

BAZIN Thomas
GALLAY Jules
GARNIER Aurélien
GUILLOTIN Louis
MAZERON Lucas
RAVIER Sébastien

RAPPORT
-
PROJET DÉVELOPPEMENT D'APPLICATION WEB
-
LES BLAIREAUX ET LES KÉKÉS



SOMMAIRE

- INTRODUCTION
- GESTION DES JOUEURS
- JEU
- GESTION DES PARTIES
- LIMITES ET CONCLUSION

I- Introduction

Le but de notre projet de développement Web était de réaliser un site internet proposant l'accès à un jeu accessible sur navigateur. Le jeu proposé se nomme « Les blaireaux et les kékés ». Il se présente comme un jeu par équipe, l'équipe des blaireaux et celle des kékés. Les blaireaux auront ici pour but d'éliminer les kékés, mais à intervalle régulier les rôles sont inversés et c'est aux kékés d'éliminer les blaireaux. Les différents joueurs auront la possibilité de bénéficier d'objets bonus répartis sur le plateau de jeu.

Pour réaliser ce jeu, il nous était imposé de travailler à l'aide de Three.js lors de la confection du jeu. De plus, le site devait proposer des protocoles de connexion et d'inscription pour accéder au jeu et permettre à certains utilisateurs d'avoir un statut d'administrateur. Nous avons immédiatement compris qu'il s'agissait ici de stocker les différentes informations concernant nos utilisateurs dans une base de données afin de rendre ces données persistantes. Le site devait comporter trois thèmes.

Concernant les contraintes liées intrinsèquement au jeu et à la manière d'y jouer, il était demandé de modéliser les joueurs sous forme de modèles géométriques simples, et d'ajouter des obstacles ainsi qu'un certain nombre d'objets bonus :

- une cape d'invisibilité rendant le joueur invisible pendant un temps donné,
 - des bottes doublant la vitesse du joueur,
 - un bouclier rendant le joueur invulnérable,
 - une potion rendant le joueur neutre,
 - une super-vue permettant au joueur, lorsqu'il appuie sur une touche, de prendre de la hauteur sur la map. Le joueur ne peut pas se déplacer pendant la super-vue.
- Il était également demandé l'ajout d'un mode spectateur.

En ce qui concerne la répartition des tâches, nous n'avons pas réalisé une répartition fixe et intransigeante des tâches, nous nous sommes tous naturellement tournés vers la partie avec laquelle nous étions le plus à l'aise, mais rien n'était attribué d'avance. Nous avons simplement créé un espace partagé via GitHub, sur lequel chacun d'entre nous a publié ses avancées sur le projet au fur et à mesure. Nous avons eu la chance de former un groupe suffisamment motivé, et où personne n'a délégué la totalité du travail à fournir aux autres membres du groupe.

Louis Jules et Aurélien se sont orientés dans un premier temps sur l'implémentation des pages PHP puis sur la base de données de notre jeu. Lucas, Sébastien et Thomas se sont eux plutôt orientés vers le Three.js et donc l'implémentation du jeu dans notre site.

Il est à noter qu'au fur et à mesure de l'avancement du projet, chaque personne s'est

permise d'apporter sa touche personnel dans les tâches polyvalentes: ainsi les pages PHP, HTML et CSS ont donc été faites par l'intégralité de notre groupes. Il en va de même pour le jeu.js.

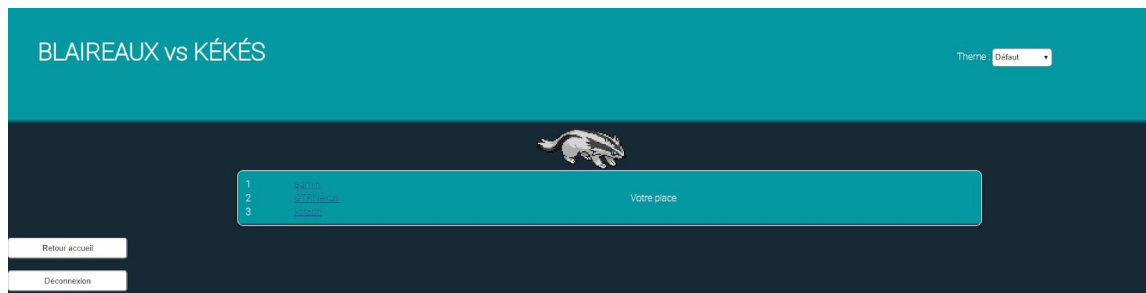
II- Gestion des joueurs

L'accès au site est disponible uniquement après que les utilisateurs se soient inscrits ou identifiés. Les données des utilisateurs sont stockées dans une base de données. Les données des joueurs sont enregistrées dans une table User comportant leur ID, leur login, leur mot de passe, leur score et leur classement. Cette table comporte un champ supplémentaire permettant de déterminer si l'utilisateur est un administrateur ou non.

