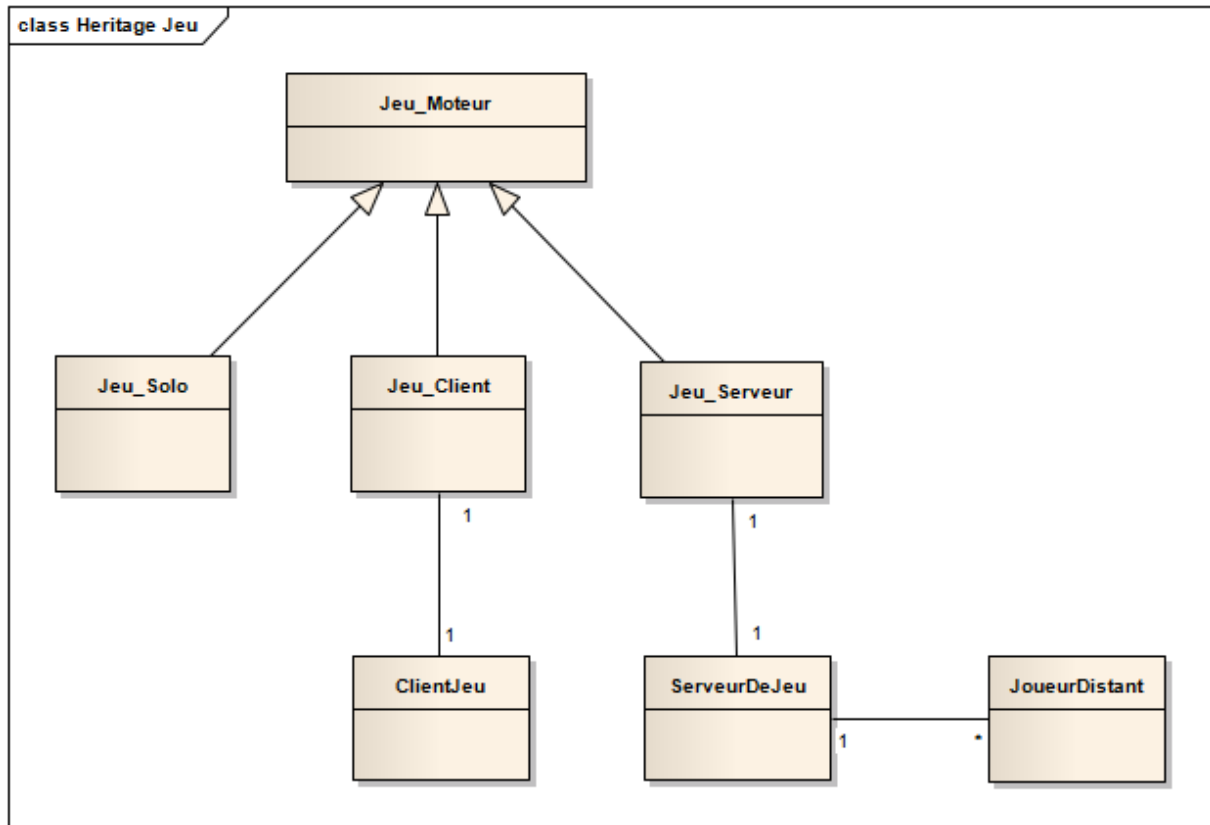


Figure 3.4.3 :Classes dérivants du moteur et gestionnaires réseaux

Le schéma de la figure 3.4.3 illustre également les classes de gestion des échanges de messages en les clients et le serveur. Notez que le Serveur de jeu (gestionnaire réseau) possède pour chaque client connecté un classe de gestion d'envoi de messages.

3.4.4 Réseau

Dans la figure suivante, nous présentons le diagramme de modélisation de domaine pour le paquetage *Reseau*, responsable de fournir les fonctionnalités réseau de base.

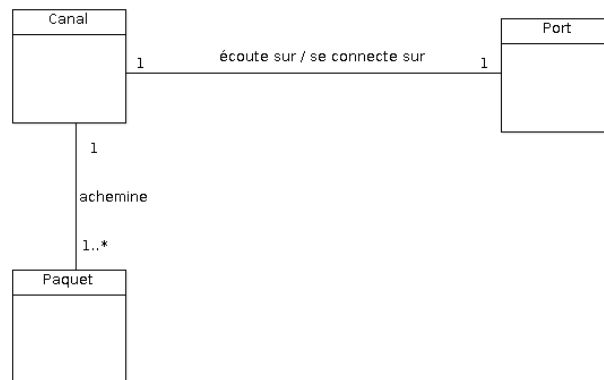


Figure 3.4.4 : Classe primitive du réseau

3.5 Charte graphique

Nous présentons ici nos premières idées de l'élaboration des interfaces utilisateurs de notre jeu. Nous nous concentrerons principalement sur la partie réseau.



Menu principal

Premier menu lors du lancement du jeu. Trois choix sont offerts au joueur :

1. Partie solo
2. Créer une partie multi-joueurs
3. Rejoindre une partie multi-joueurs

1. Mode solo

Cette fenêtre permet au joueur de faire une partie seul. 4 terrains s'offre à lui. Il en sélectionne un et la partie commence.

Note : La fenêtre de jeu solo ne sera pas présentée dans se rapport



2. Créer une partie réseau

Le joueur introduit les caractéristiques du jeu (terrain, équipes aléatoires, etc.) et créer une partie réseau.



2.1 Attente de joueurs

La partie est créée, il faut maintenant attendre les joueurs. Le joueur qui crée la partie est considéré comme un modérateur de la partie en question. Il peut notamment changer la composition des équipes et démarrer le jeu à tous moments.



3.1 Attente de joueurs

...Le client arrive alors dans le formulaire d'attente d'autres joueurs où il peut voir les joueurs déjà connectés. Il peut changer d'équipe si de la place est disponible. Il attend ensuite que tout le monde soit connecté ou que le modérateur décide de lancer la partie.

La partie commence...



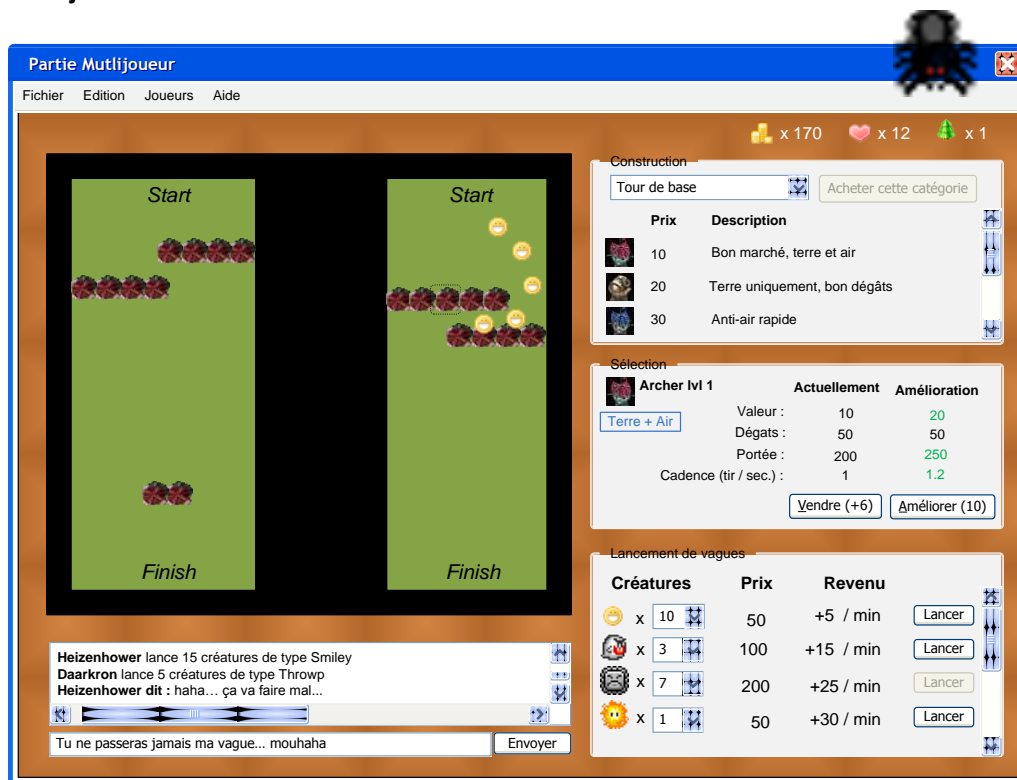
3. Rejoindre une partie réseau

Plaçons nous maintenant du côté du client qui veut se connecter. Si le serveur d'enregistrement est atteignable, le client verra la liste des serveurs disponibles avec leurs caractéristiques.

Le client sélectionne une partie et la rejoint...



