

Rapport de projet de Développement d'Applications Web

*Création d'un jeu en ligne :
les blaireaux et les kékés*



*Alexis Guyot | Alexandre Labrosse | Arnaud François
Galvin Bernard | Thinhinane Saidi | Timothé Batut*

Table des matières

Table des matières	2
I. Introduction.....	4
II. Fonctionnalités	5
a. Gestion des joueurs.....	5
b. Gestion des parties.....	7
c. Le jeu	8
La partie.....	8
Les joueurs.....	9
Les déplacements et les interactions possibles.....	11
d. Les règles ++	11
III. Modélisation.....	14
a. Diagrammes de cas d'utilisation	14
b. Maquette et navigation de notre site web	17
c. Mise en place du MVC.....	18
d. Diagramme de packages	19
e. Diagramme de classe.....	20
f. Schéma de la base de données	22
IV. Développement.....	26
a. Chargement automatique des classes	26
b. Echanges de données : SimpleXML et JSON.....	28
c. Partage de données entre utilisateurs : sessions et cookies.....	30
d. Utilisation de la base de données : PDO	31
e. Communications entre le client et le serveur : AJAX	32
f. MVC : Les différents contrôleurs.....	34
g. Envoi de mails.....	36
h. Vérifications Javascript et PHP	36
i. Création du jeu : ThreeJS.....	37
j. Bugs connus.....	39
k. Difficultés rencontrées	40
l. Améliorations possibles.....	40
V. Organisation	41
a. La méthode utilisée	41
Réunion n°1 : 13 février 2019.....	41
Réunion n°2 : 27 février 2019.....	41
Réunion n°3 : 7 mars 2019	42

Réunion n°4 : 21 mars 2019	42
Réunion n°5 : 4 avril 2019	42
Réunion n°6 : 10 avril 2019	43
Réunion n°7 : 2 mai 2019	43
Conclusion	44
b. Les outils utilisés.....	44
VI. Conclusion	46
Annexes	47
Diagramme de classes complet.....	47

I. Introduction

Dans le cadre du deuxième semestre de la L3 Informatique à l'UFR Sciences et Techniques de Dijon, nous avons mené un projet dans le module de Développement Web. Ce projet consiste en la création d'un site permettant de jouer à un jeu de type « Pac-Man » en ligne.

Créer ce site va nous permettre de mettre en pratique toutes les connaissances acquises lors de ce module en utilisant les technologies vues en cours, en travaux pratiques et en travaux dirigés. Ces technologies sont liées aux langages JavaScript, PHP et HTML/CSS. En lien avec l'autre module de L3 Informatique qu'est Image pour le Web, certaines fonctionnalités concernant le jeu lui-même seront implémentées grâce à « Three.js », qui est une bibliothèque JavaScript pour créer des scènes en 3D sur un navigateur.

Notre groupe pour réaliser ce projet est composé d'Alexis Guyot, Alexandre Labrosse, Arnaud François, Galvin Bernard, Thinhinane Saidi et Timothé Batut. Le chef de projet est Alexis Guyot et le responsable qualité Alexandre Labrosse.

Nous allons, tout au long de ce rapport, vous décrire quelles étaient les fonctionnalités attendues pour ce projet, comment nous avons procédé pour le réaliser, et comment nous nous sommes organisés. Notre site est accessible en ligne à l'adresse <http://www.projet-web-l3.fr>. Pour que vous puissiez mener à bien vos différents tests, deux utilisateurs ont été créés :

- Pseudo : JoueurTest | Mot de passe : ProjetWebL3_
- Pseudo : AdminTest | Mot de passe : ProjetWebL3_

Comme son nom l'indique, AdminTest possède les droits d'administrateur et peut par conséquent accéder au contenu réservé à ce statut.

II. Fonctionnalités

Dans cette partie, nous allons vous décrire toutes les consignes données dans notre cahier des charges et menées à bien lors de notre projet. Nous allons séparer ces fonctionnalités en plusieurs parties : la gestion des joueurs, la gestion des parties, le jeu en lui-même et enfin les règles supplémentaires implémentées pour ce dernier. Il y a en plus une fonctionnalité plus générale qui ne correspondait à aucune des sous-parties suivantes, nous tenons donc à la présenter ici. Il s'agit du fait que le site web possède trois styles différents. Le style choisi est défini dans le fichier `php/Config/config.php`.

a. Gestion des joueurs

Pour jouer à notre jeu, il faut posséder un compte sur notre site. Un système d'inscription et d'authentification ont donc dû être mis en place. Ce compte doit ensuite être validé grâce au mail de confirmation envoyé par le système, puis reçu par l'utilisateur. Au niveau des règles à l'inscription, le mot de passe du joueur doit être composé d'au moins un chiffre ainsi que de lettres avec au moins une majuscule, pour un total de 8 à 20 caractères. Son pseudo doit faire la même taille mais ne possède pas de restriction sur son contenu. Il peut en plus éventuellement contenir un point d'interrogation ou d'exclamation ainsi qu'un tiret du haut ou du bas. Il est nécessaire d'être authentifié sur notre site pour pouvoir accéder à autre chose que la page de connexion. Le mot de passe du joueur n'est pas stocké en clair dans la base de données, mais est crypté avant son enregistrement.

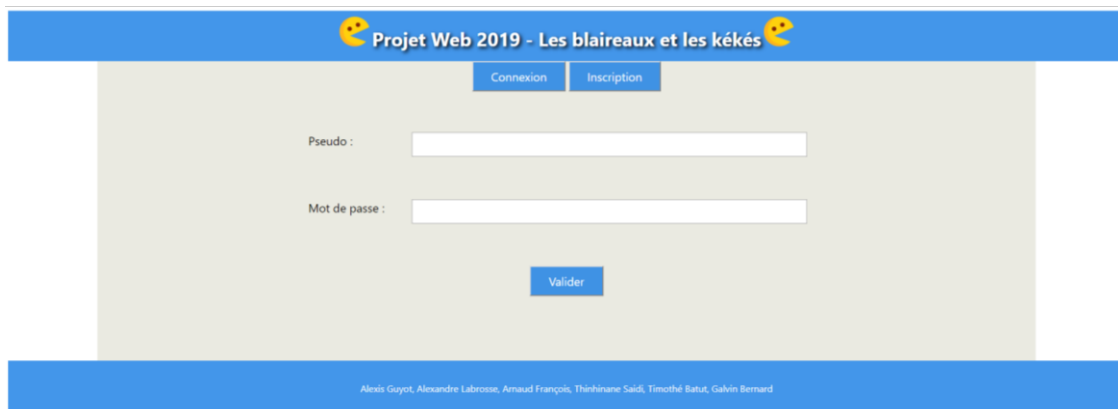


Figure 1 - Page d'accueil du site, mode connexion

The screenshot shows the registration page of a website. At the top, there is a blue header with the site's name and two yellow smiley face icons. Below the header, there are two buttons: 'Connexion' and 'Inscription'. The 'Inscription' button is highlighted. The main form area is light gray and contains several input fields: 'Pseudo:', 'Adresse email:', 'Confirmation de l\'adresse email:', 'Mot de passe:', and 'Confirmation du mot de passe:'. There is also a small grid of dots next to the password confirmation field. At the bottom of the form, there is a blue 'Envoyer' button. The footer is a blue bar with the names of the team members: Alexis Guyot, Alexandre Labrosse, Amaud François, Thinhinane Saidi, Timothé Batut, Galvin Bernard.

Figure 2 - Page d'accueil du site, mode inscription

Un comportement correct est exigé sous peine de bannissement en cas de non-respect des autres utilisateurs et/ou du site/jeu. Pour s'assurer de la bonne utilisation de notre projet, une interface d'administration a été mise en place, offrant ainsi la possibilité de bannir des inscrits. A noter que cette page n'est accessible que si l'utilisateur est déclaré comme administrateur dans la base de données. Si c'est le cas, il disposera d'un bouton de plus sur la page principale pour accéder à l'interface d'administration.

The screenshot shows the main page of the website in administrator mode. At the top, there is a blue header with the site's name and two yellow smiley face icons. Below the header, there is a blue bar with the text 'Bonjour AlexisGuyot' and two buttons: 'deconnexion' and 'admin'. The main content area is light gray and contains two blue buttons: 'Créer une partie' and 'Rejoindre une partie'. The footer is a blue bar.

Figure 3 - Page principale du site, mode administrateur

Administration du site

Rechercher un joueur :

Alexandre	alexandre_labrosse@etu.u-bourgogne.fr
AlexisGuyot	alexis.guyot39@gmail.com
AlexisGuyot2	ok-gio@gmail.com
AlexisGuyot3	weshgleelumonfrere@gmail.com
AlexisGuyot4	truc@machin.fr

Motif du ban :

[Revenir à la page principale](#) [Valider](#)

Alexis Guyot, Alexandre Labrosse, Arnaud François, Thinhinane Saidi, Timothé Batut, Galvin Bernard

Figure 4 - Page d'administration

b. Gestion des parties

Les utilisateurs peuvent créer des parties. Une fois cela fait, ils sont renvoyés dans une salle d'attente où ils devront attendre au moins deux autres joueurs pour pouvoir lancer le jeu (le nombre de joueurs minimum est donc égal à 3). Seul le créateur de la partie peut la lancer. A chaque création d'une partie, une vérification est faite pour supprimer celles qui sont toujours en attente mais sans joueur à l'intérieur. Dans les salles d'attente, les joueurs peuvent inviter d'autres utilisateurs. Ils sont également en mesure de rejoindre une partie en attente grâce à la page dédiée, accessible depuis la page principale. Sur cette même page, ils pourront également retrouver toutes les invitations qui leur ont été envoyées et visualiser les parties publiques en cours, pour éventuellement en rejoindre une et la regarder en tant que spectateur.

Création d'une partie

Nom partie :

Partie privée : ☐ Partie publique : ☒

Nombre de joueurs :

Durée en minutes :

[Créer la partie](#) [Page principale](#) [Réinitialiser les paramètres](#)

Alexis Guyot, Alexandre Labrosse, Arnaud François, Thinhinane Saidi, Timothé Batut, Galvin Bernard

Figure 5 - Création d'une partie

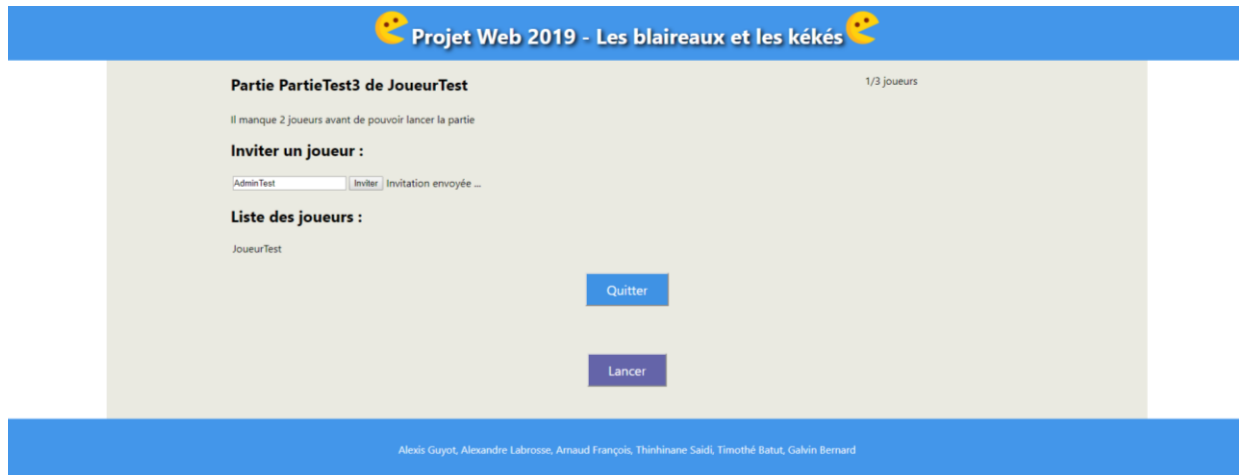


Figure 6 - Salle d'attente d'une partie

Les parties créées peuvent être définies de deux manières différentes : privées ou publiques. Les parties privées sont des parties où les joueurs ont été invités par le créateur ou qui possèdent simplement le mot de passe pour entrer. Sur la page pour rejoindre une partie, celles-ci apparaissent avec un champ pour pouvoir renseigner ce dernier. Un adhérent ne pourra rejoindre la salle d'attente que s'il possède le bon mot de passe, ou s'il la rejoint depuis sa liste d'invitations. Il est impossible d'être spectateur d'une partie privée, celles-ci n'apparaissent même pas dans l'interface pour rejoindre une partie en cours. Les parties publiques, quant à elles, peuvent être rejointes par n'importe qui, en tant que joueur ou en tant que spectateur.

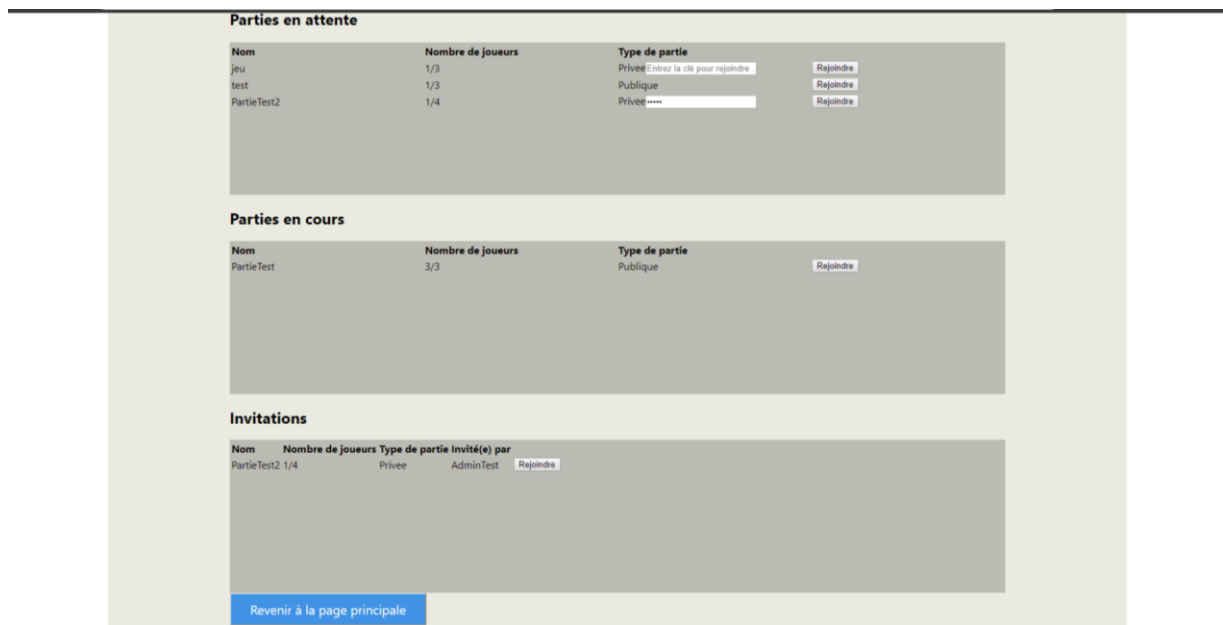


Figure 7 - Interface pour rejoindre une partie

c. Le jeu

La partie

Le jeu se déroule de la manière suivante : les joueurs sont répartis de manière aléatoire en deux équipes, les kékés et les blaireaux. Les kékés sont en surnombre. Il y a en moyenne 1/3 de blaireaux pour 2/3 de kékés (d'où les 3 joueurs minimum pour commencer une partie). Il existe deux façons de

terminer une partie : arriver au bout du décompte ou éliminer complètement l'équipe adverse. Pour ce faire, le joueur doit posséder le rôle « blaireau ». Si c'est le cas, il pourra alors dévorer les joueurs de l'équipe d'en face en trouvant un joueur et en se plaçant sur la même case que lui. S'il essaye de dévorer un autre blaireau, le joueur réussira mais mourra en conséquence. Si les deux équipes ne peuvent se départager, la victoire est donnée à l'équipe ayant le plus de joueurs à la fin du temps imparti.

Il peut arriver, à certains moments de la partie, qu'un gong sonne (dans notre cas le gong est un son de vache) et inverse les rôles des deux équipes. Les kékés deviennent alors les blaireaux et vice versa.