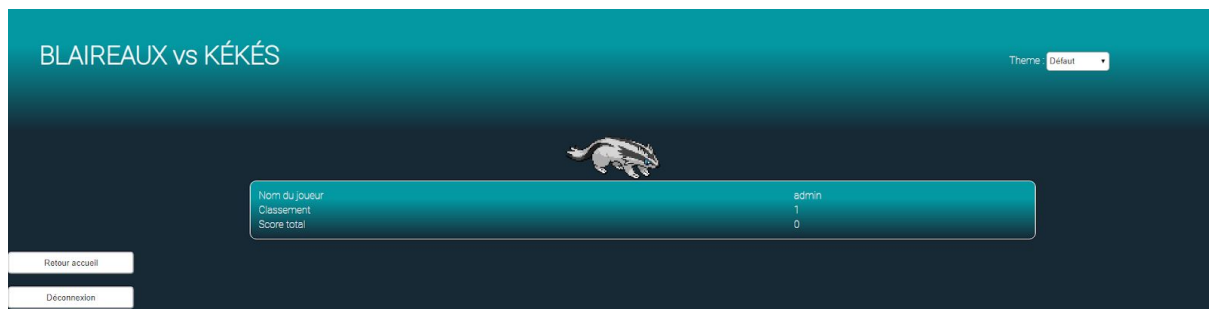
Lorsqu'un utilisateur est connecté, on crée une variable de sessions de type USER qui va contenir toutes les informations de l'utilisateur. Le type USER est défini grâce à la classe User.php, et un USER contient toutes les informations d'une ligne de la table User décrite précédemment.

Lors de l'inscription, la page de login va aller interroger la base de données pour vérifier que le pseudo n'est pas déjà pris, et si ce n'est pas le cas, le compte est créé. En ce qui concerne la connexion, la page va construire un objet USER avec les données présentes dans les champs login et mot de passe, puis elle appelle une fonction existe() sur cet objet. Cette fonction va aller interroger la base de données pour vérifier qu'un utilisateur avec le même login et le même mot de passe existe bien.

Le leaderboard, une page récapitulant les scores des différents joueurs enregistrés, peut être lui consulté en appuyant sur le bouton "Leaderboard" de la page principale. Le classement se présente alors sous la forme d'un tableau rempli automatiquement à l'aide d'un script PHP. On va chercher dans la table User de notre base de donnée le nombre d'éléments de cette table à l'aide d'une première requête via PDO, puis on réalise une seconde requête dans une boucle, qui permet de récupérer les joueurs dans l'ordre selon leur classement, et on les affiche dans un tableau HTML. On compare aussi le login de l'utilisateur de la session courante avec le login des utilisateurs dans la base de données afin d'indiquer à l'utilisateur où il se situe dans le classement. On notera cependant que la requête ainsi effectuée dans une boucle ne tolère pas que deux joueurs se placent ex-equo.



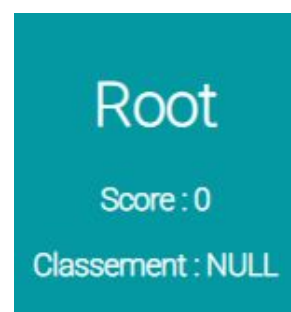
Depuis le leaderboard, il est possible en cliquant sur le pseudonyme d'un utilisateur dans le tableau de scores d'accéder à sa page profil, donnant des informations sur le score du joueur, son classement et son pseudonyme. Pour ajouter cette fonctionnalité nous avons fait en sorte que chaque pseudonyme du leaderboard soit encapsulé dans une balise `<a>` renvoyant lorsqu'on clique sur le pseudonyme à la page "profil.php?profil=pseudonymedujoueur". Ensuite, à l'aide de la variable globale `$_GET` on récupère le login, et on va chercher les informations correspondantes au login donné dans la table User de la base de données via une requête PDO. Il suffit une fois de plus ensuite d'afficher les informations récupérées dans un tableau HTML.



