



RAPPORT FINAL DE STAGE

DÉVELOPPEMENT DE JEUX SÉRIEUX

RÉALISÉ PAR

BARREAU Anthony

SOUS LA DIRECTION DE

GOUAICH Abdelkader

ANNÉE UNIVERSITAIRE 2009 - 2010

RAPPORT FINAL DE STAGE

DÉVELOPPEMENT DE JEUX SERIEUX

RÉALISÉ PAR

BARREAU Anthony

SOUS LA DIRECTION DE

GOUAICH Abdelkader

ANNÉE UNIVERSITAIRE 2009 - 2010

Remerciements

Je tiens premièrement à remercier mon tuteur, Monsieur Abdelkader GOUAICH, qui m'a assisté et soutenu tout au long de la période de stage.

Je tiens également à remercier Madame Ines DI LORETO, Madame Huguette ALBERNHE-GIORDAN, Monsieur Michael BERGERET et Monsieur Fabien MICHEL pour leurs conseils et l'expérience qu'ils ont su m'apporter.

Enfin mes remerciements s'adressent à l'ensemble du personnel du LIRMM et de l'IUT de Montpellier qui ont mis à ma disposition leurs moyens techniques ainsi que leurs connaissances.

Table des matières

1– Introduction.....	10
2– Présentation du stage	11
2.1 – Le LIRMM et l'équipe SMILE :	11
2.2 – Les jeux sérieux :	12
2.3 – School Society en détail	12
2.4 – Principe de fonctionnement de gameQuery	16
2.4.1 – Le DOM : gestion des sprites	16
2.4.2 – Les animations	17
2.4.3 – RegisterCallback : une boucle infinie.....	19
2.5 Outils utilisés	19
2.5.1 Redmine	19
3– Cahier des charges.....	22
4– Développement de jeux sérieux	24
4.1 Analyse du besoin	24
4.1.1 Modèle relationnel des données.....	24
4.2 Conception	25
4.2.1 Ajout automatique d'un mini jeu.....	25
4.2.2 Mini jeu n°1 : Castle Defender	26
4.2.2.1 Initialisation.....	26
4.2.2.2 Les monstres	27
4.2.2.3 Lancement du jeu	28
4.2.3 Mini jeu n°2 : Castle Defender compétitif.....	29
4.2.3.1 La salle d'attente	29
4.2.3.2 Le jeu.....	30
4.2.3.3 Détection des différents évènements	31
4.2.4 Mini jeu n°3 : Pandémie.....	33
4.2.4.1 L'alternance des tours de jeu.....	33
4.2.4.2 Les déplacements des joueurs et évènements.....	34
4.2.4.3 L'expansion de la menace	36
4.3 Résultats	38
4.3.1 Castle Defender :.....	38
4.3.2 Castle Defender compétitif :	40

4.3.3 Pandémie :	41
5– Back office	44
5.1 Analyse du besoin	44
5.1.1 Diagramme de cas d'utilisations :	44
5.1.2 Modèle relationnel des données :	45
5.1.3 Diagramme de séquences :	45
5.2 Conception	46
5.2.1 Gestion des quiz.....	46
5.2.2 Gestion des membres	47
5.2.3 Gestion de la gazette.....	47
5.2.4 Statistiques	47
5.3 Résultats	48
5.4 Impact du back office	52
6– Discussions.....	54
7– Conclusion	55
8– Sitographie	56
9– Annexes.....	57

Glossaire

- **AJAX** (Asynchronous Javascript And XML) : ce terme évoque l'utilisation conjointe de plusieurs technologies comme le HTML, CSS, DOM et Javascript.
- **API** (Application Programming Interface) : interface de programmation permettant de développer des applications.
- **ATH** (Affichage Tête Haute) : ensemble des éléments relatifs au statut du joueur affichés à l'écran dans un jeu.
- **Back-office** : zone d'un site web réservé à son administrateur et qui permet de gérer / apporter du contenu au site.
- **DOM** (Document Object Model) : description de la structure d'un document HTML et XML.
- **Framerate** : Taux de rafraîchissement : nombre d'images par seconde affiché dans un jeu.
- **Framework** : ensemble de bibliothèques et d'outils permettant de développer une application.
- **Gameplay** : terme qui définit la jouabilité d'un jeu : sensations, déplacements, type de jeu etc.
- **Invader-like** : style de jeu où le joueur subit des attaques de monstres organisées en vagues que celui-ci doit repousser.
- **Lag** : ralentissement dans un jeu entraîné par un engorgement du traitement des données.
- **phpMyAdmin** : application web permettant de gérer une base de données.
- **PvE** (Player Versus Environnement) : mode de jeu où le joueur doit se battre contre les différents éléments de l'environnement tels que les animaux, les conditions climatiques, etc.
- **PvP** (Player Versus Player) : mode de jeu où le joueur doit se battre contre d'autres joueurs
- **Sprite** : élément graphique dans un jeu qui peut se déplacer
- **Tuple** : désigne un enregistrement dans une table d'une base de données

Table des illustrations

Figure 1 : modèle d'analyse de système social	11
Figure 2 : alternance phase ludique et pédagogique dans les jeux sérieux	12
Figure 3 : points de caractéristiques du joueur.....	13
Figure 4 : classement par expérience des joueurs	14
Figure 5 : page d'accueil des quiz	15
Figure 6 : interface de gestion des messages privés.....	15
Figure 7 : section Mes Potes	16
Figure 8 : vue en 3 dimensions de la superposition de sprites	17
Figure 9 : image source pour les animations d'un personnage	18
Figure 10 : ligne d'animation par états.....	18
Figure 11 : liste des tâches rapportées.....	19
Figure 12 : visualisation d'une tâche.....	20
Figure 13 : ajout d'une tâche	21
Figure 14 : modèle relationnel des données des jeux compétitif et pandémie	24
Figure 15 : une masse (en bleu) : élément de collision	27
Figure 16 : calcul d'angle entre deux points et l'axe vertical	28
Figure 17 : connexion à une partie existante.....	29
Figure 18 : attente d'un joueur en mode compétitif.....	30
Figure 19 : synchronisation des deux joueurs	30
Figure 20 : fin de partie après déconnexion de l'adversaire	32
Figure 21 : réception d'un malus envoyé par l'adversaire.....	32
Figure 22 : représentation du terrain de jeu.....	35
Figure 23: calcul de la nouvelle case malade.....	36
Figure 24 : ATH de Castle Defender	39
Figure 25 : fin de vague dans Castle Defender	39
Figure 26 : malus envoyé par l'adversaire	40
Figure 27 : chat du mode PvE	41
Figure 28 : système de voisinage d'une case.....	42
Figure 29 : représentation des cases dans le mode jeu Pandémie	43
Figure 30 : diagramme de cas d'utilisations de programmation.....	44
Figure 31 : modèle relationnel des données du back-office	45
Figure 32 : diagramme de séquence d'ajout d'une catégorie	46

Figure 33 : diagramme de séquence de consultation des statistiques	46
Figure 34 : section quiz du back-office	49
Figure 35 : panneau d'édition d'une question	49
Figure 36 : panneau d'édition d'un membre.....	50
Figure 37 : page de gestion des gazettes.....	50
Figure 38 : édition d'une gazette.....	51
Figure 39 : panneau de création de graphes.....	51
Figure 40 : génération d'un graphe avec deux histogrammes.....	52
Figure 41 : fin d'un quiz	53

1 – Introduction

Au cours de la deuxième année du DUT informatique à l'IUT de Montpellier, chaque étudiant doit réaliser un stage d'une durée de deux mois et demi au sein d'une entreprise. Celui-ci a pour but de préparer l'étudiant à son arrivée dans le monde du travail en mettant en pratique les connaissances qu'il a acquies durant sa scolarité.

Ce rapport va présenter le stage de deux mois et demi que j'ai réalisé au sein de l'équipe SMILE dans les locaux du LIRMM. Mon but a été de participer au développement d'un projet de jeu sérieux sur lequel l'équipe travaille : School Society. Il m'a été demandé de modifier le moteur de jeu existant pour permettre la création de minis jeux et d'ajouter du contenu à ce moteur.

Dans un premier temps je vais clarifier le thème de ce stage au sein de l'équipe qui m'a accueillie, puis dans un second temps, le cahier des charges défini avec le maître de stage. Dans une troisième partie, j'aborderai le thème de la conception de jeux sérieux puis ensuite celui du back-office produit et enfin je discuterai des travaux réalisés.

2 – Présentation du stage

2.1 – Le LIRMM et l'équipe SMILE :

Le LIRMM, Laboratoire d'Informatique de Robotique et de Microélectronique de Montpellier est une entité de recherche de l'université de Montpellier 2 (UM2) et du Centre de la Recherche National Scientifique (CNRS).

Au sein de cette unité, l'équipe SMILE, composée de 16 personnes, porte ses travaux sur plusieurs projets dont notamment School Society dans lequel Mr. Abdelkader GOUAICH, Mr. Michael BERGERET et Mme. Ines DI LORETO sont impliqués.

School Society est une expérimentation mise en place auprès d'étudiants de première année de DUT Informatique afin d'étayer la thèse de Mme DI LORETO. Cette thèse a pour but de démontrer la validité d'un modèle d'analyse de systèmes sociaux dans les logiciels et de proposer un framework permettant la création de ce genre de systèmes. Voici comment celui-ci s'organise.

L'analyse d'un système social se fait autour de 4 dimensions :

- l'identité : a quel point le système peut refléter la personnalité de l'utilisateur
- l'espace : le degré de personnalisation de l'espace alloué à l'utilisateur par le système
- la persistance : le système propose t-il une sauvegarde des informations dans le temps
- l'action : le nombre d'actions que peut réaliser l'utilisateur, si celui-ci est passif ou non

Ainsi, en énumérant les différentes fonctionnalités d'un système et en quantifiant chacune d'entre elles, il est possible d'obtenir un diagramme reflétant la configuration du système.

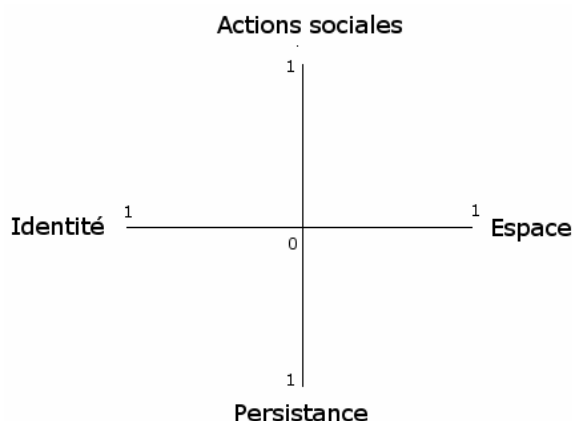


Figure 1 : modèle d'analyse de système social

School Society a été créé suivant une forme précise et, au début de l'expérimentation, différents groupes d'utilisateurs ont été créés dans le but de répartir l'accès aux fonctionnalités : certains groupes n'avaient pas accès à des éléments particuliers comme le chat par exemple.

En réalisant une étude de satisfaction auprès des utilisateurs, le modèle peut ainsi être démontré.

2.2 – Les jeux sérieux :

Le principe d'un jeu sérieux est de permettre la transmission d'une information au joueur dans un but généralement pédagogique en utilisant des mécanismes liés aux jeux vidéo traditionnels. Il s'agit généralement d'une application informatique où un joueur manipule un avatar à travers plusieurs épreuves pédagogiques et ludiques.

La difficulté dans la conception d'un jeu sérieux est que les phases ludiques et pédagogiques doivent s'enchaîner de façon régulière et équilibrée afin de maintenir la motivation du joueur et il ne faut surtout pas que ce dernier puisse trouver un moyen de rester uniquement dans l'une des deux phases, la phase ludique par exemple.

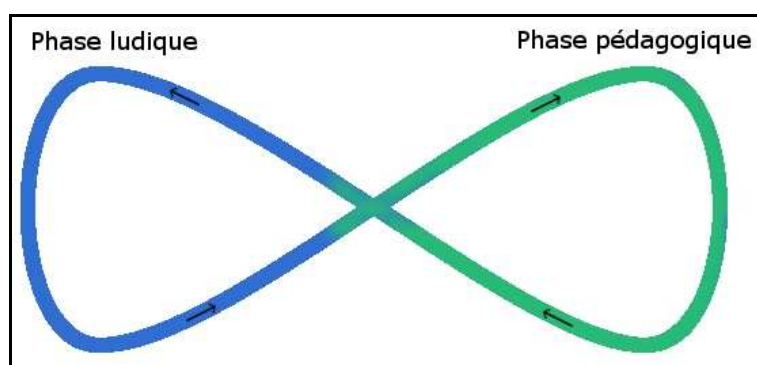


Figure 2 : alternance phase ludique et pédagogique dans les jeux sérieux

Il est possible par exemple de mettre en place un système de vases communicants : le seul moyen pour que le joueur puisse progresser dans la phase ludique est de réaliser une action dans la phase pédagogique.

Ce genre de verrou ne doit pas être trop contraignant au risque de démotiver le joueur.

2.3 – School Society en détail

Cette partie décrit la situation de School Society à mon arrivée dans ce stage.

Le projet School Society est un site web composé de plusieurs fonctionnalités. Chaque joueur possède des caractéristiques telles que l'expérience, la force ou

encore l'agilité et toutes sont quantifiées par des points qu'il est possible d'augmenter afin de devenir plus puissant.

Utilisateur: abserver.abserver	Niveau: 5
Email: [REDACTED]	EXP: 2/780 (0%)
Inscription: February 4, 2010, 1:12 am	HP: 170/170
Age: 56 days	Energie: 8/10
Assassinats/Morts: 19/27	PO:589
Jetons d'examen: 47	Force: 3
	Vitalité: 4
	Agilité:4
<i>Vous avez 3 points de stat à dépenser !</i>	
<i>Utilisez les point stat ici</i>	

Figure 3 : points de caractéristiques du joueur


Le site web est organisé de façon à pouvoir alterner phase ludique et phase pédagogique :

La phase ludique est assurée par les rubriques « Mode PvP » et « Mode PvE ».

Dans le mode PvP, chaque joueur peut défier un autre étudiant dans un combat afin de gagner de l'expérience. Chaque joueur ne peut défier que des adversaires se trouvant dans une tranche de niveau bien précise. En effet, seules les personnes ayant un niveau de moins ou cinq de plus maximum que la personne qui engage le combat peuvent être défiées, ceci afin d'éviter aux joueurs qui auraient pris de l'avance de ne combattre uniquement que les plus faibles et ainsi déséquilibrer le jeu.

Plus un joueur a d'expérience, plus celui-ci monte en niveau. Plus un joueur a un niveau élevé, plus celui-ci est puissant. Par ailleurs, un classement des joueurs par niveau existe dans la catégorie du même nom.

Members (classement par XP):



[Previous Page](#) | [Next Page](#)

Montrer [5](#) | [10](#) | [20](#) | [30](#) | [40](#) | [50](#) | [100](#) membres par page.

Username	Niveau	Action
[blurred]	41	Mail Défier
[blurred]	24	Mail Défier
[blurred]	19	Mail Défier
[blurred]	14	Mail Défier
[blurred]	12	Mail Défier
[blurred]	12	Mail Défier
[blurred]	11	Mail Défier
[blurred]	10	Mail Défier
[blurred]	10	Mail Défier

Figure 4 : classement par expérience des joueurs

Dans le mode PvE, le joueur se retrouve sur différentes cartes sur lesquelles il peut déplacer son avatar. Suivant le type de carte choisie, les objectifs ne seront pas les mêmes. Par exemple, sur la carte nommée **Tréant**, le joueur devra réaliser plusieurs quêtes dans le but de pouvoir progresser et découvrir de nouveaux items.

Sur une carte comme **Castle Defender**, le joueur va uniquement combattre des monstres jusqu'à atteindre un seuil qui lui permettra d'obtenir un objet essentiel dans l'accomplissement d'une quête annexe sur une autre carte.

Lorsqu'un joueur perd l'ensemble de ses points de vie, celui-ci a deux alternatives : soit il peut aller à l'hôpital et payer pour se faire soigner, soit il peut attendre le lendemain que l'ensemble de ses points de vie soient régénérés.

La phase pédagogique elle est assurée par le système de quiz et de récompense. En effet, la seule et unique source de revenus pour les joueurs est matérialisée par les pièces d'or gagnées lors des réponses aux différents quiz proposés. Lorsqu'un joueur répond à un quiz, suivant sa difficulté et le nombre de réponses justes qu'il aura donné, celui-ci gagne un certain nombre de pièces d'or qui sont nécessaires pour l'achat de nouveaux objets ou de nouvelles cartes pour le mode PvE dans la boutique.