Les joueurs

Tous les joueurs possèdent un avatar, représenté à l'écran par une forme géométrique simple. Les équipes sont différenciées grâce à une couleur propre à chacune. La première équipe est représentée par des cubes rouges, la deuxième par des cylindres bleus.

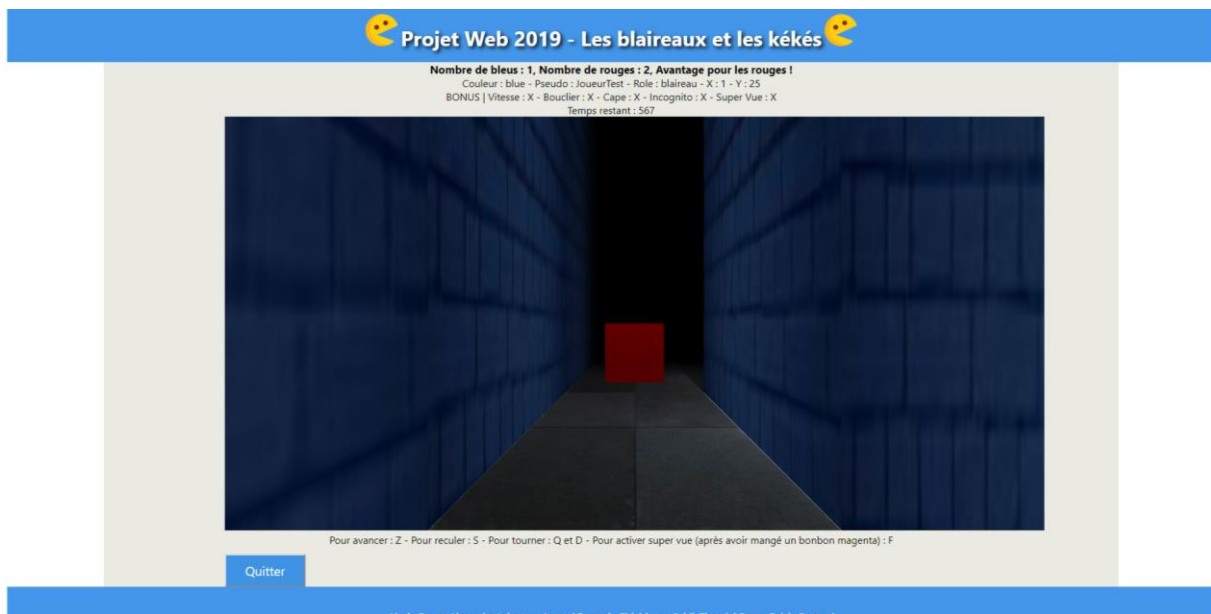


Figure 8 - Interface du jeu pour un joueur de l'équipe bleue



Figure 9 - Interface du jeu pour un joueur de l'équipe rouge

Les avatars se déplacent sur un terrain constitué de cases selon une grille. Les bords du terrain se rejoignent à la manière d'un tore. Si l'on sort d'un côté du terrain, cela entraîne un retour du côté opposé de la grille. En début de partie, les joueurs sont répartis aléatoirement sur le terrain. La vision de chaque joueur est en première personne (on voit ce que voit notre avatar). La distance de vision est limitée à trois cases.

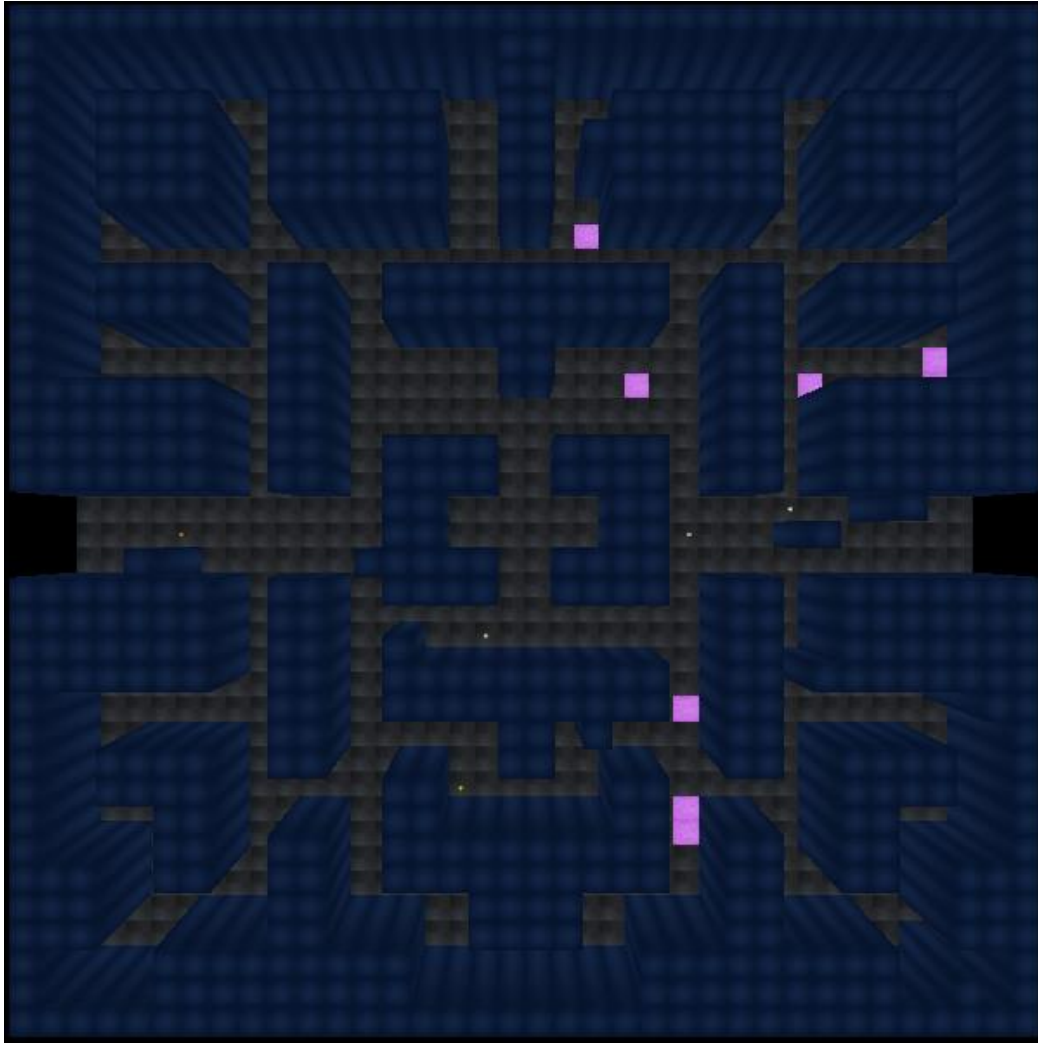


Figure 10 - Carte du jeu

Les déplacements et les interactions possibles

Au niveau des contrôles du jeu, les touches Z, Q et D sont utilisées. Les touches Q et D serviront à tourner la caméra du joueur, respectivement d'un quart de tour à gauche et d'un quart de tour à droite. La touche Z servira elle à avancer d'une case suivant la direction du regard du joueur. Ces touches ont été privilégiées par rapport aux flèches directionnelles demandées dans le sujet car celles-ci avaient parfois tendance à faire descendre ou monter la page en plus de ce qu'on leur demandait.

d. Les règles ++

Il y a sur le terrain des obstacles qui limitent les déplacements dans l'espace de jeu, représentés par des murs. On peut également retrouver sur celui-ci des flaques magiques (cases roses) qui font réapparaître le joueur sur une autre flaque magique disposée autre part sur la carte.

De temps en temps, des bonus qui confèrent des supers pouvoirs pour une durée limitée dans le temps (30 secondes) apparaissent :

- La cape d'invisibilité, représentée par une sphère jaune.

- Les bottes de 7 lieues, qui permettent un déplacement deux fois plus rapide, représentées par une sphère noire.
- Le bouclier d'or, qui permet au joueur d'être intouchable pendant un certain temps, représenté par une sphère blanche.
- La potion super vue, qui permet au joueur, via l'appui sur la touche F, de passer en vue du dessus et donc de voir la totalité du terrain. Cependant, aucun déplacement ne sera possible durant l'utilisation de ce bonus tant que le joueur n'est pas revenu à sa vue « classique » en première personne. Ce changement de caméra est possible pour le joueur autant de fois qu'il le souhaite tant que le bonus a encore de l'effet. Il est représenté par une sphère violette.
- La potion incognito, qui permet au joueur d'apparaître de façon neutre sur la carte et donc de ne pas être détecté par les autres joueurs, représentée par une sphère orange.

Il est possible de rejoindre une partie en cours dans laquelle on n'est pas joueur. Dans ce cas, on se connecte en tant que spectateur, et on dispose en conséquence d'un ensemble de boutons pour accéder aux vues en première personne de tous les joueurs, ainsi qu'un autre supplémentaire pour observer la partie en vue du dessus.

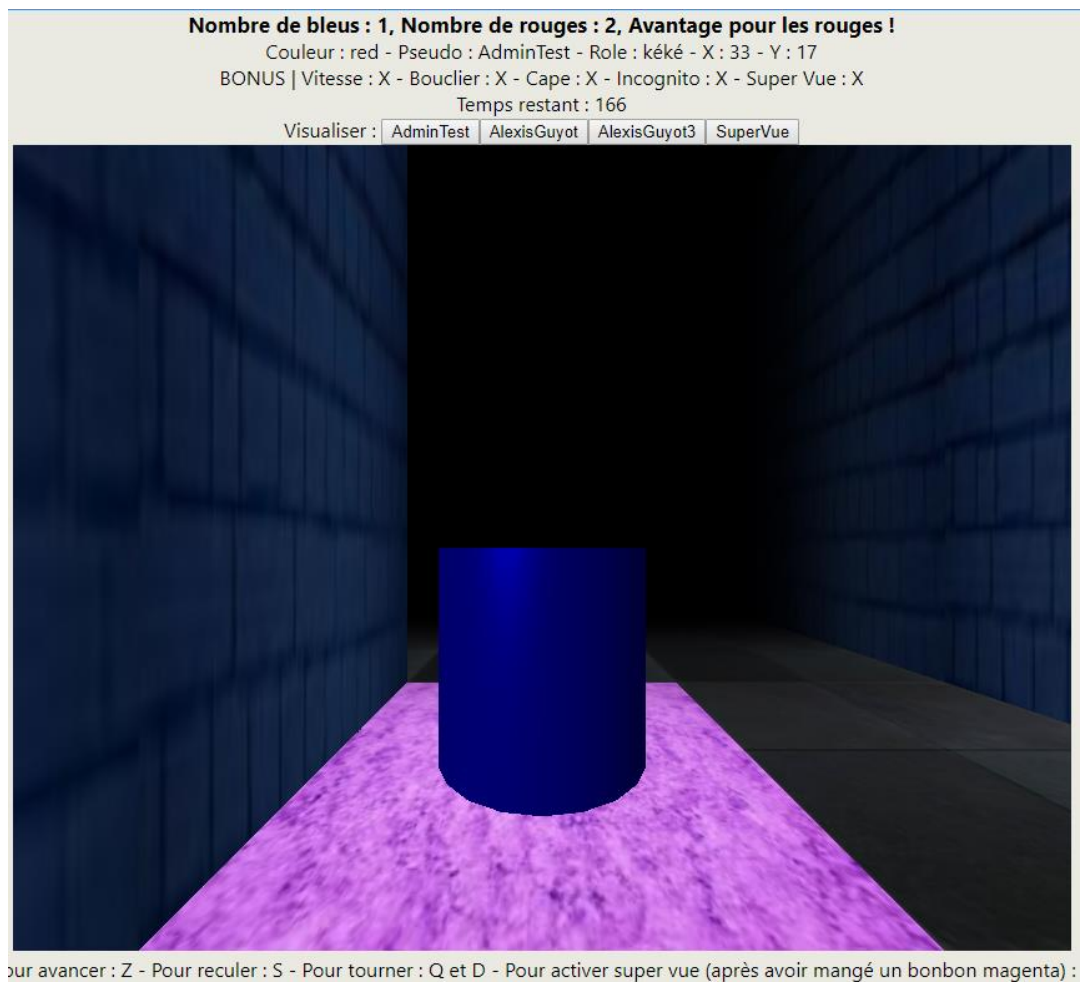


Figure 11 - Interface du jeu pour un spectateur

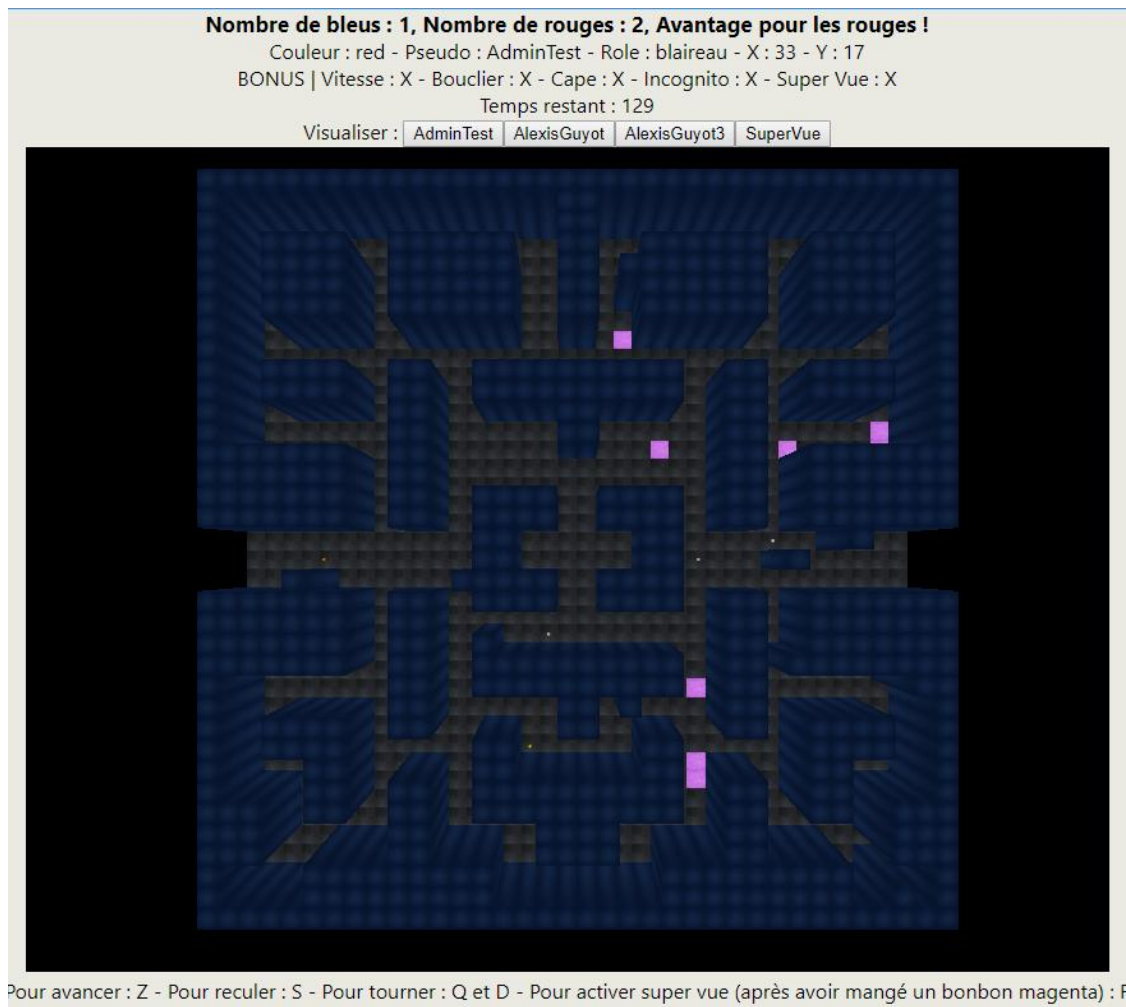


Figure 12 - Les spectateurs peuvent accéder à la visualisation de la carte (mode super vue)

III. Modélisation

a. Diagrammes de cas d'utilisation

Afin d'identifier les différentes fonctionnalités attendues pour notre site web, nous avons commencé par réaliser des diagrammes de cas d'utilisation. Ces derniers vont permettre de nous donner une vision d'ensemble du comportement que devra avoir notre site web. Il est intéressant d'effectuer différents diagrammes de cas d'utilisations, ayant chacun un niveau d'abstraction plus ou moins important. Cela nous permettra d'avoir une vision plus ou moins détaillée de chacune des fonctionnalités attendues, sans surcharger un seul diagramme principal. C'est ce que nous avons décidé d'effectuer.

