# 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.

## 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 Environnement)

## 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.

## Test n°01 : établir une connexion client/serveur d’enregistrement avec échange de messages

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

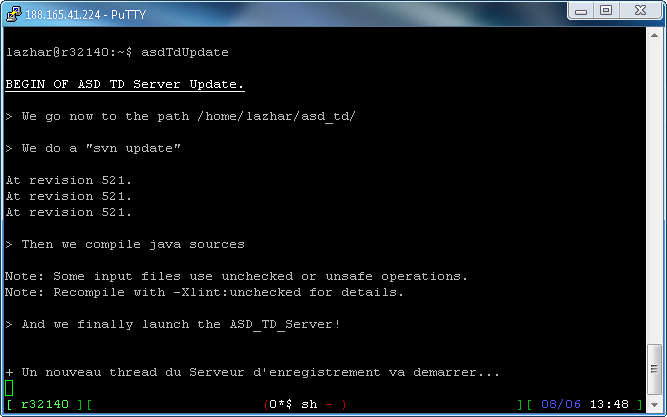


Figure . : lancement du serveur d’enregistrement sur un serveur dédié (188.165.41.224)

Une fois le serveur lancé, comme on le voit sur la , 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 :

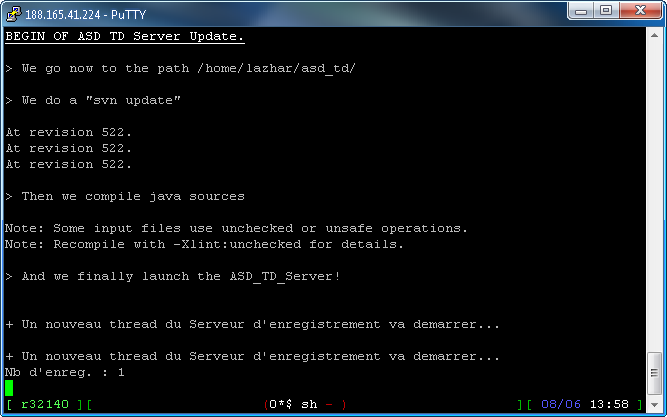


Figure . : Le serveur d’enregistrement a réagit et possède maintenant un nouvel enregistrement, celui de la partie nouvellement créée

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

## 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 :

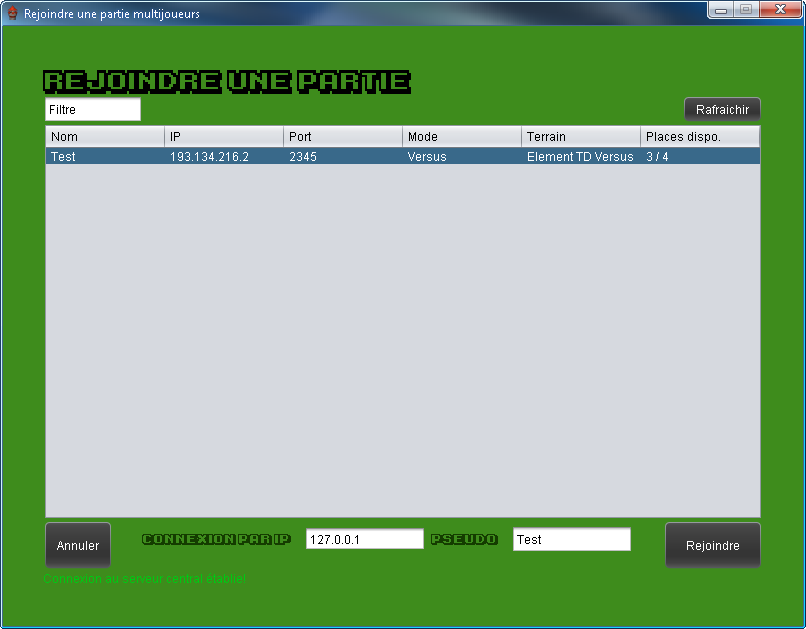


Figure . : La partie enregistrée dans la est maintenant visible par d’autres clients. C’est bien la partie « Test » précédemment créée. L’adresse IP que l’on voit est celle du serveur de Jeu (adresse Internet et non locale).