Figure 5 : page d'accueil des quiz

Plus un joueur répond à un quiz, moins le nombre de pièces d'or gagnées est important. Également, un joueur ne peut répondre à un quiz que toutes les trente minutes et ne peut en cumuler que trois dans la journée.

Pour dialoguer entre eux, les étudiants possèdent deux systèmes de communication.

Un dit privé : les joueurs peuvent s'envoyer des mails et dialoguer de façon confidentielle.

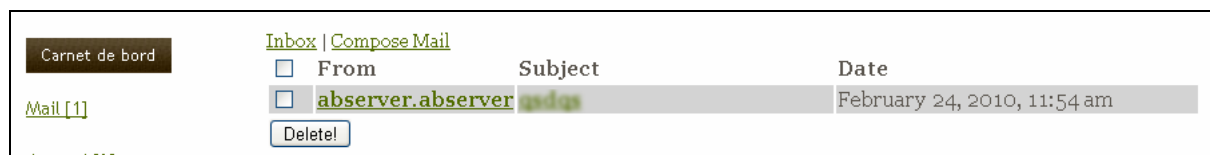


Figure 6 : interface de gestion des messages privés

Un dit publique : la section Mes Potes est une sorte de chat général où les joueurs peuvent dialoguer rapidement entre eux.

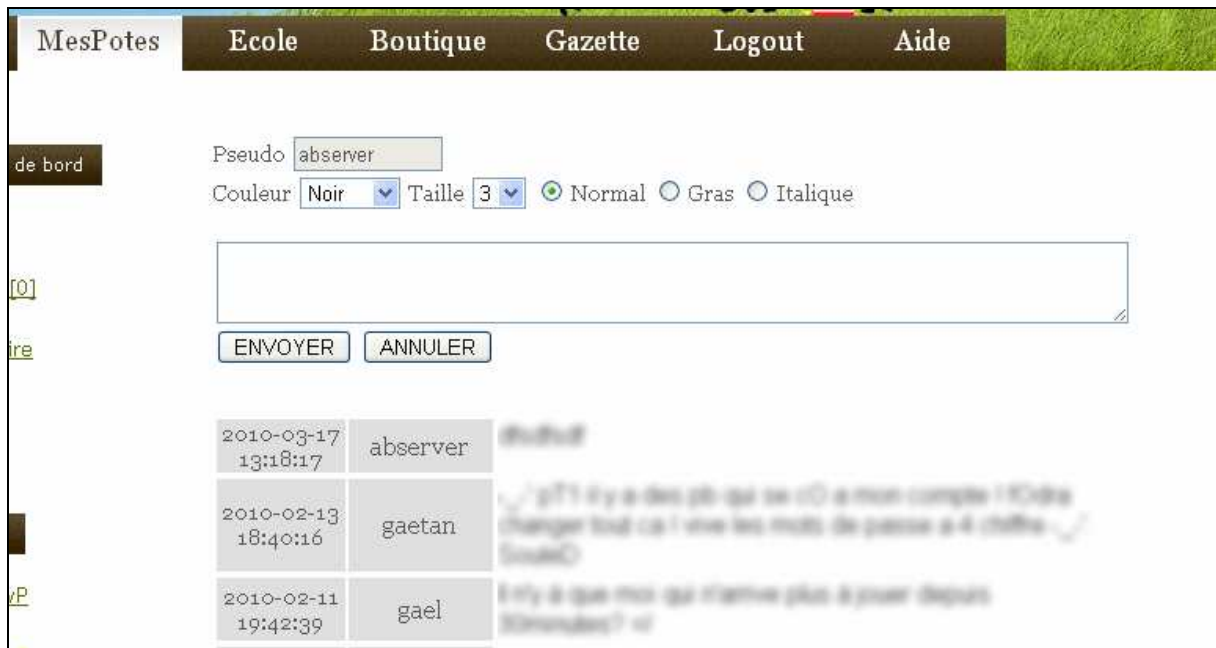


Figure 7 : section Mes Potes

2.4 – Principe de fonctionnement de gameQuery

gameQuery est un framework Javascript basé sur l'API **jQuery** et permet de développer des petits jeux plus facilement. En effet, celui-ci propose plusieurs structures permettant un gain de temps considérable lors du développement des fichiers sources. Cette librairie a été utilisée pour la conception de tous les jeux figurant dans la partie PvE.

2.4.1 – Le DOM : gestion des sprites

gameQuery propose une vision en couche des images. Chaque image, appelée sprite, correspond à une division HTML et est positionnée de façon absolue dans la page.

Chaque élément a son propre calque et chaque calque appartient à un groupe qui est lui aussi une division HTML.

Ainsi, pour générer un empilement d'image, il faut d'abord créer un empilement de groupes et ensuite associer chaque sprite à un groupe.

L'ensemble de ces groupes appartient à une division spécifique : **playground**

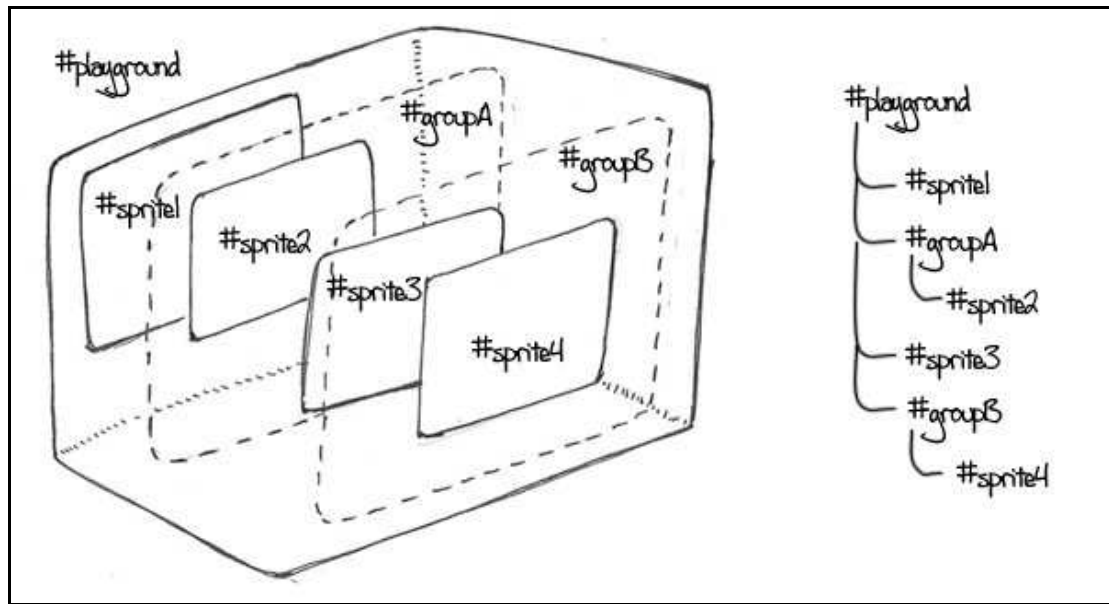


Figure 8 : vue en 3 dimensions de la superposition de sprites

Voici le code HTML correspondant :

```
<div id="playground">
  <div id="sprite1"></div>
  <div id="groupA">
    <div id="sprite2"></div>
  </div>
  <div id="sprite3"></div>
  <div id="groupB">
    <div id="sprite4"></div>
  </div>
</div>
```

2.4.2 – Les animations

Les animations sont en réalité une succession de portions d'images. Voici le fichier PNG source d'un personnage :



Figure 9 : image source pour les animations d'un personnage

Chaque ligne correspond à l'ensemble des images à mettre les unes à la suite des autres suivant l'état du personnage. L'enchaînement de ces différentes portions donnera l'illusion que le personnage est en mouvement.

Dans gameQuery, cette gestion des animations se fait d'une façon très simple. Pour créer une nouvelle animation il suffit d'utiliser le constructeur de la classe **Animation** auquel il faut fournir plusieurs paramètres :

```
new Animation({imageURL: "./images/licorne2.png", numberOfFrame: 4, delta: 80, rate: 180, distance: 80, type: Animation.HORIZONTAL | Animation.MULTI});
```

- imageURL : lien absolu ou relatif de l'image source de l'animation
- numberOfFrame : nombre d'images sur une même ligne : ici 4
- delta : hauteur de l'image du personnage
- distance : largeur de l'image du personnage
- rate : nombre de successions d'images par secondes
- type : si l'ensemble des images de l'animation est stocké sur une ou plusieurs lignes. Animation.HORIZONTAL signifie qu'il faut lire les images de façon horizontale et Animation.MULTI signifie qu'il y a plusieurs lignes

Ensuite, lorsque l'on souhaite modifier à l'écran l'animation d'une entité possédant cette animation au préalable, il suffit d'utiliser **setAnimation**. Si les différentes images sont disposées sur plusieurs lignes dans l'image source alors l'appel de la fonction setAnimation va modifier la ligne d'images affichée à l'écran.

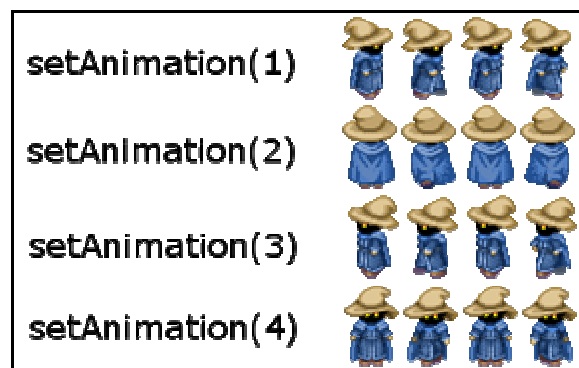


Figure 10 : ligne d'animation par états

2.4.3 – RegisterCallback : une boucle infinie

La dernière particularité de gameQuery est que le framework propose une fonction particulière que l'on peut considérer comme une boucle infinie : **registerCallback**.

Cette fonction est utile car lorsque celle-ci est placée dans la fonction principale du jeu, elle permet de faire tourner le jeu jusqu'à ce qu'un évènement vienne l'arrêter et ceci à un taux de rafraîchissement particulier spécifié en paramètre. Cela permet de fixer un nombre d'images par seconde facilement et d'éviter de devoir utiliser une autre fonction comme setInterval de Javascript qui réalise la même chose, mais d'une façon plus contraignante pour le programmeur.

2.5 Outils utilisés

Certains logiciels comme l'éditeur de texte Notepad++ ont été utilisés au cours de ce stage. Leur utilisation étant très basique, je ne vais pas m'attarder sur leur présentation. Intéressons nous plutôt à l'outil de gestion de projet qui a été utilisé.

2.5.1 Redmine

RedMine est un système de gestion de projets en mode Web. Il propose plusieurs fonctionnalités permettant de gérer et de coordonner l'ensemble des personnes l'utilisant.

Parmi celles-ci, en voici quelques unes :

- Gestion d'un wiki
- Système de gestion des problèmes/bugs
- Diagrammes de Gantt
- Hébergement et envoi de fichier par formulaire

Dans mon cas, ce logiciel m'a été très utile pour informer les autres personnes travaillant sur School Society de l'avancée des éléments que nous avons prévu de mettre en place.

A chaque nouvel élément était créée une tâche dans le RedMine et au cours du développement, l'état d'avancement de celle-ci était mis à jour. A chaque modification, un mail est envoyé par le logiciel à l'ensemble des développeurs avec la liste des modifications effectuées.

✓ #	Tracker	Status	Priority	Subject	Assigned to	Updated
416	Task	Closed	Normal	Système de timeout	barreau anthony	04/02/2010 01:24 PM
415	Bug	Closed	Normal	Problème de déplacement	barreau anthony	04/02/2010 03:06 PM
414	Bug	New	Normal	Changement de joueur	barreau anthony	04/01/2010 12:55 PM
402	Task	Closed	Normal	Pandémie : la menace	barreau anthony	04/01/2010 12:53 PM
400	Task	Closed	Normal	Pandemie : bases	barreau anthony	03/29/2010 02:31 PM
390	Task	Closed	Normal	Images for PVE	Ines Di Loreto	03/18/2010 12:12 PM
389	Task	New	Normal	Mettre le log	Ines Di Loreto	03/16/2010 03:34 PM

Figure 11 : liste des tâches rapportées

Toutes les tâches rapportées peuvent être consultées et mises à jour par l'ensemble des utilisateurs du logiciel.

Task #402 Update Log time Watch Copy

Pandémie : la menace

Added by barreau anthony 13 days ago. Updated 4 days ago.

Status :	Closed	Start :	03/23/2010
Priority :	Normal	Due date :	
Assigned to :	barreau anthony	% Done :	<div style="width: 100%; background-color: green; height: 10px;"></div> 100%
Category :	-	Spent time :	-
Target version :	release-2.0		

Description

- Choisir comment matérialiser la menace à l'écran
- Déterminer un algorithme d'expansion (jeu de la vie ?)
- Éclaircir le gameplay : comment les joueurs sont sensés enrayer la pandémie (d'un point de vue technique) ?

Related issues Add

History

03/30/2010 02:37 PM - barreau anthony #1

- % Done changed from 0 to 50

- La menace s'étend maintenant de façon aléatoire mais dans un intervalle de cases restreint de sorte à avoir une progression du centre vers l'extérieur.
 - Elle est matérialisée par des cases rouges à l'écran
 - Elle ne peut apparaître sur la case d'un joueur

- Reste à gérer la régression de la menace grâce aux actions des joueurs

04/01/2010 12:53 PM - barreau anthony #2

- Status changed from *New* to *Closed*
- % Done changed from 50 to 100

Si une case malade a dans ses voisins au moins deux joueurs, celle ci disparaît.

Figure 12 : visualisation d'une tâche

Afin d'être le plus efficace possible, j'ai choisi d'utiliser le diagramme de Gantt fourni par RedMine.

Lorsqu'une tâche est créée, il est possible d'ajouter le temps nécessaire à la réalisation de celle-ci et de spécifier la version du logiciel ciblée. La visualisation du diagramme permet ensuite de voir l'intervalle de temps dont nous disposons avant de devoir fournir la prochaine version de School Society.

Tracker * Task

Subject * Pandémie : fin de partie

Description * **B** *I* U ~~S~~ **C** H1 H2 H3 pre

Ajouter un évènement de fin de partie dans le mode de jeu Pandémie.

Status * New Start 2010-04-06

Priority * Normal Due date 2010-04-14

Assigned to barreau anthony Estimated time 5 Hours

Category New category % Done 0 %

Target version release-2.0

Files Aucun f... choisi *Optional description*
Add another file (Maximum size: 5 MB)

Figure 13 : ajout d'une tâche

Pour plus d'informations sur les différentes tâches de School Society, il est possible de visualiser mon diagramme de Gantt en annexes.

3 – Cahier des charges

Voici le cahier des charges que j'ai défini à l'aide de mon tuteur :

- Modifier le moteur de jeu actuellement utilisé pour permettre la création de minis jeux sans avoir à tout réécrire : il fallait fournir aux futurs développeurs une sorte de boîte à outils leur permettant de créer très rapidement leur propre jeu sans devoir modifier les fichiers source.
- Créer la documentation du moteur de jeu de School Society et ajouter des commentaires aux différentes fonctions : cette tâche, non négligeable, permet un gain de temps considérable pour un utilisateur lambda lors de la compréhension de l'architecture du moteur.
- Créer un minis jeu de type invader-like pour apporter un peu de nouveauté au mode PvE existant : Castle Defender. Il s'agit d'une carte de défense de château où le joueur subit des vagues de monstres successives de plus en plus rapprochées. S'il parvient à rester en vie et à défendre son château jusqu'à la dernière vague de monstres, alors le joueur a gagné.
- Ajouter des fonctionnalités au site web existant :
 - o Permettre aux joueurs de personnaliser leur espace avec une description d'eux même
 - o Choisir un avatar parmi une galerie existante qui sera intégré dans le mode de jeu PvE.
 - o Créer un nouveau type de classement par points gagnés dans les quiz : ce classement sera celui retenu pour le gain de l'objet final.
- Mettre en place un gameplay multi-joueurs dans le mode PvE. L'interaction entre les différents joueurs se limitant au mode PvP jusqu'à présent, il était intéressant de l'étendre au mode PvE en proposant plusieurs types de jeu :
 - o Créer un mode de jeu compétitif : les deux joueurs sont en compétition sur la même carte et le premier à avoir rempli un objectif gagne la partie. Chaque joueur peut lancer des malus à l'autre joueur au fur et à mesure de la partie pour ralentir la progression de son adversaire.
 - o Créer un mode de jeu de type pandémie : jeu en tour par tour qui permet de développer le côté coopératif de School Society : plusieurs joueurs doivent s'entraider pour freiner et enrayer l'expansion d'une menace.
- Structurer le planning de développement

J'ai dû faire face à plusieurs contraintes techniques comme notamment les langages de programmation déjà utilisés :

- Le Javascript : employé pour créer les minis jeux du mode PvE. Le moteur est entièrement basé sur ce langage dynamique à objets.
- Le PHP : pilier de tout site web dynamique, il sert notamment à récupérer des données sur un serveur distant et les affiche au client. Il s'agit également d'un langage à objets.