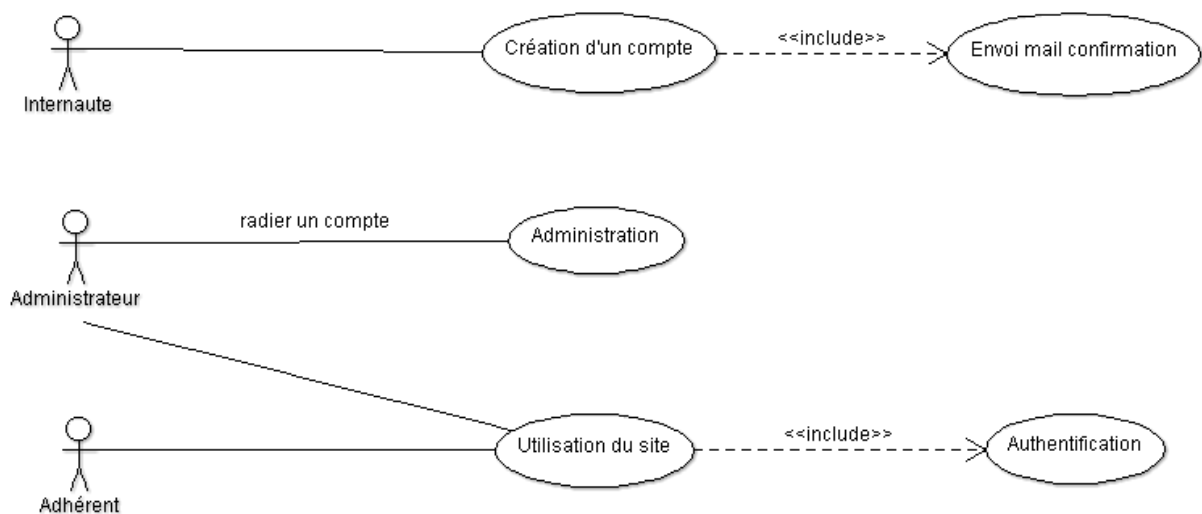


Figure 13 - Diagramme de cas d'utilisation de niveau 1

Le diagramme Use Case ci-dessus est le diagramme de premier niveau. Il représente notre site Web d'un point de vue global. Trois acteurs sont présents : l'internaute, l'administrateur et l'adhérent. L'internaute représente toute personne pouvant accéder à notre site. Il pourra alors se créer un compte. Afin de valider celui-ci, un email de confirmation lui sera envoyé. Ensuite, nous avons représenté l'acteur Adhérent. Ce dernier possède un compte validé sur notre site et peut donc l'utiliser lorsqu'il s'est authentifié. Enfin, l'administrateur hérite des fonctionnalités accordées à l'adhérent. Cependant, il aura en plus comme rôle d'administrer le site, comme son nom l'indique.

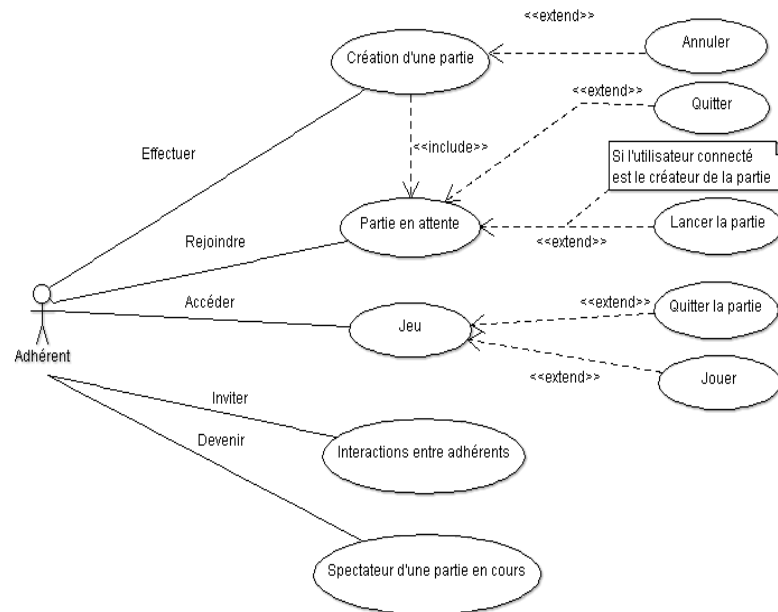


Figure 14 - Diagramme de cas d'utilisation de niveau 2 (utilisation du site)

Voici le diagramme de cas d'utilisation de second niveau. Il permet de détailler ce que peut faire un adhérent connecté à notre site.

Dans un premier temps, l'utilisateur peut créer une partie. La relation d'extension avec le cas d'utilisation « Annuler » spécifie le cas où l'utilisateur voudrait mettre un terme à la partie qu'il est en train de créer.

Ensuite, l'adhérent peut également rejoindre une partie en attente de joueurs. Lorsqu'une partie est créée, elle est automatiquement mise en attente, d'où le lien include entre « Création d'une partie » et « Partie en attente ». L'utilisateur peut choisir de lancer la partie s'il est le créateur de celle-ci, ou bien la quitter s'il en a envie.

De plus, lorsque la partie est lancée, il peut logiquement jouer mais aussi quitter la partie en cours. Aucune relation n'a été spécifiée dans le graphique ci-dessus, mais il est logique qu'il soit obligatoire qu'une partie ait été créée avant de pouvoir y jouer.

Enfin, l'adhérent peut inviter d'autre adhérents pour jouer ensemble. Il peut également devenir spectateur d'une partie en cours de déroulement.

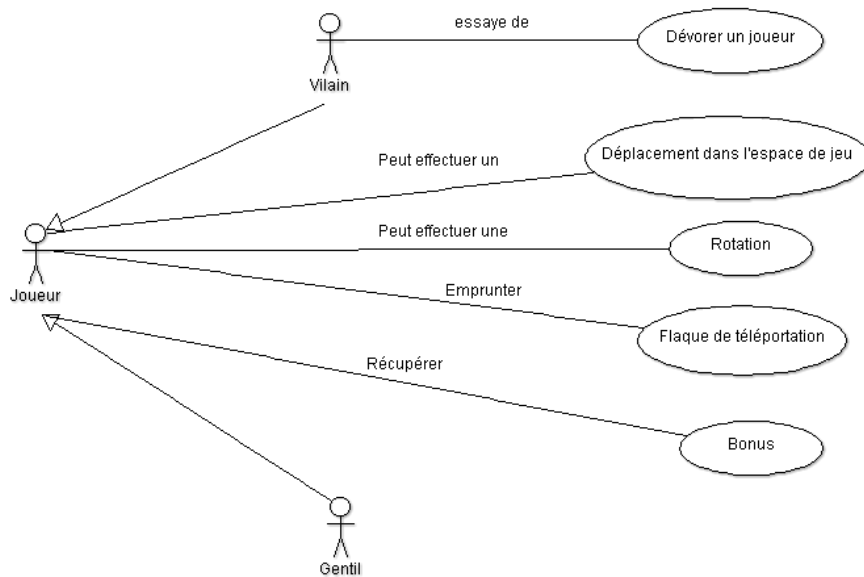


Figure 15 - Diagramme de cas d'utilisation de niveau 3 (Jeu)

Le diagramme ci-dessus est un diagramme de troisième niveau détaillant ce que peut faire un joueur au cours d'une partie. Le joueur peut logiquement se déplacer dans l'espace de jeu, effectuer une rotation, emprunter une flaque de téléportation et récupérer divers bonus comme nous avons pu le voir dans le cahier des charges. De plus, les utilisateurs peuvent avoir le rôle de vilain (blaireau) ou de gentil (kéké). Ces deux derniers sont représentés dans notre diagramme sous la forme de deux acteurs distincts, héritant tous deux de Joueur. La différence entre les deux est que le but d'un joueur Vilain est de dévorer les joueurs Gentil. Nous avons donc ajouté un cas d'utilisation spécifique à cette action pour l'acteur Vilain.

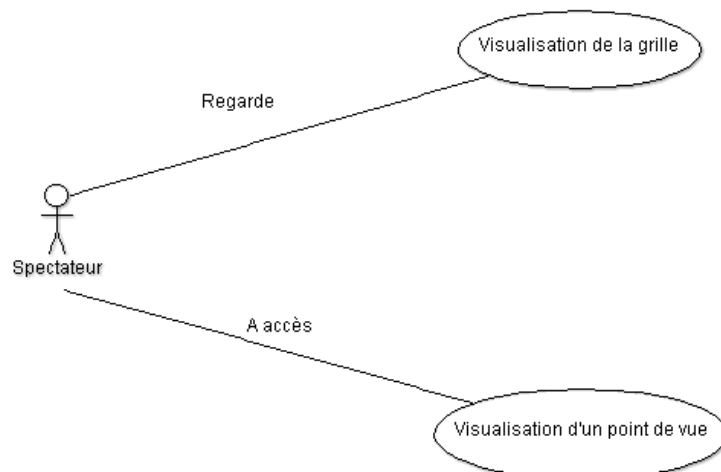


Figure 16 - Diagramme de cas d'utilisation de niveau 3 (Spectateur d'une partie en cours)

Enfin, voici le deuxième diagramme de troisième niveau détaillant le rôle d'un spectateur sur notre site web. Comme indiqué dans le cahier des charges, ce dernier a la possibilité de regarder une partie du point de vue d'un des joueurs ou bien d'un point de vue global en visualisant la totalité de la grille.

b. Maquette et navigation de notre site web

Afin de bien visualiser la navigation entre les différentes pages de notre site, nous avons établi un diagramme de navigation. Ce dernier nous a permis d'établir les différentes pages à réaliser, tout en commençant à penser au design et à l'enchaînement de ces dernières pour la navigabilité. Nous l'avons réalisé tous ensemble au cours de l'une de nos réunions de projet (voir partie Organisation). Voici comment nous l'avons représenté.

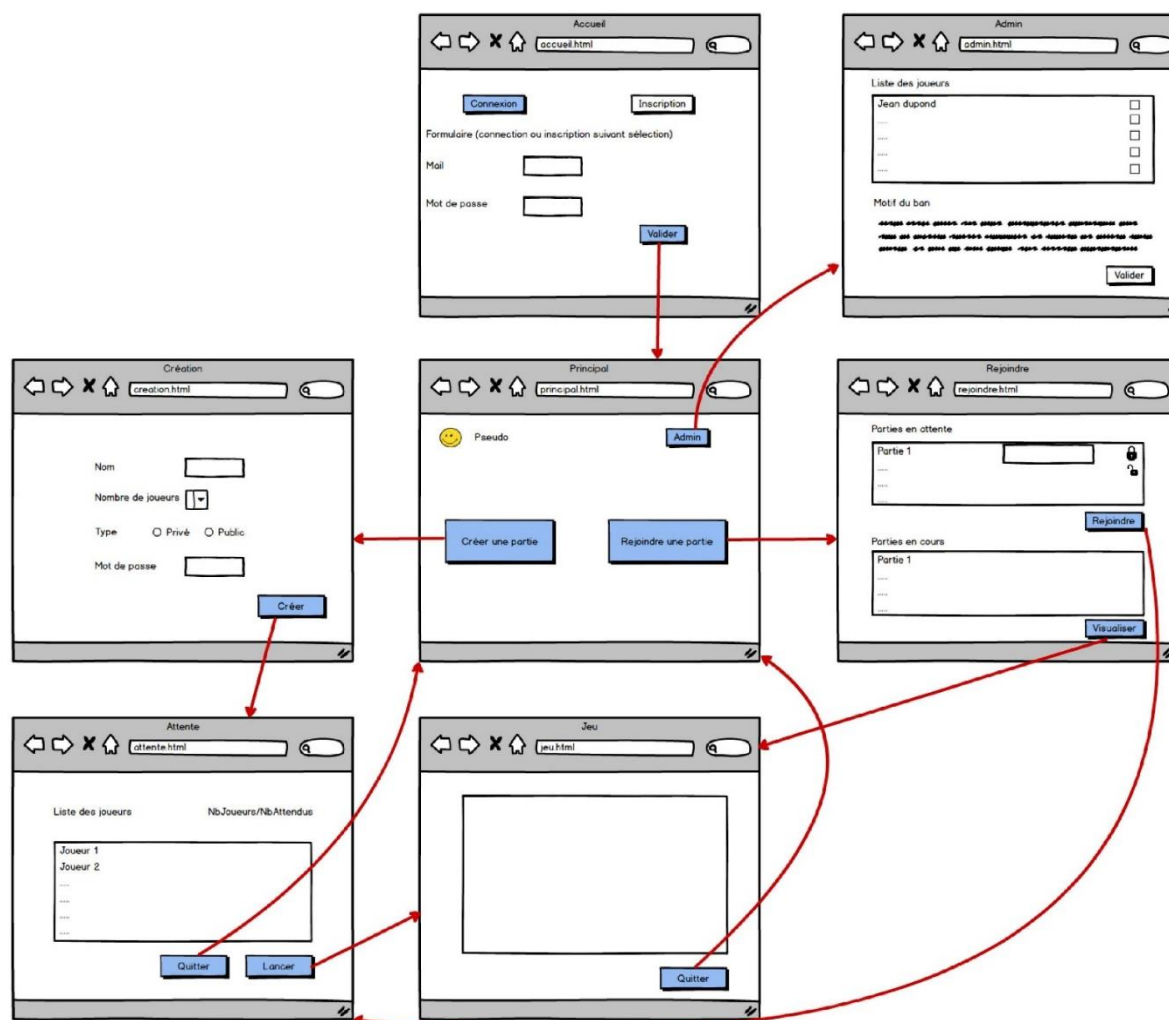


Figure 17 - Diagramme de navigation

Tout d'abord, lorsque l'utilisateur se rend sur notre site web, il se trouve sur la page « accueil » sur laquelle il peut se connecter ou se créer un compte. De base, le formulaire de connexion s'affiche. Si l'internaute veut s'inscrire, il doit cliquer sur le bouton « inscription » et le formulaire d'inscription apparaîtra. Ensuite, lorsque l'utilisateur se connectera, il arrivera sur la page « principal » et verra apparaître deux boutons au milieu de son écran : l'un pour créer une partie, l'autre pour en rejoindre une. Dans le cas où l'utilisateur serait un administrateur, le bouton « admin » apparaîtra également. Ce dernier permet de se rendre sur la page « admin » comme son nom l'indique si bien. Cette dernière va permettre à l'administrateur de bannir un joueur s'il estime que son comportement est incorrect. Il pourra d'ailleurs saisir le motif de son bannissement grâce à une zone de saisie.

Si l'utilisateur connecté choisi de créer une partie (en cliquant sur le bouton « Créer une partie »), la page « creation » sera lancée. Il pourra ainsi donner un nom, un nombre de joueurs, un

type de partie (privée ou publique) et un mot de passe (dans le cas où la partie est privée). S'il valide la création de sa partie, la page « attente » sera affichée à l'écran, indiquant les joueurs participant à la partie créée. Lorsque le nombre de joueurs nécessaires sera atteint, le créateur de la partie pourra lancer le jeu. La page « Jeu » sera alors appelée et les joueurs pourront commencer à jouer.

Ensuite, si un joueur souhaite rejoindre une partie (en cliquant sur le bouton « Rejoindre une partie »), la page « rejoindre » sera affichée. Celle-ci affiche les parties qui sont en attente de joueurs et celles déjà en cours. L'utilisateur pourra alors choisir celle qu'il souhaite rejoindre, en tant que joueur pour celles en attente ou en tant que spectateur pour les autres. Il a également accès sur cette page aux invitations qui lui ont été envoyées (pas représentées sur le diagramme ci-dessus pour des raisons de lisibilité, une capture complète de cette page est présente dans la partie Fonctionnalités).