Sur la page principale de notre projet, nous pouvons y retrouver un onglet récapitulatif de l'utilisateur. Cette partie se fait assez simplement grâce à la classe User.php qui donne toutes les informations nécessaires sur les utilisateurs. Ainsi il a suffit de regarder l'utilisateur courant, et d'interroger la base de données sur celui ci afin d'afficher ses informations. (Schéma 1)

```
$user = $_SESSION['user'];
$login = $user->getLogin();
echo "<h1> $login </h1>";
echo "Score : ";
echo $user->getScore();
echo "<p></p>";
echo "Classement : ";
echo $user->getClassement();
```

(Schéma 1)



(Schéma 1)

A chaque instant, l'utilisateur en cours a la possibilité de se déconnecter. L'appui sur le bouton déconnexion appellera la fonction `deconnexion.php`, qui détruira la session actuelle et redirigera l'utilisateur sur la page de connexion. (Schéma 2)

```
<?php
    session_start();
    if(isset($_SESSION['user'])){
        session_destroy();
        header("Location: //localhost/BlaireauxEtKeke/pages/login.php");
    }
?>
```

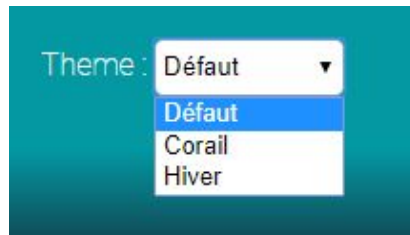
(Schéma 2)

A l'instar du bouton déconnexion, l'utilisateur pourra à tout moment retourner sur la page principale via le bouton "Retour accueil". Ce bouton appelle la fonction `RetourAccueil.php` qui récupère l'utilisateur courant et le redirige sur l'index de notre site. (Schéma 3)

```
<?php
    session_start();
    if(isset($_SESSION['user'])){
        header("Location: //localhost/BlaireauxEtKeke/index.php");
    }
?>
```

(Schéma 3)

On retrouve aussi sur chaque page de notre projet une liste déroulante permettant de changer le thème de la page courante lors de la sélection d'un thème de la liste. Pour changer de thème, on appelle une fonction JavaScript stockée dans un autre fichier, changeant la couleur de l'animation en arrière plan. Nous avons implémenté trois thèmes différents : un thème par défaut composé de bleu et noir, un thème hivernal, sobrement orienté vers le blanc et le noir, et un thème corail, beaucoup plus vif, teinté de rouge. Nous avons ensuite essayé de rendre le thème sélectionné persistant lors d'un changement de page au sein de notre projet, à l'aide de la variable globale `$_POST`, en plaçant notre liste déroulante dans une balise `<form method="post"></form>`, puis en appelant au chargement du HTML de la page suivante la liste déroulante avec l'option choisie à la page précédente présélectionnée grâce à une série de conditions PHP, puis appliquer le script JavaScript au chargement de la page. Nous n'avons pas réussi à effectuer ceci, car pour stocker des informations dans la variable `$_POST` il fallait un bouton de validation, qui remettait automatiquement à zéro le formulaire. Nous n'avons pas trouvé d'autre solution qui aurait pu nous permettre d'implémenter cette fonctionnalité.



Enfin, nous avons ajouté un onglet “Foire aux Questions” en bonus dans notre projet. Sur cette onglet nous disposons de 5 boutons, et le clic sur l’un de ces boutons fait appelle à une fonction JavaScript qui affiche la réponse correspondante au bouton.

III- Jeu

Le personnage

L’avatar utilisé par le joueur est représenté par un cube, dont la couleur détermine son appartenance à une équipe. Une caméra lui est également associée, celle-ci est placée au centre du cube afin de réaliser une vue à la première personne. De plus, un brouillard est ajouté à la scène afin de limiter le champ de vision du joueur à 3 cases autour de lui, comme explicité dans le sujet.

- Mouvements : déplacements et rotation

Les mouvements pouvant être effectués par le joueur sont gérés via la fonction `keyPressed`, prenant en compte les actions suivantes : faire avancer le personnage (flèche du haut), effectuer un quart de tour (flèches gauche et droite), utiliser un objet présent dans l’inventaire (flèche du bas).

Afin d’effectuer le déplacement vers l’avant, il a fallu tout d’abord créer une variable de direction, qui définit où regarde le joueur. Celle-ci prend les valeurs 0, 1, 2 et 3, pour respectivement gauche, haut, droite, bas, relatif à l’origine de la map. Cela a permis de savoir, en fonction de la valeur de direction, s’il fallait incrémenter ou bien décrémenter la position de 1, et s’il fallait le faire selon l’axe des X ou des Z.