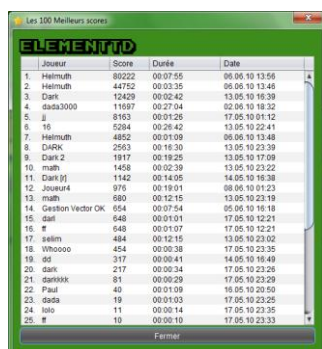
4. Jeu multi-joueurs en mode Versus



Voici la fenêtre de jeu réseau. Grâce aux divers menus, le joueur pourra entre autre créer et gérer des tours, voir des informations sur les créatures (Box Sélection) et envoyer des vagues de créatures à l'équipe adverse. Un petit chat fera peut-être son apparition si le temps le permet (à priori non planifié dans les itérations).

3.6 S rialisation des scores et des terrains de jeu

Lorsqu'on joue   un jeu, il y a dans la plupart des cas un score ainsi que d'autres informations qui doivent  tre persistantes tant que l'utilisateur garde l'application sur son ordinateur. Dans notre cas, il s'agit des scores, que l'utilisateur doit retrouver d'une partie   l'autre, et ceci m me apr s avoir quitt  et relanc  l'application. Pour ce faire, nous avons utilis  la possibilit  qu'il existe en Java de « s rialiser » des morceaux de code (en g n ral des classes) afin de pouvoir directement les sauvegarder dans des fichiers persistants sur le disque dur. En effet, gr ce   cette possibilit , on peut enregistrer l' tat de l'ex cution de l'application dans un fichier afin de retrouver cet  tat plus tard.

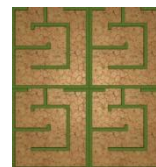


	Joueur	Score	Dur�e	Date
1.	Heimuth	80222	00:07:55	06.05.10 13:56
2.	Heimuth	44752	00:03:35	06.05.10 13:46
3.	DanK	12429	00:02:42	13.05.10 16:39
4.	data3000	11697	00:27:04	02.05.10 16:32
5.	J	8163	00:01:26	17.05.10 01:12
6.	16	5284	00:26:42	13.05.10 22:41
7.	Heimuth	4852	00:01:09	06.05.10 13:48
8.	DARK	2583	00:16:30	13.05.10 23:39
9.	DanK 2	1917	00:19:25	13.05.10 17:09
10.	math	1458	00:02:39	13.05.10 23:22
11.	DanK [?]	1142	00:14:05	14.05.10 16:38
12.	Jochem4	975	00:19:01	08.05.10 01:23
13.	math	880	00:12:15	13.05.10 23:19
14.	Gaston Vector OK	654	00:07:54	05.05.10 16:18
15.	dan	548	00:01:01	17.05.10 12:21
16.	#	548	00:01:07	17.05.10 12:21
17.	selim	484	00:12:15	13.05.10 23:02
18.	thiboo	454	00:00:38	17.05.10 23:35
19.	85	317	00:00:41	14.05.10 16:49
20.	dan	217	00:00:34	17.05.10 23:26
21.	darkkk	91	00:00:29	17.05.10 23:29
22.	Paul	40	00:01:09	16.05.10 20:50
23.	data	19	00:01:03	17.05.10 23:25
24.	jolo	11	00:00:14	17.05.10 23:35
25.	#	10	00:00:10	17.05.10 23:33

Pour la gestion des scores, nous avons donc enregistr  dans un fichier l' tat de ces scores   un temps t du jeu, en principe lors de la fin d'une partie. Ainsi, quand un joueur d cide d'enregistrer son score, nous « s rialisons » les donn es associ es et nous les enregistrons dans un fichier dit « s rialis  ». Par la suite, lors du lancement de l'application, cette derni re se charge de r cup rer ce fichier et d'en reprendre les informations dans l' tat o  elles avaient  t  laiss es. On peut ainsi r cup rer facilement les scores d'un joueur. Les fichiers de scores s rialis s portent l'extension « .ms » et se trouvent dans le dossier « donnees » se trouvant   la racine de l'arborescence de l'application.

Fen tre des scores

Le m me principe a  t  utilis  pour les terrains de jeu. En effet, nous les avons « s rialis  » et enregistr  dans des fichiers dont l'extension est « .map ». Ces fichiers se trouvent dans le dossier « maps »   la racine de l'arborescence de l'application.



Ainsi, lorsque nous cr ons un nouveau terrain de jeu, nous le s rialisons dans un fichier .ms afin de pouvoir ensuite le redistribuer aux joueurs. On peut donc par la suite proposer en t l chargement ces fichiers .ms directement sur le site Internet de l'application, ce qui permet au joueur d'installer de nouveaux terrains de jeu en quelques clics.

Cette possibilit  est tr s pratique et donne un c t  dynamique au jeu qui a toujours un bon effet aupr s de l'utilisateur et  veille encore son int r t ainsi que sa curiosit  pour le jeu.

L'avantage de la s rialisation est qu'elle est simple, mais  galement que les fichiers produits, dits « s rialis s » sont dans un format binaire, illisible pour l'utilisateur. Il ne peut donc ni tricher, ni modifier ses scores comme il pourrait le faire si nous avions simplement sauve  les scores dans un fichier texte. En effet, dans ce dernier cas, le joueur pourrait simplement ouvrir le fichier texte en question et le modifier   sa guise, ce qui ne serait pas correct.

En ce qui concerne les terrains, on peut  galement imaginer par la suite cr er une application permettant de cr er des terrains de jeu. Cette application aurait une fonction de sauvegarde du terrain cr   par l'utilisateur, et le fichier qui en r sulterait serait lui aussi « s rialis  » au format « .map ». Il suffirait ensuite de copier ce fichier dans le dossier « maps » de notre application et le joueur pourrait avoir ses propres terrains qu'il s'est lui-m me cr  es. Il pourrait  galement les distribuer   d'autres joueurs sur Internet.

Il faut encore noter que le syst me de s rialisation a  t  utilis  comme substitut d'une base de donn es. En effet, il  tait demand  au d part d'interagir avec une base de donn es, ce qui n'est malheureusement pas compatible   premi re vue avec notre projet. Nous avons donc d cid  de substituer ce point par celui-ci, c'est- -dire la s rialisation des donn es.

4 Gestion de projet

4.1 Rôle des participants au sein du groupe

Voici un tableau présentant les différents rôles (standard) des membres du groupe.

	Aurélien	Lazhar	Pierre-Do.	Romain
Représentants des utilisateurs		✕		✕
Chef de projet	✕			
Analyste	✕			✕
Architecte, concepteur	✕	✕	✕	✕
Programmeur	✕	✕	✕	✕
Responsable des tests			✕	
Responsable de la configuration		✕		
Responsable Apéro				✕

4.2 Plan d'itérations initial

Dans ce chapitre nous vous présentons les différentes itérations prévues pour ce projet. Celui-ci se déroule sur **8 semaines** (phase d'initialisation comprise) à raison d'environ 4 périodes de 45 minutes par semaine. Le projet a débuté le **19 avril 2010** et il est à rendre le **11 juin 2010** avant la présentation.

Comme pour le partage des responsabilités, les personnes citées comme **responsables** s'occuperont de gérer les ressources humaines pour mener à bien les parties qui leur incombent. Bien évidemment, chaque membre apportera sa contribution pour chacune des parties à mettre sur pied.

4.2.1 Itération 1 – Serveur d'enregistrement + Interface graphique

Durée : 1 semaine – 30 avril 2010 au 7 mai 2010

Temps : environ 8 heures / membre soit environ 32 heures

Implémenter complètement la partie Serveur d'enregistrement ainsi que son interface graphique.


Le serveur d'enregistrement permet d'enregistrer les serveurs de jeu sur un serveur central afin de fournir la liste de ces serveurs aux clients. Ces derniers peuvent ensuite choisir la partie qu'ils veulent rejoindre.

Pourquoi cette itération prend-elle place ici ?

C'est une petite partie fournissant une bonne introduction à la notion de communication CLT-SRV notamment pour les points suivants :

- Création de notre premier protocole réseau (fixation des standards)
- Première communication CLT-SRV en Java (Création des classes de base)
- Intégration du tout dans une interface graphique cohérente. (Schéma d'interface)

Résultat attendu : le client peut enregistrer ses parties sur le serveur d'enregistrement et voir la liste de toutes les parties en attente de joueur(s) depuis une interface graphique.