Ainsi, nous avons détaillé le fonctionnement global de notre site et l'interconnexion de chacune de nos pages. Deux pages n'apparaissent pas dans ce diagramme : la page de validation du mail et la page de notification d'envoi du mail. Ces dernières n'ont pas une importance primordiale dans notre site. Par conséquent, nous ne les avons pas fait apparaître afin d'alléger le diagramme.

c. Mise en place du MVC

Nous avons décidé de mettre en place le patron d'architecture « Modèle Vue Contrôleur » pour notre site web. Ce type d'architecture est constitué de trois modules ayant chacun une responsabilité spécifique :

- Le modèle contient les données à afficher et effectue leur traitement.
- La vue contient la présentation de l'interface graphique.
- Le contrôleur contient la logique concernant les actions effectuées par l'utilisateur et possède un rôle d'intermédiaire entre les deux autres modules.

Cette architecture présente plusieurs avantages. Le premier est qu'elle permet une conception claire et efficace grâce à la séparation des données de la vue. Le deuxième est le gain de temps en cas d'évolution et de maintenance du site. En effet, le fait de ne pas inclure les données au sein de la vue permet de ne pas avoir à la modifier dans le cas où l'on voudrait changer la source de nos données (base de données vers XML par exemple). Enfin, elle permet une grande souplesse de développement puisqu'elle sépare le front-end (ce que voit l'utilisateur) et le back-end (ce qu'il ne voit pas). Ceci est un avantage lorsque l'on travaille à plusieurs sur un projet.

Le schéma ci-dessous résume le fonctionnement du MVC comme décrit précédemment :

Model-View-Controller

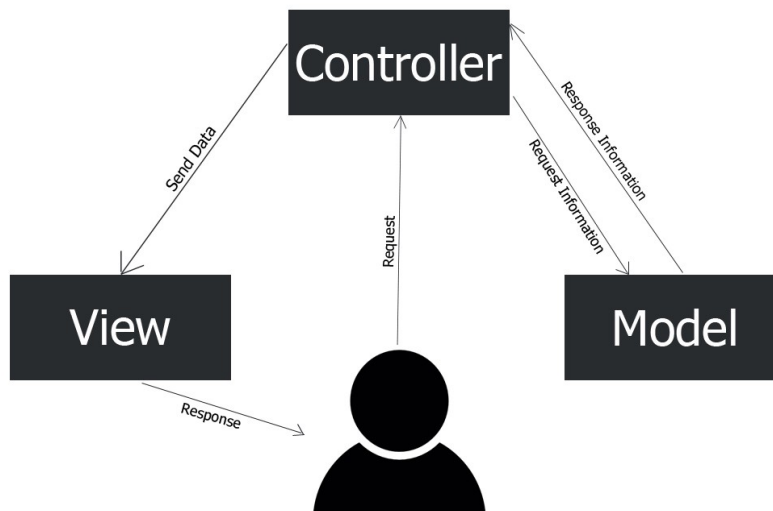


Figure 18 - Principe du MVC

d. Diagramme de packages

Nous avons décidé d'effectuer un diagramme de packages afin de présenter les relations entre les différents répertoires de notre projet.

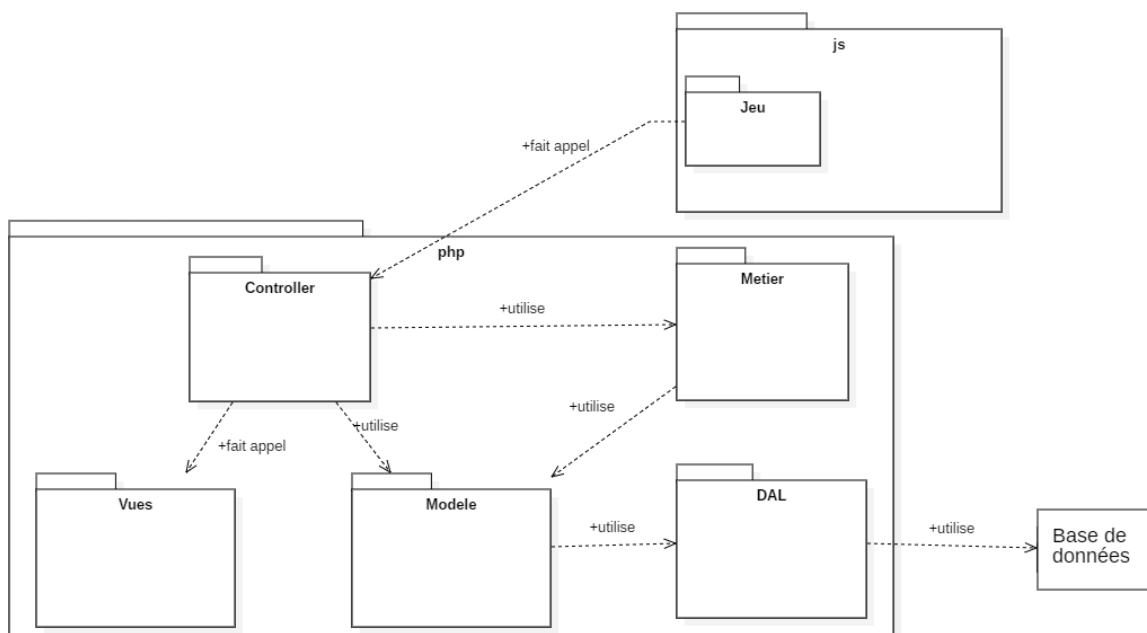


Figure 19 - Diagramme de packages

Dans un premier temps, nous pouvons remarquer la mise en place du MVC que nous avons évoqué précédemment. Les fichiers du répertoire « Controller » sont chargés de mettre à jour la vue en fonction de l'action de l'utilisateur et des données qu'ils ont récupérées via le modèle. Une description plus complète de ceux-ci sera faite plus tard dans ce rapport, dans la partie Développement. Les données manipulées par les contrôleurs sont récupérées par l'intermédiaire de la DAL (Data Access Layer). Cette dernière est chargée d'interroger la base de données pour récupérer

les informations nécessaires. Le « Modele » sert à effectuer des pré-traitements sur les données brutes récupérées par la BD (lever des exceptions, analyser la valeur de retour de la requête et en déduire quelque chose, ...). Il est utilisé par les contrôleurs pour retourner aux vues les données à afficher mais aussi par la partie « Metier » pour ce qui concerne le lancement et l'arrêt de la partie. L'avantage de passer par un modèle comme cela est que les contrôleurs et la partie « Metier » n'auraient pas à être modifiés si on venait à changer la base de données par exemple. Ceci est donc une autre forme d'application du MVC. Tous les packages évoqués précédemment sont placés dans le répertoire « php ». Ils sont exécutés côté serveur. Le package Jeu situé dans le répertoire « js » est chargé d'exécuter le jeu en lui-même côté client. Il fait appel au package « Controller » afin de mettre à jour sur le serveur les actions effectuées par l'utilisateur côté client. Encore une fois, cela est l'application du principe MVC car ce qui se passe côté client peut être assimilé au concept de vue. Ces scripts ne modifient pas directement le modèle mais passent par un contrôleur.

e. Diagramme de classe

Afin de ne pas surcharger notre diagramme de classe, nous avons décidé de ne représenter que la partie métier de notre application sous cette forme. En effet, puisque nous avons déjà détaillé les relations entre les différents packages dans la partie précédente, nous estimons qu'il n'est pas nécessaire de détailler chacune des classes créées dans ces packages à travers un diagramme de classes pour chaque. Concernant celui pour la partie « Metier » (le code du jeu donc), nous l'avons découpé en deux parties afin de rendre la description plus aisée et d'améliorer la lisibilité. Vous retrouverez le diagramme dans son intégralité en annexe. Voici la première partie :

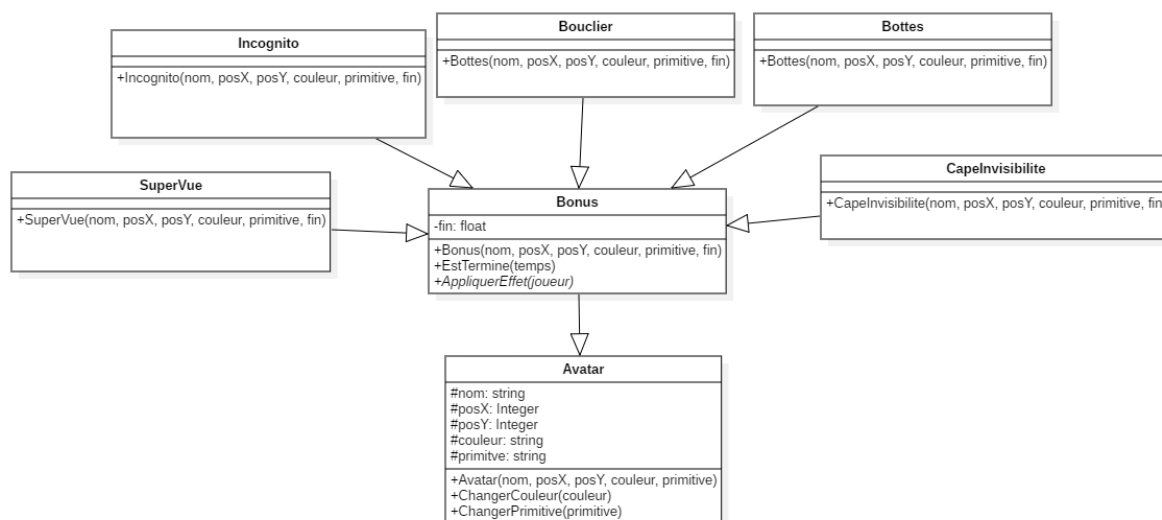


Figure 20 - Diagramme de classes, première partie

La classe Avatar correspond à tout objet pouvant être représenté dans notre jeu. Elle possède donc des attributs correspondant à sa position sur la carte : posX et posY. La classe Bonus dérive de cette classe puisque ceux-ci sont visibles dans le jeu. Elle possède une méthode abstraite appelée AppliquerEffet, qui sera à redéfinir dans toutes ses sous-classes. Cela va permettre de spécifier un effet différent pour chacun des Bonus (SuperVue, Incognito, Bouclier, Bottes et CapelInvisible). Cette modélisation est intéressante puisqu'elle est évolutive. En effet, dans le cas où nous souhaiterions ajouter un nouveau bonus dans notre jeu, il suffirait de le faire hériter de la classe Bonus et de spécifier la méthode AppliquerEffet lui correspondant. Cela permet donc de faire évoluer notre jeu très

rapidement, sans avoir à modifier tout le code, ce qui est un réel avantage. Passer par une classe et une méthode abstraite permet aussi de forcer le développeur à redéfinir certaines fonctions. Cela permet donc de lui offrir un cadre bien défini.

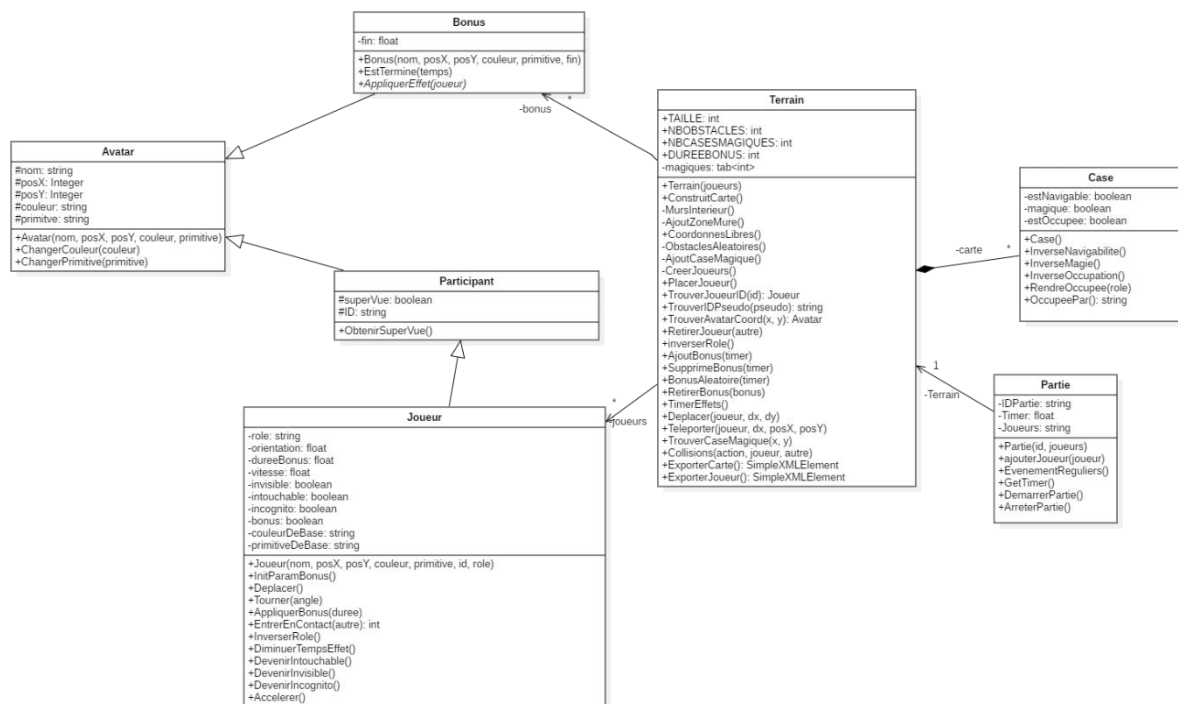


Figure 21 - Diagramme de classes, deuxième partie

Cette deuxième portion du diagramme de classes nous permet de compléter ce que nous avons pu dire sur le premier.

On retrouve ainsi ici la classe Avatar, qui comme tout à l'heure est une généralisation de tous les objets affichables à l'écran. Cette dernière a cependant ici une nouvelle classe fille, Participant, qui a elle-même une autre classe fille, celle qui représente les joueurs de la partie. Ce choix de modélisation devrait normalement vous interroger puisque la classe Participant est ici complètement inutile. Elle est en fait un reste de notre toute première modélisation, où nous avons prévu que la classe Participant ait en fait deux filles, Joueur et Spectateur. En effet, nous jugions à l'époque que le mode spectateur devait être géré depuis le serveur, et pas uniquement en Javascript comme nous l'avons fait finalement (voir partie Développement). Et pour le coup, un spectateur n'était rien d'autre qu'un joueur qui ne pouvait pas se déplacer. L'idée était alors de factoriser les points communs entre les deux classes dans leur classe mère Participant, puis après uniquement de laisser les informations propres à chacune dans leur classe respective. Après coup, cette idée a été abandonnée alors que le développement avait déjà commencé. Nous avons donc décidé de la laisser pour nous éviter des problèmes, quitte à avoir une classe inutile dans notre modélisation finale.

Concernant le reste du diagramme, la classe Joueur contient un ensemble de variables qui représentent l'état du joueur ainsi que des getters/setters. Elle contient également la fonction qui permet de gérer les collisions entre les joueurs ou avec les bonus.

La classe Terrain est la classe principale de notre jeu. C'est elle qui contient la liste des joueurs, des bonus, et bien sûr le terrain, qui n'est autre qu'une grille de cases. Elle contient l'ensemble des fonctions pour construire et instancier les listes précédemment citées. Son rôle est également de placer correctement les avatars qui la composent sur la grille. Deux fonctions supplémentaires sont

également présentes pour exporter les informations importantes de la classe au format XML, afin de communiquer avec les clients.

La grille est composée de cases, représentées par la classe Cases justement. Cette dernière possède plusieurs états possibles, décrits par ses variables, et un ensemble de getters/setters.

Enfin, on retrouve la classe Partie, qui est celle qu'on va instancier depuis l'extérieur pour lancer une partie. C'est elle qui contient le terrain ainsi que le timer. Elle possède également la liste des pseudos des joueurs inscrits à la partie, à la fois pour pouvoir la transmettre à la classe Terrain, qui va instancier un objet de type Joueur pour chaque, mais aussi pour vérifier si un joueur qui se connecte à la page de la partie est un joueur ou non (auquel cas il s'agirait d'un spectateur). Quant aux fonctions qui la composent, elles gravitent autour du timer et autour de l'évolution de l'état de la partie. La fonction GetTimer permet de décrémenter le temps restant et d'obtenir en retour une version XML décrivant l'état du timer, ainsi que celui des événements réguliers (les bonus) et des joueurs présents sur le terrain. Les fonctions DemarrerPartie et ArrêterPartie permettent de modifier la base de données pour refléter le statut de la partie en cours.

f. Schéma de la base de données

Après avoir analysé le sujet, nous avons déduit que très peu d'informations avaient besoin d'être stockées dans la base de données. En réalité, seules celles relatives aux utilisateurs doivent être enregistrées afin de pouvoir procéder à leur authentification, ainsi que celles liées aux parties en attentes et en cours de déroulement. Nous avons donc établi un schéma de données dans lequel nous ne faisons apparaître que les éléments que nous avons jugés nécessaires d'être stockés. Les **clés primaires sont soulignées** et les **clés étrangères sont en rouge**.