Pour la rotation, le même mécanisme a été mis en place : appuyer la flèche gauche ou bien la droite affecte une valeur au sens de rotation, permettant de savoir si celle-ci doit être de 90° ou de -90° selon l’axe des Y. Il fallait également actualiser la direction regardée après avoir effectué cette rotation : valeur de direction ± 1 , modulo 4.

- Mouvements : animations

Les mouvements étaient fonctionnels mais leur réalisation était assez brutale rendant difficile l'orientation sur la map. Afin de les rendre plus progressifs dans le but de lisser ces déplacements et rotations, des vitesses de déplacement et de rotation ont été mises en place, correspondant à la valeur de mouvement par frame.

Ainsi, le personnage se déplace de 0,1 sur le plateau par frame, et la rotation se fait par intervalles de 9° par frame. Ces animations utilisent des compteurs par mouvement, permettant de les arrêter. Ainsi, l'animation continue (via requestAnimationFrame) tant que les compteurs de déplacement / rotation, qui s'incrémentent de 0,1 / 9°, n'ont pas atteint 1 / 90°. Ceci étant fait, nous avons immédiatement pensé à mettre en place des booléens d'animation, empêchant le joueur d'effectuer un nouveau mouvement tant qu'une animation était toujours en cours.

```
function move() {
    moving = true;
    if(direction == 0) {
        if(cube.position.x <= 0) {
            cube.position.x = mapWidth;
            cam.position.x = mapWidth;
        }
        cube.position.x -= vitDep;
        cam.position.x -= vitDep;
    }
    else if(direction == 1) {
        if(cube.position.z <= 0) {
            cube.position.z = mapHeight;
            cam.position.z = mapHeight;
        }
        cube.position.z -= vitDep;
        cam.position.z -= vitDep;
    }
    else if(direction == 2) {
        if(cube.position.x >= mapWidth - 1) {
            cube.position.x = -1;
            cam.position.x = -1;
        }
        cube.position.x += vitDep;
        cam.position.x += vitDep;
    }
    else if(direction == 3) {
        if(cube.position.z >= mapHeight - 1) {
            cube.position.z = -1;
            cam.position.z = -1;
        }
        cube.position.z += vitDep;
        cam.position.z += vitDep;
    }

    moveCpt += vitDep;
}
```

```
// corrige pb de precision
moveCpt = roundNumber(moveCpt,1);
cube.position.x = roundNumber(cube.position.x,1);
cube.position.z = roundNumber(cube.position.z,1);
cam.position.x = roundNumber(cam.position.x,1);
cam.position.z = roundNumber(cam.position.z,1);

if(moveCpt >= 1) {
    moving = false;
    moveCpt = 0;
    console.log(cube.position);
    console.log("\n");

    checkTeleportFlaque();
    checkCol();
    checkBonus();
}
else requestAnimationFrame(move);
}
```

```
// pour changement de direction
function rotate() {
    rot = true;
    if(rotDir == 0) {
        rotCpt += THREE.Math.degToRad(vitRot);
        cube.rotation.y += THREE.Math.degToRad(vitRot);
        cam.rotation.y += THREE.Math.degToRad(vitRot);
    }
    else {
        rotCpt += THREE.Math.degToRad(vitRot);
        cube.rotation.y -= THREE.Math.degToRad(vitRot);
        cam.rotation.y -= THREE.Math.degToRad(vitRot);
    }

    if(rotCpt >= THREE.Math.degToRad(90)) {
        if(rotDir == 0) direction = mod(direction-1,4);
        else direction = mod(direction+1,4);
        debugDir();
        rot = false;
        rotCpt = 0;

        checkCol();
    }
    else requestAnimationFrame(rotate);
}
```


- Mouvements : collisions

Nous avons également eu à gérer les collisions du joueur avec les autres éléments présents sur la map : les murs, les flaques, les objets et les joueurs. Ces entités seront présentées juste après.

