logo.png

ASD Tower Defense

Version 2.0 – En réseau

Intégration d’un mode mutlijoueurs

2010

GEN

Heig-vd - GEN

01/06/2010

**mouton.png**

Vous avez l’esprit joueur ?

Recherchez les 7 créatures qui se cachent dans ce document.



Table des matières

[1 Introduction 3](#_Toc263898750)

[2 Analyse 3](#_Toc263898751)

[2.1 Règles du jeu 3](#_Toc263898752)

[2.1.1 Les créatures 3](#_Toc263898753)

[2.1.2 Les tours 3](#_Toc263898754)

[2.1.3 Les modes de jeu 3](#_Toc263898755)

[2.1 Partage des responsabilités 4](#_Toc263898756)

[2.2 Etapes de lancement du jeu 5](#_Toc263898757)

[2.2.1 Cas d’utilisation 5](#_Toc263898758)

[2.2.2 Acteurs principaux 6](#_Toc263898759)

[2.2.3 Scénario principal (succès) 6](#_Toc263898760)

[2.4 Cas d’utilisation 7](#_Toc263898761)

[2.4.1 Schéma global des acteurs 7](#_Toc263898762)

[2.6 Serveur d’enregistrement 8](#_Toc263898763)

[2.6.1 Acteurs principaux 8](#_Toc263898764)

[2.6.2 Scénario principal (succès) 9](#_Toc263898765)

[2.7 Serveur de Jeu 10](#_Toc263898766)

[2.8 Serveur Web 10](#_Toc263898767)

[3 Conception du projet 12](#_Toc263898768)

[3.1 Protocoles d’échange 12](#_Toc263898769)

[3.1.1 Serveur d’enregistrement 12](#_Toc263898770)

[3.1.2 Serveur de jeu 12](#_Toc263898771)

[3.2 Architecture de l’application 13](#_Toc263898772)

[3.2.1 Moteur du jeu 14](#_Toc263898773)

[3.2.2 Intégration du pattern Observable / Observé 15](#_Toc263898774)

[3.2.3 Réseau 17](#_Toc263898775)

[3.3 Charte graphique 18](#_Toc263898776)

[4 Gestion de projet 21](#_Toc263898777)

[4.1 Rôle des participants au sein du groupe 21](#_Toc263898778)

[4.2 Plan d’itérations initial 22](#_Toc263898779)

[4.2.1 Itération 1 – Serveur d’enregistrement + Interface graphique 22](#_Toc263898780)

[4.2.2 Itération 2 – Serveur de Jeu + Architecture 23](#_Toc263898781)

[4.2.3 Itération 3 –Intégration du serveur de jeu + Interface du Jeu en réseau 23](#_Toc263898782)

[4.2.4 Itération 4 – Lifting de la GUI + Game Design + Amélioration Mode Solo 24](#_Toc263898783)

[4.2.5 Itération 5 – Serveur Web (facultatif) 24](#_Toc263898784)

[5 Conclusion 25](#_Toc263898785)

[6 Annexes 25](#_Toc263898786)

# Introduction

Ce projet prend place durant notre 4e semestre aux seins 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 pensée à l’amélioration d’un jeu que nous avions créé durant nos cours d’Algorithmes et Structures de Données (ASD2) suivi durant notre 3e semestre. En effet, nous avions 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 final du projet reposant sur une gestion de projet basé 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.*

# Analyse

## 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.

### 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.

### 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 est de ne pas la créer là ou se trouve une créature à ce moment et il doit toujours y avoir un chemin entre la zone de départ et celle de fin.

### Les modes de jeu

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

Une première analyse nous a permis de déterminer plusieurs types de partie 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 d’exécution des opérations réduite.

Nous avons retenu le mode versus uniquement 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 ».

## Partage des responsabilités

Nous vous présentons ici les personnes **responsables** des différentes implémentations du projet. Bien évidement, chaque membre apportera ça 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 incombe.