Il m'a également fallu apprendre à utiliser le framework **jQuery** car le moteur **gameQuery**, utilisé pour la conception du mode PvE, était basé sur cette API.

Des contraintes de temps m'ont également été imposées. Suivant le planning établi, il fallait mettre à disposition du publique les différentes releases du projet, généralement au rythme d'une tous les 10 ou 14 jours environ.

Enfin, s'adapter au code déjà écrit a été la contrainte majeure. Cette tache de rétro ingénierie n'a pas été la plus simple car il n'existait pas à proprement parler de documentation sur la logique de programmation des jeux School Society. A mon arrivée, de nombreux éléments avaient déjà été codés comme par exemple les déplacements du joueur ou des monstres. Même si les jeux déjà réalisés se basent sur l'API gameQuery, cette boite à outils ne fournit qu'une structure permettant de manipuler un empilement de sprites. Toute la logique de quête, de déplacement des monstres et du joueur était propre à School Society.

Voici comment va s'organiser la suite de ce rapport. Nous allons aborder les deux thèmes principaux du stage à savoir la conception de jeux sérieux et du back office. Pour chaque partie, je vais d'abord présenter l'analyse du besoin réalisée, puis expliquer la phase de conception avec une explication détaillée de la logique de programmation et enfin nous verrons les résultats produits.

4 – Développement de jeux sérieux

Cette partie va présenter l'ensemble des travaux réalisés ayant attiré aux minis jeux de School Society.

4.1 Analyse du besoin

La création des minis jeux de type compétitif et pandémie a nécessité de revoir légèrement l'architecture de la base de données déjà existante, c'est pourquoi j'ai réalisé un modèle relationnel des données et non un diagramme de classes.

4.1.1 Modèle relationnel des données

Le diagramme ci-dessous ne correspond pas au modèle relationnel complet de la base de données : il ne contient que les tables relatives à l'étude des jeux créés. Pour le consulter en entier, veuillez vous reporter aux annexes.

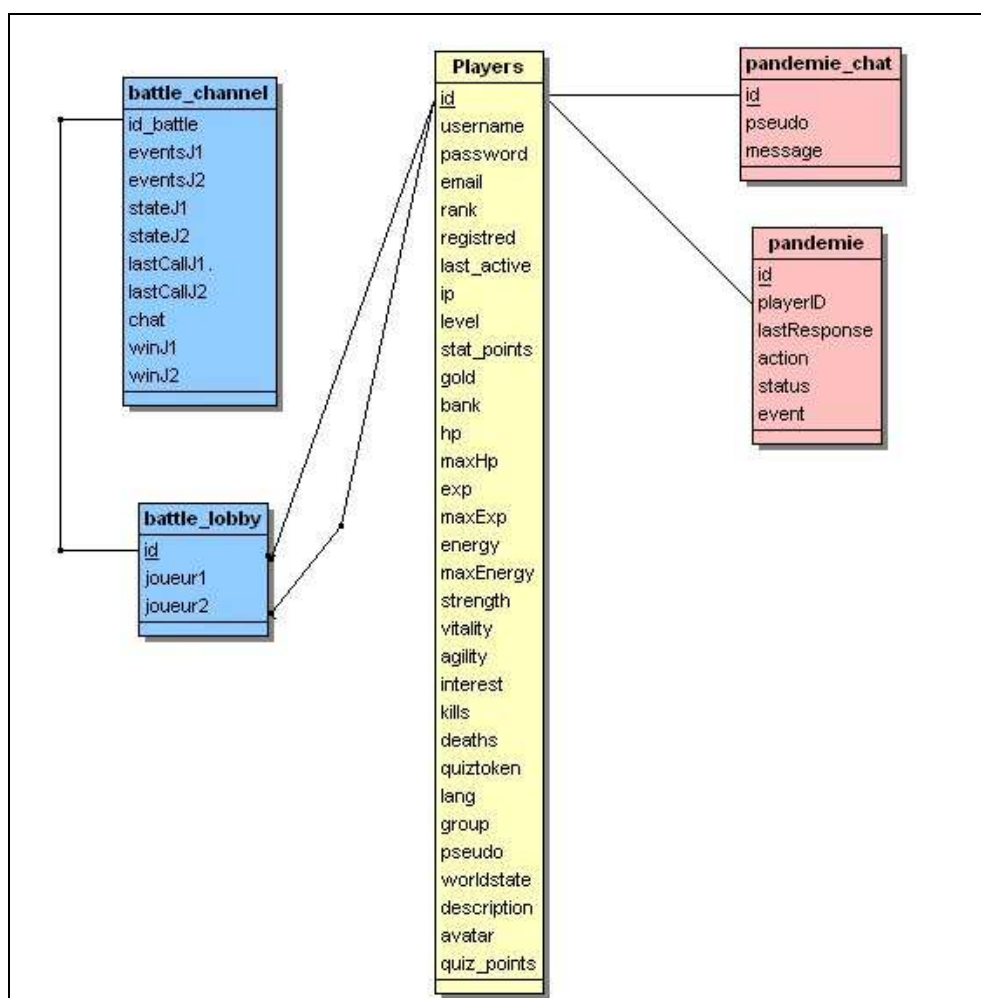


Figure 14 : modèle relationnel des données des jeux compétitif et pandémie

En bleu figurent les tables ajoutées pour le mode de jeu compétitif et en rouge les tables ajoutées pour le mode de jeu pandémie. Afin de ne pas entrer dans une

description trop complète de l'ensemble des attributs de la base de données, je ne vais expliquer que ceux des tables rajoutées.

Pour le jeu compétitif, la table `battle_lobby` matérialise la salle d'attente des joueurs. Elle contient les deux ID des joueurs connectés et un identifiant unique qui va servir pour la clé étrangère `id_battle` dans la table `battle_channel`. Le reste des attributs de cette table sera expliqué plus tard lors de la conception.

Concernant le mode de jeu pandémie, la table `pandemie` sert à regrouper l'ensemble des joueurs connectés au jeu. **LastResponse** correspond au dernier temps de réponse du joueur, **action** correspond à sa position sur la grille, **status** matérialise le fait que le joueur est actif ou non (dans la partie ou non) et **event** sert à savoir si le jeu est terminé.

La table `pandemie_chat` est là pour le chat du mode de jeu. Elle contient l'ensemble des messages tapés par les joueurs avec leurs pseudos.

4.2 Conception

4.2.1 Ajout automatique d'un mini jeu

Afin de faciliter la mise en ligne de contenu, il a fallu réfléchir à une façon générique d'ajouter des minis jeux. Cette réflexion a nécessité de se placer dans un contexte abstrait et d'imaginer une structure générique suffisamment souple pour permettre l'intégration de jeux plus ou moins complexes.

La page `game.php` s'occupe de faire l'ajout automatique des fichiers d'un mini jeu. Lorsqu'une nouvelle carte est ajoutée dans la boutique du site et que l'utilisateur l'a acheté, l'intitulé de la carte figure dans le menu déroulant de la page `game.php`

Lorsque le formulaire est validé, le nom du dossier du jeu sélectionné est envoyé à la page appelée par le formulaire. Celle-ci s'occupe d'ouvrir le dossier et lit le fichier `listing.php`. En fonction du nombre de fichiers Javascript inscrits dans ce fichier, une boucle va générer les balises nécessaires à leur inclusion dans le code source HTML de la page.

```
if(isset($_POST['map'])) {  
    $arr = array("70" => "space_invaders", // tableau pour la correspondance id -> nom de dossier  
  
               "treant" => "treant",  
  
               "75" => "space_invaders_2",  
  
               "76" => "space_invaders_collab");  
  
    include('./scripts/'.$arr[$_POST['map']].'/listing.php'); // fichier qui contient la liste des fichiers du mini jeu  
  
    foreach($files as $value) { // on inclue les scripts nécessaires au jeu
```

```
        echo '<script type="text/javascript"
src="./scripts/.$arr[$_POST['map']].'.$value.'"></script>';
    }
}
```

Nous allons maintenant aborder la création des minis jeux qui m'ont été demandés.

4.2.2 Mini jeu n°1 : Castle Defender

4.2.2.1 Initialisation

Afin d'initialiser notre fond de carte et de l'intégrer au jeu, il faut placer en début de main.js une variable globale **MAP** qui contient l'url de l'image utilisée. Il faut également définir deux autres constantes qui correspondent à la largeur et la hauteur de la carte en pixels : **MAP_WIDTH** et **MAP_HEIGHT**.

L'appel de la fonction **initAnimations** dans la fonction principale permet de créer l'animation de la carte (une image unique non animée) ainsi que celles des autres éléments du jeu (monstres, joueur, objets, etc).

A ce stade notre joueur est créé et notre fond de carte est visible à l'écran. Pour délimiter les zones jouables, il est possible de créer des blocs qui vont simuler les collisions. Lorsque le joueur ou un monstre entrera en collision avec ce bloc, il ne pourra pas le traverser. La création de ce bloc se fait en spécifiant 4 paramètres : l'abscisse et l'ordonnée du point de départ du bloc et sa largeur et sa hauteur. Ce bloc se traduit par la création d'un div dans le DOM de la page avec un attribut **sprite_type** initialisé à la valeur **masse** (ceci pour pouvoir identifier quel type de sprite est ciblé : monstre, joueur, masse, etc.)

```
// Bordures de la carte
addBlock(190,180,1220,5); // haut
addBlock(1380,160,5,1310); // droite
addBlock(230,160,5,1310); // gauche
addBlock(190,1440,1220,5); // bas
```

La détection des collisions fonctionne suivant ce principe : lorsqu'une entité (joueur, monstres, etc.) se déplace, une prévision de ses futures coordonnées est réalisée en fonction de la direction qu'elle emprunte. Si ces futures coordonnées vont chevaucher une autre entité de type masse, alors le déplacement ne se produit pas et on garde les anciennes coordonnées.

Une note concernant les positionnements en pixels : tous les positionnements en pixels sont fait par rapport au coin supérieur gauche de l'image de la carte : cela correspond au point de coordonnées (0, 0).



Figure 15 : une masse (en bleu) : élément de collision

4.2.2.2 Les monstres

Pour plus de clarté dans le code, toutes les fonctions concernant les monstres sont placées dans un fichier `mob.js`.

Sur le *Castle Defender*, les monstres ont un comportement particulier : lorsqu'ils sont attaqués ils ne réagissent pas à l'attaque et continuent quoi qu'il arrive de se déplacer vers leur cible, le château.

Ce n'est pas le comportement par défaut qui a été défini dans le moteur du jeu c'est pourquoi il a fallu redéfinir la fonction de comportement **act** en passant par l'attribut **prototype** en Javascript (le comportement par défaut est que les monstres se déplacent de façon complètement aléatoire).

Celle-ci appelle la fonction **move** des monstres qui sert aux déplacements de ces derniers et qui elle aussi a été redéfinie, nous verrons plus tard pourquoi.

Afin de diriger les monstres automatiquement vers le château, il faut calculer la direction que doivent adopter les monstres en fonction de leur position. Pour cela, la fonction **calcAngle** intervient et permet lorsqu'on lui fournit l'abscisse et l'ordonnée des deux points d'origine et d'arrivée, de calculer l'angle entre l'axe vertical, le point de départ et celui d'arrivée.

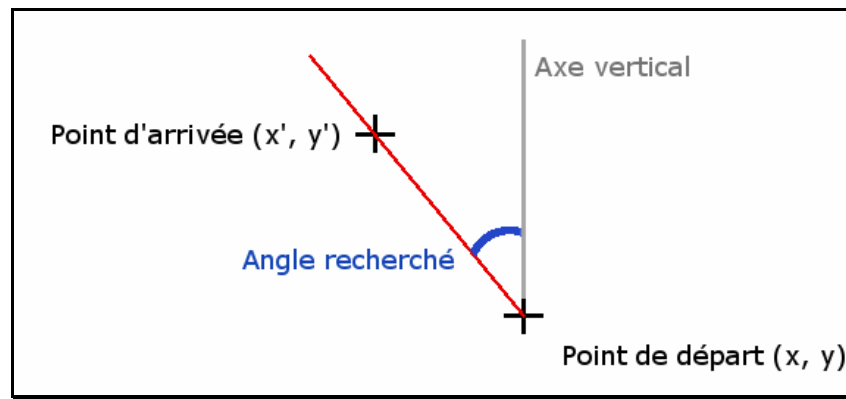


Figure 16 : calcul d'angle entre deux points et l'axe vertical

Une fois l'angle récupéré, il suffit de déterminer la direction à emprunter pour le monstre à l'aide de plusieurs conditions

Pour éviter que le monstre change tout le temps de direction et pour apporter un caractère imprévisible à ce dernier, un nombre aléatoire compris entre 0 et 1 est généré à chaque fois que la fonction **move** est appelée. Si ce chiffre est compris entre 0.99 et 1, alors le monstre change de direction, sinon il garde sa direction actuelle.

1% de chance de changer de direction peut sembler très faible mais ce taux permet d'avoir un comportement naturel.

Enfin, les monstres sont générés aléatoirement dans 4 zones différentes. Ces zones sont définies par la fonction **zone** qui prend 4 paramètres : abscisse et ordonnée du coin supérieur gauche de la zone et abscisse et ordonnée du coin inférieur droit.

```
function initZones()
{
    zones[0] = new zone(250, 250, 350, 350); // haut gauche
    zones[1] = new zone(1150, 250, 1250, 350); // haut droite
    zones[2] = new zone(250, 1150, 350, 1250); // bas gauche
    zones[3] = new zone(1150, 1150, 1250, 1250); // bas droite
}
```

4.2.2.3 Lancement du jeu

Lorsque le jeu est lancé, la fonction **popMob** s'occupe de gérer l'apparition des monstres. Elle vérifie tout d'abord que 60 tours de boucle se sont effectués depuis le dernier appel, ceci pour éviter de créer un nouveau monstre à chaque début de boucle. 60 tours correspondent à 2 secondes de temps de jeu.

Le nombre de tours de boucle a été retenu pour créer un intervalle plutôt que la fonction **setInterval** car cela permet en cas de problème de framerate du jeu de malgré tout ne pas faire apparaître les monstres trop vite.

Dans cette fonction un chiffre entier aléatoire entre 0 et le nombre de zones créées non compris est généré pour déterminer dans quelle zone va être créé le monstre. Une fois celle-ci déterminée, on choisit au hasard une abscisse et une ordonnée dans cet espace pour déterminer les coordonnées d'apparition du monstre.

Afin d'étudier plus en détails les divers processus de fonctionnement des jeux de School Society, intéressons-nous maintenant à un autre mini jeu.

4.2.3 Mini jeu n°2 : Castle Defender compétitif

4.2.3.1 La salle d'attente

Pour permettre la rencontre de deux joueurs lors d'une partie, j'ai créé un système de salle d'attente qui va se charger de la mise en relation des deux joueurs.

Lorsqu'un premier joueur clique sur le bouton pour lancer le jeu compétitif, la page **game_coop.php** va s'occuper de vérifier s'il existe des parties en cours en attente d'un joueur dans la table **battle_lobby**. Si c'est le cas, alors le joueur 1 est mis en relation avec la partie déjà créée, sinon une salle d'attente est créée.



Figure 17 : connexion à une partie existante

Si le joueur 1 est en attente d'un adversaire, un message d'information lui est affiché à l'écran lui spécifiant qu'il est en attente.