Les murs étant présents dans le but de limiter le déplacement, nous avons utilisé un booléen blocked, qui se met à jour à la fin de chaque déplacement et rotation, pour empêcher le joueur d'avancer si un mur se trouve devant lui. Selon la position de l'avatar et la direction regardée, on détermine la case placée juste devant lui, puis l'on regarde si un mur est présent sur les mêmes coordonnées que cette case, et l'on ajuste la valeur de blocked en fonction du résultat.

```
// met à jour le booléen blocked
// en fonction de la présence ou non d'un mur face au joueur
function checkCol() {
    var p = new Point();
    switch(direction) {
        case 0:
            p.x = cube.position.x - 1;
            p.y = cube.position.y;
            p.z = cube.position.z;
            break;
        case 1:
            p.x = cube.position.x;
            p.y = cube.position.y;
            p.z = cube.position.z - 1;
            break;
        case 2:
            p.x = cube.position.x + 1;
            p.y = cube.position.y;
            p.z = cube.position.z;
            break;
        case 3:
            p.x = cube.position.x;
            p.y = cube.position.y;
            p.z = cube.position.z + 1;
            break;
    }

    blocked = false;
    listemurs.forEach(function(element) {
        if(p.x == element.x && p.z == element.y) {
            blocked = true;
        }
    });
}
```

Pour chacun des autres éléments, un évènement leur est associé et celui-ci se déclenche uniquement lorsque le joueur marche dessus. C'est pourquoi il suffit uniquement de vérifier si les coordonnées de l'objet et de l'avatar sont égales avant d'effectuer l'action associée.

Le plateau, les obstacles et les bonus

Le plateau de jeu (map) est caractérisé par sa taille, définie par des variables `mapWidth` et `mapHeight`, dont les valeurs sont choisies par l'hôte de la partie, lorsque celle-ci est créée. Ce terrain est limité en taille, mais ces bords se rejoignent de manière à former un tore. La sortie d'un joueur par l'un des côtés entraîne son retour par le côté opposé. Toutefois, le plateau est parsemé d'obstacles et de bonus positionnés aléatoirement. Ces éléments sont stockés dans des tableaux lorsque la map se génère et ne se superposent pas : une boucle parcourt chacun de ces tableaux afin de vérifier que les coordonnées aléatoires de l'entité ne soient pas déjà utilisées. Puis, de même pour le joueur, qui est placé au hasard sur le plateau.

Une fois la partie lancée, le joueur pourra se déplacer sur le terrain de jeu, en contournant les murs bloquant la route, jusqu'à découvrir les nombreux bonus. Le premier est un élément statique de la carte : la flaque magique. La puissante lumière violette émanant du sol téléportera forcément le joueur ailleurs, sur une autre flaque. Effectuer le chemin inverse ne permet pas obligatoirement de revenir au point de départ. À vos risques et périls ! D'autres objets sont disponibles, mais ceux-ci sont ramassables, c'est à dire que marcher dessus le retire de la map (`scene.remove(...)`) et l'ajouter à votre inventaire. Lorsqu'un joueur termine son déplacement, on vérifie si un objet se trouve sur sa case et si son inventaire est vide pour pouvoir le prendre. L'inventaire une simple variable stockant l'identifiant du type d'objet tenu. Il suffit d'appuyer sur la flèche du bas pour utiliser ce bonus.

Les bonus ramassables sont des objets dynamiques, animés et utilisables par le joueur. Chaque type d'objet possède un identifiant, permettant de savoir quelle action doit être effectuée. Le premier objet, représenté par un oeil, permet d'offrir au joueur le pouvoir de super-vue. Lorsqu'il est utilisé, les déplacements de l'avatar sont bloqués mais une vision globale du terrain lui est offerte. Ceci est réalisé à l'aide d'un booléen pour bloquer les mouvements, ainsi qu'une série de rotations afin de positionner la caméra dans les airs, orientée par rapport à la direction du joueur. Activer à nouveau le bonus permet d'annuler le pouvoir et se déplacer à nouveau. Le joueur pourra également trouver une botte ailée, lui offrant le pouvoir de super-vitesse pendant un temps limité. Activer ce pouvoir lance un compte à rebours, durant lequel la vitesse de déplacement et de rotation du joueur est doublée (les valeurs d'animation vues à la partie précédente). De la même manière, une cape offre au joueur le pouvoir d'invisibilité temporairement, les lunettes noires le rendent incognito, enfin le bouclier le rend invincible pendant un court instant. L'invisibilité place simplement le paramètre "visible" du joueur sur false, ne l'affichant plus par le `render`, et le pouvoir incognito lui change sa couleur. Les autres joueurs ne sauront donc plus à quelle équipe il appartient, et risquent même de le confondre avec un mur à cause de son camouflage. Ces bonus limités dans le temps ont une