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) | araignee.png |  |  |  |

## 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.

### Cas d’utilisation



Figure : Système de lancement et de fin de jeu

### Acteurs principaux

* Joueur
* Serveur de jeu

### Scénario principal (succès)

|  |  |
| --- | --- |
| **Joueur** | **Serveur de jeu** |
| 1. Le joueur choisis le jeu multijoueur qu’il veut créer ou rejoindre avec l’interface |  |
| 2. Le joueur choisis de rejoindre une partie ou en crée une |  |
| 3. Le joueur se connecte au serveur de jeu |  |
|  | 4. Le serveur accepte la demande et la traite (création de la partie ou ajout du joueur à une partie |
|  | 5. Le cas échéant, informe les autres joueurs de l’arrivée du dernier |
| 6. Le joueur choisis son équipe |  |
|  | 7. Le serveur valide le choix du joueur et informe les autres de ce changement |
|  | 8. Le serveur lance la partie a proprement dit |
| 9. Le joueur joue sa partie |  |
|  | 10. Lorsque la fin de partie est détectée, les joueurs sont informés et déconnectés de la partie |

## Cas d’utilisation

### Schéma global des acteurs

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



Figure : Schéma global des acteurs

## 

## Serveur d’enregistrement

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



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

### Acteurs principaux

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

### Scénario principal (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 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 joueur attend que la partie commence (que d’autres joueurs se connectent) |  |
|  | 5. L’hébergeur signale au système que la partie est complète et qu’elle va commencer |
| 6. La partie commence | 6. Le système efface l’enregistrement de sa base de données |

## 

## 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 : Système de serveur de gestion des parties

## Serveur Web

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

# Conception du projet

## Protocoles d’échange

### 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 avons décidé d’utiliser comme format de message le JSON (*JavaScript Object Notation),* 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.

*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*.

### 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 crées par le client cible ou d’autre 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.*

## Architecture de l’application

L’architecture d’ASD - Tower Defense à subie d’énormes changements dans le but d’intégrer correctement la notion de partie réseaux. 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 du intégrer différentes notions et adapter toute la partie solo pour quelle 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. Effectivement, la plupart des membres du projet portent un intérêt particulier à continuation de ce jeu.

### Moteur du jeu

Voici le diagramme de classes représentant le moteur du jeu. Il correspond au Model & Controleur dans une architecture MVC. Celui-ci et utilisé tel-quel 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 fourni. Le client utilise tout de même les gestionnaires mais uniquement pour calculer certains élément du jeu afin d’évité que le serveur doive envoyer en tout temps l’état du jeu.

|  |  |
| --- | --- |
| C:\Users\Dark\Desktop\Class Model - Server.png  Figure : Schéma du moteur du jeu  *Ce schéma de plus grande taille est fourni en annexe.* | 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 fourni les **équipes** initiales et les **vague de créature**s 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. |

### 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 5 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 : Qui observe quoi ?

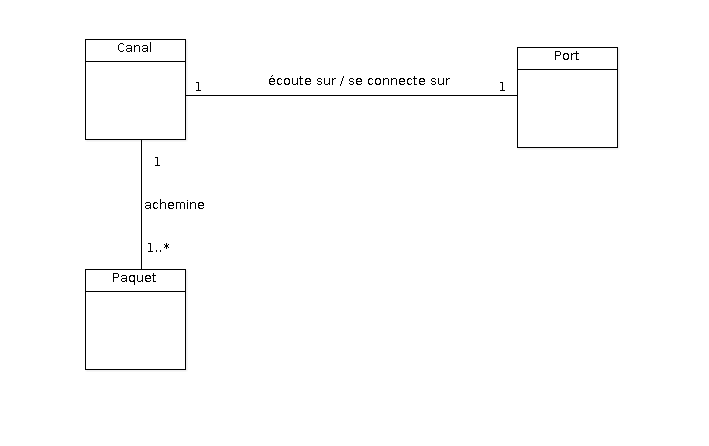
Ce schéma permet d’illustrer les différentes classes observatrices de l’application. Nous pouvons remarquer ici la présence du jeu qui en fait et une sorte de contrôleur qui écoutes les principaux éléments du model. S’il vient à recevoir une notification, il effectue les opérations qu’il lui semble nécessaires 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 notifier l’utilisateur final des changements d’états du jeu. Le gestionnaire réseaux est également notifié de ces changements afin de transmettre aux clients connectés les divers modifications qui on eut lieux. 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 réalisée 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 |