Fonctionnalités attendues		
Ok	Responsable(s)	Fonctionnalité 
<input type="checkbox"/>	Lazhar	Etablir une connexion client / serveur avec échange de message
<input type="checkbox"/>	Lazhar	Enregistrer une partie sur le serveur d'enregistrement.
<input type="checkbox"/>	Lazhar	Voir les parties inscrites sur le serveur d'enregistrement
<input type="checkbox"/>	Lazhar	« Dénregistrer » d'une partie sur le serveur d'enregistrement.
<input type="checkbox"/>	Lazhar	Mettre à jour les informations d'une partie.
<input type="checkbox"/>	Aurélien	Interface graphique pour l'enregistrement de la partie
<input type="checkbox"/>	Aurélien	Interface graphique pour voir les parties inscrites

4.2.2 Itération 2 – Serveur de Jeu + Architecture

Durée : 1 semaine - 7 mai 2010 au 14 mai 2010

Temps : environ 10 heures / membre soit environ 40 heures

Pourquoi cette itération prend-t-elle place ici ?

On prépare tous les éléments pour les fusionner ensuite (avec adaptations quasi certaines)

Création de l'application client / serveur pour le jeu. En parallèle nous commencerons la restructuration de l'architecture pour correspondre à un jeu multi-joueurs.

Il s'agit de mettre place (sans interface) une communication entre un joueur et le serveur de Jeu.

Résultat : Un protocole de communication mis en place pour l'échange de message entre le client et le serveur de jeu. Concernant la restructuration, on attend un mode 1 joueur avec exactement les mêmes fonctionnalités mais avec une architecture beaucoup plus propre.

Fonctionnalités Attendues		
Ok	Responsable(s)	Fonctionnalité
<input type="checkbox"/>	Pierre-Dominique & Romain	Implémentation de tous les messages fournis par le protocole. (liste trop exhaustive pour les citer tous, <i>se référer au protocole en annexe</i>)
<input type="checkbox"/>	Aurélien	Le mode 1 joueur fonctionne correctement et comme avant. Présentation du schéma de classe ou de domaine.

4.2.3 Itération 3 –Intégration du serveur de jeu + Interface du Jeu en réseau

Durée : 2 semaines - 14 mai 2010 au 28 mai 2010

Temps : environ 24 heures / membre soit environ 96 heures

Intégration du serveur dans l'architecture et le jeu fonctionne.

Le serveur de jeu devra être intégré à l'architecture de l'application (au noyau du jeu). Le client et le serveur pourront alors interagir avec le modèle (point de vue MVC) du jeu. L'interface du jeu permettra d'illustrer ces changements.

Pourquoi cette itération prend-t-elle place ici ?

Il est temps de faire fusionner tous les éléments et en faire un programme plus cohérent.

Résultat : A la fin de cette itération, le jeu doit fonctionner et tous les messages transitant entre le client et le serveur doivent être correctement traités par l'entité réceptrice.

Fonctionnalités Attendues		
Ok	Responsable(s)	Fonctionnalité
<input type="checkbox"/>	Aurélien	L'interface permet de solliciter des actions du jeu.
<input type="checkbox"/>	Aurélien	Le client peut se connecter à une partie de jeu.
<input type="checkbox"/>	Pierre-Dominique & Romain	Le serveur et le client interprètent les messages réseau et modifie correctement le model. Ceci est visible grâce au

	changement du terrain de jeu.
--	-------------------------------

4.2.4 Itération 4 – Lifting de la GUI + Game Design + Amélioration Mode Solo

Durée : 1 semaine - 28 mai 2010 au 4 juin 2010

Temps : environ 10 heures / membre soit environ 40 heures

Revoir le design et faire de notre logiciel un « vrai » jeu vidéo

Actuellement, au niveau de l'interface graphique, notre jeu ressemble plus un à logiciel applicatif qu'à un jeu vidéo. Nous aimerions dans cette itération rendre notre jeu plus attractif en créant une interface plus agréable. Il serait aussi intéressant de concevoir nos propres ressources (images / sons / etc.) car actuellement, une bonne partie de nos images sont reprises d'autres jeux.

Une bonne chose serait de revoir également toutes les valeurs liées au jeu pour le rendre plus « jouable » (*level design*). Cette partie peut paraître bénigne mais elle est cruciale et très complexe pour ce genre de jeu car il y a énormément d'éléments qui influencent la durée de vie du joueur.

Nous voulons également implémenter un système de progression dans le mode solo pour que le joueur ressente l'envie de finir complètement le jeu. Le système sera basé sur des étoiles que le joueur gagnera en fonction de son score. Les étoiles donneront l'accès à de nouveaux terrains de jeux.

Résultat : Un programme plus esthétique, plus jouable et avec un système de progression.

Fonctionnalités Attendues		
Ok	Responsable(s)	Fonctionnalité
<input type="checkbox"/>	Lazhar	Lifting de l'interface, celle-ci ressemble plus à un jeu.
<input type="checkbox"/>	Pierre-Dominique & Romain	Adaptation des valeurs, jeu plus agréable.
<input type="checkbox"/>	Aurélien	Système de progression mise en place.

4.2.5 Itération 5 – Serveur Web (facultatif)

Durée : moins d'une semaine - 4 juin 2010 au 9 juin 2010

Temps : environ 6 heures / membre soit environ 24 heures

Serveur web de stockage des meilleurs scores

Il s'agit de mettre en place un serveur de web fournissant un service web de sauvegarde et récupération des meilleurs scores pour les différents terrains de jeu. Le but étant de motiver le joueur à s'améliorer (il doit avoir envie de rejouer le plus souvent possible).

Résultat : Un système permettant de sauver et voir les meilleurs scores de tous les joueurs (du monde).

Fonctionnalités Attendues		
Ok	Responsable(s)	Fonctionnalité
<input type="checkbox"/>	Romain	Mise en place du serveur web
<input type="checkbox"/>	Aurélien	Intégration du service dans l'application

4.3 Suivi du projet : itérations par itérations

Tout au long des itérations que nous avons planifiées, nous nous sommes confrontés à divers soucis, problèmes ou voir même bonnes surprises qui ont conduit à diverses replanifications. Nous avons également tiré un bilan de nos erreurs et des choses que nous avons bien faites.

4.3.1 Itération 1 – Serveur d'enregistrement + Interface graphique

Dans cette itération, le but était de mettre en place le serveur d'enregistrement en utilisant les protocoles et outils réseaux développés, ainsi que d'intégrer dans le programme la partie qui utilise ce serveur, permettant de voir les parties enregistrées.

4.3.1.1 Problèmes rencontrés

Nous avons rencontré des soucis lors de la mise en place des tests unitaires JUnit sur cette partie, du fait des méthodes bloquantes de la classe TCPCanal développée pour faire l'interaction TCP entre le client et le serveur.

Nous nous sommes également confrontés à des problèmes conceptuels concernant l'adresse IP à transmettre au serveur d'enregistrement. En effet, nous avons le choix entre l'adresse locale (celle du réseau local visiblement uniquement par les utilisateurs sur le même broadcast) et l'adresse Internet (l'adresse du routeur, vue par un éventuel client ou serveur externe). Nous avons dû mettre en place un protocole de contournement, ainsi qu'une fonctionnalité permettant de se connecter à un serveur directement depuis une adresse IP, sans passer par le serveur d'enregistrement.

4.3.1.2 Bilan

Nous tirons de cette itération un bilan très positif : en effet nous avons à notre disposition un serveur d'enregistrement tel que décrit dans le cahier des charges fonctionnel. Cependant, les soucis d'adresses rencontrés nous ont bien rendus attentif à ce genre de détails pour la suite du projet.

4.3.1.3 Replanifications

Malgré la résolution des soucis inattendus relatifs aux adresses IP, nous avons pu rester dans les délais et ne pas avoir eu à replanifier les itérations suivantes.

4.3.1.4 Remarques

Lors de la livraison de l'itération, le client nous a semblé content du résultat. Les retours que nous avons eu nous ont permis de rester motivé pour la suite du projet.

4.3.2 Itération 2 – Serveur de Jeu + Architecture

Dans cette itération, le but était de mettre en place le serveur de jeu ainsi que l'architecture associée, tant coté client que serveur, incluant les protocoles de communications entre le serveur et le client.

4.3.2.1 Problèmes rencontrés

Malheureusement le protocole tel que présenté dans le rapport préliminaire et le cahier des charges n'était pas totalement adapté à nos besoins. Nous avons donc nous replonger dedans pour corriger quelques incohérences et ajouter quelques fonctionnalités nécessaires. Ce n'était pas à proprement parlé un problème, mais une rapide remise en question et adaptation du travail fourni précédemment.

Nous avons également rencontré au niveau protocole des soucis concernant non pas le protocole d'échange, mais le protocole de connexion (échange des versions, des ID des clients, etc....). En effet, ce protocole était défini de manière empirique et suivant les règles de bons sens. Nous avons donc du nous mettre d'accord sur un bref ensemble de messages à changer à la connexion.

Cependant, nous avons expérimenté un léger stress avant la présentation de l'itération du à ces refontes. Cependant nous avons su en n'ajoutant pas d'heures supplémentaires mais en augmentant l'efficacité de notre travail durant les heures planifiées arriver à bout des soucis que nous avons rencontré.