PARTIE(IDPartie, Nom, Createur, MaxJoueur, Type, Mdp, Duree, Statut, DateDebut, DateFin, Gagnants)

UTILISATEUR(pseudo, mail, mdp, isAdmin, cle, isActive)

PARTICIPE(IDPartie, IDJoueur, Couleur)

INVITATION(IDPartie, Expediteur, Destinataire, Date)

BANNIS(Mail, Raison, Date)

Quelques indications sur ce schéma :

- L'attribut Createur est une clé étrangère faisant référence au pseudo d'un utilisateur.
- Les attributs IDPartie et IDJoueur de la table PARTICIPE font référence aux colonnes du même nom dans les tables PARTIE et UTILISATEUR.
- L'attribut IDPartie de la table INVITATION fait référence à la colonne du même nom dans la table PARTIE. Les attributs Expediteur et Destinataire font eux référence au champ « pseudo » de la table UTILISATEUR.
- L'attribut Mail de la table BANNIS fait référence à la colonne du même nom dans la table UTILISATEUR.
- Le type de la partie peut prendre deux valeurs : « Privée » ou « Publique ».

- L'attribut Statut dans PARTIE peut prendre trois valeurs : « EnAttente », « EnCours » ou « Terminée ».
- L'attribut Gagnants correspond à la couleur des gagnants : « Red » ou « Blue ».
- La clé présente dans la table Utilisateur est la clé permettant la validation du compte par mail.
- L'attribut IsActive a pour valeur 1 si le compte a été validé par l'adresse mail, 0 sinon.
- La table PARTICIPE fait le lien entre les tables PARTIE et UTILISATEUR grâce aux clés étrangères IDPartie et IDJoueur.

Quelques captures d'écran de la base de données :

IDPartie	Nom	Createur	MaxJoueur	Type	Mdp	Duree	Statut	DateDebut	DateFin	Gagnants
1556822885	Test	AlexisGuyot	4	Publique		10	EnCours	02 05 2019 20:48		
1556824603	Test2	AlexisGuyot	3	Publique		5	EnCours	02 05 2019 21:16		
1556825016	Test3	AlexisGuyot	4	Publique		5	EnCours	02 05 2019 21:23		
1556825940	Test3	AlexisGuyot	3	Publique		5	EnCours	02 05 2019 21:39		
1556826195	Test4	AlexisGuyot	3	Publique		5	Terminee	02 05 2019 21:43	02 05 2019 21:49	egalite
1556826714	Test6	AlexisGuyot	5	Publique		3	EnCours	02 05 2019 21:51		
1556827430	Test7	AlexisGuyot	3	Publique		1	Terminee	02 05 2019 22:03	02 05 2019 22:05	rouges

Figure 22 - Table Partie

pseudo	mail	mdp	isAdmin	cle	isActive
AdminTest	admin-test@test.fr	\$2y\$10\$WLUgtFTronk8mF8VjMcx.tkN5Zr6o6/NW9RNcrARwn...	1	51f88b08410829705df90c22c3cb0c9a	1
Alexandre	alexandre_labrosse@etu.u-bourgogne.fr	\$2y\$10\$En1PAkQ5tiz6wpXiw0CCNeOwB4BE/1Ke7Lrg3z0d5f...	1	b1d1f7a8af82410bac7d958a8b5cd351	1
AlexisGuyot	alexis.guyot39@gmail.com	\$2y\$10\$ShCnJgG6d3G4W0dcOKzbt.ekik.Ux1FwuqxAf6Hs5UOo...	1	98e6261b887ae9df7a0dc80713e43427	1
AlexisGuyot2	ok-gio@gmail.com	\$2y\$10\$7J4mcbGBY8rtfV2PBF/eB38sCn73L.A2Y9Xq4uQ1G...	0	36bb940b9cea1685366b3c5f0a9a3e05	1
AlexisGuyot3	weshgleelimonfrere@gmail.com	\$2y\$10\$PUqdBclsj8nlp3fQB9Rg6OOiVblspQltCz0HSqn7Xkd...	0	09ec8d12f04ce1b923aea2b179f0504b	1
AlexisGuyot4	truc@machin.fr	\$2y\$10\$3HG9Ndvx5BSTbudRCg2OesFqgAj.Kw8rZV8mHghYIU...	0	8e81a41dfa65f30f2bb05e4f5722ffef	1
BlablaTruc	blabla@blabla.com	\$2y\$10\$C654Tzzsd/JIS2/mK8WFzOXfr9bnXnNRBDDCTIVQB3j...	0	4d64d8ede7af5c373d4cae314d407daa	0
Galvin71	galvin.bernard71@gmail.com	\$2y\$10\$EVFa.tqJkZ.JJOr0blob2beqmk0vp2tPjBHLAW/t0UGC...	0	39a1deb2b4e66c60c86da648cd566864	1
JoueurTest	joueur-test@test.fr	\$2y\$10\$8h4oGEctdQ3x4PUA.KJFxeKSnwcQnhJl8g71K.dmk9...	0	ea6294d84f9c017c398287e5fefe29bf	1
KijlodeYOP	matthieu.lefrancoise@hotmail.fr	\$2y\$10\$Em7Pgbzpr53XrzEEqpN8ge1UYDovAp6bnPmtu3T/7/A...	0	db52498a5c03e57f7deba507d3c20726	1
testprojet	test@yopmail.com	\$2y\$10\$IIDEsIDNxcxexFFCcn.deQQsv9HKCPTTs62.UD6Bx...	0	4c4b93b847c5af236a3e99e9d4204307	1
testtest	testprojetweb13@gmail.com	\$2y\$10\$OG.rS6f1bgqC4UByrnLbes7UCHEvMas1xixal6d6vv...	0	7106b502bfa0ff8e6c4606f7cf8996b0	1
testtimo	timothe.batut98@gmail.com	\$2y\$10\$6k4xp5D.JWUpuaRWFTY/3OoQy5clPIQu2EpxRA1fuKD...	1	5272a999ab8054ed579478c39616869f	1

Figure 23 - Table Utilisateur

IDPartie	IDJoueur	Couleur
1556822885	AlexisGuyot	red
1556822885	AlexisGuyot3	red
1556822885	AlexisGuyot4	blue
1556824603	AlexisGuyot	red
1556824603	AlexisGuyot3	red
1556824603	AlexisGuyot4	blue

Figure 24 - Table Participe

IDPartie	Expediteur	Destinataire	Date
1556828219	AlexisGuyot	AdminTest	02 05 2019 22:17

Figure 25 - Table Invitation

Mail	Raison	Date
machintruc@truc.fr	Gratuit	02 05 2019 12:14
timothe.batut98@gmail.com	Gratos	27 04 2019 16:19

Figure 26 - Table Bannis

IV. Développement

Nous allons dans cette partie détailler la façon dont nous avons utilisé les différentes notions vues en cours de Web ce semestre pour notre projet, les principales difficultés rencontrées et finalement nous terminerons avec quelques idées d'améliorations possibles, notamment en ce qui concerne la sécurité de notre site.

a. Chargement automatique des classes

Afin de faciliter le développement de notre site, de rendre notre code plus propre et d'éviter les effets de bord, nous avons décidé d'implémenter un autoloader qui a pour mission de charger automatiquement les différentes classes de notre projet. Son implémentation se fait en deux temps, et est répartie dans les fichiers **Autoload.php** et **config.php**, tous deux disponibles dans le dossier **php/Config**.

Dans un premier temps, nous avons chargé les chemins vers toutes nos classes dans la variable globale **\$classes** de **config.php**. Pour cela, nous avons utilisé la fonction **scandir** de PHP, qui retourne un tableau contenant les noms de tous les fichiers et répertoires trouvés au chemin passé en paramètre. Nous avons fait cela pour tous nos répertoires contenant du code PHP, à savoir **Modele**, **Metier**, **Controller**, **DAL** et **Autre**. Pour savoir où trouver ces scripts par rapport au fichier de configuration, nous avons utilisé la constante magique **__DIR__**, qui contient le nom du dossier où se trouve le fichier courant. Nous avons stocké le chemin relatif vers le répertoire où trouver les fichiers PHP dans une autre constante globale, **\$rep**.



```
3 $rep=__DIR__.'../../';
```

Figure 27 - Récupérer le chemin vers le dossier « php » du projet (config.php)

```

19 //Tableau associatif de classes : clé : nom de la
20 //Pareil que pour Metier
21 foreach (scandir($rep.'Modele/') as $value)
22 {
23     if($value!='.' && $value!='..')
24     { $classes[$value]='Modele'; }
25 }
26 //On parcourt les fichiers et dossiers du réperto
27 foreach (scandir($rep.'Metier/') as $value)
28 {
29     //On ne prend pas en compte les répertoires
30     if($value!='.' && $value!='..')
31     //On ajoute au tableau le nom de la classe en
32     { $classes[$value]='Metier'; }
33 }
34 //Pareil que pour Metier
35 foreach (scandir($rep.'Controller/') as $value)
36 {
37     if($value!='.' && $value!='..')
38     { $classes[$value]='Controller'; }
39 }
40 foreach (scandir($rep.'DAL/') as $value)
41 {
42     if($value!='.' && $value!='..')
43     { $classes[$value]='DAL'; }
44 }
45 foreach (scandir($rep.'Autre/') as $value)
46 {
47     if($value!='.' && $value!='..')
48     { $classes[$value]='Autre'; }
49 }

```

Figure 28 - Récupérer les chemins vers toutes les classes du projet (config.php)

Dans un second temps, dans **Autoload.php**, nous avons créé une classe statique (et donc accessible de n'importe où sans instantiation) contenant deux procédures de chargement. La première, `charger()`, utilise simplement la fonction `spl_autoload_register` de PHP qui permet de définir une surcharge à appeler à chaque fois que la fonction magique `__autoload` est appelée au chargement d'une classe. Ainsi, à chaque chargement, toutes les classes appelleront la procédure **chargementClasses** de l'autoloader, qui se contente de récupérer les chemins vers les classes récupérés précédemment et d'effectuer l'inclusion nécessaire.

```

12 class Autoload
13 {
14     public static function charger()
15     {
16         //On spécifie que la fonction chargementClasses est celle
17         spl_autoload_register(self::chargementClasses(), false);
18     }
19
20     public static function chargementClasses()
21     {
22         //On récupère les variables globales rep et classes spéci
23         global $rep;
24         global $classes;
25         //Pour chaque classe, on inclue le fichier
26         foreach ($classes as $k => $v){
27             $file=$rep.$v.'/'.'$k';
28             if (file_exists($file))
29             {
30                 include_once $file;
31             }
32         }
33     }
34 }

```

Figure 29 - Définition de l'autoloader (autoloader.php)

b. Echanges de données : SimpleXML et JSON

Pour notre projet, nous avons décidé d'utiliser le format XML comme support dans l'échange de données du serveur vers le client. Pour créer nos fichiers XML à faire transiter entre les deux entités, nous avons utilisé le module SimpleXML de PHP. Celui-ci permet très simplement de créer une arborescence XML grâce à sa classe principale, **SimpleXMLElement**. A l'instanciation de celle-ci, on lui donne la racine de notre document, puis on lui ajoute simplement des fils grâce à la fonction **addChild** comme illustré ci-dessous.

```

456 /**
457  * Exporte une version XML du terrain
458  * @return \SimpleXMLElement
459  */
460 public function ExporterCarte(){
461     $c = new SimpleXMLElement('<carte/>');
462     for($i=0; $i<self::TAILLE; $i++){
463         for($j=0; $j<self::TAILLE; $j++){
464             $case = $c->addChild('case');
465             $case->addChild('x', $i);
466             $case->addChild('y', $j);
467             $case->addChild('navigable', boolval($this->carte[$i][$j]->estNavigable()) ? 'O' : 'X');
468             $case->addChild('magique', boolval($this->carte[$i][$j]->estMagique()) ? 'M' : 'O');
469             $case->addChild('occupee', $this->carte[$i][$j]->OccupeePar());
470         }
471     }
472     return $c;
473 }
474
475

```

Figure 30 - Exportation du terrain au format XML (Terrain.php)

Dans cette fonction, on crée une arborescence représentant la carte du jeu, en indiquant pour chaque case ses coordonnées, si elle est magique, si elle est navigable et si elle est occupée. L'objet retourné est de type **SimpleXMLElement**, et il suffit ensuite dans le contrôleur d'appeler la fonction **asXML()** de ce dernier pour obtenir la version chaîne de caractères de l'arborescence, et ainsi pouvoir la renvoyer au client.

```

49
50  /**
51   * Retourne une version XML du terrain
52   */
53  function GetTerrain() {
54      header("Content-type: text/xml");
55
56      $id = "P".$REQUEST["IDPartie"];
57      echo $_SESSION[$id]['partie']->GetTerrain()->ExporterCarte()->asXML();
58  }
59

```

Figure 31 - Envoi de l'arborescence à la vue par le contrôleur (JeuController.php)

A noter qu'on indique tout de même dans l'en-tête de la réponse qu'il s'agit d'un document XML, afin de pouvoir la traiter plus facilement en Javascript à la réception. Nous avons créé en tout deux autres fonctions dans le genre afin de retourner l'état des joueurs et l'état de la partie. Une fois la réponse reçue par le Javascript à travers AJAX, que nous verrons dans une partie suivante, nous l'avons traitée de la façon suivante. Tout d'abord, nous récupérons sous la forme d'un tableau l'ensemble des cases, en récupérant toutes les balises qui portent le nom « case », grâce à la fonction **getElementsByTagName**.

```

158  creerTerrain(reponse.documentElement.getElementsByTagName("case"));

```

Figure 32 - Récupération des toutes les balises "case" (communicationServeur.js)

A partir du tableau obtenu, on traite chaque case en récupérant, toujours avec la même fonction, les champs qui nous intéressent.

```

63  for(var i=0; i<cases.length; i++){
64      //On récupère les données du XML
65      x = cases[i].getElementsByTagName("x")[0].textContent;
66      y = cases[i].getElementsByTagName("y")[0].textContent;
67      navigable = cases[i].getElementsByTagName("navigable")[0].textContent;
68      magique = cases[i].getElementsByTagName("magique")[0].textContent;

```

Figure 33 - Récupération en Javascript des attributs d'une case (affichageThreeJS.js)

Ces traitements ont lieu respectivement dans les fichiers Javascript **js/Jeu/communicationServeur.js** pour le premier et **js/Jeu/affichageThreeJS.js** pour le deuxième.

Pour varier un peu et pour nous faire manipuler d'autres concepts, nous avons également utilisé un peu de JSON pour faire transiter un tableau PHP récupéré d'une requête à la base de données à un de nos fichiers Javascript (**admin.js**). Pour faire cela, nous avons utilisé la fonction PHP **json_encode** qui exporte un tableau passé en paramètre en structure JSON. Nous avons notamment utilisé cette méthode dans la fonction **GetAdherents** du fichier **php/Controller/JSController.php**. Le JSON a l'avantage par rapport au XML d'être encore plus facile à manipuler en Javascript. En effet, ce dernier se manipule comme des tableaux, comme on peut le voir dans cet exemple tiré du fichier **js/admin.js** (tuple représente une ligne du tableau retourné par la requête et le nom entre crochets l'intitulé de la colonne souhaitée).

```

38 function creerTR(tuple){
39     var tr = document.createElement("tr");
40     var pseudo = creerTD(tuple['pseudo']);
41     var mail = creerTD(tuple['mail']);
42     var cb = creerCheckBox(tuple['pseudo']);
43     tr.appendChild(pseudo);
44     tr.appendChild(mail);
45     tr.appendChild(cb);
46     return tr;
47 }

```

Figure 34 - Manipulation d'un fichier JSON en Javascript (admin.js)

c. Partage de données entre utilisateurs : sessions et cookies

Pour faciliter le partage de données entre les différents utilisateurs du site, nous avons décidé de faire en sorte que toutes les personnes identifiées sur le site (qui ont donc validé leur connexion) partagent la même session, du nom de « ProjetWebL3 ». Ainsi, il est beaucoup plus simple de les faire partager des variables communes.