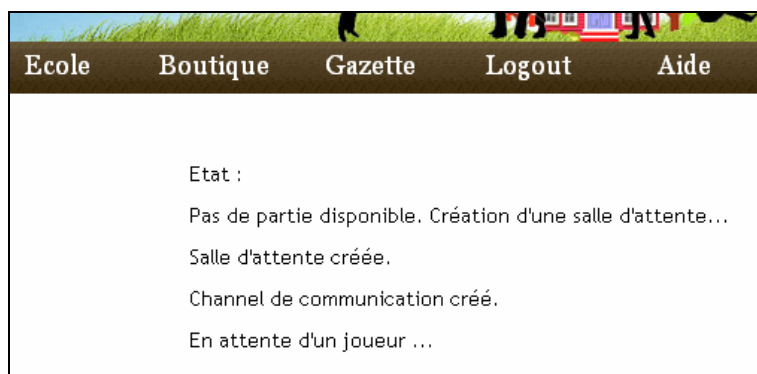


Figure 18 : attente d'un joueur en mode compétitif

Périodiquement, une fonction **AJAX** exécute un script PHP qui va vérifier si un nouveau joueur est arrivé. Cette vérification se fait en consultant le champ joueur2 dans la table battle_lobby. Si ce champ est à zéro, c'est qu'il n'y a toujours pas de joueur, sinon, c'est que quelqu'un vient d'arriver.

A l'arrivée du joueur 2, le joueur 1 va avoir un message l'informant qu'un adversaire vient de se connecter. Le jeu en Javascript va ensuite être chargé chez les deux clients et l'id de la bataille ainsi que le numéro de chaque joueur vont être stockés dans l'URL, ceci pour permettre l'initialisation de variables indispensables dans le Javascript pour le dialogue client / serveur.

4.2.3.2 Le jeu

Une fois que le jeu est chargé dans la page, il faut synchroniser les deux joueurs pour éviter qu'une des deux personnes ne prenne de l'avance dans la partie.

Ainsi, tant que les deux joueurs n'ont pas cliqué sur le message les invitant à se rendre prêt, le jeu reste figé. Chez le client, une fois que celui-ci a cliqué sur le bouton « Se rendre prêt », une fonction AJAX vérifie périodiquement que dans la table battle_channel l'autre joueur est prêt : si le champ stateJ1 vaut true dans la table, cela signifie que le joueur 1 est prêt. Tant que les deux joueurs ne sont pas prêts, il est impossible de lancer le jeu.



Figure 19 : synchronisation des deux joueurs

Une fois les deux joueurs prêts, le jeu se lance chez chaque client.

4.2.3.3 Détection des différents évènements

Lorsque le jeu est lancé, toutes les 2 secondes s'exécute la fonction **callbackAsynchronous**. Elle s'occupe de quatre points :

- vérification que l'adversaire ne nous a pas envoyé un malus
- envoi de notre dernier temps de réponse (le timestamp actuel)
- récupération du dernier temps de réponse de l'adversaire
- récupération de l'état de la partie de l'adversaire (s'il a gagné ou non)

Toutes ces vérifications se font par le biais de requêtes AJAX asynchrones : un script PHP spécifique va se lancer suivant la requête et va exécuter du code SQL sur la base de données. Les informations sont stockées dans les champs de la table **battle_channel**.

Voici un exemple de code qui va récupérer le temps de réponse de l'adversaire :

```
if($_POST['action'] == "getOpponentStatus") {  
    $query_channel = $db->execute("select ".$_POST['attr']." from battle_channel where  
id_battle = ?", array($_POST['id_battle']));  
  
    $row_channel = $query_channel->FetchRow();  
  
    if($row_channel[$_POST['attr']] != "") {  
        if(($row_channel[$_POST['attr']] + 6) < time())  
  
            echo 'false';  
  
        else  
  
            echo 'true';  
    }  
}
```

Si le dernier temps de réponse de l'adversaire est plus vieux de 6 secondes du timestamp actuel au moment où le script s'exécute, alors l'adversaire est considéré comme déconnecté et la variable **LAST_CALL_OPPONENT** dans le Javascript du jeu passe à false ce qui a pour effet de stopper le jeu avec un message d'information.



Figure 20 : fin de partie après déconnexion de l'adversaire

Le principe est le même pour l'état de la partie de l'adversaire (gagné ou non).

Lorsque le joueur 1 lit dans la table si un évènement lui a été envoyé, il récupère et stocke cette valeur dans la variable **EVENT**. Si **EVENT** a une valeur différente de 0, alors la fonction **eventTracker** va déclencher l'évènement correspondant chez le client et exécuter une requête SQL pour remettre à zéro le champ des évènements du joueur 1 dans la base de données.



Figure 21 : réception d'un malus envoyé par l'adversaire

Afin d'éviter de lancer plusieurs fois le même évènement, un système de verrou a été mis en place pour protéger l'accès à la variable **EVENT** dans la fonction **eventTracker**. Lorsque **EVENT** est différent de 0, la variable **mutexLock** passe à **false** ce qui a pour effet d'empêcher la fonction de se ré-exécuter accidentellement avant que **EVENT** soit revenu à 0.

Lorsque EVENT a été remis à zéro, mutexLock repasse à true pour re-permettre l'accès à la fonction.

```
function eventTracker() {  
    if(EVENT != 0) {  
        if(mutexLock) {  
            mutexLock = false;  
            switch (EVENT) {  
                case '1': vitesseJoueur(); print("Votre adversaire vous a  
ralenti !"); break;  
                case '2': vitesseMob(); print("Votre adversaire a accéléré  
les monstres !"); break;  
                default : break;  
            }  
            if(NUMJOUEUR == 1) var attr = "eventsJ1";  
            else attr = "eventsJ2";  
            rpcAS('./rpc/eventHandler.php?action=raz&id_battle='+ID_BATTLE+'&attr='+attr, 'RAZ');  
        }  
        EVENT = 0;  
    }  
}
```

Pour éviter tout problème de ralentissements intempestifs, nous avons décidé de séparer la boucle graphique (registerCallback) de la boucle qui s'occupe de récupérer à intervalle régulier les différents événements. Comme la fonction callbackAsynchronous s'exécute dans une boucle parallèle indépendante et exécute des requêtes AJAX asynchrones, il n'y a aucune interférence entre les deux systèmes. On évite ainsi tout problème de ralentissement également appelé lag.

4.2.4 Mini jeu n°3 : Pandémie

4.2.4.1 L'alternance des tours de jeu

Voici le principe de fonctionnement de l'alternance des tours de jeu dans le mode de jeu pandémie.

Comme il est impossible de créer une application serveur tournant en permanence qui s'occuperait de gérer l'ordre de passage des différents joueurs connectés, il a été décidé que chaque joueur dans la partie ferait à tour de rôle office de serveur.

Une table **pandemie** s'occupe de regrouper l'ensemble des joueurs connectés avec leur ID (chaque joueur est matérialisé par un enregistrement). Un tuple ayant l'ID 0 est par défaut créé dans cette table. Il va servir à connaître l'ID du joueur qui est en train de jouer et un tuple avec l'ID 1 va servir à matérialiser les cases occupées par la maladie.

Lorsqu'un joueur se connecte à la partie, un nouveau tuple est créé dans la table avec son ID. Si ce joueur est le premier arrivé sur la carte, alors par défaut le champ action du tuple d'ID 0 va être initialisé à l'ID du joueur qui vient de se connecter.

Périodiquement, chaque client va lire dans le tuple 0 la valeur du champ action. Si l'ID présent dans ce champ vaut celui du joueur, alors celui-ci sait qu'il peut jouer et prend la main. Il dispose alors de dix secondes pour réaliser ses actions.

A la fin des dix secondes, le joueur se bloque et récupère l'ID du joueur suivant dans la table. Il modifie le champ action du tuple zéro en mettant l'ID du prochain joueur afin de permettre à un autre client de se débloquent. Et le schéma se répète en boucle tout au long de la partie.

Pour déterminer quel est le joueur suivant, il a fallu créer un algorithme qui regarde dans la table **pandemie** la liste des joueurs connectés. La liste ordonnée des ID de tous les joueurs est stockée dans un tableau et l'algorithme récupère la position du joueur courant dans ce tableau. Il décale alors l'indice d'une case vers la droite pour déterminer le joueur suivant. Si le joueur courant est en bout de tableau, alors c'est la case d'indice zéro qui est renvoyée.

A l'aide d'un modulo, il est possible de ne pas utiliser de condition pour trouver le joueur suivant : on compte le nombre de joueurs connectés, on incrémente de 1 l'indice du joueur courant qui matérialise sa case et on fait ensuite un modulo entre cette addition et le nombre de joueurs. Cela renvoi toujours la case se situant après le joueur courant même si celui-ci est en bout de tableau.

Pour ne pas ralentir le temps de jeu de chaque joueur, un système de temps de réponse a été mis en place comme pour le PvE compétitif. Chaque client envoi périodiquement son dernier temps de réponse et si un joueur s'aperçoit que le joueur suivant n'a pas répondu depuis trop longtemps, alors celui n'est plus considéré comme un joueur actif : le champ status de la table passe alors de la valeur **active** à **inactive**.

4.2.4.2 Les déplacements des joueurs et évènements

Les déplacements sont différents des modes de jeu PvE vus auparavant. En effet, une grille est matérialisée sur la carte de façon semi-transparente.



Figure 22 : représentation du terrain de jeu

A chaque tour, chaque joueur peut se déplacer d'une case tout autour de lui (les déplacements en diagonale ne sont pas autorisés) à l'aide des 4 boutons qui lui sont affichés à l'écran.

Lorsque le joueur clique par exemple sur le bouton du haut, la fonction **moveTo** est appelée. En lui passant en paramètre la direction du déplacement (0, 1, 2 ou 3 pour respectivement droite, haut, gauche, bas), celle-ci va vérifier si tout d'abord le déplacement est possible : est-ce que le joueur souhaite aller sur une case dont l'indice en abscisse et en ordonnée est compris entre 0 et 23 (car la grille de jeu est un tableau de 24 cases par 24 cases) ? Lorsque c'est le cas, on décrémente ou incrémente deux variables X ou Y suivant l'axe de déplacement : ces deux indices servent à se souvenir de la position du joueur.

Si oui alors une boucle s'exécutant en parallèle est créée à l'aide de la fonction **setInterval** : toutes les 10 millisecondes, la fonction **hasMoved** va être appelée et va vérifier si le joueur a atteint la position finale ou non. Tant que le joueur n'est pas à sa place, alors celui-ci se déplace grâce à la fonction **move** ; dès qu'il est placé, on stoppe son mouvement.

Sur ce point, une astuce a été utilisée. En effet, lorsque le joueur se déplace comme dit précédemment, ce n'est en réalité pas lui qui bouge mais tous les autres éléments. La division HTML du joueur elle reste fixe et garde des valeurs constantes pour les attributs **top** et **left** du CSS.

Pour donc faciliter le calcul de la position du joueur, deux variables **POS_X** et **POS_Y** simulent la variation de position qu'aurait dû avoir le joueur si celui-ci n'était pas bloqué au centre de l'écran : **POS_Y** correspond donc à la distance entre le joueur et le bord haut de la grille et **POS_X** la distance entre le joueur et le bord gauche.

Ainsi, lors d'un déplacement horizontal par exemple, la fonction `hasMoved` vérifie si l'indice de la case en abscisse multiplié par la largeur de la case où doit se trouver le joueur correspond à la distance `POS_X` entre le joueur et le bord gauche de la grille. Il en est de même pour les déplacements verticaux.

4.2.4.3 L'expansion de la menace

Une fois que le dernier joueur du tour a effectué son déplacement, l'évolution de la maladie est calculée.

Pour cela, la fonction **nouveauTourMaladie** dans le script PHP, exécutée par la fonction AJAX périodique (la même que dans le mode de jeu compétitif : `callbackAsynchronous`), récupère la position des joueurs et des cases malades et initialise les différentes cases d'un tableau à deux dimensions avec la valeur 1 lorsqu'il s'agit d'une case occupée par un joueur et la valeur 2 lorsqu'il s'agit d'une case occupée par la maladie.

Afin de calculer la nouvelle case de maladie, il faut déterminer l'intervalle dans lequel va se situer cette case. Pour cela, on stocke dans des variables différentes le plus petit indice de case occupée par la maladie en abscisse et en ordonnée et de même pour les plus grand que l'on augmente de 1 (pour que l'aire s'agrandisse).

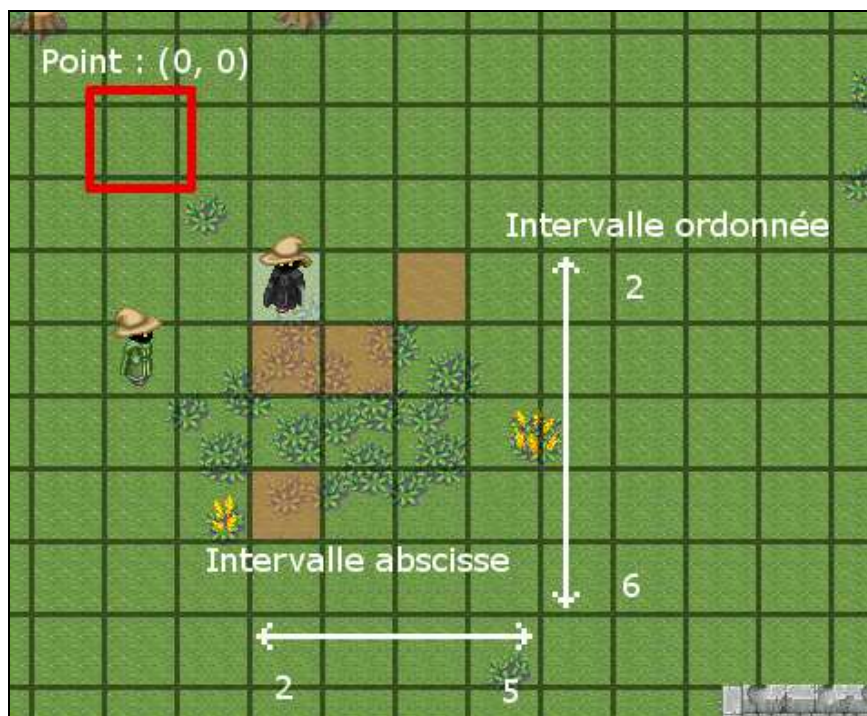


Figure 23: calcul de la nouvelle case malade

Dans ces deux intervalles d'indices sont générés deux coordonnées qui vont correspondre à la position de la future case malade. Si ces coordonnées ne correspondent pas à une case déjà occupée, alors on crée la nouvelle case.

Ensuite intervient la vérification de la position des joueurs pour la suppression des cases détruites.

Chaque case contaminée est analysée à l'aide de la fonction voisinage qui vérifie si cette case à dans son voisinage au moins deux joueurs. Si c'est le cas, alors la case est supprimée.

Une fois toutes les cases analysées, on récupère sous forme de chaîne de caractères la position des cases malades et on met à jour le champ action du tuple ayant l'ID -1.

La position des cases infectées sur la grille se met à jour automatiquement ensuite à l'aide de la fonction **refreshGrid** dans le Javascript.

```
refreshGrid() {
    getMaladie();
    $("#grid_tiles").html("");
    for(var i=0; i<40; i++) {
        for(var j=0; j<40; j++) {
            if(X == i && Y == j)
                // ajout du fond de la case du joueur
                $("#grid_tiles").addSprite("tile"+i+'-'+j, {animation:
grid_tile_white, width: 40, height: 40, posx:i*40+360, posy:j*40+280, sprite_type:"grid"});
            if(GRID_MALADIE[i][j] == true)
                // ajout de la case malade
                $("#grid_tiles").addSprite("tile"+i+'-'+j, {animation:
grid_tile_red, width: 40, height: 40, posx:i*40+360, posy:j*40+280, sprite_type:"grid"});
        }
    }
}
```

Ensuite l'algorithme vérifie que les cases à défendre ne sont pas atteintes par la maladie. Si aucune case n'a été touchée alors on vérifie qu'il reste au moins deux cases malades sur la grille. Si ce n'est pas le cas, on informe l'ensemble des joueurs qu'ils ont gagné en modifiant le champ event de la table pandémie en y affectant la valeur **pandemieWin**. Dans le cas où les joueurs auraient perdu, la valeur **gameOver** est affectée au champ.

Maintenant que la partie de conception est terminée, penchons nous sur les résultats.

4.3 Résultats

Pour de faciliter le processus de développement d'un mini jeu, un exemple contenant des explications pas à pas m'a été demandé. Celui-ci est visible en annexe, sous la forme d'un tutorial. Voyons maintenant les minis jeux.

4.3.1 Castle Defender :

Au début du jeu, le joueur apparaît dans son château et à l'aide des touches Z, Q, S et D peut se déplacer pour aller respectivement en haut, à gauche, en bas et à droite. A l'aide de la touche F il peut tirer des boules de feu qui infligeront des dégâts aux monstres si celles-ci en touche un.

Chaque type de monstre (dragon, cerf, squelette, etc) possède un nombre de points de vie en fonction de sa taille. Plus le monstre est gros, plus celui-ci a de points de vie.