## Test n°03 : « Désenregistrer » une partie sur le serveur d’enregistrement

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

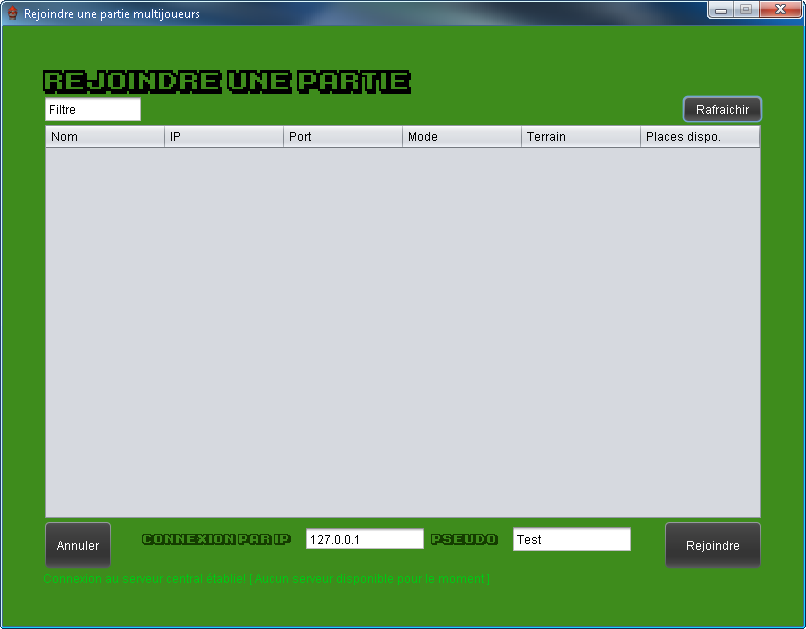


Figure . : La partie précédemment enregistrée a disparu de la liste des enregistrements lorsqu’on clique sur le bouton « Rafraichir », car la partie « Test » a été arrêtée volontairement.

## 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 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 :

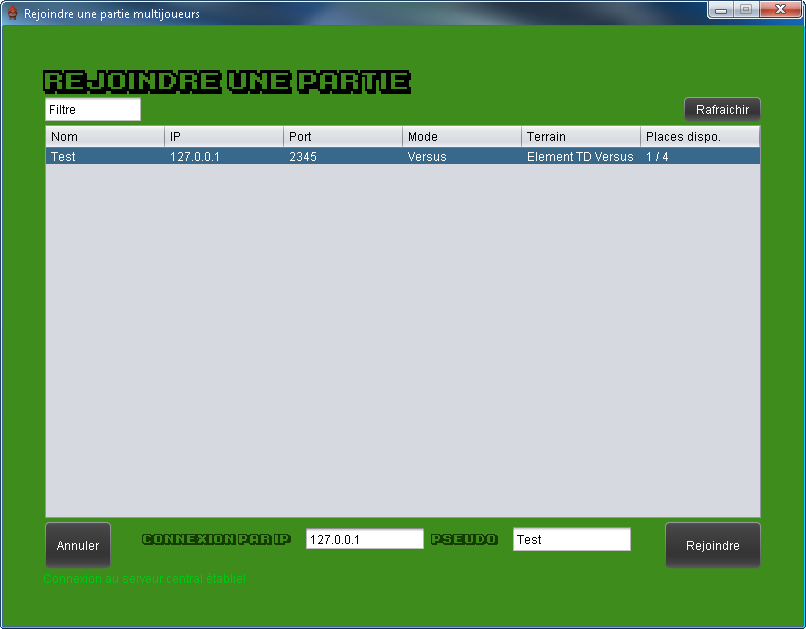


Figure . : Le nombre de places disponibles s’est bien mis à jour (l’IP est 127.0.0.1 car nous avons lancé le serveur d’enregistrement en local).

## 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 . : Première occurrence de l’application – création d’un serveur de jeu (d’une partie réseau).

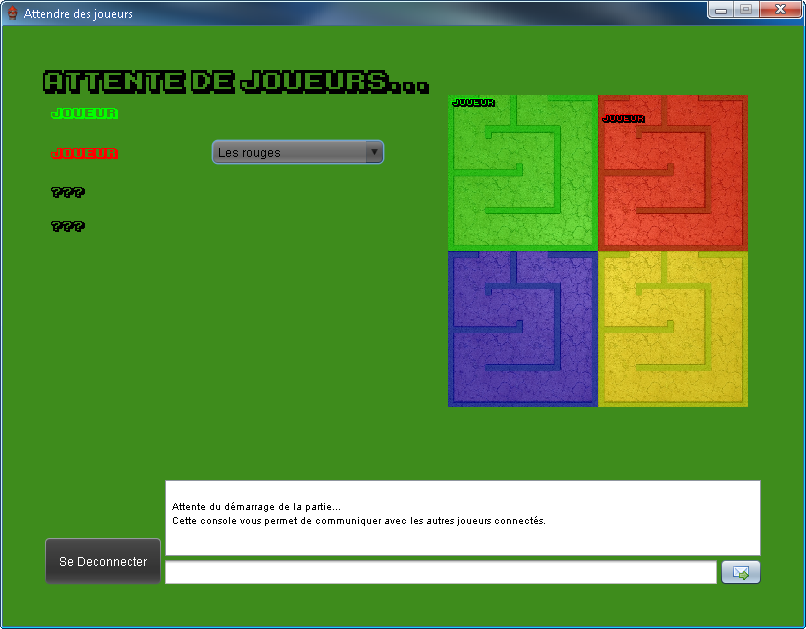


Figure . : Deuxième occurrence de l’application (rejoindre une partie réseau nouvellement créée).

Nous voyons que le joueur qui a créé une partie (Figure 1.6) 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 1.7) 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 . : 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.

## 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.

# Sérialisation des scores et des terrains de jeu

Lorsqu’on joue à un jeu, il y a dans la plupart des cas un score ainsi que d’autres informations qui doivent être 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 à l’autre, et ceci même après avoir quitté et relancé l’application.