### 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.



## 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.

|  |  |
| --- | --- |
| D:\Developpement\Java\WorkSpace\ASDTowerDefense\documents\v2.0\charteGraphique2.png | **Menu principal**  Premier menu lors du lancement du jeu. Trois choix sont offerts au joueur :  **1. Partie solo**  **2. Créer une partie multijoueurs**  **3. Rejoindre une partie multijoueurs** |
| **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 | elephant.pngD:\Developpement\Java\WorkSpace\ASDTowerDefense\documents\v2.0\interface_ModeSolo.png |
|  | **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 créer 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 moment. |  |
|  | **3. Rejoindre une partie réseau**  Plaçons nous maintenant du coté du client qui veux 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… |
| **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… |  |

|  |
| --- |
| **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). |

# 

# Gestion de projet

## 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** |  |  |  | 🞮 |

## 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 à débuté le **19 avril 2010** et il est à rendre le **11 juin 2010** avant la présentation.

### Itération 1 – Serveur d’enregistrement + Interface graphique

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

**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-t-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)** | **pigeon.pngFonctionnalité** |
| 🗆 | Lazhar | Etablir une connexion client / serveur avec échange de message |
| 🗆 | Lazhar | Enregistrer une partie sur le serveur d’enregistrement. |
| 🗆 | Lazhar | Voir les parties inscrites sur le serveur d’enregistrement |
| 🗆 | Lazhar | « Désenregistrer » d’une partie sur le serveur d’enregistrement. |
| 🗆 | Lazhar | Mettre à jour les informations d’une partie. |
| 🗆 | Aurélien | Interface graphique pour l’enregistrement de la partie |
| 🗆 | Aurélien | Interface graphique pour voir les parties inscrites |

## 

### Itération 2 – Serveur de Jeu + Architecture

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

**Pourquoi cette itération prend-t-elle place ici ?**

On prépare tous les éléments pour les fusionner ensuite (avec adaptations quasis 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 ente 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** |
| 🗆 | Pierre-Dominique & Romain | Implémentation de tous les messages fournis par le protocole. (liste trop exhaustive pour les citer tous, *se référer au protocole en annexe*) |
| 🗆 | Aurélien | Le mode 1 joueur fonctionne correctement et comme avant. Présentation du schéma de classe ou de domaine. |

## 

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

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

**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 pas l’entité réceptrice.

|  |  |  |
| --- | --- | --- |
| **Fonctionnalités Attendues** | | |
| **Ok** | **Responsable(s)** | **Fonctionnalité** |
| 🗆 | Aurélien | L’interface permet de solliciter des actions du jeu. |
| 🗆 | Aurélien | Le client peut se connecter à une partie de jeu. |
| 🗆 | 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. |

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

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

**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’a 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 propre 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.

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

|  |  |  |
| --- | --- | --- |
| **Fonctionnalités Attendues** | | |
| **Ok** | **Responsable(s)** | **Fonctionnalité** |
| 🗆 | Lazhar | Lifting de l’interface, celle-ci ressemble plus à un jeu. |
| 🗆 | Pierre-Dominique & Romain | Adaptation des valeurs, jeu plus agréable. |
| 🗆 | Aurélien | Système de progression mise en place. |

### Itération 5 – Serveur Web (facultatif)

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

**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** |
| 🗆 | Romain | Mise en place du serveur web |
| 🗆 | Aurélien | Intégration du service dans l’application |

# Conclusion

Le projet avance bien, nous avons maintenant une bonne idée des choses restantes à faire et nous sommes impatients de pouvoir mettre en œuvre tous les artefacts créés dans ce rapport intermédiaire.

# 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. Diagramme de classes du client

aigle.png