Périodiquement des monstres apparaissent dans les quatre coins de la carte pour obliger le joueur à se déplacer et pour accentuer la difficulté du jeu. Chaque monstre se dirige vers le château et dès qu'un d'entre eux est en contact avec celui-ci, la structure perd 5 points de vie.

Afin de visualiser rapidement les différents éléments du jeu, le joueur possède sur son ATH un radar ainsi que deux barres, une verte et une bleue, symbolisant respectivement la vie du joueur et les points de défense du château. Un compteur du nombre de monstres restant à tuer dans la vague courante est également affiché.

Sur le radar, le château est matérialisé par un rectangle vert, les monstres par des carrés rouges et le joueur par un carré bleu.



Figure 24 : ATH de Castle Defender

Lorsque le joueur a tué tous les monstres d'une vague, un popup s'affiche à l'écran pour l'informer de l'arrivée de la nouvelle vague. Aussitôt le compteur de monstres est mis à jour à l'écran.



Figure 25 : fin de vague dans Castle Defender

Si le joueur parvient à traverser toutes les vagues, alors un message d'information apparaît à l'écran et l'informe que celui-ci vient de gagner la partie. Un objet est alors tiré au hasard parmi l'ensemble des items figurant dans la base de données et est donné au joueur. Cet objet pourra servir pour des quêtes annexes plus tard dans l'aventure.

4.3.2 Castle Defender compétitif :

Comme dit dans le cahier des charges, le principe est le même que le Castle Defender simple. Cependant, en rajoutant un système de bonus/malus, il est possible de créer un aspect compétitif : à la mort d'un monstre, celui-ci peut faire apparaître une étoile qui provoquera un évènement au hasard chez l'adversaire lorsque le joueur passera dessus.

Ce mode de jeu se joue à deux et le premier joueur qui parvient à survivre à la totalité des vagues de monstres gagne.

Lorsqu'un joueur marche sur une étoile qu'a lâché un monstre, deux types d'évènements peuvent se produire chez son adversaire. Soit celui-ci est ralenti dans ses déplacements, soit les monstres se déplacent beaucoup plus vite.



Figure 26 : malus envoyé par l'adversaire

Pour communiquer, les joueurs ont à leur disposition un chat avec rafraîchissement automatique dans lequel ils peuvent écrire du texte.



Figure 27 : chat du mode PvE

4.3.3 Pandémie :

Les jeux en tour par tour ont un fonctionnement particulier : tant que l'ensemble des joueurs n'a pas réalisé une action, le jeu est en pause et tous les joueurs sont en attente du tour suivant. Ce genre de jeu ne propose généralement qu'une seule action par tour pour chaque joueur.

Dans School Society, ce type de gameplay a été retenu pour le mode de jeu coopératif dont voici l'histoire : une maladie s'étend sur la carte et plusieurs joueurs doivent coopérer pour stopper la progression du fléau et le réduire à néant avant que celui-ci n'atteigne une source d'eau vitale pour la survie du monde.

Pour cela, les joueurs doivent se positionner de façon stratégique autour des cases malades. En fonction du nombre de joueurs présents autour de la menace, celle-ci va être détruite ou non.

Chaque joueur occupe une case. Toutes les cases adjacentes de celle-ci font partie de son voisinage. Lorsqu'une case fait partie du voisinage d'un joueur, celle-ci prend une valeur : s'il n'y a qu'un joueur elle vaut 1, s'il y en a deux, elle vaut deux, etc.

0	0	0	0	0	0	0
0	1	1	1	0	0	0
0	1	J1	1	0	0	0
0	1	1	2	1	1	0
0	0	0	1	J2	1	0
0	0	0	1	1	1	0
0	0	0	0	0	0	0

Figure 28 : système de voisinage d'une case

Si les joueurs décident d'encadrer une case malade, des points vont ainsi se cumuler. Si la somme des valeurs attribuées à une case malade dépasse le nombre de points de défense d'une zone malade, alors la zone est détruite et redevient saine.

A la fin de chaque tour complet de jeu, autrement dit quand tous les joueurs ont effectués leurs mouvements, la menace évolue et s'étend si toutes les cases n'ont pas été détruites. La direction de progression de la menace se fait de façon aléatoire mais reste globalement proche de la zone déjà contaminée pour avoir une progression de l'intérieur vers l'extérieur et ne pas déborder trop rapidement les joueurs.

Les cases malades sont identifiées par un fond rouge, les cases à protéger par un fond vert clair et la case du joueur par un fond blanc.



Figure 29 : représentation des cases dans le mode jeu Pandémie

Lorsque toutes les cases rouges ont disparu, la partie se termine et une nouvelle recommence.

Il est tout a fait possible pour un joueur de quitter la session au cours du jeu et de revenir plus tard. Sa position est sauvegardée et quand il reviendra, il reprendra la place qu'il possédait avant de partir, du moins si la partie ne s'est pas terminée.

Même si l'ensemble des joueurs quitte la session, la position actuelle de tous les éléments du jeu (maladie et joueurs) sera sauvegardée.

Pour faciliter la coordination des joueurs, un chat est présent à coté de l'écran de jeu. Ils peuvent y taper des messages qui seront transmis à l'ensemble des autres joueurs connectés.

La partie concernant les jeux de School Society étant maintenant terminée, penchons nous sur le site web existant.

5 – Back office

L'ensemble des paragraphes constituant cette partie se rapporte à l'interface web mise à disposition de l'utilisateur.

5.1 Analyse du besoin

Au cours du stage, je me suis aperçu qu'un problème ralentissait l'ajout de contenu au site web. Il n'y avait pas de back-office pour ajouter rapidement et facilement de nouveaux éléments : lorsque l'administrateur voulait par exemple ajouter un quiz, il devait faire ça à la main dans la base de données à l'aide de **PhpMyAdmin**.

J'ai donc décidé, après discussion avec mon tuteur, de développer un back-office qui permettrait de gérer l'ensemble du contenu présent sur le site web (membres, gazette, quiz) et d'étudier les différents flux de données à l'aide de statistiques. Il m'a donc fallu analyser l'existant pour savoir sur quelles tables j'allais devoir travailler et quels seraient les cas d'utilisations possibles afin de créer les fonctionnalités nécessaires.

5.1.1 Diagramme de cas d'utilisations :

Voici le diagramme de cas d'utilisations de programmation réalisé pour prévoir les fonctionnalités à intégrer :

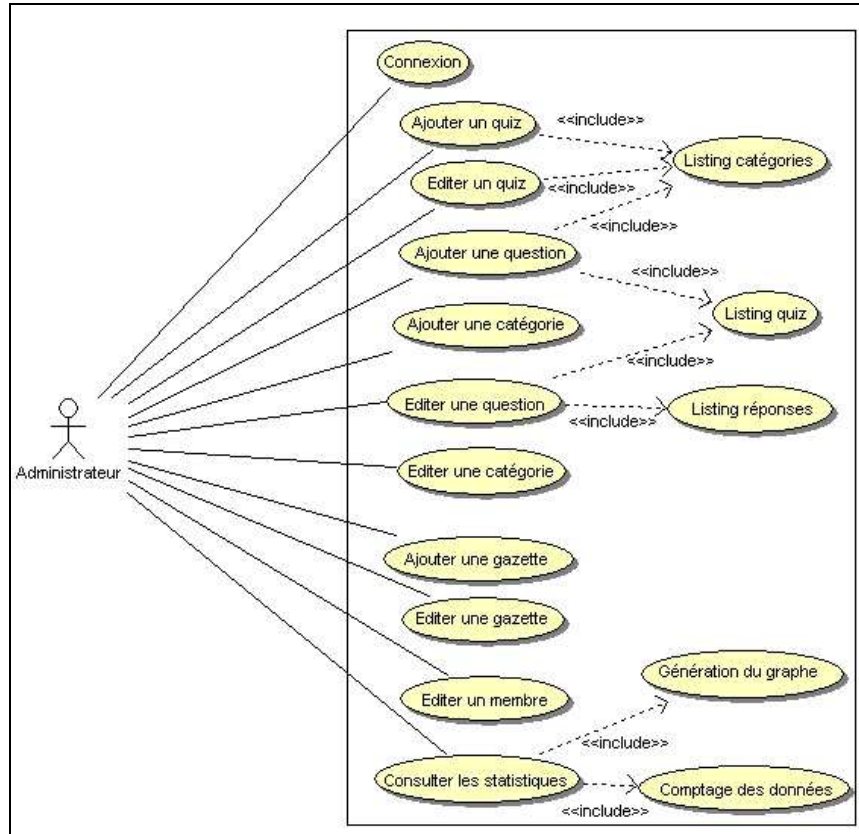


Figure 30 : diagramme de cas d'utilisations de programmation

Pour ne pas rendre illisible le schéma, je n'ai pas matérialisé l'ensemble des liens de type « extend » entre le cas « connexion » et l'ensemble des autres cas mais il va de soit que l'administrateur ne pourra utiliser les fonctionnalités que si celui-ci est connecté.

5.1.2 Modèle relationnel des données :

Voici les différentes tables qui ont été utilisées pour la conception du back-office. Encore une fois, ce diagramme ne matérialise pas l'ensemble des tables de la base de données finale. Le schéma complet est disponible en annexe.

Les attributs les plus importants sont expliqués plus bas dans le rapport lors de la phase de conception du back-office pour ne pas surcharger la lecture.

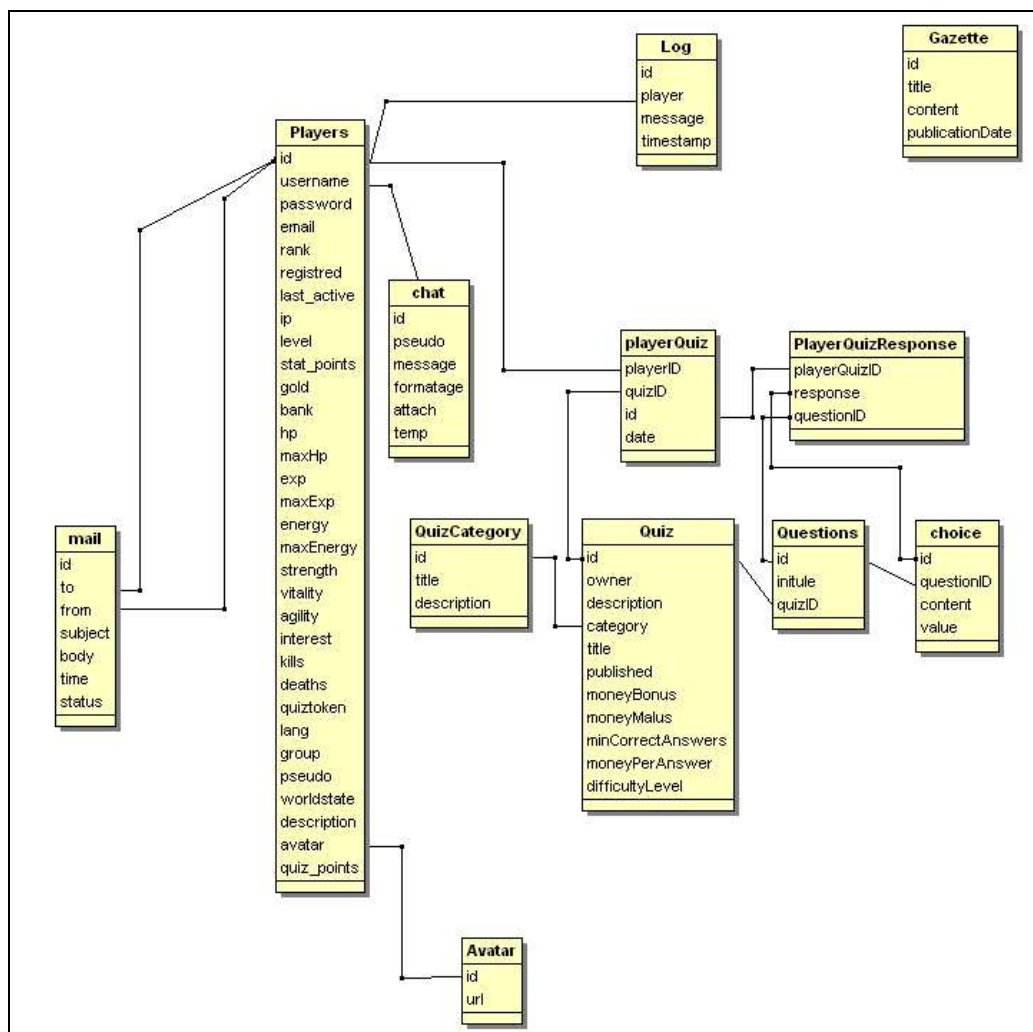


Figure 31 : modèle relationnel des données du back-office

5.1.3 Diagramme de séquences :

Plusieurs diagrammes ont été réalisés pour prévoir le comportement du back-office en fonction des cas d'utilisation. Seuls deux cas nominaux sont présentés ici

pour ne pas surcharger la lecture du rapport. Les autres diagrammes sont disponibles en annexe.

Le premier correspond à l'ajout d'une catégorie de quiz. Lorsque l'administrateur demande le formulaire d'ajout, le site lui envoie un formulaire vide. L'administrateur est sensé alors renvoyer le formulaire avec l'ensemble des champs remplis c'est pourquoi le back-office va vérifier que ces derniers sont correctement remplis. Si c'est le cas, alors les données sont insérées dans la table QuizCategory et un message de confirmation est affiché à l'écran de l'administrateur.

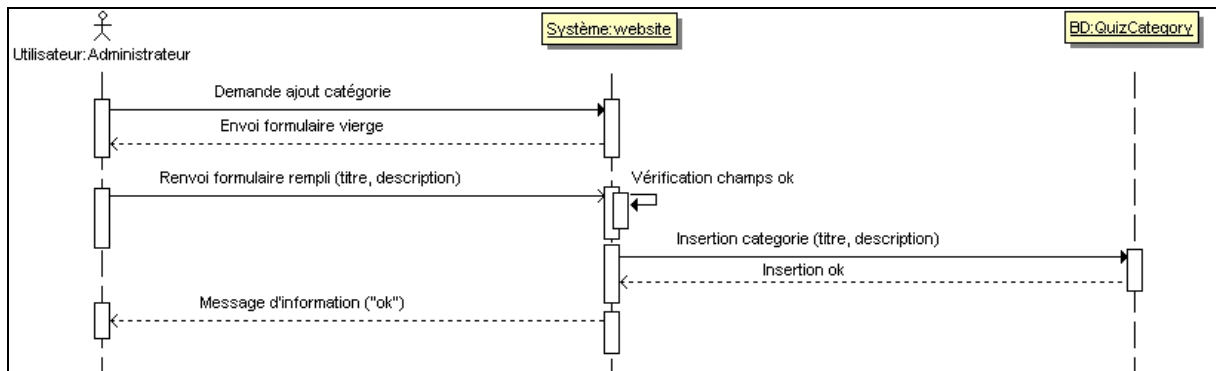


Figure 32 : diagramme de séquence d'ajout d'une catégorie

Le deuxième correspond à la visualisation des flux de données dans la section statistiques. En sélectionnant les données à comparer et la période à étudier, le système récupère les données dans la base de données et génère un graphe pour l'utilisateur.

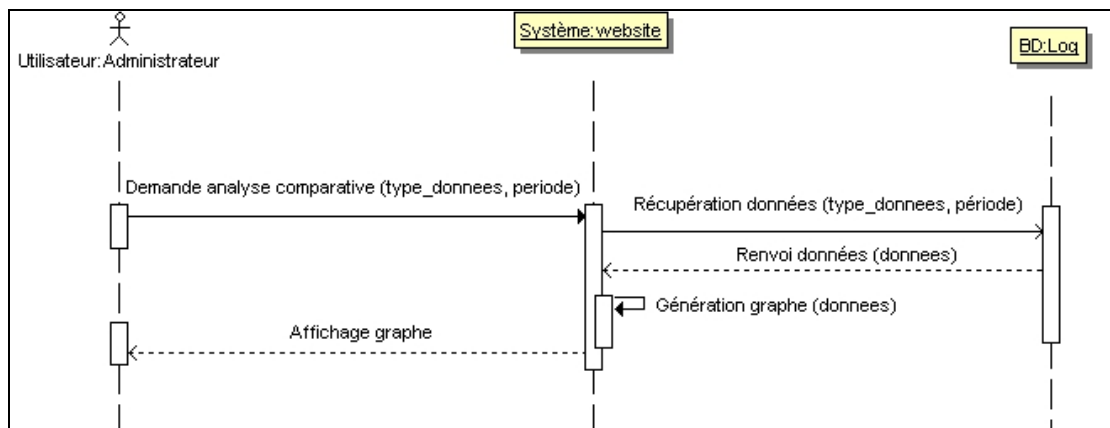


Figure 33 : diagramme de séquence de consultation des statistiques

5.2 Conception

5.2.1 Gestion des quiz

Lorsque l'administrateur ajoute une catégorie dans le panneau d'administration, cela se traduit par l'exécution d'une requête SQL qui va créer un nouveau tuple dans la table QuizCategory.