Avant l'authentification, un utilisateur possède une session qui lui est propre, générée par PHP. L'identifiant de cette session lui sert d'identifiant tant qu'il n'est pas connecté (capture provenant de `accueil.php`).

```
setcookie("IDJoueur",session_id());
```

Figure 35 - On laisse PHP ouvrir une session pour l'utilisateur non-connecté et on stocke dans son cookie l'identifiant utilisé (`accueil.php`)

Par la suite et dès sa connexion, son identifiant est modifié pour correspondre à son pseudo sur le site, et la session commune lui est imposée pour remplacer celle que PHP lui a attribuée. Ces modifications sont faites dans le fichier `index.php` afin que la session soit effective partout sur le site (sauf sur la page d'accueil où l'on recrée une autre session comme expliqué précédemment).

```

9     session_destroy();
10    session_id("ProjetWebL3");
11    session_start();

```

Figure 36 - Changement de session après l'authentification (`index.php`)

Les identifiants des joueurs ne sont cependant pas stockés dans la variable de session commune mais dans ses cookies, qui lui sont pour le coup propres. Il est ainsi facilement accessible par le champ « IDJoueur », aussi bien en PHP qu'en Javascript (ce qui est très utile pour le jeu). Pour créer le cookie, nous avons bien évidemment utilisé la fonction `setcookie`, comme nous pouvons le voir sur une des captures ci-dessus. Cette dernière correspondait à la mise en place du cookie pour un utilisateur non-connecté, mais le fonctionnement est identique après l'authentification, l'identifiant devenant juste le pseudo de l'utilisateur plutôt que son identifiant de session.

A partir de là, chaque utilisateur possède un morceau de la variable de session, identifié par son pseudo stocké et récupéré à partir du cookie. Dans cette partition, on peut alors placer des

messages d'erreurs ou des variables. On simule ainsi le fonctionnement d'une session personnelle d'un utilisateur tout en gardant le système de session globale.

```
$_SESSION[$_COOKIE["IDJoueur"]]['erreur'] = $e->getMessage();
```

Figure 37 - Chaque utilisateur possède un morceau de la variable de session générale à son nom (UserController.php)

Les instances des parties sont également stockées dans la variable de session, avec une partition pour chaque comme pour les utilisateurs. Tout l'intérêt de faire ce système de double session était là, car tous les utilisateurs identifiés sur le site peuvent récupérer facilement l'instance de la partie de son choix, pour pouvoir l'afficher et la modifier s'ils sont joueurs, ou juste la regarder s'ils sont spectateurs.

```
$_SESSION["P".$id]["partie"] = new Partie($id,$_COOKIE["IDJoueur"],$duree,$joueurs, new UserModel());
```

Figure 38 - Création d'une partie et stockage dans la variable de session (UserController.php)

Voici un schéma résumant la façon dont nous nous sommes servis des variables de session pour notre projet. Le grand rectangle représente la variable de session, chaque petit un morceau utilisé par un joueur ou une partie.

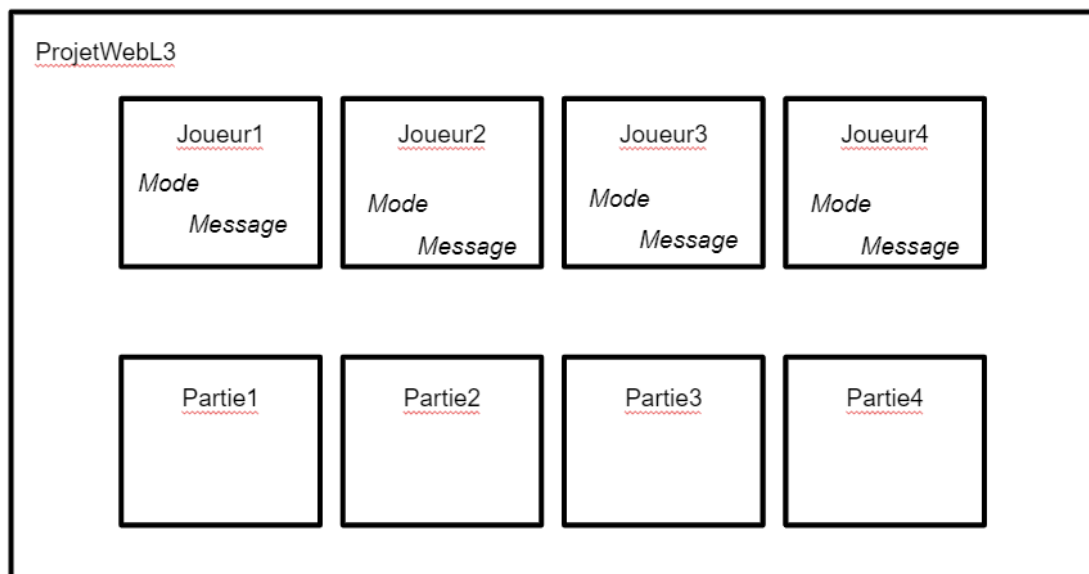


Figure 39 – Utilisation de la variable de session

d. Utilisation de la base de données : PDO

Afin de nous connecter à notre base de données MySQL, nous avons utilisé le module PDO de PHP. L'accès se fait en deux temps et à travers deux classes différentes : **UserGateway** et **UserModele**. La classe **UserGateway** est celle qui crée le moteur PDO et qui envoie et reçoit les requêtes SQL. La classe **UserModele** est celle qui est instanciée et manipulée par les utilisateurs, pour récupérer des données dans la BD. L'idée derrière le fait de créer puis de séparer ces deux classes était de séparer au maximum l'aspect récupération de l'aspect traitement des données.

A l'instanciation d'un objet de type **UserGateway**, la classe **Connection** du fichier **php/Autre/Connection.php** est appelée pour créer un objet PDO de connexion à une base. En cas de

changement pour un autre type de base (PostgreSQL par exemple), il suffirait donc de modifier cette dernière classe (ou d'en créer une similaire). Les paramètres de connexion sont stockés dans des constantes, déclarées dans le fichier **php/Config/config.php**.

```
define('DSN','mysql:host=db5000038371.hosting-data.io;dbname=db33334');
define("USER",'dbu78701');
define("PASSWORD",'ProjetWeb2019_');
```

Figure 40 - Paramètres de connexion à la BD (config.php)

```
class Connection
{
    public $dbh;

    function __construct()
    {
        try{
            $this->dbh=new PDO(DSN,USER,PASSWORD);
        } catch (Exception $e) {
            print "Erreur !: " . $e->getMessage() . '
            die();
        }
    }
}
```

Figure 41 - Connexion à la BD (Connection.php)

C'est avec cet objet PDO que nous communiquons avec notre BD, avec une requête préparée pour chaque besoin.

```
//Recherche si un utilisateur possède déjà l'adresse mail passée en paramètre
public function recherche_email($mail)
{
    $stmt = $this->connexion->dbh->prepare("SELECT * FROM UTILISATEUR where mail=:email");
    $stmt->bindParam(':email', $mail);
    if($stmt->execute()) {
        if ($stmt->rowCount() == 1) { return true; }
    }
    return false;
}

//Insert un utilisateur dans la base
public function insert_user($pseudo,$mdp,$email,$cle)
{
    $stmt = $this->connexion->dbh->prepare("INSERT INTO UTILISATEUR VALUES(:pseudo,:email,:mdp,0,:cle");
    $stmt->bindParam(':pseudo', $pseudo);
    $stmt->bindParam(':email', $email);
    $stmt->bindParam(':mdp', $mdp);
    $stmt->bindParam(':cle', $cle);
    if($stmt->execute()) { return true; }
    return false;
}
```

Figure 42 - Exemples de récupérations de données dans la BD (UserGateway.php)

e. Communications entre le client et le serveur : AJAX

Lorsque le client a besoin de récupérer des données sur le serveur, nous utilisons un moteur AJAX. Ceux-ci permettent de récupérer des données depuis un script PHP appelé à travers une requête GET ou POST envoyée en Javascript. Nous les avons notamment utilisés pour récupérer les données de la partie (le terrain, l'état des joueurs, les bonus, le timer, ...) au format XML, afin de pouvoir les afficher ensuite grâce à ThreeJS. Les requêtes AJAX ont toutes la forme suivante :


```

58 function getTerrain(){
59     objetXHR = creationXHR();
60     objetXHR.onreadystatechange = recupererTerrain;
61     objetXHR.open("get", "php/Controller/JeuController.php?IDPartie="+IDPartie+"&fct=GetTerrain", true);
62     objetXHR.send(null);
63 }

```

Figure 43 - Envoi d'une requête AJAX (communicationServeur.js)

L'envoi d'une requête AJAX respecte les étapes suivantes :

- Création du moteur.
- Attribution d'une fonction de callback à appeler quand un retour sera reçu (facultatif si la requête est une mise à jour à effectuer sur le serveur).
- Indication du script à contacter sur le serveur, avec éventuellement quelques paramètres passés en GET.
- Envoi de la requête.

Toutes les requêtes AJAX passent par un contrôleur (soit JeuController.php ou JSController.php, voir partie suivante), qui se charge de récupérer les données, ou de les mettre à jour, puis de répondre au client une fois cela fait. Cela nous permet ainsi de respecter le principe du MVC. La bonne fonction du contrôleur à appeler est spécifiée à travers le paramètre « fct ».

```

$fct = $_GET['fct'];
switch($fct){
    case "GetTerrain":
        GetTerrain();
        break;
    case "RecupererJoueurs":
        RecupererJoueurs();
        break;
    case "Timer":
        Timer($_GET['joueur']);
        break;
    case "SeDeplacer":
        SeDeplacer($_GET['id'], intval($_GET['dh']), intval($_GET['dv']));
        break;
    case "TournerJoueur":
        TournerJoueur($_GET['id'], $_GET['angle']);
        break;
    case "QuitterPartie":
        QuitterPartie($_GET['id']);
        break;
    case "GetID":
        GetID($_GET['pseudo']);
        break;
    case "TerminerPartie":
        TerminerPartie();
        break;
}

```

Figure 44 - Choix de la bonne fonction à appeler selon le paramètre passé par la requête AJAX (JeuController.php)

```

49
50  /**
51   * Retourne une version XML du terrain
52   */
53  function GetTerrain() {
54      header("Content-type: text/xml");
55
56      $id = "P".$_REQUEST["IDPartie"];
57      echo $_SESSION[$id]['partie']->GetTerrain()->ExporterCarte()->asXML();
58  }
59

```

Figure 45 - Exemple de fonction appelée par le Javascript (JeuController.php)

Dans les callbacks, on récupère ensuite les données retournées par le contrôleur à travers les champs `responseText` ou `responseXML`, en fonction du type de la donnée retournée, puis on effectue les traitements nécessaires. La condition au début de la fonction permet de lancer le traitement uniquement si l'état de la réponse est 4, soit réponse retournée, et le statut 200, autrement dit que la requête est un succès.

```

/* Récupère la réponse du serveur concernant l'id du joueur
 * @returns {undefined}
 */
function recupererID() {
    if(objetXHR.readyState===4 && objetXHR.status ===200){
        var response = objetXHR.responseText; //On récupère la réponse
        if(response !== null){ //Si elle n'est pas nulle
            if(response !== "spec") ID = response; //Si elle n'est pas "spec"
            else { ID = "0"; spectateur = true; } //Si elle est "spec"
            initJeu(); //On lance le jeu
        }
    }
}

```

Figure 46 - Récupération d'une réponse à une requête AJAX (communicationServeur.js)

f. MVC : Les différents contrôleurs

Nous avons tout au long de ce projet essayé de respecter le principe du patron de conception appelé MVC, qui consiste à séparer le stockage, le traitement et l'affichage des données à l'aide d'intermédiaires que sont les contrôleurs. L'entièreté de notre site communique à travers cinq contrôleurs, qui ont des tâches différentes. En effet, **AdminController**, **FrontController** et **UserController** sont liés et servent à l'utilisation générale du site et aux communications entre les scripts PHP sur le serveur, alors que **JSController** et **JeuController**, eux, permettent la communication entre les clients Javascript et le serveur. Nous allons détailler rapidement leurs rôles.

AdminController et **UserController** sont constitués et fonctionnent de la même manière. Les deux sont des classes PHP qui, quand on les crée, récupèrent le paramètre « action » de la requête http et appellent en conséquence la bonne fonction. Leur présence permet de n'utiliser en apparence que le fichier **index.php** pour la navigation sur notre site. En effet, c'est l'action précisée dans l'URL qui indiquera la bonne page à inclure, avec avant cela un certain nombre de traitements et de vérifications éventuellement, en fonction des besoins. La seule différence entre les deux est que **AdminController** contient les fonctionnalités uniquement accessibles aux administrateurs, alors que **UserController** contient celles accessibles à tout le monde. L'instanciation de ces contrôleurs est faite par le **FrontController**. Sa mission est de vérifier qu'il y a bien une action précisée dans la requête et que cette dernière existe dans les contrôleurs. Il possède pour cela la liste de celles qui sont possibles et il

crée, en fonction de s'il s'agit d'une action d'utilisateur lambda ou d'administrateur, le contrôleur adéquat. Si aucune action n'est demandée, son rôle est aussi d'en ajouter une pour renvoyer à la page d'accueil du site. Le **FrontController** est créé au chargement de **l'index.php**, juste après la création des sessions. Pour illustrer le fonctionnement expliqué dans ce paragraphe, nous allons détailler un exemple autour **d'AdminController**.