Pour ce faire, nous avons utilisé la possibilité qu’il existe en Java de « sérialiser » des morceaux de code (en général des classes) afin de pouvoir directement les sauvegarder dans des fichiers persistants sur le disque dur. En effet, grâce à cette possibilité, on peut enregistrer l’état de l’exécution de l’application dans un fichier afin de retrouver cet état plus tard.

Pour la gestion des scores, nous avons donc enregistré dans un fichier l’état de ces scores à un temps *t* du jeu, en principe lors de la fin d’une partie. Ainsi, quand un joueur décide d’enregistrer son score, nous « sérialisons » les données associées et nous les enregistrons dans un fichier dit « sérialisé ». Par la suite, lors du lancement de l’application, cette dernière se charge de récupérer ce fichier et d’en reprendre les informations dans l’état où elles avaient été laissées. On peut ainsi récupérer facilement les scores d’un joueur. Les fichiers de scores sérialisés portent l’extension « .ms » et se trouvent dans le dossier « donnees » se trouvant à la racine de l’arborescence de l’application.

Le même principe a été utilisé pour les terrains de jeu. En effet, nous les avons « sérialisé » et enregistré dans des fichiers dont l’extension est « .map ». Ces fichiers se trouvent dans le dossier « maps » à la racine de l’arborescence de l’application.

Ainsi, lorsque nous créons un nouveau terrain de jeu, nous le sérialisons dans un fichier .ms afin de pouvoir ensuite le redistribuer aux joueurs. On peut donc par la suite proposer en téléchargement 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 possibilité est très pratique et donne un côté dynamique au jeu qui a toujours un bon effet auprès de l’utilisateur et éveille encore son intérêt ainsi que sa curiosité pour le jeu.

L’avantage de la sérialisation est qu’elle est simple, mais également que les fichiers produits, dits « sérialisés » 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 sauvé 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 à sa guise, ce qui ne serait pas correct.

En ce qui concerne les terrains, on peut également imaginer par la suite créer une application permettant de créer des terrains de jeu. Cette application aurait une fonction de sauvegarde du terrain créé par l’utilisateur, et le fichier qui en résulterait serait lui aussi « sérialisé » 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-même créées. Il pourrait également les distribuer à d’autres joueurs sur Internet.

Il faut encore noter que le système de sérialisation a été utilisé comme substitut d’une base de données. En effet, il était demandé au départ d’interagir avec une base de données, ce qui n’est malheureusement pas compatible à première vue avec notre projet. Nous avons donc décidé de substituer ce point par celui-ci, c’est-à-dire la sérialisation des données.

# Cas d’utilisation : serveur d’enregistrement

[Le schéma est ok]

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

|  |  |
| --- | --- |
| **Joueur d'une partie** | **Hébergeur d'une partie** |
|  | 1. L'hébergeur crée une partie |
|  | 2. Le système enregistre la partie nouvellement créée dans sa base de données |
|  | 3. L'hébergeur attend que des joueurs rejoignent la partie qu'il vient de créer |
| 1. le joueur demande la liste des parties (hébergeurs) disponibles |  |
| 2. Le système fournit la **liste des parties** (hébergeurs) en attente de joueurs |  |

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

|  |  |
| --- | --- |
| **Joueur d'une partie** | **Hébergeur d'une partie** |
|  | 1. L'hébergeur crée une partie |
|  | 2. Le système enregistre la partie nouvellement créée dans sa base de données |
|  | 3. L'hébergeur attend que des joueurs rejoignent la partie qu'il vient de créer |
| 1. le joueur demande la liste des parties (hébergeurs) disponibles |  |
| 2. Le système fournit la liste des parties (hébergeurs) en attente de joueurs |  |
| 3. Le joueur choisit voit le **nombre de parties** créées dans la liste qui s’affiche |  |

## Scénario 03 : Enregistrement d’une partie (succès)

|  |  |
| --- | --- |
| **Joueur d'une partie** | **Hébergeur d'une partie** |
|  | 1. L'hébergeur crée une partie |
|  | 2. Le système **enregistre** la partie nouvellement créée dans sa base de données |
|  | 3. L'hébergeur attend que des joueurs rejoignent la partie qu'il vient de créer |

## Scénario 04 : Suppression d’une partie (succès)

|  |  |
| --- | --- |
| **Joueur d'une partie** | **Hébergeur d'une partie** |
|  | 1. L'hébergeur crée une partie |
|  | 2. Le système enregistre la partie nouvellement créée dans sa base de données |
|  | 3. L'hébergeur attend que des joueurs rejoignent la partie qu'il vient de créer |
| 1. Les joueurs se connectent à la partie |  |
|  | 4. L’hébergeur signale au système que la partie est complète et qu’elle va commencer |
| 2. La partie commence | 5. Le système **efface** l’enregistrement de sa base de données |

## Scénario 05 : Mise à jour des informations d’une partie

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

## Scénario 06 : Fermeture de la connexion (succès)

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