Comme l'ajout d'un quiz ou d'une question est assez proche de la création d'une catégorie et que l'édition d'un élément déjà existant est plus intéressante à étudier, je ne vais pas m'attarder sur l'explication technique de cette partie.

Quand l'administrateur arrive sur la page d'accueil d'édition des quiz, un affichage de l'ensemble des catégories ainsi que des quiz y étant rattachés est effectué.

On récupère d'abord l'ensemble des catégories existantes et pour chaque catégorie on va récupérer l'ensemble des quiz correspondant. De même pour les questions.

A chaque lien affiché est associé l'ID de l'élément pour permettre son édition. Ainsi, lorsque l'utilisateur clique sur le bouton d'édition de l'élément sélectionné, il est redirigé sur une page et dans l'url figure l'ID correspondant.

Cet ID permet d'exécuter une requête SQL qui va récupérer dans la base de données les différentes informations de l'élément édité pour initialiser les champs du formulaire aux valeurs déjà saisies.

Lors de la validation du formulaire, une requête SQL de type UPDATE est exécutée pour mettre à jour les informations dans la base de données.

5.2.2 Gestion des membres

L'édition d'un membre est similaire à celle d'un quiz. Lorsque l'administrateur clique sur le membre à modifier, l'ID de celui-ci est passé dans l'URL pour initialiser les différents champs du formulaire. A la validation du formulaire, une requête de type UPDATE est exécutée.

5.2.3 Gestion de la gazette

Lorsque le formulaire d'ajout d'une gazette est validé, une requête SQL est exécutée pour insérer un nouveau tuple dans la table gazette avec les différentes valeurs saisies.

L'édition d'une gazette déjà rédigée suit le même principe que les deux autres parties ci-dessus.

```
mysql_query('UPDATE gazette SET content = "'.addslashes($_POST['content'])."', title = "($_POST['title'])." WHERE id = "'.addslashes($_POST['id_gazette'])."')
```

5.2.4 Statistiques

Les graphiques sont générés à l'aide d'un framework PHP nommé **Artichow**. Lorsqu'on lui fournit sous forme de tableau les différentes données numériques à mettre en page, il génère automatiquement le graphe correspondant (histogramme, camembert ou courbe suivant le type de graphe choisi).

Cependant, cette boîte à outils ne s'occupe pas de récupérer les informations directement. C'est au codeur de s'atteler à cette tâche.

Ainsi pour chaque donnée à analyser, plusieurs fonctions ont été créées. Elles permettent suivant la date spécifiée de récupérer dans la base de données les différents tuples correspondant aux critères de sélection.

Tous les enregistrements sont ensuite comptés et stockés dans un tableau qui sera fourni au framework plus tard pour la génération du graphe.

La majorité des événements détectés (lecture de l'aide, modification du profil, etc.) sont stockés dans la table log à l'aide de la fonction **track**. Cette fonction permet lors de son appel de créer un nouveau tuple dans la table log qui contiendra l'ID du joueur suivi, la page qu'il a visualisé et la date de visualisation.

```
track($player->id, "game/pve",$db );
```

Voici la fonction permettant de récupérer les informations dans la table log dans un intervalle de jours précis :

```
function compareLog($message, $moisDebut, $jourDebut, $heureDebut, $moisFin, $jourFin, $heureFin) {  
  
    $query = mysql_query('SELECT COUNT(*) AS nbr FROM log WHERE message =  
    ".$message." AND UNIX_TIMESTAMP(timestamp) >= ".mktime($heureDebut, 00, 00,  
    $moisDebut, $jourDebut, 2010)." AND UNIX_TIMESTAMP(timestamp) < ".mktime($heureFin,  
    00, 00, $moisFin, $jourFin, 2010)."' or die(mysql_error());  
  
    if(mysql_num_rows($query) > 0)  
  
        $data = mysql_fetch_array($query);  
  
        return (int)$data['nbr'];  
  
}
```

Il n'y a pas de limite théorique concernant le nombre d'informations à comparer sur le graphe. Au niveau technique, il a été constaté que le graphe ne se génère plus au-delà de 5 informations en même temps.

5.3 Résultats

Le back-office propose une interface permettant d'ajouter facilement des quiz. Chaque quiz doit être attaché à une catégorie et possède plusieurs questions, chaque question ayant plusieurs réponses.

Si l'utilisateur souhaite ajouter un nouveau quiz dans une nouvelle catégorie, il doit d'abord au préalable créer la nouvelle catégorie. Cette action se fait en se rendant sur la rubrique « Ajouter une nouvelle catégorie ».

L'ajout d'un quiz se fait ensuite par la rubrique « Créer un nouveau quiz ». L'utilisateur doit sélectionner la catégorie auquel celui ci va être rattaché via le menu déroulant puis saisir son intitulé.

Enfin, la création de nouvelles questions se fait par le menu « Créer de nouvelles question pour un quiz ». Une formulaire contenant la liste des quiz disponibles ainsi que l'intitulé de la question et 4 réponses possibles est alors affiché à l'écran.

Chaque information est entièrement éditable via le panneau d'accueil de gestion des quiz où un affichage par catégorie des différents quiz, catégories et questions est réalisé.



Figure 34 : section quiz du back-office

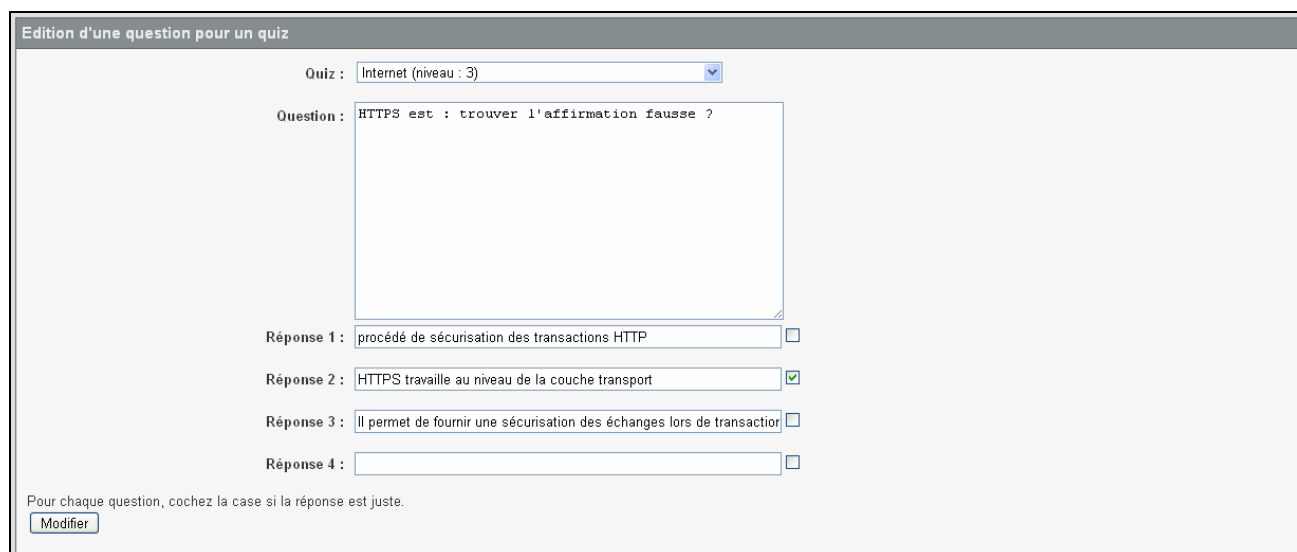
The screenshot shows the 'Edition d'une question pour un quiz' form. At the top, there is a dropdown menu for 'Quiz' set to 'Internet (niveau : 3)'. Below it is a text input field for 'Question' containing 'HTTPS est : trouver l'affirmation fausse ?'. There are four 'Réponse' fields, each with a checkbox. The first response is 'procédé de sécurisation des transactions HTTP' with an unchecked checkbox. The second response is 'HTTPS travaille au niveau de la couche transport' with a checked checkbox. The third response is 'Il permet de fournir une sécurisation des échanges lors de transaction' with an unchecked checkbox. The fourth response is empty with an unchecked checkbox. At the bottom, there is a note: 'Pour chaque question, cochez la case si la réponse est juste.' and a 'Modifier' button.

Figure 35 : panneau d'édition d'une question

Concernant la gestion des membres et de la gazette, ces deux fonctionnalités sont proches de la gestion des quiz. En effet, les différents membres et gazettes rédigées sont éditables par le panneau d'administration proposé.

Pour les membres, l'utilisateur a la possibilité de modifier toutes les informations du membre étudié à l'exception de son mot de passe, pour des raisons de confidentialité.

The screenshot shows a web form titled "Modification d'un membre". The form contains the following fields and values:

- Username : abserver.abserver
- Pseudo : abserver
- Description : Voici ma nouvelle description. Yes oué

- Email : gouaich@lirmm.fr
- Level : 5
- Points de stat : 3
- Gold : 589
- Bank : 793
- HP : 170
- MaxHP : 170
- Exp : 2
- MaxExp : 780
- Energy : 8
- MaxExenergy : 10
- Strength : 3
- Vitality : 4
- Agility : 4
- Quiz Token : 47
- Worldstate : 5
- Quizz points : 20

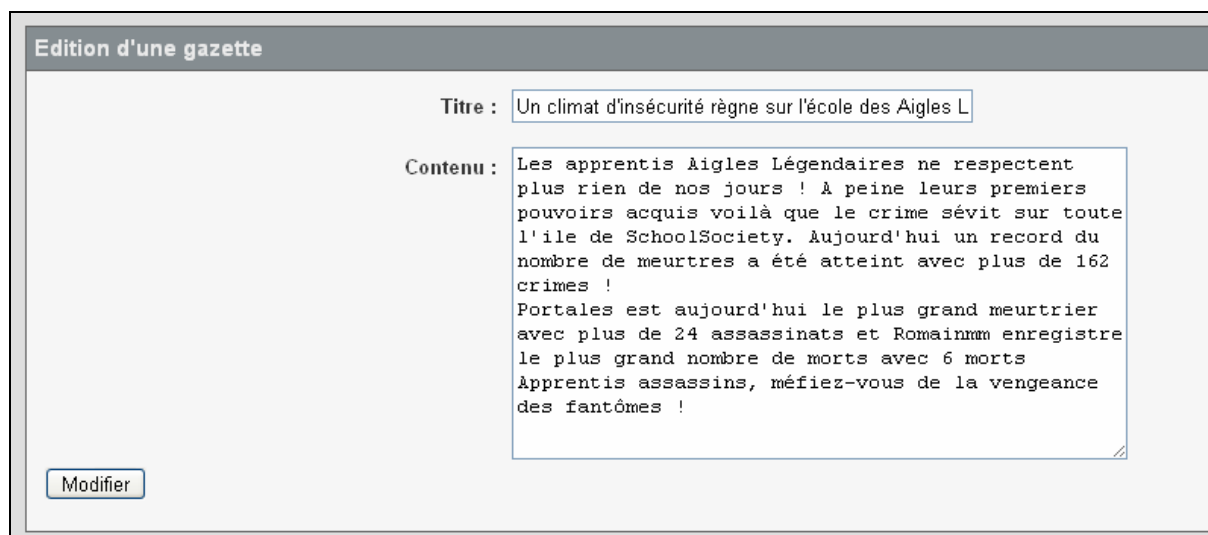
Figure 36 : panneau d'édition d'un membre

Au sujet des gazettes, l'utilisateur peut ajouter ou modifier un nouveau billet. Dans les deux cas, les informations à saisir sont le titre du billet ainsi que sa description.

The screenshot shows the "School Society" website's management interface. The top navigation bar includes "Accueil", "Quiz", "Membres", "Gazette", and "Statistiques". The main content area is divided into two sections:

- Base de données**: A list of gazettes with titles and dates:
 - Nouvelle version du jeu bientôt prête - 2010-03-15
 - Un climat d'insécurité règne sur l'école des Aigles L'égendaires! - 2010-02-04
 - Portales ce h?ro! - 2010-02-03
- Actions**: A button labeled "Ajouter une gazette".

Figure 37 : page de gestion des gazettes



Edition d'une gazette

Titre : Un climat d'insécurité règne sur l'école des Aigles L

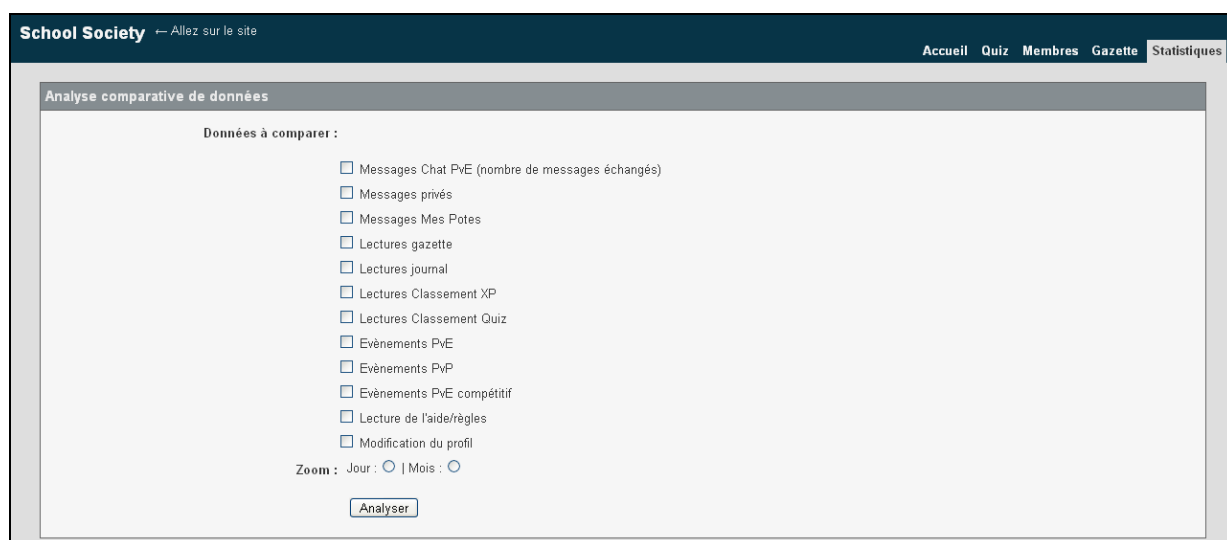
Contenu : Les apprentis Aigles Légendaires ne respectent plus rien de nos jours ! A peine leurs premiers pouvoirs acquis voilà que le crime sévit sur toute l'île de SchoolSociety. Aujourd'hui un record du nombre de meurtres a été atteint avec plus de 162 crimes !
Portales est aujourd'hui le plus grand meurtrier avec plus de 24 assassinats et Romainmm enregistre le plus grand nombre de morts avec 6 morts
Apprentis assassins, méfiez-vous de la vengeance des fantômes !

Modifier

Figure 38 : édition d'une gazette

Afin de pouvoir étudier et interpréter les différents événements se déroulant sur School Society, l'administrateur a à sa disposition une interface lui permettant de consulter et de mettre en page sous forme d'histogrammes ces informations.

Pour réaliser des analyses comparatives, l'administrateur peut s'il le souhaite, afficher plusieurs histogrammes sur la même période. Il lui suffit pour cela de sélectionner les informations qui l'intéressent en cochant les cases affichées à l'écran. A l'aide du champ « zoom », il est possible d'étudier les données au mois ou à la journée suivant la précision recherchée.



School Society — Allez sur le site

Accueil Quiz Membres Gazette Statistiques

Analyse comparative de données

Données à comparer :

- Messages Chat PvE (nombre de messages échangés)
- Messages privés
- Messages Mes Potes
- Lectures gazette
- Lectures journal
- Lectures Classement XP
- Lectures Classement Quiz
- Evènements PvE
- Evènements PvP
- Evènements PvE compétitif
- Lecture de l'aide/règles
- Modification du profil

Zoom : Jour : | Mois :

Analyser

Figure 39 : panneau de création de graphes

Lorsque le formulaire est validé, un histogramme est affiché à l'écran avec la légende correspondante et en titre, le mois ou jour étudié.