Dans notre exemple, un utilisateur qui est connecté sur notre site a appuyé sur le bouton « admin » de la page **principal.php**. S'il y a eu accès, il y a fort à parier qu'il est administrateur puisque le bouton n'est accessible qu'à ce type d'utilisateur. Cependant, il est intéressant de refaire une vérification. En cliquant sur ce bouton, l'utilisateur a validé un formulaire qui a pour action de renvoyer à la page « **index.php?action=ban_joueurs** ». Au chargement de la page, le **FrontController** va être créé et va constater que l'action demandée est « **ban_joueurs** ». En fouillant dans ses listes, il va voir qu'il s'agit d'une action d'administrateur, et va créer en conséquence un objet de type **AdminController**. Comme on peut le voir sur la capture ci-dessous, la création d'un tel objet va automatiquement lancer l'aiguillage vers la fonction adéquate, grâce à une structure « switch » sur la valeur de l'action. La fonction **bannir_joueurs** de cet objet va donc être appelée. Dans celle-ci, on va créer un **UserModele** pour pouvoir contacter la base de données et savoir si l'utilisateur est bien admin ou non (en récupérant son identifiant dans son cookie). Si oui, on affiche la page « admin ». Sinon, on affiche la page « principal ». Pour éviter d'avoir à tout recharger partout dans le code le chemin vers les différentes pages de notre site en cas de changement, nous avons déclaré dans le fichier **config.php** un tableau les contenant tous. Ce tableau est stocké dans la variable globale **\$vues**.

```

10  class AdminController
11  {
12      function __construct()
13      {
14          global $vues;
15          try {
16              $action = $_REQUEST['action'];
17              switch ($action) {
18                  case 'ban_joueurs':
19                      $this->bannir_joueurs();
20                      break;
21                  case 'valider_ban':
22                      $this->valider_ban_joueur();
23                      break;
24              }
25          }
26          catch(Exception $e) { }
27      }
28
29      /**
30       * Quand on clique sur le bouton "admin" de la page principale
31       * Accessible seulement pour un utilisateur qui est Admin
32       */
33      private function bannir_joueurs()
34      {
35          global $vues;
36          $m = new UserModele();
37          if($m->est_admin($_COOKIE["IDJoueur"])){ require($vues['admin']); }
38          else { require($vues['principal']); }
39      }

```

Figure 47 - Extrait du code de l'AdminController (AdminController.php)

Quant aux contrôleurs **JeuController** et **JSController**, la différence avec les autres est qu'ils ne sont pas dans des classes, mais simplement dans un script qui n'a donc pas besoin d'être instancié. En effet, comme précisé précédemment, leur rôle sera d'être accessibles aux moteurs AJAX, qui eux ne peuvent pas les instancier. Sinon le fonctionnement est le même, à savoir un switch qui choisit la bonne

fonction du script selon l'action demandée par l'utilisateur, ici précisée par le paramètre « fct » pour ne pas confondre avec les contrôleurs précédents. La différence entre les deux est purement organisationnelle puisque **JeuController** est le contrôleur appelé pour les fonctions qui concernent l'instance de la partie, alors que **JSController** est celui appelé par les autres scripts Javascript du site quand ils ont besoin de récupérer une donnée dans la BD par exemple.

g. Envoi de mails

Pour l'envoi de mails lors de la confirmation d'un compte, nous avons utilisé la fonction PHP **mail** dans la classe **EnvoiMail** du fichier **php/Autre/EnvoiMail.php**. Cette classe permet de construire tous les éléments nécessités par la fonction PHP pour envoyer un mail, à savoir un objet (appelé ici sujet), un entête et un message. L'adresse mail du destinataire est récupérée lors de l'inscription et la clé pour valider l'authentification est générée au même moment puis stockée dans la BD. A la réception de ce mail, l'utilisateur n'aura plus qu'à cliquer sur le lien qui va le ramener vers **l'index.php**, avec comme action celle qui va valider le mail et comme informations supplémentaires l'adresse mail et la clé. Le contrôleur va alors se charger de vérifier que toutes les informations sont conformes à ce qui a été enregistré dans la BD et va mettre à jour cette dernière pour activer le compte.

```

class EnvoiMail
{
    function __construct($destinataire,$cle)
    {

        $sujet = "Activer votre compte";
        $entete = 'From: noreply@projet-web-l3.fr' . "\r\n" .
            'Reply-To: noreply@projet-web-l3.fr' . "\r\n" .
            'X-Mailer: PHP/' . phpversion();

        //Sur le serveur
        $message = 'Bienvenue sur projet-web-l3.com,
            Pour activer votre compte, veuillez cliquer sur le lien ci dessous ou copier/colle
            http://www.projet-web-l3.fr/index.php?action=validation\_mail&mail=" .
            -----
            Ceci est un mail automatique, Merci de ne pas y répondre.';

        //En local
        /*$message = 'Bienvenue sur projet-web-l3.com,
            Pour activer votre compte, veuillez cliquer sur le lien ci dessous ou copier/colle
            http://localhost/projet\_web/Dev/index.php?action=validation\_mail&mail="
            -----
            Ceci est un mail automatique, Merci de ne pas y répondre.';*/

        return mail($destinataire, $sujet, $message, $entete);
    }
}

```

Figure 48 - Envoi d'un mail (EnvoiMail.php)

h. Vérifications Javascript et PHP

Deux niveaux de vérifications ont été mis en place pour vérifier les entrées des utilisateurs sur le site. Une première protection est fournie en Javascript pour le formulaire d'inscription de l'accueil (**accueil.js**) et celui de création de partie (**creation.js**). Pour le formulaire de la page d'accueil, du code Javascript permet d'analyser le mot de passe que l'utilisateur entre, pour en estimer sa force et le lui montrer visuellement grâce à la barre située au bas de la page. Cette force est estimée grâce à une correspondance avec du code regex qui permet de faire gagner des points au mot de passe entré. Un

point est gagné par le mot de passe dès qu'il contient une lettre minuscule, une lettre majuscule, un chiffre, un caractère spécial et quand le mot de passe entré pour la confirmation est identique. Cela pousse l'utilisateur à choisir un mot de passe plus sécurisé.

```
function checkPassword(){
    var password= document.getElementById("input_password").value;
    var password_conf= document.getElementById("input_password_conf").value;
    var strength = 0;
    if(password.match(/[a-z]{1,18}/)){strength += 1 ;}
    if(password.match(/[0-9]{1,18}/)){strength += 1 ;}
    if(password.match(/[A-Z]{1,18}/)){strength += 1 ;}
    if(password.match(/^(?=.*{8,20}$)(?=.*?[A-Z]){1,18})(?=.*?[a-z])(?=.*?[0-9]){1,18}).*$$/){ strength += 1
    if(password==password_conf){ strength += 1 ;}
    if(password.length==0){ strength=0;}
    changeBar(strength);
}
```

Figure 49 - Détermination de la force d'un mot de passe

Du côté du PHP, une vérification est faite sur le mot de passe, le pseudo et sur l'adresse mail avant la validation de l'inscription et l'ajout de l'utilisateur dans la base de données. Ces vérifications sont rangées dans la classe **Validation** du fichier **php/Autre/Validation.php**. Pour le pseudo, on vérifie qu'il contient de 8 à 20 caractères et qu'il est constitué uniquement de lettres minuscules ou majuscules, de chiffres, d'un point d'exclamation ou d'interrogation ou d'un tiret du haut ou du bas. Pour le mot de passe, on vérifie qu'il contient au moins une lettre majuscule, d'un chiffre et de lettres minuscules, le tout ne faisant pas moins de 8 caractères et pas plus de 20 non plus. Enfin pour le mail, on utilise la fonction **filter_var** avec la constante **FILTER_VALIDATE_EMAIL**, pour vérifier que la chaîne entrée par l'utilisateur ressemble à une adresse mail.

i. Création du jeu : ThreeJS

Nous avons utilisé ThreeJS pour construire notre environnement 3D à partir des données XML reçues du serveur grâce aux requêtes AJAX. Ce qui est important de comprendre avec le fonctionnement de notre jeu, c'est que l'ensemble du jeu tourne sur le serveur, et que la partie Javascript ne sert qu'à afficher l'état de la partie et à contacter le contrôleur via AJAX pour mettre à jour des données dans cette dernière (notamment la position et l'orientation du joueur).

Au niveau des spécificités de la scène, celle-ci possède un brouillard noir qui empêche le joueur de voir à plus de 3 cases de distance. Elle contient deux caméras, une contrôlée par le joueur pour faire de la première personne, et une deuxième qu'on active pour obtenir la super vue. Deux lumières éclairent la scène, un spot et une d'atmosphère.

Le terrain n'est récupéré et construit qu'une seule fois au début de la partie. Par la suite, seuls les joueurs et les bonus sont visuellement mis à jour sur celui-ci, respectivement toutes les 500ms et toutes les secondes. Le ThreeJS se contente dans un premier temps de créer le sol du terrain, qui est la traduction directe de la représentation 2D stockée sur le serveur. Il applique sur les cases navigables la texture par défaut et sur les magiques celle associée à cet état. Quand il croise une case non-navigable, il construit par-dessus un mur.

```

//Construction de la primitive qui va représenter le terrain
var solObjet = new THREE.BoxGeometry(1,1,1);

//Récupération des textures
var loader = new THREE.TextureLoader();
loader.crossOrigin = '';
var solTexture;
if(magique === 'M' && navigable === 'O') solTexture = loader.load("img/solmagique.jpg");
if(magique === 'O' && navigable === 'O') solTexture = loader.load("img/sol.jpg");
if(navigable === 'X') { creerMur(x,y); solTexture = loader.load("img/solmur.jpg");}
solTexture.wrapS = solTexture.wrapT = THREE.RepeatWrapping;

//Création du matériau qui va envelopper la primitive
var solMatériau = new THREE.MeshBasicMaterial({map: solTexture});

//Création de l'objet sol (combinaison de la primitive et du matériau)
var sol = new THREE.Mesh(solObjet, solMatériau);

//On place la case au bon endroit
sol.position.set(parseInt(x,10),-1,parseInt(y,10));

//On ajoute la case à la scène
scene.add(sol);
}

//On réeffectue le rendu de la scène après ajout du terrain
rafraichirScene();

```

Figure 50 - Construction du terrain (affichageThreeJS.js)

Chaque joueur possède une couleur et une primitive de base qui sert à identifier son appartenance à une équipe. Il y a deux équipes, les cubes rouges et les cylindres bleus. Il possède en plus de cela une autre couleur et une autre primitive qui sont la plupart du temps identiques à celles de base, mais qui peuvent varier à la récupération de certains bonus, la potion incognito notamment qui change la primitive pour un cône et la couleur pour du blanc. Ce sont cette deuxième couleur et cette deuxième primitive qui servent à construire le personnage dans la scène. Les joueurs ne se déplacent pas à proprement parler dans l'environnement 3D, ils mettent à jour leur position sur le serveur, changement qui sera répercuté 500 ms plus tard au nouvel affichage des joueurs. Quand l'affichage des joueurs tombe sur le joueur courant (identifié par son ID stocké dans son cookie), il ne crée pas de primitive mais déplace la caméra aux coordonnées du joueur et la tourne pour correspondre à son orientation enregistrée sur le serveur. Seule exception à cela quand l'utilisateur passe en super vue. Son personnage est alors représenté sur le terrain par sa primitive de base mais colorée en vert pour qu'il puisse savoir où il se trouve.

Il existe 5 bonus, qui sont tous représentés par des petites sphères qui flottent, comme les gomme dans le jeu Pac-Man. Ils possèdent chacun une couleur propre. Les bottes de sept lieux, qui augmentent la vitesse, sont la gomme noire. Le bouclier d'or, qui rend invulnérable, est la gomme blanche. La cape d'invisibilité est la gomme jaune. La potion incognito, qui rend la primitive et la couleur du joueur neutre, est la gomme orange. La potion de super vue, qui permet de changer la caméra pour un plan large, est la gomme violette. Chaque bonus reste 30 secondes sur le terrain avant de disparaître si personne ne le récupère.

Quand l'utilisateur possède le bonus de super vue et qu'il appuie sur la touche 'F', la caméra change pour un plan large en vue du dessus. Le brouillard est désactivé pour que l'utilisateur puisse voir. Tant qu'il est dans la caméra super vue, le joueur ne peut pas se déplacer.

Concernant le mode spectateur, un joueur est considéré comme tel dès qu'il possède dans son cookie un identifiant qui n'est pas enregistré comme joueur dans la partie sur laquelle il se connecte. L'identifiant du premier joueur créé lui est alors attribué et il reçoit sa vue. Il ne peut cependant pas mettre à jour la partie (se déplacer ou se tourner), juste recevoir les structures XML et donc les afficher

en ThreeJS. En cliquant sur les boutons au-dessus de la scène, le spectateur peut changer son ID provisoire et profiter de la vue d'un autre joueur, voire de la super vue s'il le souhaite.

Lorsque les rôles sont inversés (de temps en temps pendant la partie de manière aléatoire), un son est joué, grâce à ThreeJS et à la classe **AudioContext** de Javascript.

```
function jouerSon(){
    //On crée un contexte audio pour pouvoir jouer le son
    var context = new AudioContext();
    context.resume();

    //La caméra ThreeJS va jouer le son
    var listener = new THREE.AudioListener();
    camera.add(listener);

    //On crée puis charge puis joue le son
    var sound = new THREE.Audio(listener);
    var audioLoader = new THREE.AudioLoader();
    audioLoader.load("img/sound.mp3",function(buffer){
        sound.setBuffer(buffer);
        sound.setLoop(false);
        sound.setVolume(0.5);
        sound.play();
    });
}
```

Figure 51 - Création d'un son (affichageThreeJS.js)

j. Bugs connus

Malgré tous nos efforts pour essayer de rendre notre projet le plus fonctionnel possible, quelques bugs subsistent toujours alors que le temps nous est de plus en plus compté. Nous tenons alors à rajouter cette partie pour deux raisons. Tout d'abord, afin de nous servir de tableau de bord concernant ce qu'il nous reste à faire en termes de débogage. Et ensuite, si nous ne parvenons pas à tout corriger avant de rendre le projet, pour vous prévenir des quelques dysfonctionnements de notre projet. Voici donc la liste des bugs restants :

- A la connexion sur la page principale, les variables liées à la session du joueur (nom et privilèges) ne sont parfois pas chargées. Concernant le nom, cela n'est pas trop gênant puisque son utilité est purement esthétique sur la page (afficher le pseudo du joueur pour « personnaliser » l'expérience). Mais pour les privilèges, cela l'est un peu plus puisque si ceux-ci ne sont pas chargés depuis la BD, l'utilisateur est par défaut non-admin. S'il l'est normalement, il n'aura alors pas accès au bouton pour accéder à la page d'administration. Cependant, ce bug n'est pas si gênant que cela puisqu'il intervient seulement quand on passe de la page d'accueil vers la page principale (via la connexion donc). Cela signifie donc qu'il suffit de rafraîchir la page (passer de la page principale à la page principale) pour que tout se recharge correctement et que le bouton apparaisse (ou accéder à une autre page et revenir).

- Quand le créateur de la partie se déconnecte ou meurt, le timer s'arrête. Ce bug est un peu plus gênant puisqu'il n'est pas rare que le créateur soit éliminé de la partie lors de son déroulement. Si c'est le cas, cette dernière peut continuer et les joueurs peuvent toujours se dévorer entre eux, mais plus aucun bonus n'apparaîtra et la partie ne pourra pas se terminer tant qu'il restera au moins un joueur dans une des deux équipes. Concernant ce bug, on sait très bien d'où il vient. En effet, cela est dû à une mauvaise conception du système de timer dès le départ. Puisque PHP ne possède pas de fonction pour simuler un timer, nous avons utilisé Javascript et `setInterval` qui elle, le permet. C'est donc le créateur, depuis son client, qui à chaque fois qu'il essaye de récupérer la valeur du timer, le décrémente par la même occasion. S'il meurt ou se déconnecte, les communications avec le serveur sont coupées et la décrémentation ne se fait plus, coupant par la même occasion les événements aléatoires dans le temps (changement des rôles et bonus). Pour corriger le problème, il faudrait modifier une grosse partie de notre code, ce qui est toujours dangereux en fin de projet. Mais si on le faisait, il faudrait très certainement retirer ce système basé sur `setInterval` et passer plutôt par un Thread en PHP, qui serait créé au début de la partie et qui décrémenterait le timer jusqu'à ce qu'il atteigne 0.

k. Difficultés rencontrées

Au niveau des difficultés rencontrées pendant le développement, le plus difficile a surtout été de déterminer d'où venaient les problèmes liés à PHP. A part cela, nous avons rencontré quelques problèmes avec le côté asynchrone de Javascript et avec certains moments où il fallait bien comprendre la manière dont le client et le serveur communiquaient à différents endroits dans notre projet (pas toujours de la même manière d'ailleurs). Concernant ce point, une des difficultés a été par exemple de comprendre pourquoi nos classes, qui étaient pourtant auto-chargées au chargement de la page, n'étaient pas disponibles lors d'un appel AJAX. Il fallait alors comprendre que le moteur AJAX n'utilisait simplement pas **l'index.php** sur lequel était l'utilisateur, mais « une instance » différente et donc qu'il fallait recharger les sessions et l'autoloader. A part ces quelques points très spécifiques, les difficultés ont plus été de l'ordre conceptuel que portées sur le développement.

l. Améliorations possibles

A la fin du développement de notre projet, plusieurs idées d'améliorations nous restent quand même en tête. On souhaite alors en faire part dans cette partie du rapport.

D'abord, le site et le jeu mériteraient une bonne mise à jour concernant la sécurité. En effet, aucune protection n'a été mise en place contre de l'injection de code par exemple. Concernant la partie, il y a quelques failles de sécurité autour du mode spectateur. En effet, puisque ceux-ci empruntent l'identifiant d'un autre joueur pour obtenir leur vue, il suffirait que l'un d'entre eux passent la variable JS « spectateur » à false pour pouvoir usurper l'identité de quelqu'un, et ainsi déplacer son personnage à sa place. Il faudrait, pour empêcher cela, rajouter une protection du côté du serveur pour que seul un utilisateur dont l'ID stocké dans son cookie est enregistré dans la partie puisse utiliser les fonctions de déplacement ou de rotation.

A part la sécurité, il pourrait également être intéressant de donner la possibilité aux joueurs éliminés de la partie de pouvoir la regarder en tant que spectateur. Il pourrait être bien aussi de faire en sorte qu'une partie se termine automatiquement si plus aucun joueur n'est en train de jouer dessus.