4.3.2.2 Bilan

Encore une fois cette itération s'est bien passée. Nous avons également effectué des **tests de non régressions** sur la version solo pour vérifier que l'adaptation de la structure ne supprime pas de fonctionnalités déjà vérifiées dans la version précédente.

L'implémentation du protocole tant coté client que serveur s'est également bien passée, nous avons très rapidement pu arriver à une version fonctionnelle du dialogue entre le serveur et le client. Par exemple le client demande l'ajout d'une tour, le serveur reçoit correctement la requête et donne en retour une valeur témoin.

4.3.2.3 Replanifications

Malgré les changements souvent esthétiques du protocole, nous n'avons pas subis de retard entrainement une replanifications futures.

4.3.2.4 Remarques

Encore une fois la présentation du résultat au client s'est correctement déroulée. Nous avons mis en place une petite application de test en interface par lignes de commandes pour l'échange de messages texte à la manière d'un t'chat IRC pour illustrer le fonctionnement correcte de notre protocole.

4.3.3 Itération 3 – Intégration du serveur de jeu + Interface du Jeu en réseau

Dans cette itération, notre but a été de mettre en place le lien entre les modules clients/serveur et la structure de base du jeu (modèles, vues, contrôleurs) déjà mise en place dans les itérations précédentes.

4.3.3.1 Problèmes rencontrés

Nous nous sommes confrontés à des soucis de compréhension des bases déjà mises en place au sein de l'équipe de développement. En effet, les responsables de chaque parties ont du comprendre et par la lecture de la documentation mise en place et par le dialogue direct la manière dont chaque partie fonctionne.

Cette mise à niveau de chaque membre sur les sections des autres s'est révélée être complexe du fait du nombre de concepts mis en place par chacun. Cependant, nous avons pu grâce à une réunion de crise éclaircir pour chacun les points problématiques ainsi que de pouvoir recentrer nos efforts.

4.3.3.2 Bilan

Dernière étape avant la terminaison du projet, cette itération s'est révélée délicate. Malgré quelques soucis de communication au sein du groupe, nous avons pu mettre en place des moyens de résolution rapides des problèmes survenus, et ceux à moindre coût humain et en temps.

Nous avons donc pu présenter dans les temps l'itération selon le cahier des charges.

4.3.3.3 Replanifications

Ici encore une fois pas de replanification nécessaire mais plusieurs heures supplémentaires ont dues être réalisées par les membres du projet (environ au total une bonne dizaine d'heures) ainsi qu'une heure supplémentaire de réunion obligatoire pour se mettre d'accord sur les façons de lier le serveur/client aux parties applicatives. Après coup nous avons remarqué que nous avons été beaucoup trop optimistes lors du plan initial des itérations. En effet cette partie était la partie la grande du projet, nous aurions du l'étendre sur deux semaines.

4.3.3.4 Remarques

La présentation au client de l'itération s'est bien passée. Nous avons pu faire une démonstration d'une pose de tour par le client ainsi que l'affichage de l'objet chez chaque client connecté.

4.3.4 Itération 4 - Lifting de la GUI + Game Design + Amélioration Mode Solo

Dans cette itération nous avons prévu de mettre en places des refontes cosmétiques de notre application, tant au niveau de l'interface qu'au niveau de la jouabilité.

4.3.4.1 Problèmes rencontrés

Pas de problèmes majeurs rencontrés, à part l'hétérogénéité des bonnes idées d'améliorations (changement du prix des créatures, de la manière dont nous faisons gagner de l'argent aux joueurs, etc.) entre les membres du groupe. En effet, M. Da Campo en tant que chef de projet ainsi que M. Farjallah en tant que testeur ont énormément apporté de très bonnes idées permettant d'améliorer grandement le côté ludique du jeu. Dû au côté prenant et motivant de cette itération, quelques heures supplémentaires ont été faites (dans ce genre de projet, jouer c'est travailler ☺).

4.3.4.2 Bilan

Après cette itération nous avons un jeu pleinement jouable malgré notre faible expérience dans le développement de jeux vidéo. La difficulté des différents modes est contrôlée, la jouabilité est également au rendez-vous. Nous ressortons enthousiastes de cette itération et très fière de présenter notre travail au client.

4.3.4.3 Replanifications

Ici également pas de replanifications nécessaire du à l'investissement d'heures supplémentaires ((environ au total une bonne dizaine d'heures) pour peaufiner les détails ludiques du jeu. Cependant des membres du projet par forcément concerné au premier degré par l'équilibrage du jeu sont venus en renfort des membres responsables de cette partie.

4.3.4.4 Remarques

Plusieurs collègues et amis nous ont fait ressentir leurs intérêts pour ces changements divers qui améliorent considérablement le « gameplay » du jeu.

4.3.5 Itération 5 – Serveur Web (facultatif)

Dans cette itération nous avons prévu de mettre en ligne l’affichage des scores avec une gestion client/administrateur tel que suggéré dans la structure du projet.

4.3.5.1 Problèmes rencontrés

Le principal problème que nous avons rencontré est le délai de livraison du projet ne nous laissant pas le loisir de mettre en place cette itération définie également avec le client comme facultative. Durant la période de cette itération, nous avons consacré notre temps à la correction des derniers bugs restants dans l’itération 3.

4.3.5.2 Bilan

Cette itération ne sera donc pas livrée à la date butoir fixée avec le client. Si nous avions eu de l’avance dans les précédentes itérations, nous aurions pu avoir le temps de mettre en place l’itération cinq. Cela n’a malheureusement pas été le cas.

Cette itération reste donc non livrée, en conformité avec le cahier des charges.

4.3.5.3 Replanifications

Cette itération va être replanifiée dans une amélioration future du projet hors délai accordé par le client pour le projet.

4.3.5.4 Remarques

Rien d’autre à ajouter, sinon insister sur le fait que la non livraison de cette itération était un cas prévu dans le cahier des charges avec le client. Nous avons donc en tous points respectés le contrat précédemment admis par les deux partis.

4.4 Stratégie de tests

Nous présentons dans cette section la description de tous les tests effectués sur l'application finale. Pour chacun de ces tests, nous donnons une capture d'écran du résultat avec un commentaire expliquant les conclusions à tirer ou les éventuels échecs.

Ces tests ont été faits pour qu'ils soient suivis dans l'ordre. En effet, un test est souvent la conséquence du précédent (sa suite logique). Il est donc intéressant ici de suivre le cheminement du début à la fin sans sauter d'un test à l'autre, au risque de ne pas tout comprendre.

Nous commencerons par tester le serveur d'enregistrement, puis nous testerons le serveur de jeu en créant une partie et en y connectant des joueurs.

4.4.1 Condition des tests effectués

Les tests ont été effectués dans les conditions suivantes :

- La version de l'application utilisée est la 2.0 (beta) disponible en téléchargement à l'adresse suivante :
http://code.google.com/p/asd-tower-defense/downloads/detail?name=ASD_TD_v2.0_beta.zip
- Le serveur de jeu ainsi que les clients sont exécutés sur une machine localement.
- Le serveur d'enregistrement est exécuté sur un serveur dédié disponible à l'adresse IP suivante : 188.165.41.224, ainsi qu'en local à l'adresse 127.0.0.1
- Le système d'exploitation de la machine locale est Windows XP Professional SP3
- Le système d'exploitation du serveur dédié (188.165.41.224) est GNU/Linux Ubuntu Server 9.04
- L'application est déployée sous forme d'une archive JAR exécutable (Java)
- Une installation de la JRE 1.6 ou supérieure est nécessaire (Java Runtime Environment)

4.4.2 Stratégie des tests

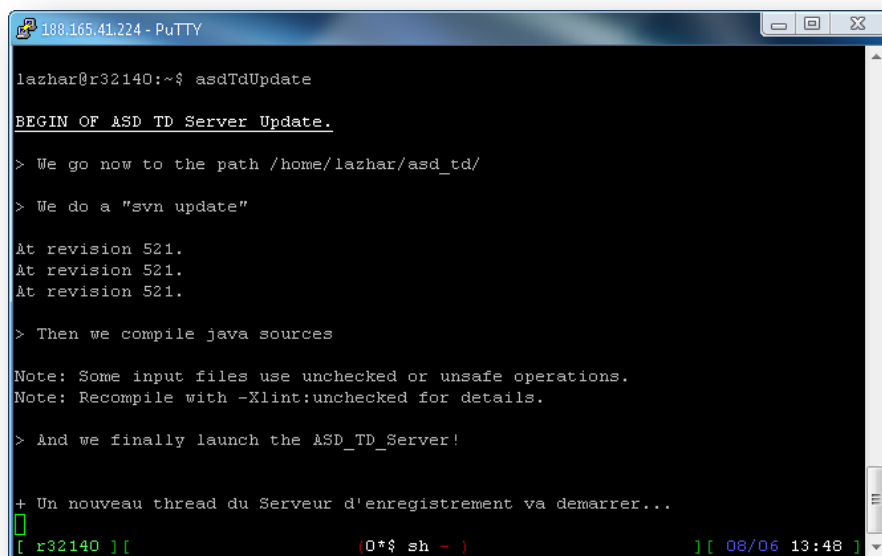
Nous allons faire une série de tests les un après les autres dans un ordre logique. Nous n'effectuons ici que les tests minimaux afin de démontrer le bon fonctionnement du jeu (serveur d'enregistrement et jeu en réseau).