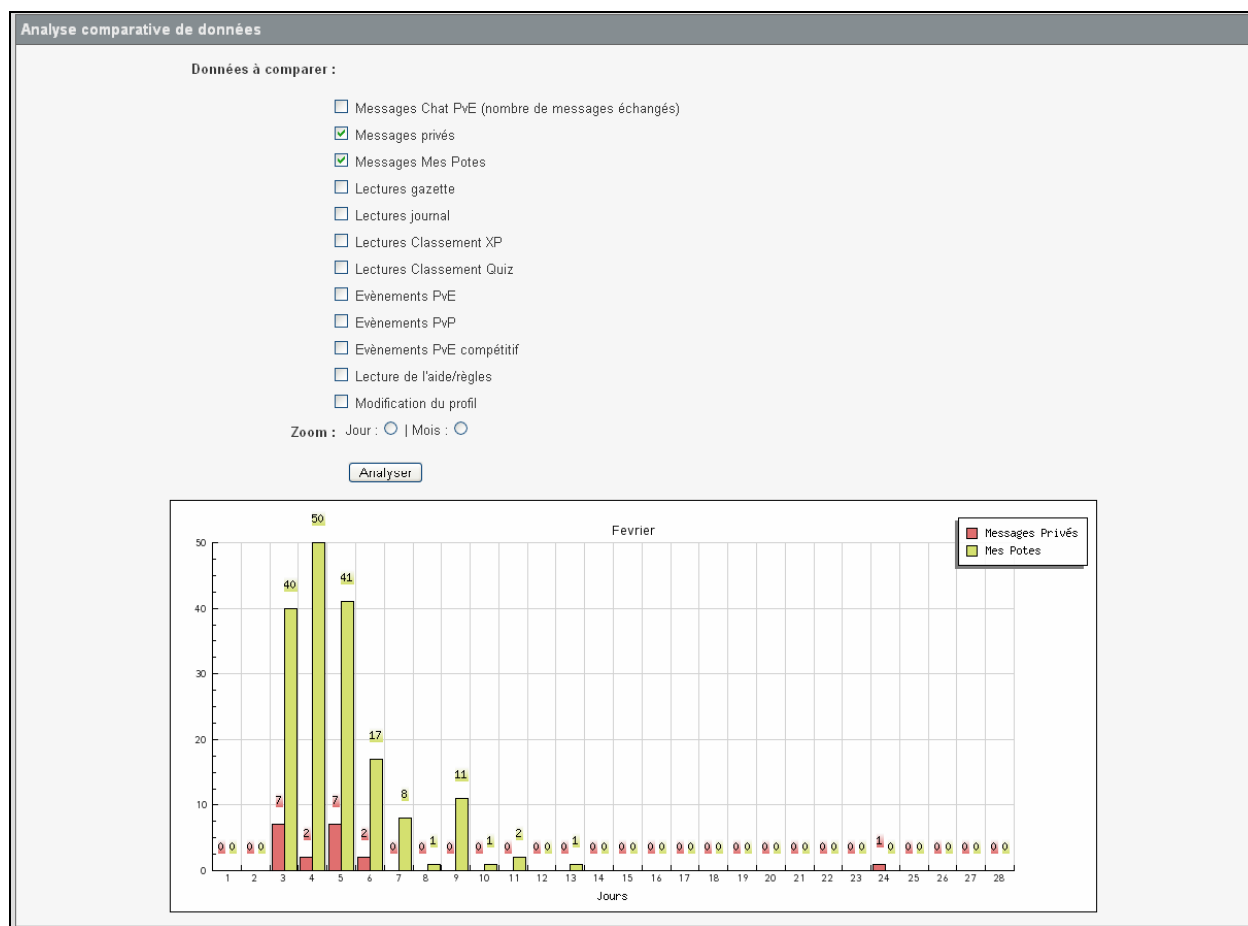


Figure 40 : génération d'un graphe avec deux histogrammes

5.4 Impact du back office

Au cours de l'expérimentation, nous nous sommes rendu compte grâce aux statistiques qu'un élément du jeu était déséquilibré. En effet, les joueurs délaissaient la partie pédagogique de School Society.

Pour remédier à cela, nous avons mis en place un système de points : lorsqu'un utilisateur répond à un quiz celui-ci gagne un nombre de points variable qui alimentera un compteur. A la fin de l'expérimentation, l'utilisateur ayant le plus de points gagnera l'objet.

Lorsqu'un utilisateur répond à un quiz, celui-ci gagne des points qui lui serviront de monnaie dans le jeu d'une part et pour le classement par points quiz d'autre part.

Vos résultats:

Réponses correctes

HTTPS est : trouver l'affirmation fausse ?
La bonne réponse est: HTTPS travaille au niveau de la couche transport
C'était juste !

Quel protocole est d?di? ? la transmission de fichiers sur Internet ?
La bonne réponse est: FTP
C'était juste !

Qu'est-ce que le SMTP ?
La bonne réponse est: Un protocole de transmission de courrier ?lectronique
C'était juste !

Tu as obtenu un Bonus!
Vous avez fait le test 3 fois. Votre gain est de $130/4=33$ et de 2 points quizz.
[Retour](#)

Figure 41 : fin d'un quiz

Plus un utilisateur répond souvent à un quiz, moins son nombre de points quiz gagnés est important, jusqu'à tendre vers 1.

Ce système a été choisi pour empêcher les joueurs de ne répondre qu'aux mêmes quiz à chaque fois et pour les pousser à répondre à des quiz d'un niveau plus élevé.

Le calcul du nombre de points gagnés se fait sur la base du nombre de réponses justes à un quiz.

Le nombre de réponses justes est divisé par le nombre de fois que l'utilisateur a répondu au quiz. Ce chiffre est arrondi à l'entier inférieur et on ajoute 1 pour obtenir un seuil minimum de points.

```
echo "Vous avez fait le test ". $history ." fois. Votre gain est de
".$oldgain."/".($history+1).".".$gain." et de ".floor($numCorrect/$history+1)." points quizz.";

// numCorrect : nombre de réponses justes

// $history : nombre de fois que le joueur a fait le quiz
```

6 – Discussions

Au début du stage a été fixé un cahier des charges précis que je pensais réaliser intégralement.

Lorsque l'on regarde les résultats obtenus, on constate que l'ensemble des fonctionnalités et nouveaux éléments de jeu du site web fonctionnent et semblent répondre au besoin exprimé au début de ce rapport.

Cependant, plusieurs éléments mériteraient quelques heures de développement supplémentaires comme par exemple le mode de jeu pandémie : agrémenter le jeu de quelques éléments de gameplay supplémentaires le rendrait encore plus intéressant à jouer. Rajouter des éléments sur l'ATH serait également intéressant : informer l'ensemble des joueurs du nom du joueur courant.

Il en est de même pour le back-office : au niveau des statistiques, il aurait été intéressant d'apporter des fonctionnalités supplémentaires comme par exemple un affichage à la semaine, des statistiques détaillées par joueur ou encore des valeurs comme la moyenne, la variance, l'écart type, etc.

Enfin, je pense que mettre l'ensemble du site aux normes W3C serait un plus non négligeable : l'accessibilité du site en serait améliorée et le portage de l'application sur différents navigateurs serait plus simple.

7 – Conclusion

Ce stage dans le cadre de ma deuxième année de DUT Informatique est terminé. Il m'a permis de découvrir le monde professionnel tout en améliorant mes connaissances apprises en première et deuxième année et a permis à l'équipe SMILE du LIRMM de développer leur projet School Society.

D'un point de vue technique, l'absence de commentaires dans le code déjà réalisé à un été un vrai challenge. En effet, ces indications laissées par le programmeur ont pour but de faciliter la compréhension des scripts. Sans ces informations, il est beaucoup plus difficile de comprendre les liens entre les différentes fonctions et le mode de fonctionnement de l'application étudiée.

L'analyse du besoin d'un back-office pour ajouter du contenu a été judicieuse et a permis de faire gagner du temps aux administrateurs.

En revanche, comme cet élément n'était pas prévu dans le planning de départ, j'ai dû me limiter dans le temps pour éviter de déborder sur le temps de développement alloué au reste. Sentant que je ne pourrai pas finir les tâches plus importantes qui m'étaient assignées, j'ai préféré fixer une limite technique au back-office pour ainsi être sûr de pouvoir tout réaliser.

Grâce à ce stage, j'ai appris à travailler en autonomie et à me former tout seul. En effet, je ne connaissais pas du tout le langage Javascript ni le framework jQuery.

L'utilisation de l'outil de gestion de projet RedMine a été très efficace, notamment pour faciliter la communication de type informelle. De plus, le fait d'avoir un planning assez strict et peu flexible m'a permis d'apprendre à respecter les délais et à bien évaluer le temps de développement nécessaire pour créer une application.

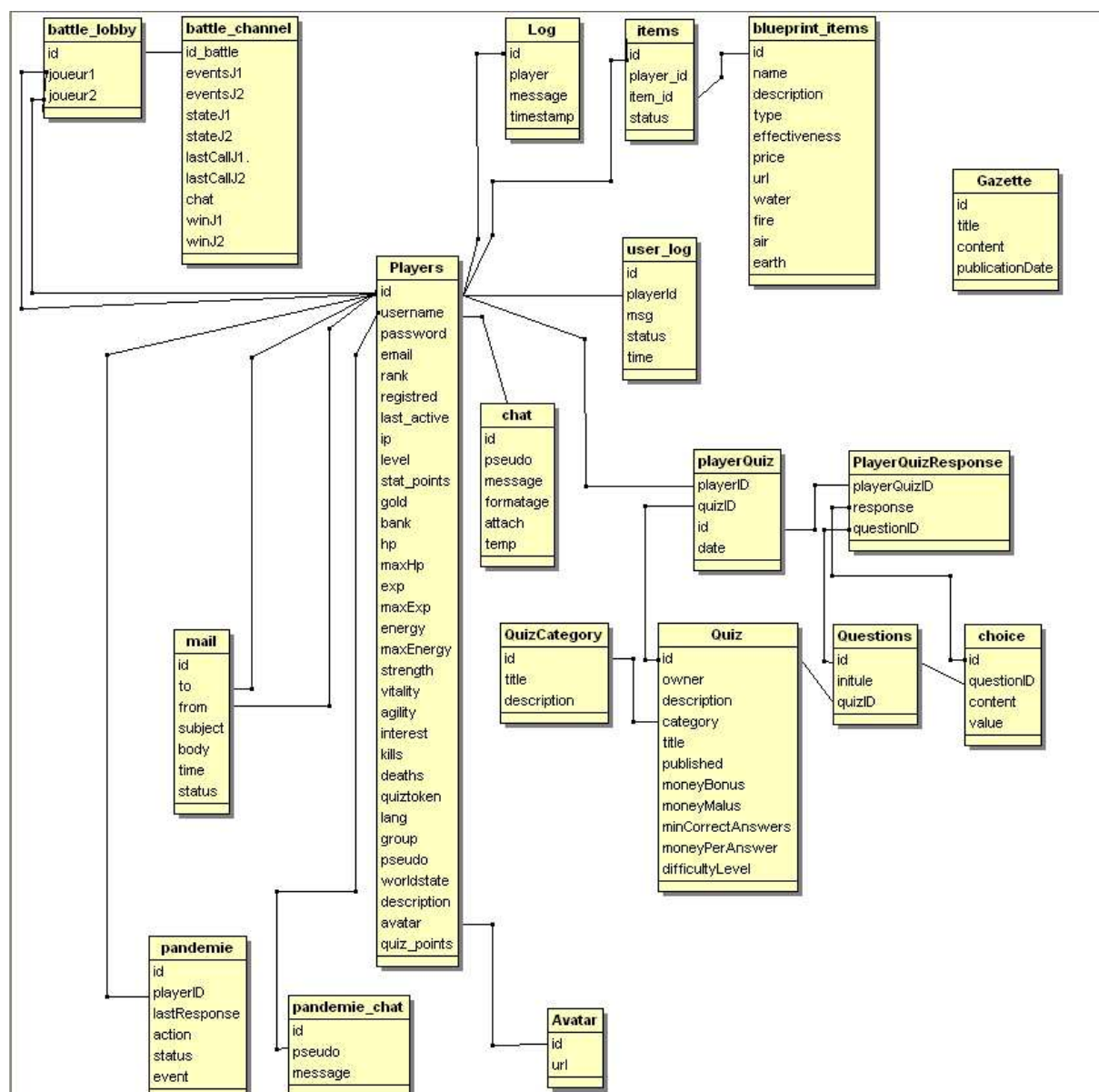
Enfin, plusieurs fois j'ai été confronté à des problèmes de performance à cause des limites du langage Javascript. Peut être aurait-il été plus judicieux d'utiliser dès le départ du Flash afin de tirer partie de la performance de son moteur graphique et physique.

8 – Sitographie

- **www.siteduzero.com** : site communautaire contenant des cours et tutoriaux sur divers langages de programmation
- **www.php.net** : manuel technique du langage PHP
- **www.developpez.com** : site communautaire contenant des explications d'algorithmes
- **www.toutjavascript.com** : manuel technique du Javascript
- **Api.jquery.com** : manuel technique de jQuery
- **gamequery.onaluf.com** : manuel technique de gameQuery
- **www.artichow.org** : manuel technique d'Artichow

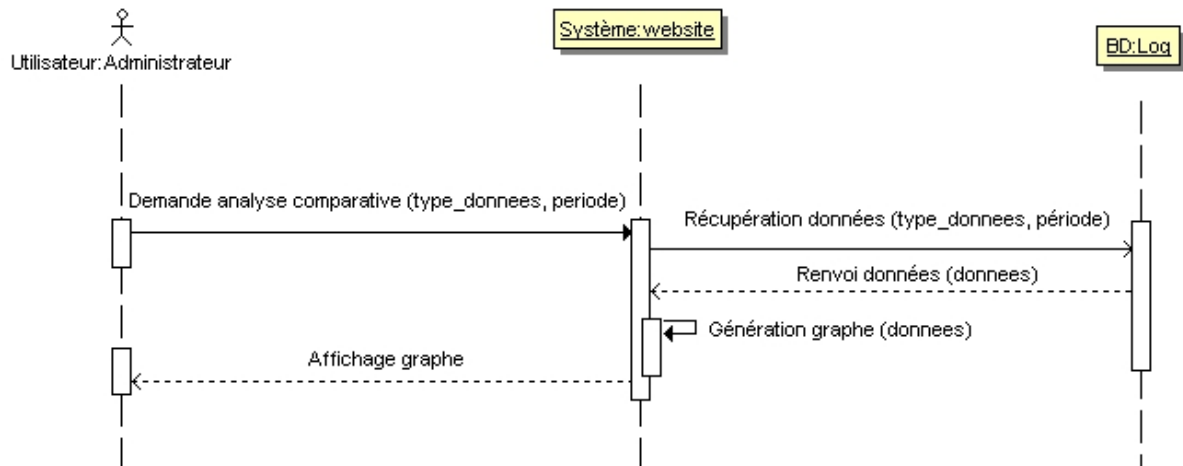
Annexe 2 : modèle relationnel des données complet

Voici le schéma de la base de données de School Society à la fin du stage.

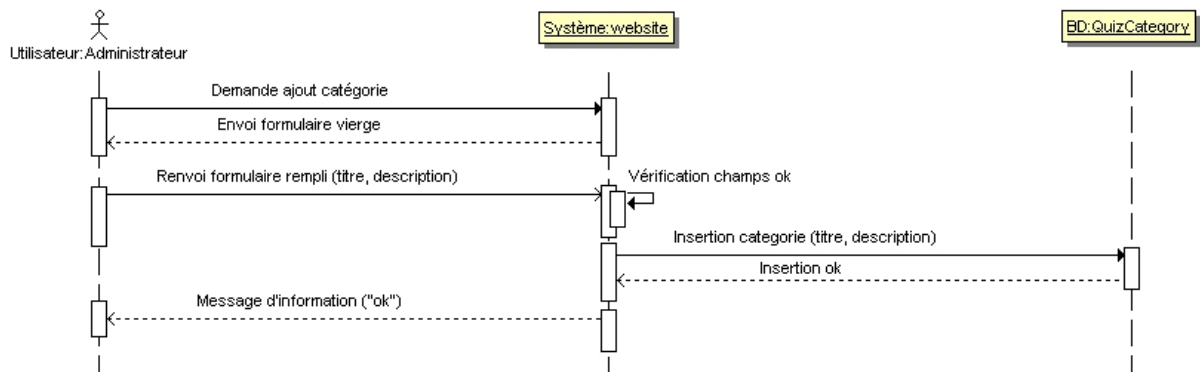


La suite des annexes correspond aux différents diagrammes de séquence réalisés pour le back-office. Les intitulés des diagrammes étant relavements explicites, je ne vais pas expliquer les différents schémas présents.