V. Organisation

a. La méthode utilisée

Afin de mener à bien ce projet, et ce dès le moment où nous avons reçu le sujet, nous nous sommes organisés au travers de réunions régulières, si possible une fois par semaine, sinon une fois tous les 10-15 jours, afin de respecter le principe de cycles de la méthode agile Scrum. L'objectif de chaque réunion était de voir les avancées sur le projet concernant les tâches que nous nous étions réparties la réunion précédente, et de déterminer puis de nous répartir les tâches à effectuer pour la semaine d'après. Nous fournissons avec ce rapport un court descriptif de chaque réunion, pour vous donner une idée de la façon dont nous avons avancé sur le projet, et de la façon dont nous nous sommes réparti les tâches.

Réunion n°1 : 13 février 2019

Dès la distribution du projet par M.Gentil, nous avons décidé de mettre en place un système d'une réunion de groupe par semaine. Cette organisation nous a paru la plus optimale pour progresser et pour faire part de nos difficultés le plus régulièrement possible. La première réunion nous a surtout permis d'interpréter le sujet et de mettre en commun nos idées et remarques concernant la modélisation. C'est également lors de celle-ci que nous avons décidé des outils que nous allions utiliser par la suite. A la fin de la réunion, Alexis a délégué des tâches pour terminer la modélisation commencée pendant la séance (chacun devait finir la partie qu'il avait commencée pendant le TD qui a servi de première réunion).

Réunion n°2 : 27 février 2019

Pendant cette séance, nous avons mis en commun et corrigé les diagrammes Use Cases et le diagramme de classes produits pendant les vacances de février. Ensuite, nous avons réfléchi au diagramme de navigation de notre futur site, c'est à dire les liens entre les différentes pages. Nous avons réussi à le finir pendant la réunion, puis nous nous sommes de nouveau partagé le travail afin que chacun puisse commencer à développer une des pages trouvées pour le diagramme (ainsi qu'éventuellement un début de code Javascript pour l'accompagner). Nous avons suivi la distribution suivante :

PAGE HTML	ATTRIBUTION
ACCUEIL	Arnaud
ADMIN	Arnaud
CREATION	Timothée
PRINCIPAL	Alexandre
REJOINDRE	Alexis
ATTENTE	Galvin
JEU	Thinhinane

Réunion n°3 : 7 mars 2019

En raison des examens, nous n'avons pas pu nous voir pendant plus d'une semaine, ce qui explique le décalage de cette réunion. Durant cette dernière, nous avons surtout énormément réfléchi et débattu sur comment nous allions implémenter notre site puis notre jeu, notamment en ce qui concerne le stockage des données, les communications entre le serveur et le client ainsi que celles des clients entre eux. Concernant ce premier point, nous avons décidé après discussions d'utiliser plutôt des fichiers pour ne pas s'embêter avec une base de données. Pour le deuxième point, nous ne savions pas encore trop comment faire, nous avons alors décidé d'attendre que le cours de Web avance un petit peu plus pour aborder certains concepts qui nous paraissaient utiles (Ajax, PDO, sessions, ...). Pour finir cette réunion, nous avons essayé de tout de même réeffectuer un découpage en plus petites tâches de ce qu'il nous restait à faire vis-à-vis du sujet. A la fin de la séance, chacun devait réfléchir à ce qu'il avait envie de faire pour la suite et devait se renseigner sur les différents concepts qui nous manquaient.

Réunion n°4 : 21 mars 2019

Nous nous sommes retrouvés le 21 mars afin de faire le point sur les précédentes parties à attribuer à chacun. La solution de la création de compte grâce à un fichier a été remplacée après débat par une base de données, bien plus adaptée finalement. Nous avons également recommencé à répartir les tâches pour créer les 3 css demandés dans le sujet (Alexandre, Arnaud, Galvin) et le javascript permettant de vérifier les champs dans les formulaires (Thinhinane). Pour les autres membres du groupe, Alexis devait essayer de se renseigner sur le MVC et commencer à le mettre en place. Timothé, quant à lui, devait se renseigner sur un moyen d'obtenir une base de données et sur comment on allait s'en servir pour effectuer les authentifications/inscriptions. Ces étapes étaient obligatoires pour ensuite s'occuper de la partie jeu.

TACHE HTML	ATTRIBUTION
CSS PRINCIPAL	Alexandre
CSS SECONDAIRE 1	Arnaud
CSS SECONDAIRE 2	Galvin
MVC	Alexis
JAVASCRIPT	Thinhinane
BD	Timothé

Réunion n°5 : 4 avril 2019

Pour cette cinquième réunion, nous avons fait le point sur le serveur web fraîchement loué par Alexis pour accueillir notre projet, afin qu'on puisse tous y accéder de n'importe où et profiter d'une base de données. Il a, par la même occasion, montré ce sur quoi il avait travaillé en remplacement du fait qu'il avait du mal à mettre au point le MVC de notre projet : la synchronisation de différents clients grâce aux sessions PHP et à AJAX. A ce stade du projet, nous avons alors compris que notre site était, au niveau de l'apparence terminé, et que nous commençons à voir de plus en plus clair concernant ce que nous allions devoir faire pour le jeu. Nous avons alors commencé à nous répartir quelques

nouvelles tâches à son propos. Arnaud s'est trouvé en charge de commencer à modéliser un terrain en ThreeJS, Thinhinane devait terminer les quelques finitions restantes au niveau du Javascript, Alexandre s'est proposé pour faire le système d'authentification, d'inscription et d'envoi de mail et Alexis pour implémenter en pratique le diagramme de classes construit durant les premières réunions (classes qui allaient par la suite devenir la partie « Métier » de notre MVC, voir les parties [Modélisation](#) et [Développement](#) pour plus de détails). Pour prendre de l'avance, Alexis a demandé à Timothé de commencer ce rapport. Enfin, Galvin n'a pas reçu de nouvelle tâche pour ce cycle puisqu'il était exceptionnellement absent pour la réunion et qu'il n'avait pas complètement terminé son CSS du cycle précédent.

TACHE	ATTRIBUTION
TERRAIN THREEJS	Arnaud
FINITIONS JAVASCRIPT	Thinhinane
AUTHENTIFICATION	Alexandre
CLASSES « METIER »	Alexis
RAPPORT	Timothé
TERMINER CSS	Galvin

Réunion n°6 : 10 avril 2019

Lors de cette dernière réunion avant les vacances, nous nous sommes mis d'accord pour la création du jeu. Etant donné le temps qu'il restait et les indisponibilités de certains dues aux vacances et aux examens, la répartition des fonctionnalités paraissait compliquée. Alexis nous a alors proposé de s'occuper du jeu. Il a alors expliqué qu'il avait une idée assez précise de ce qu'il restait à faire après avoir fait la partie « Métier » du développement, et qu'en se basant sur le début de travail effectué par Arnaud sur le terrain et sur le diapo mis à notre disposition sur ThreeJS, il n'aurait pas trop de mal à terminer cette partie seul. Le reste du groupe a alors accepté et s'est trouvé en charge de devoir finir le rapport, à part la partie « Développement » qu'il allait faire une fois le développement terminé. Alexandre s'est également proposé pour mettre en place une bonne fois pour toutes un MVC propre pour faciliter le travail d'Alexis. La répartition des parties pour le rapport laissant Galvin sans travail, créer un template pour le diaporama de notre soutenance lui a alors été confié.

Réunion n°7 : 2 mai 2019

Lors de cette dernière réunion avant la soutenance, nous avons fait le point sur tout ce qui avait été fait depuis la dernière réunion. Pour cela, Alexis a fait une démonstration complète pour vérifier que toutes les fonctionnalités attendues dans le sujet étaient implémentées et fonctionnelles. Une fois cela fait, nous avons établi ensemble un plan pour notre soutenance, que nous avons ensuite équitablement réparti entre les différents membres du groupe. Nous avons alors distribué deux sous-parties par personne, et Alexis s'est proposé pour relire et corriger le rapport une fois avoir rajouté sa partie sur le développement.

Conclusion

Pour conclure sur cette partie concernant les réunions, nous pouvons la résumer en disant que nous avons essayé au mieux d'appliquer les principes fondamentaux de la gestion de projet agile, notamment ceux de la méthode Scrum. Pour cela, nous avons fait des réunions très régulièrement, une fois tous les 15 jours dans le pire des cas, pendant lesquelles nous avons à chaque fois fait le point sur ce qui avait été fait et sur ce qui restait à faire. Pour le travail restant, nous avons essayé à chaque répartition de le subdiviser pour obtenir au maximum des tâches simples, que nous pouvions ensuite nous déléguer le plus équitablement possible, tout en essayant à chaque fois de tirer profit des préférences et des compétences de chacun. A chaque réunion, le but était d'obtenir une version améliorée par rapport au travail effectué à la précédente, de manière à se rapprocher de l'objectif final de façon incrémentale. Ainsi, nous avons d'abord fait la modélisation, puis un squelette de site non-navigable, puis nous avons rajouté l'ergonomie (style, navigabilité, interactivité), puis l'authentification et enfin l'administration. Une fois notre site fonctionnel, nous avons ajouté le jeu et pour finir la gestion des parties. Tout le long du semestre, nous avons essayé au maximum d'effectuer une répartition équitable des tâches, même si la fin du projet et quelques imprévus nous auront forcés à dégrader à peine cette dernière pour tout pouvoir finir à temps.

b. Les outils utilisés

Comme dit précédemment, nous avons, lors de la première réunion, réfléchi à un ensemble d'outils à utiliser pour nous aider à nous organiser sur ce projet. Lors de notre réflexion, il nous a paru indispensable d'utiliser un outil pour mettre en commun notre travail, un pour communiquer, ainsi qu'un autre pour nous organiser et pour visualiser nos tâches.

Pour la mise en commun de notre travail, nous avons alors pensé aux logiciels « drive » et « git ». Alexandre étant très familier avec ce dernier, nous avons donc décidé de travailler avec. L'avantage de git est sa gestion des branches, qui peuvent nous permettre de travailler de notre côté sans affecter la branche principale du projet (appelée master). Un historique et une description des modifications effectuées est disponible, permettant ainsi un suivi de l'avancement du projet. Ces caractéristiques font de git l'outil idéal pour un projet collaboratif. Cependant et puisqu'il faut nuancer, un de ses défauts est qu'il demande une prise en main et une connaissance assez poussée des commandes à utiliser, contrairement à drive par exemple qui a une prise en main très rapide mais qui montre rapidement ses limites quant au suivi des modifications.

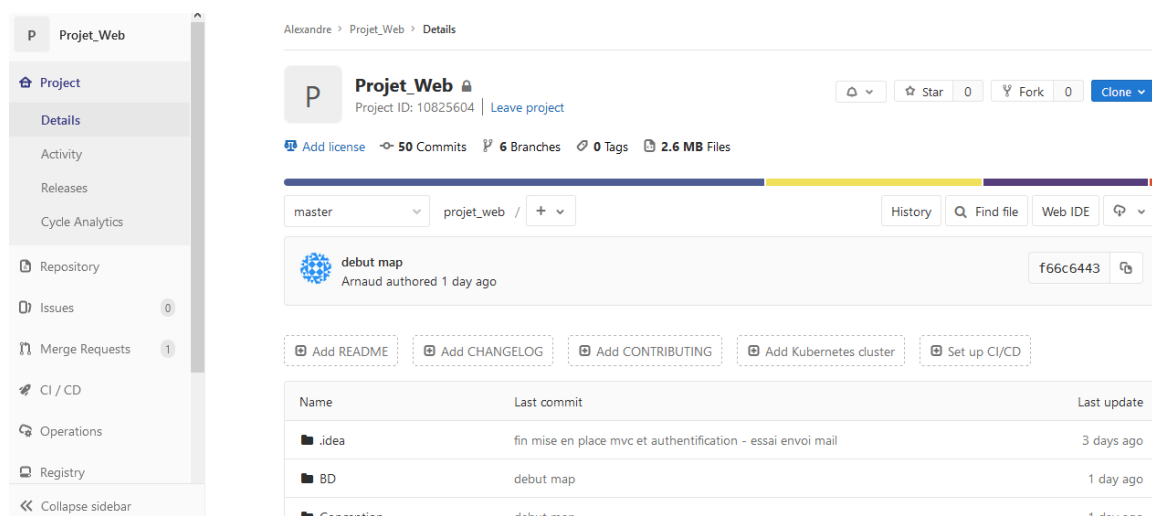


Figure 52 - GitLab

Un autre de nos besoins a été de trouver un moyen de s'organiser en classant les objectifs. Timothée nous a alors suggéré d'utiliser Trello, un site permettant de représenter les tâches sous forme de cartes à organiser comme bon nous le semble sur un tableau virtuel. Nous avons donc choisi cet outil qui nous permettait de mettre en œuvre efficacement la méthode Scrum, dont le cœur est souvent un tableau physique sur lequel on colle un post-it par tâche à effectuer.

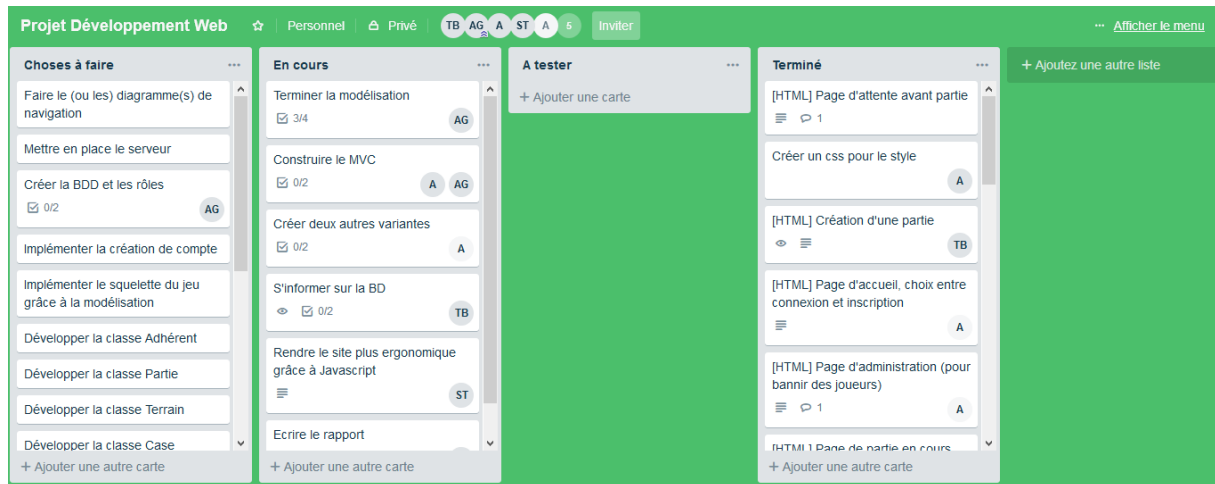


Figure 53 - Trello

Enfin, pour communiquer entre nous, l'application Messenger nous a paru être la plus simple et adaptée, puisque tous les membres de notre groupe étaient déjà inscrits sur Facebook, réseau social dont Messenger est la plateforme de messagerie instantanée.

Grâce à tous nos outils, nous avons pu mettre en place une organisation solide qui nous a permis d'avancer régulièrement et sans gros accrocs au cours de ce semestre.

VI. Conclusion

Dans ce rapport, nous avons exposé les étapes de conception et de développement de notre site web, qui consiste à gérer un jeu en ligne de type "Pac-Man". Notre problématique consistait donc à développer ce jeu, et pour atteindre cet objectif nous nous sommes servis des langages html, css, javascript et php.

Notre travail s'est déroulé sur plusieurs étapes. Dans un premier temps, nous avons commencé par une étude globale de la problématique et du cahier des charges. Puis, nous avons spécifié les besoins fonctionnels que devait respecter le site. Une fois ceux-ci fixés, nous avons enchaîné avec la conception de notre site en utilisant des diagrammes de cas d'utilisations, de classes et de navigation. Enfin, nous avons utilisé les différents concepts vus en cours durant l'implémentation de notre site et de notre jeu, le tout en essayant au mieux de respecter une organisation solide et équitable.

Ce projet était une véritable expérience de travail en collaboration, qui nous a permis d'apprendre à bien gérer la répartition des tâches et à renforcer l'esprit d'équipe et de partage de connaissances. Il nous aura donc permis de renforcer nos compétences techniques, mais aussi organisationnelles.

Annexes

Diagramme de classes complet