Pendant la phase finale du projet, nous avons passé une grande partie du temps à tester notre jeu directement en jouant entre membres du groupe. Nous avons de ce fait pu déceler divers problèmes et nous avons pu les corriger. Il faut bien être conscient que certains tests ne peuvent pas figurer directement ici ou seraient trop complexes pour être présentés dans un tel rapport.

En ce qui concerne les tests unitaires, il suffit de les lancer avec l'outil JUnit et de constater le résultat retourné. Pour tous les tests unitaires que nous avons effectués, il va de soi qu'aucun n'échoue au stade actuel du projet.

4.4.3 Test n°01 : connexion client/serveur d'enregistrement avec échange de messages

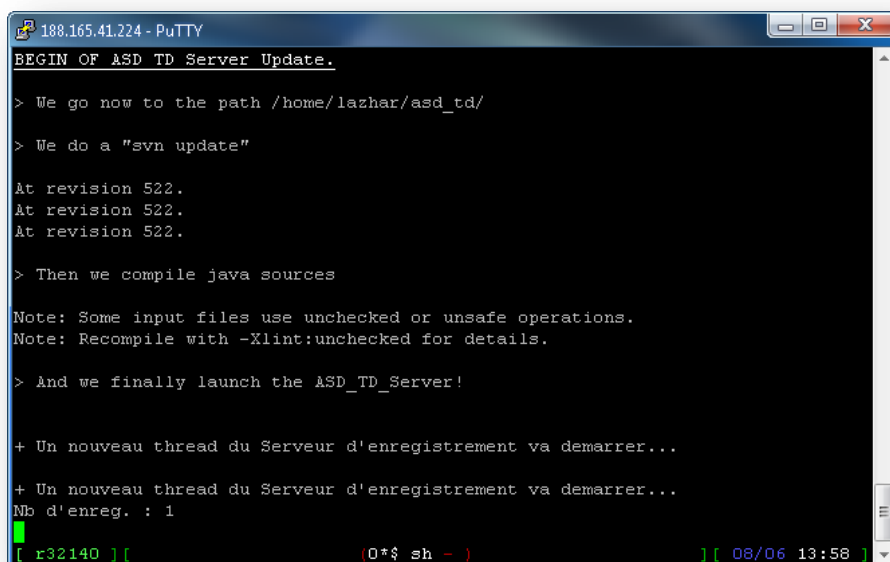
Nous commençons par lancer le serveur d'enregistrement sur le serveur dédié :



```
188.165.41.224 - PuTTY
lazarhar@r32140:~$ asdTdUpdate
BEGIN OF ASD TD Server Update.
> We go now to the path /home/lazarhar/asd_td/
> We do a "svn update"
At revision 521.
At revision 521.
At revision 521.
> Then we compile java sources
Note: Some input files use unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
> And we finally launch the ASD_TD_Server!
+ Un nouveau thread du Serveur d'enregistrement va demarrer...
[r32140 ] [ (0*$ sh - ) ] [ 08/06 13:48 ]
```

Figure 4.4.1 : lancement du serveur d'enregistrement sur un serveur dédié (188.165.41.224)

Une fois le serveur lancé, comme on le voit sur la Figure 4.4.1, nous lançons l'application cliente depuis le fichier JAR. Nous cliquons ensuite sur le bouton « Créer » afin de créer une nouvelle partie nommée « Test » de capacité 4 (4 joueurs max). Le résultat est le suivant :



```
188.165.41.224 - PuTTY
BEGIN OF ASD TD Server Update.
> We go now to the path /home/lazarhar/asd_td/
> We do a "svn update"
At revision 522.
At revision 522.
At revision 522.
> Then we compile java sources
Note: Some input files use unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
> And we finally launch the ASD_TD_Server!
+ Un nouveau thread du Serveur d'enregistrement va demarrer...
+ Un nouveau thread du Serveur d'enregistrement va demarrer...
Nb d'enreg. : 1
[r32140 ] [ (0*$ sh - ) ] [ 08/06 13:58 ]
```

Figure 4.4.2 : Le serveur d'enregistrement a réagit et possède maintenant un nouvel enregistrement, celui de la partie nouvellement créée

Nous avons donc établi une connexion client/serveur d'enregistrement et des messages ont été échangés.

4.4.4 Test n°02 : voir les parties inscrites sur le serveur d'enregistrement

Nous lançons maintenant une nouvelle application cliente (fichier JAR). Nous cliquons ensuite sur « Rejoindre » pour voir la liste des parties enregistrées. Le résultat devrait être une entrée, celle du test précédent :

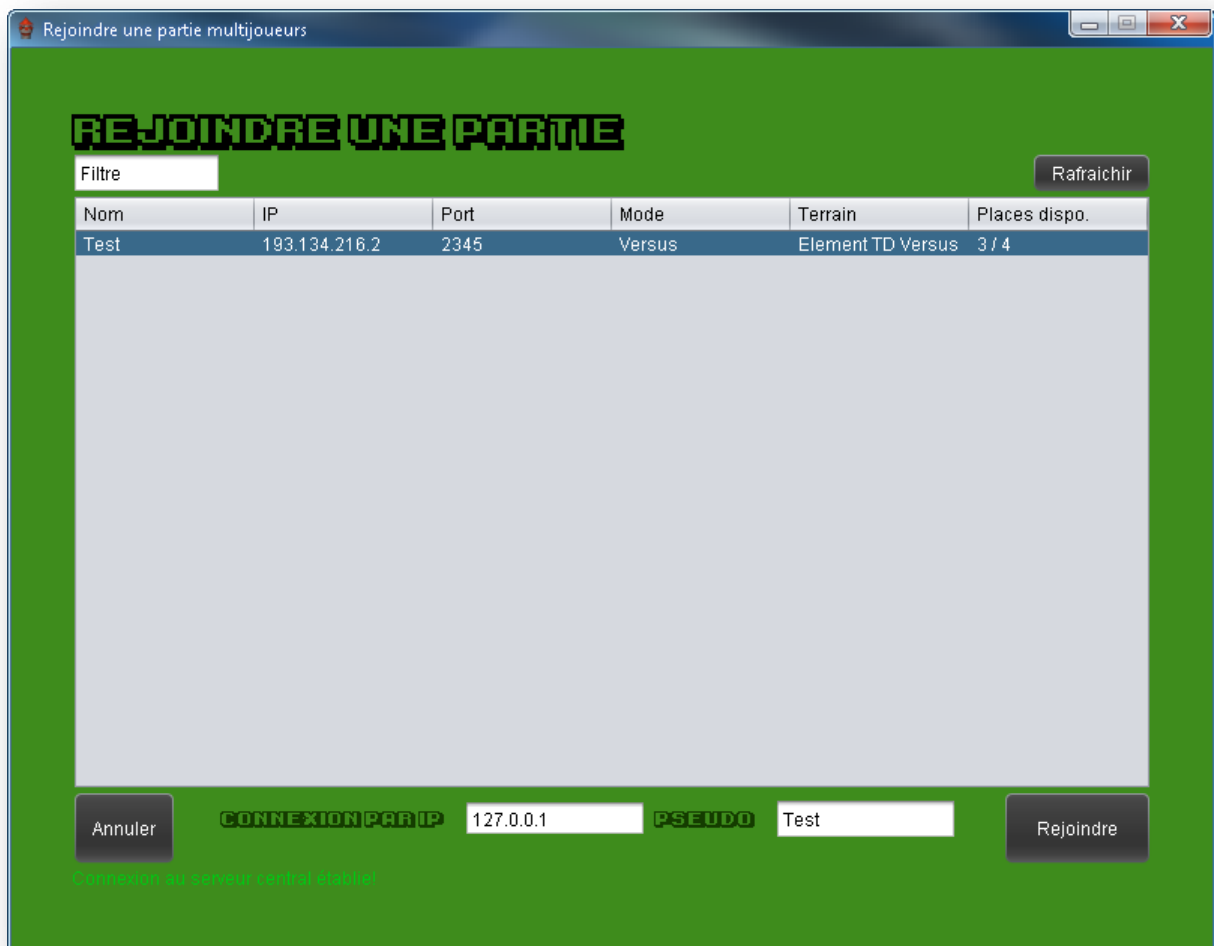


Figure 4.4.3 : La partie enregistrée visible par d'autres clients.