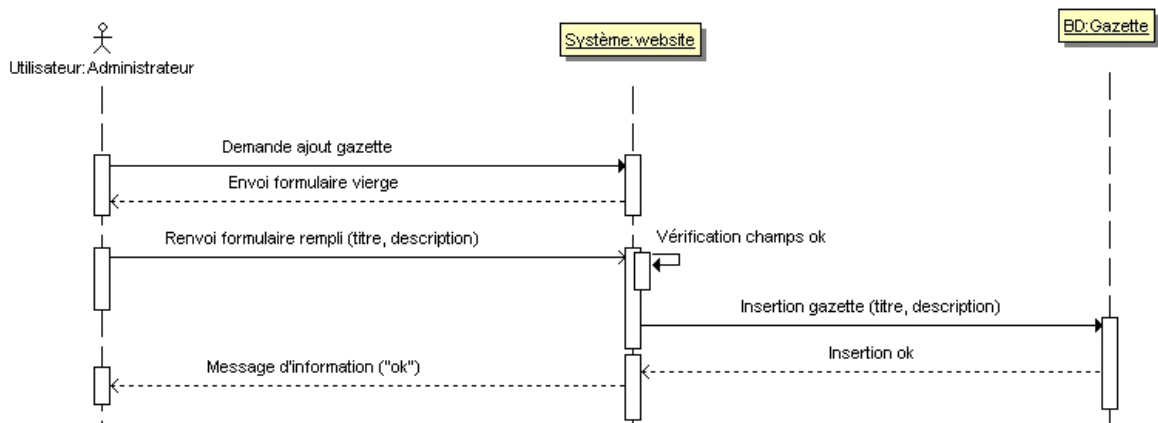
Annexe 3 : diagramme de séquence back-office : consultation des statistiques



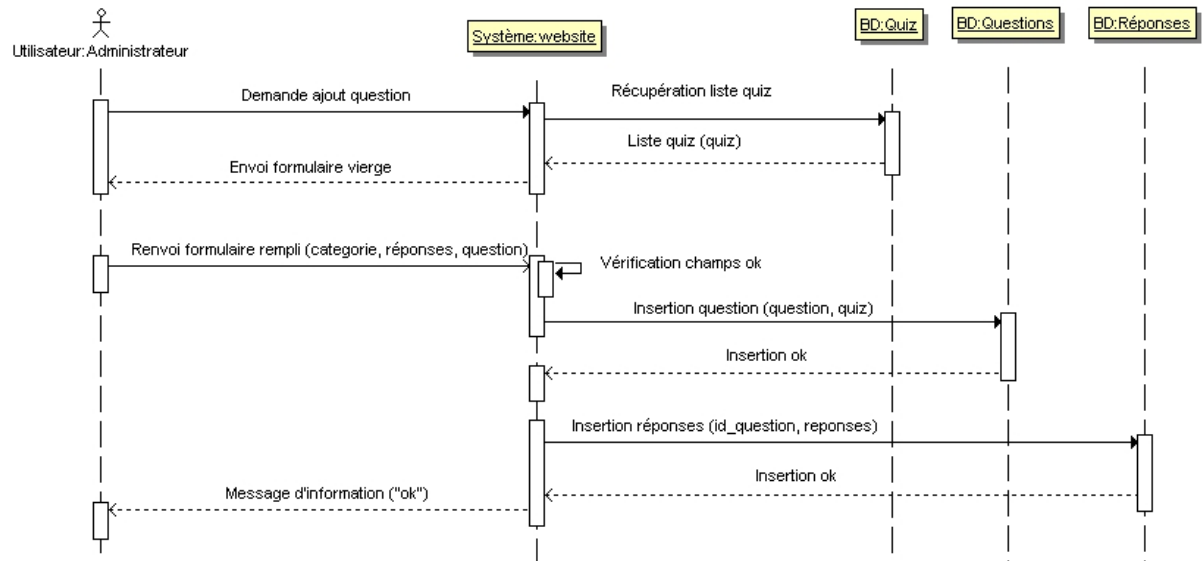
Annexe 4 : diagramme de séquence back-office : ajout d'une catégorie



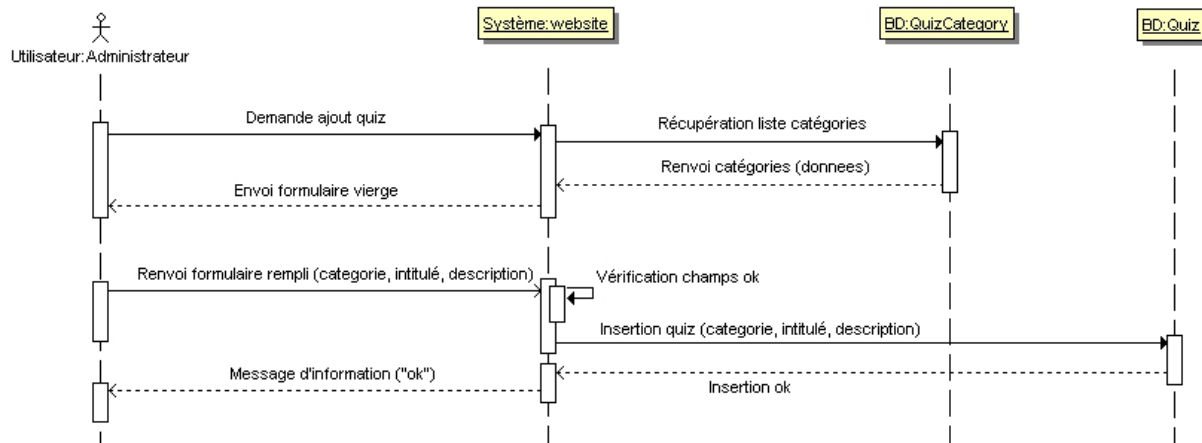
Annexe 5 : diagramme de séquence back-office : ajout d'une gazette



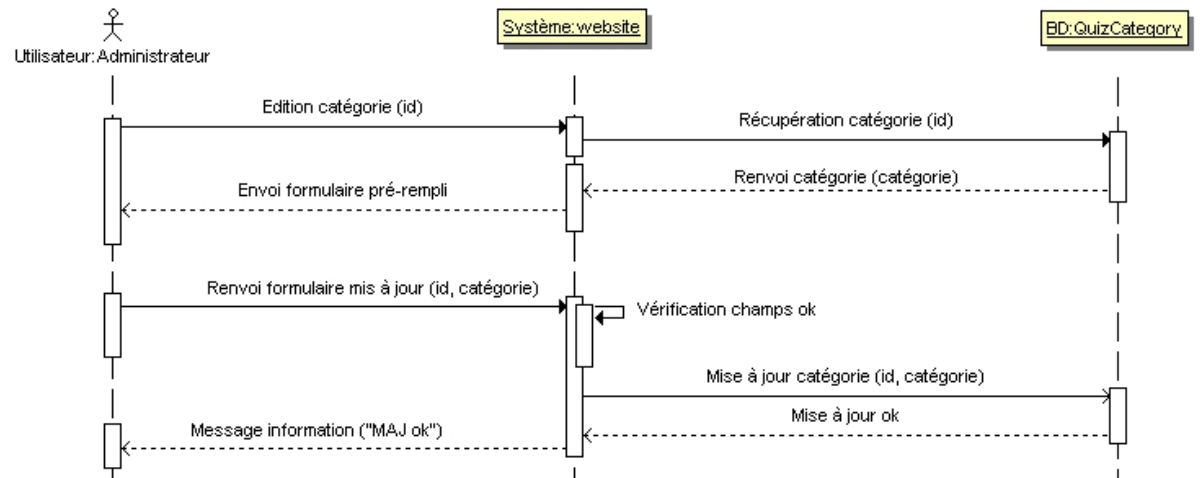
Annexe 6 : diagramme de séquence back-office : ajout d'une question



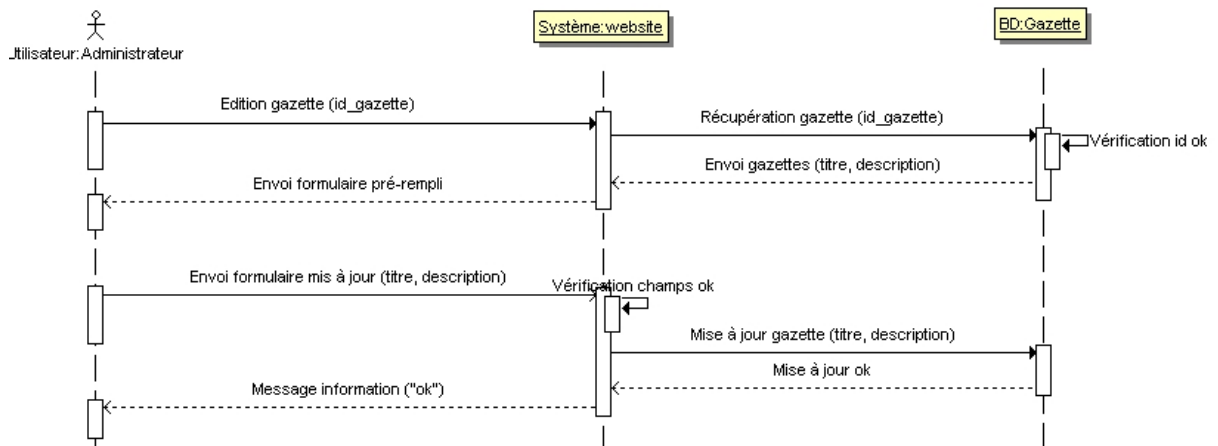
Annexe 7 : diagramme de séquence back-office : ajout d'un quiz



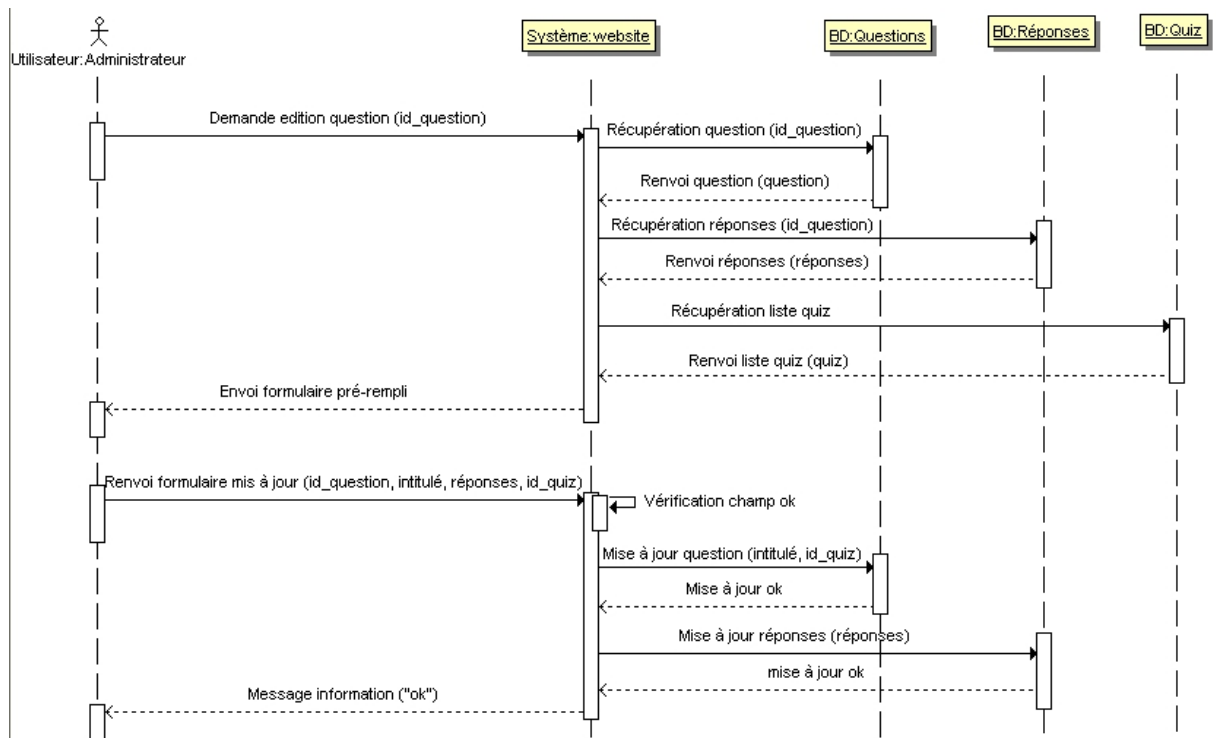
Annexe 8 : diagramme de séquence back-office : édition d'une catégorie

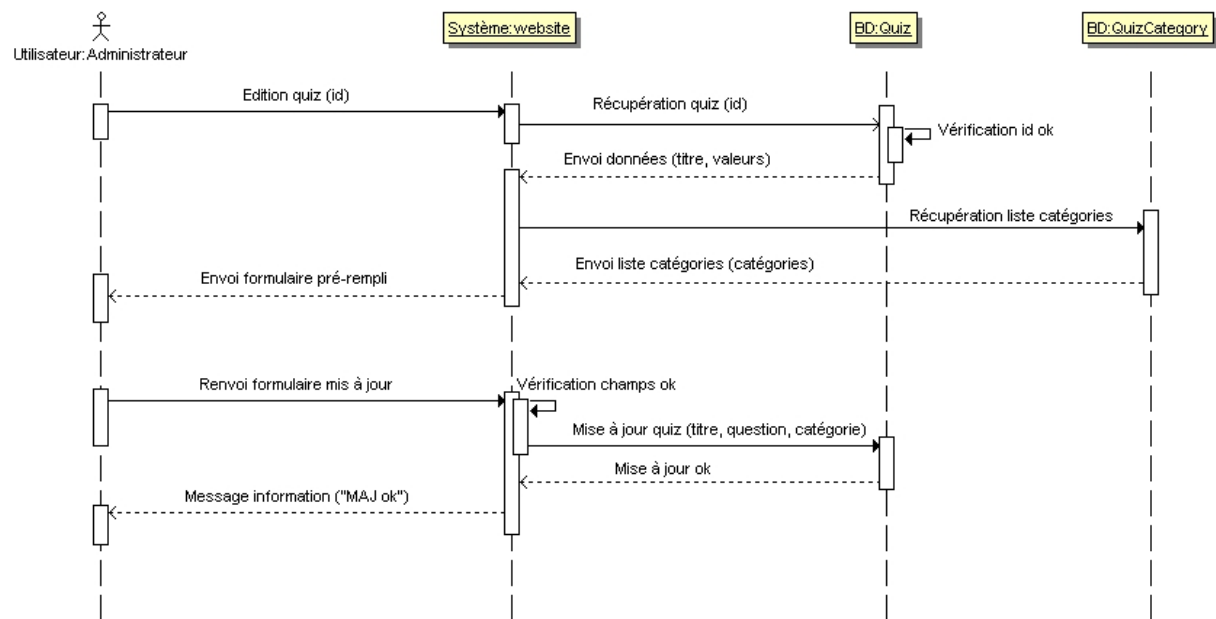


Annexe 9 : diagramme de séquence back-office : édition d'une gazette



Annexe 10 : diagramme de séquence back-office : édition d'une question



Annexe 11 : diagramme de séquence back-office : édition d'un quiz

La suite des annexes correspond au tutorial ainsi qu'à la documentation de School society que j'ai réalisé.

Résumé

L'équipe SMILE du LIRMM a lancé une expérimentation auprès d'étudiants de première année en DUT Informatique permettant de développer un framework d'analyse et de conception de systèmes informatiques sociaux.

Cette expérimentation, nommée School Society, est un jeu sérieux interactif utilisant une architecture web.

Le stage réalisé au sein de cette unité permettrait, d'une part, de développer le système School Society en y apportant du contenu et en développant de nouvelles fonctionnalités et, d'autre part, de prouver la validité du modèle proposé.

Mots clés : jeu sérieux, stage, expérimentation, framework.

Summary

The SMILE team working at LIRMM launched a testing with first year students in bachelor degree in order to develop a Social System framework.

This experimentation, named School Society, is an interactive serious game using a web structure.

The training period realized in this unit would, on one hand, develop the School Society system by adding content and by developing new features and, on the other and, prove the validity of the design proposed.

Key words : serious game, training period, testing, framework.