La partie enregistrée dans la Figure 4.4.2 est maintenant visible par d'autres clients. C'est bien la partie « Test » précédemment créée. L'adresse IP que l'on voit est celle du serveur de Jeu (adresse Internet et non locale).

4.4.5 Test n°03 : « Désenregistrer » une partie sur le serveur d'enregistrement

Nous allons maintenant quitter la partie nouvellement créée dans la Figure 4.4.2. Nous cliquons ensuite sur le bouton « Rafraichir » de la Figure 4.4.3. Le résultat devrait être la disparition de l'entrée visible dans cette dernière :

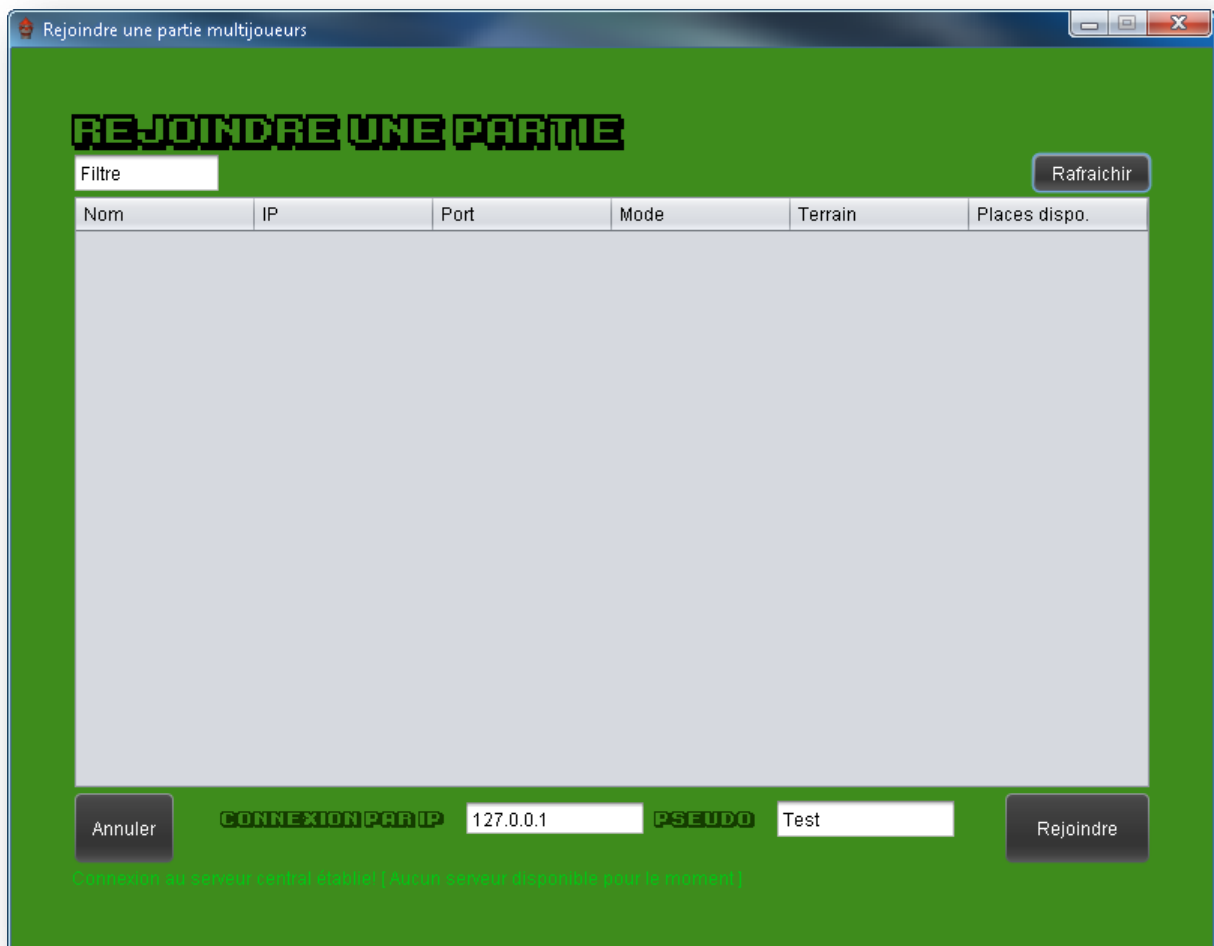


Figure 4.4.4 : Résultat du désenregistrement

La partie précédemment enregistrée a disparu de la liste des enregistrements lorsqu'on clique sur le bouton « Rafraichir », car la partie « Test » a été arrêtée volontairement.

4.4.6 Test n° 04 : mettre à jour les informations d'une partie

Nous démarrons cette fois le serveur d'enregistrement en local.

Nous recréons la partie « Test » et nous y connectons cette fois 2 clients. Le résultat de la liste des parties devrait être alors le même que celui de la figure 4.4.3 excepté la colonne « Places dispo. » qui devrait maintenant être de 1/4 et non de 3/4, vu que nous avons connecté 2 clients cette fois :

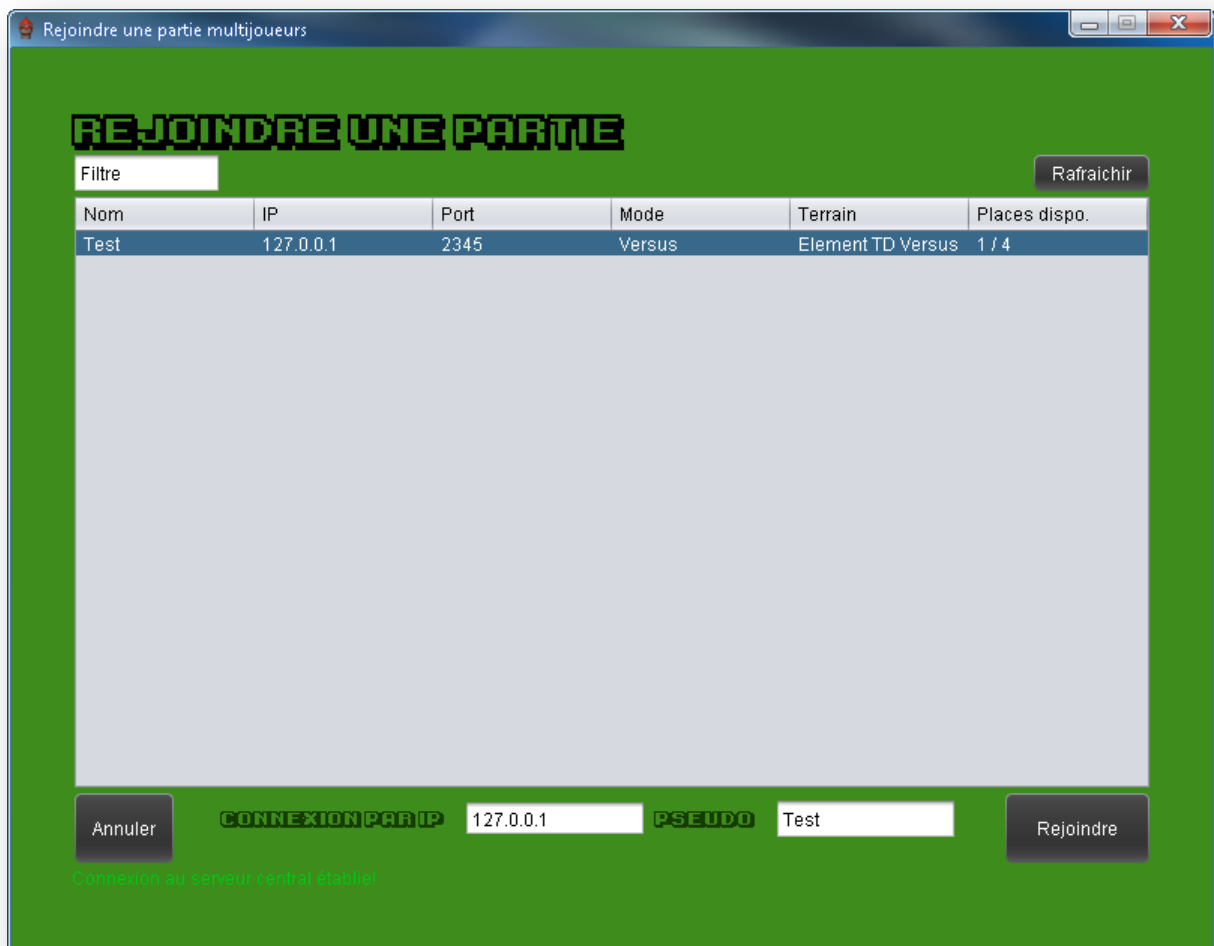


Figure 4.4.5 : Mise à jour du nombre de places disponibles

Le nombre de places disponibles s'est bien mis à jour (l'IP est 127.0.0.1 car nous avons lancé le serveur d'enregistrement en local).

4.4.7 Test n° 05 : le client peut se connecter et jouer une partie en réseau

Dans ce test, nous commençons par créer un serveur de jeu. Pour se faire, nous lançons l'application en double-cliquant sur le fichier JAR de l'application puis nous cliquons ensuite sur le bouton « Créer ». Nous lançons une deuxième occurrence du jeu en double-cliquant à nouveau sur le fichier JAR puis nous cliquons cette fois sur le bouton « rejoindre ». Comme tout se fait en local, l'adresse IP de la partie créé est 127.0.0.1 (ou localhost). Nous cliquons ensuite sur le bouton « rejoindre » depuis la deuxième occurrence du jeu lancée précédemment. Le résultat des deux fenêtres est le suivant :



Figure 4.4.6 : Première occurrence de l'application – création d'un serveur de jeu (d'une partie réseau).



Figure 4.4.7 : Deuxième occurrence de l'application (rejoindre une partie réseau nouvellement créée).

Nous voyons que le joueur qui a créé une partie (Figure 4.4.6) peut, quand il le souhaite, cliquer sur le bouton « Démarrer maintenant » situé en bas à droite. Ceci est normal vu que c'est lui-même qui a créé la partie, il est donc l'administrateur de la partie. Son rôle est d'attendre qu'il y ait un nombre suffisant de joueurs (2, 3 ou 4) selon son souhait et de lancer la partie.

Par contre, le joueur qui a rejoint la partie ne peut lui qu'attendre (Figure 4.4.7) que la partie commence. Cependant, un chat est disponible sur la partie du bas de la fenêtre pour pouvoir communiquer avec les autres joueurs pendant l'attente de tous les joueurs. Le créateur de la partie a également accès à ce chat.

Une fois que quatre joueurs sont connectés (le créateur de la partie ainsi que trois autres joueurs), le créateur de la partie peut démarrer effectivement la partie. Il clique donc sur le bouton « Démarrer maintenant ». Le résultat est le suivant :

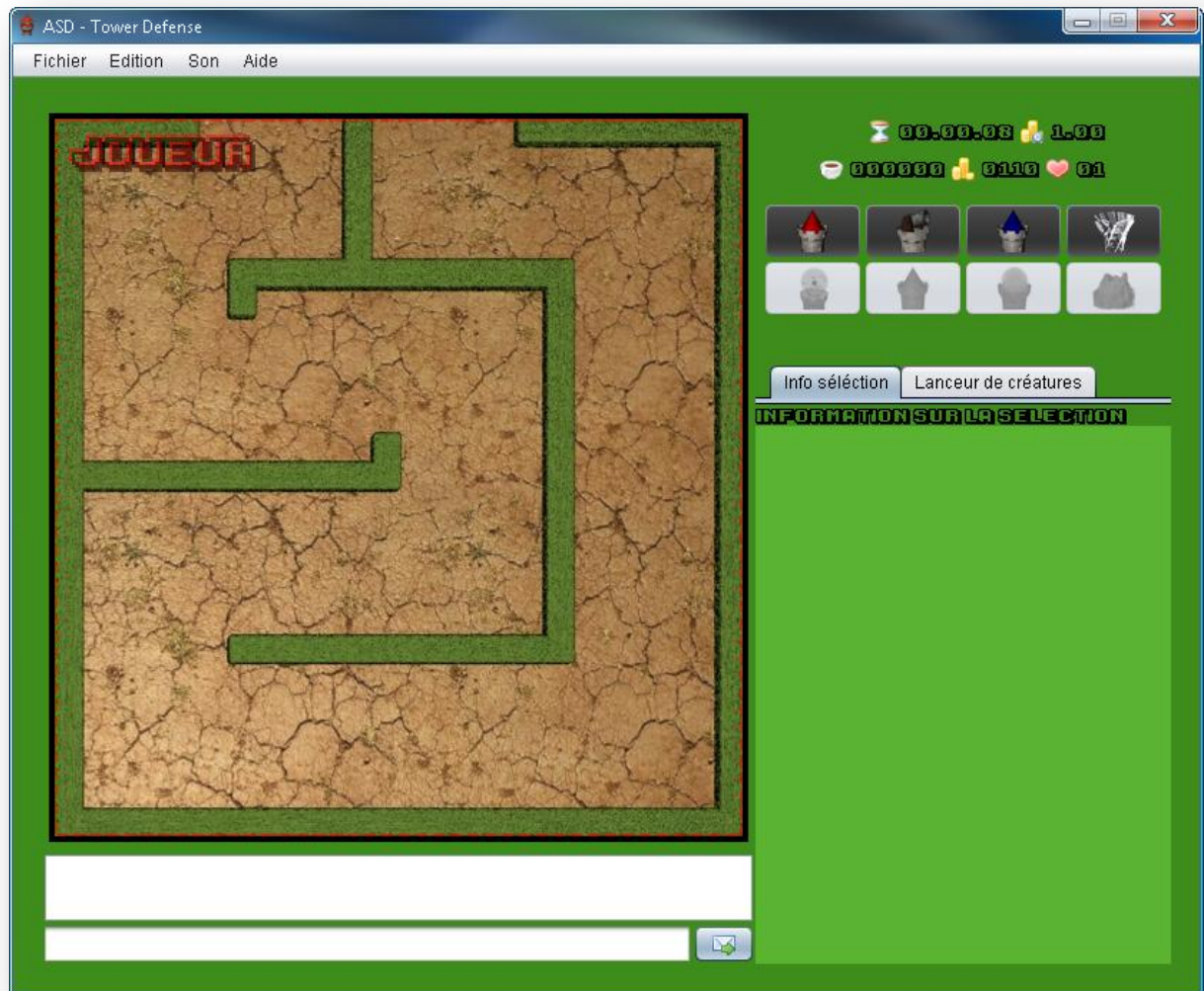


Figure 4.4.8 : La partie en réseau commence !

Les joueurs arrivent donc tous sur cette fenêtre et peuvent commencer à jouer. Chacun possède dans son focus central sa zone attribuée et peut commencer à placer des tours et/ou lancer des créatures sur les zones adverses.

Faute de temps, les autres tests n'ont pas pu être décrits en détails dans ce chapitre. Ces derniers sont néanmoins listés dans le résultat des tests dans le chapitre état des lieux. Merci de votre compréhension.

4.4.8 Conclusion des tests

Ces quelques tests ont tous été effectués avec succès. Les principales fonctionnalités décrites dans les itérations en début de projet sont opérationnelles. Nous sommes très satisfaits et nous avons évidemment pris beaucoup de plaisir à jouer (tester) le jeu dans sa version réseau finale.

4.5 Stratégie d'intégration du code de chaque participant



Repartant d'un projet déjà existant, nous avons une structure de code déjà mise en place sur notre SVN (un serveur public fourni par Google : <http://code.google.com/p/asd-tower-defense/>)

La stratégie a été la suivante : nous avons fait un clone du code existant de la version 1.0 pour conserver, par soucis historique, une base tel qu'elle a été présentée dans le cours d'ASD2. Puis nous avons restructuré le code pour permettre la mise en place des paquets destinés au réseau (créations des outils par M. Farjallah, écriture des parties clients et serveur par M. Poulain et M. Putallaz, intégration de la gestion du réseau dans l'ancienne architecture par M. Da Campo).

A la suite de la mise en place des bases de travail, chacun a modifié les parties dont il était responsable au fil des itérations en concurrence avec toujours SVN. Cette méthode nous permettait d'avoir un suivi très régulier des modifications par les notes que nous avons intégrés à chaque *commit*, de pouvoir revenir en arrière en cas d'erreur grâce à la fonction *revert* ainsi que de gérer les éditions concurrente du même fichier ou du même paquetage.

5 Etat des lieux

5.1 Ce qui fonctionne (résultat des tests)

Nous exposons dans cette section la liste complète des tests effectués avec pour chacun son état actuel. Ceux-ci sont en partie repris des objectifs des itérations du projet.

Description du test	Etat	Remarques
Etablir une connexion client / serveur avec échange de message	OK	Fonctionnement correct.
Enregistrer une partie sur le serveur d'enregistrement.	OK	Fonctionnement correct.
Voir les parties inscrites sur le serveur d'enregistrement	OK	Fonctionnement correct.
« Désenregistrer » une partie sur le serveur d'enregistrement.	OK	Mais si le serveur de jeu quitte brutalement, le serveur d'enregistrement garde toujours la partie. Il ne se met pas à jour périodiquement.
Mettre à jour les informations d'une partie.	OK	Fonctionnement correct.
Interface graphique pour l'enregistrement de la partie	OK	Fonctionnement correct.
Interface graphique pour voir les parties inscrites	OK	Fonctionnement correct.
Test du mode solo en profondeur.	OK	Fonctionnement correct.
Test de l'interface (permet de solliciter des actions du jeu).	OK	Fonctionnement correct.
Le client peut se connecter à une partie de jeu.	OK	Fonctionnement correct dans l'ensemble. Un bug restant : le client qui héberge le serveur reçoit évidemment le message du nouveau joueur plus rapidement que les autres joueurs car le message passe en local. Du coup le joueur hébergeur peut lancer la partie alors que le joueur qui veut se connecter n'est pas encore (pour lui) dans la partie. Du coup lorsqu'un joueur hébergeur voit

		un nouveau joueur se connecter il doit attendre quelques secondes avant de lancer la partie. Nous n'avons pas eu le temps de corriger cette erreur.
Le serveur et le client interprètent les messages réseau et modifient correctement le modèle. Ceci est visible grâce au changement du terrain de jeu.	OK	Fonctionnement correct.
Test de l'interface ; celle-ci ressemble plus à un jeu.	OK	Oui il nous semble.
Test de l'adaptation des valeurs (jeu plus agréable).	OK	Oui il nous semble.
Test du système de progression.	OK	Fonctionnement correct.
Mise en place de l'ensemble des constantes listées dans le protocole, plus d'éventuelles modifications à apporter par la suite.	OK	Fonctionnement correct.
Envoi de messages textes entre le client et serveur, sous forme de ping/pong ou de messages de discussions.	OK	Fonctionnement correct.
Envoi des messages de modification de l'état d'une partie par un joueur, en jeu ou hors jeu.	OK	Fonctionnement correct.
Envoi des messages de modification de l'état d'un joueur, en jeu ou hors jeu.	OK	Fonctionnement correct.
Envoi du signal de lancement d'une vague par un joueur au serveur, avec la réponse associée.	OK	Fonctionnement correct.
Création d'une tour par le client, avec le message de retour associé en cas de réussite ou d'erreur.	OK	Fonctionnement correct.
Amélioration d'une tour par son propriétaire déjà en place, avec comme retour un message associé en cas de réussite ou d'erreur.	OK	Fonctionnement correct.
Suppression d'une tour par son propriétaire, avec message de retour associé en cas de réussite ou d'erreur.	OK	Fonctionnement correct.
Ici prendra place la partie la plus complexe. En effet cette partie concernera le retour du serveur à chaque client permettant d'afficher les objets de jeu pour maintenir un affichage cohérent entre chaque client et le serveur.	OK	Fonctionnement correct.
Information lorsqu'un joueur réseau perd la partie	OK	Fonctionnement correct mais pourrait être amélioré au niveau de l'affichage.
Information de résultat d'une partie réseau	OK	Fonctionnement correct mais pourrait être amélioré au niveau de l'affichage.

5.2 Ce qu'il resterait à développer

5.2.1 Peaufinage et recontrôle de l'application réseau

Temps estimé nécessaire pour la conception / réalisation : environ 50 heures

Notre application réseau fonctionne dans sa version actuelle. Malheureusement le temps alloué pour ce projet ne nous a pas permis de finir complètement le mode de jeu réseau. Nous pensions notamment à encore implémenter les points suivants :

- Gestion de la fin de partie (affichage du résultat complet avec l'état des joueurs)
- Nouveau mode de jeu (Coopération) basé sur le mode versus actuellement implémenté
- Panel d'affichage de l'état de tous les joueurs en jeu (vies, argent, etc.).
- Adaptions de la difficulté
- Création de nouveaux terrains de jeu multi-joueurs

5.2.2 Maillage

Temps estimé nécessaire pour la conception / réalisation : environ 20 heures

Hérité du projet d'ASD2, le maillage parcouru par un algorithme de recherche de chemin le plus court d'un point à un autre par Dijkstra pourrait être grandement amélioré.

En effet, la version actuelle souffre d'un manque de performance flagrant pour un grand nombre de créatures, du fait de la redondance de certains calculs. L'algorithme de Dijkstra est très performant pour trouver un chemin d'un point A à un point B dynamiques pour chaque créatures à chaque position sur le maillage. Cependant, dans notre problème, nous n'avons d'un seul point d'arrivée et d'un seul point de départ pour toutes les créatures.

Nous aurions donc la possibilité de restreindre les contraintes du problème et de trouver une solution plus performante pour la recherche de chemin à travers le graphe. L'idée serait de considérer l'entier du maillage (avec les arcs dynamiques) comme un système de flot. Le flot partirait du point de départ des créatures et se terminerait dans leur point d'arrivée. Chaque créature suivrait donc ce flot déjà calculé (et recalculé de manière différentielle à chaque modification du maillage – pose de tour, suppression d'une tour, etc. -) au lieu de recalculer à chaque modification son propre chemin de son point courant au point invariant d'arrivée.

Un gros travail algorithmique avancé serait donc nécessaire pour optimiser les performances de la recherche de chemin. Cependant le travail se limiterait uniquement à la classe *Maillage*, les méthodes d'accès à cette classe ainsi que la manière de l'utiliser ne seraient pas modifiées par cette refonte du fait de l'architecture de notre projet.

5.2.3 Traduction en plusieurs langues (principalement anglais)

Temps estimé nécessaire pour la conception / réalisation : environ 40 heures

Notre jeu est disponible actuellement que dans la langue de Molière ce qui peut limiter considérablement le nombre de joueurs d'autres pays. C'est pourquoi l'implémentation de plusieurs langues serait réellement un plus.

5.2.4 Plus de ressources

Temps estimé nécessaire pour la conception / réalisation : environ 40 heures

Nous avons maintenant un moteur qui tourne, mais la durée de vie de notre jeu pourrait être considérablement améliorée en ajoutant d'autres ressources telles que de nouvelles tours, de nouvelles créatures, de nouveaux terrains et de nouvelles animations.

Toutes les ressources (à part les musiques) de ce jeu ont été créées par les membres de ce projet. Nous ne sommes pas des graphistes ni des level designers, c'est pourquoi ces créations nous ont pris beaucoup de temps pour les créer. Il serait intéressant pour nous de proposer via des forums une aide précieuse à des personnes de métier pour ce genre d'itération.

6 Auto-critique

Dans l'ensemble, les délais ont été respectés et la qualité du travail nous paraît correcte. Mais cela est en grande partie dû aux heures supplémentaires des membres du projet. En effet, certaines itérations nécessitaient plus d'heures de travail. Par exemple, nous avons très clairement été trop optimistes dans l'itération numéro 3 qui nous demandait l'intégration de la communication complète du protocole avec le moteur du jeu ainsi que la réalisation de l'interface du jeu multi-joueurs. A notre avis, nous aurions dû séparer cette partie en plusieurs sous-itérations (qui deviendrait chacune une itération) et étendre le tout sur 3 semaines (2 initialement).

Cela dit, nous avons tout de même pressenti ce problème et c'est pour cela que nous avons dès le départ choisi de rendre la dernière itération facultative. Ce choix prudent nous a permis de terminer à l'heure malgré que points encore à terminer.

7 Conclusion

Ce travail fut une excellente occasion pour nous de mettre en pratique les notions acquises durant nos cours théorie. En effet, de la gestion de projet UP à la création d'artefacts en passant par la mise en place d'un modèle MVC et d'une gestion de visionnage de fichiers SVN; ce travail nous a permis de nous rendre compte de l'importance quasi essentielle de ces idées pour la mise en place d'un projet de développement de logiciel.

Tous les membres du projet sont très satisfaits du travail réalisé pour ce projet. Notre jeu fonctionne et nous venons tout juste de sortir la version 2.0 (beta) disponible à l'adresse suivante :

<http://code.google.com/p/asd-tower-defense/>

Les membres du projet vont à coup sûr continuer le développement de cette application interactive. Une partie des itérations future seront probablement réalisé soit dans le cadre d'un autre projet et à titre personnel. Nous avons également commencé à faire de la publicité sur Internet afin de recevoir des avis et nouvelle idées des aficionados de ce genre de jeu.

Merci de nous avoir permis d'améliorer ce jeu et de nous avoir fait découvrir toutes ces technologies.

Si notre projet vous à plus et vous souhaitez voir son évolution, nous vous proposons de venir voir de temps à autre les changements sur notre hébergeur de code à l'adresse ci-dessus.

8 Annexes

1. Rapport de la version 1.0 réalisée durant le cours ASD2 - 2009-2010
2. Protocole du serveur d'Enregistrement
3. Protocole du serveur de Jeu
4. Diagramme de classes du Serveur
5. Manuel d'installation et d'utilisation