durée gérée par des variables, et utilisent la fonction `setInterval`. Le temps d'utilisation restant est affiché en temps réel sur la page html. Une fois l'inventaire rempli, il n'est pas possible de ramasser d'autres objets tant que celui tenu par le joueur n'a pas été utilisé. Chaque objet possède une animation simple le faisant flotter dans les airs en tournant sur lui même, permettant d'être visible facilement.

```
// bottes de vitesse
if(objetTenu == 2) {
    if(bottesActives==0){
        bottesActives = 1;
        tempsActuelBottes = dureeBottes;
        vitDep *= 2;
        vitRot *= 2;
        boostBottes = setInterval(boostVitesse, 1000);
    }
}
```

```
function boostVitesse() {
    tempsActuelBottes--;
    if(tempsActuelBottes <= 0) {
        clearInterval(boostBottes);
        boostBottes = null;
        vitDep /= 2;
        vitRot /= 2;
        objetTenu = 0;
        bottesActives = 0;
        tempsActuelBottes = dureeBottes;
        document.getElementById("nbBonus").innerHTML = "Rien";
    }
}
```

Multijoueur et changement d'équipe

Au moment de rejoindre une partie, une requête ajax va appeler un script PHP permettant de récupérer la localisation des murs, des flaques et des autres joueurs (de la partie) dans la base de données. Une fois la partie lancée, la page se recharge à intervalle de temps régulier pour que les déplacements des joueurs soit pris en compte. Dès qu'un joueur se déplace, sa position est mis à jour dans la base de données.

Lorsqu'une partie commence, les joueurs sont répartis en deux équipes : les Kékés et les Blaireaux. L'une des deux équipes possède la capacité de manger les membres de l'autre, mais ceci est temporaire. En effet, un gong retentit de temps en temps, permettant d'inverser les équipes. Chaque joueur voit sa couleur changer, excepté pour ceux sous l'effet du bonus incognito, qui ne changeront de couleur qu'à la fin de l'effet (toutefois, leur équipe change quand même).

Ce gong est géré comme les bonus à temps limité : un temps restant, décompté chaque seconde pendant la partie. Lorsque ce compte à rebours atteint zéro, les changements d'équipes à opérer sont effectués et le décompte se relance, ceci se répétant jusqu'à la fin de la partie. Une partie est terminée une fois que le temps est écoulé, déterminé par un compteur diminuant en parallèle à celui du gong, ou bien lorsqu'une équipe a été éliminée.

Lorsque deux joueurs se rencontrent sur une même case, l'un des deux est éliminé. En effet, si celui qui vient d'arriver sur la case est "méchant", il mangera l'autre joueur, même s'il s'agit de son allié ! En revanche, si le joueur est "gentil" et rencontre un méchant, il se fera manger. Ces cas-là s'appliquent uniquement si le joueur en position de faiblesse ne possède pas d'invincibilité via le bonus de bouclier. À partir de là, celui qui a été dévoré a perdu. Il ne joue plus, est retiré de la liste des joueurs et passe en mode spectateur. Lorsqu'une équipe n'a plus de joueur, la partie est terminée.

Mode spectateur

Lorsqu'un joueur est mangé mais que la partie n'est pas terminée, celui-ci passe en mode spectateur. Il est également possible pour un utilisateur externe à la partie de visionner le déroulement de celle-ci via ce mode. Le spectateur se voit proposer deux façons d'observer la partie : le mode caméra libre, et le mode caméra de joueur. En caméra libre, l'utilisateur est placé en vue aérienne, de façon similaire à la super-vue, mais peut se déplacer librement avec les flèches directionnelles dans les quatre directions. S'il souhaite entrer dans le feu de l'action, il peut observer la partie à travers la caméra des joueurs. Pour cela, il lui suffit de passer dans ce mode à l'aide de la touche A ou Z, puis d'utiliser ces touches pour naviguer à travers la liste des joueurs. Ces touches servent respectivement à passer au joueur précédent et au joueur suivant. Appuyer sur la touche espace lui permet de repasser en caméra libre.

Pour mettre en place cette fonctionnalité utilisant beaucoup les changements de caméra, une caméra active est définie, et est initialement celle du mode libre. Lorsque le spectateur observe un joueur, la caméra de dernier est utilisée en caméra active, et est remplacée à chaque fois qu'il change de joueur.

Difficultés et améliorations

Nous voulions ajouter des textures au jeu via le TextureLoader afin de donner plus de vie aux parties, mais il pouvait arriver que la sécurité de certains navigateurs

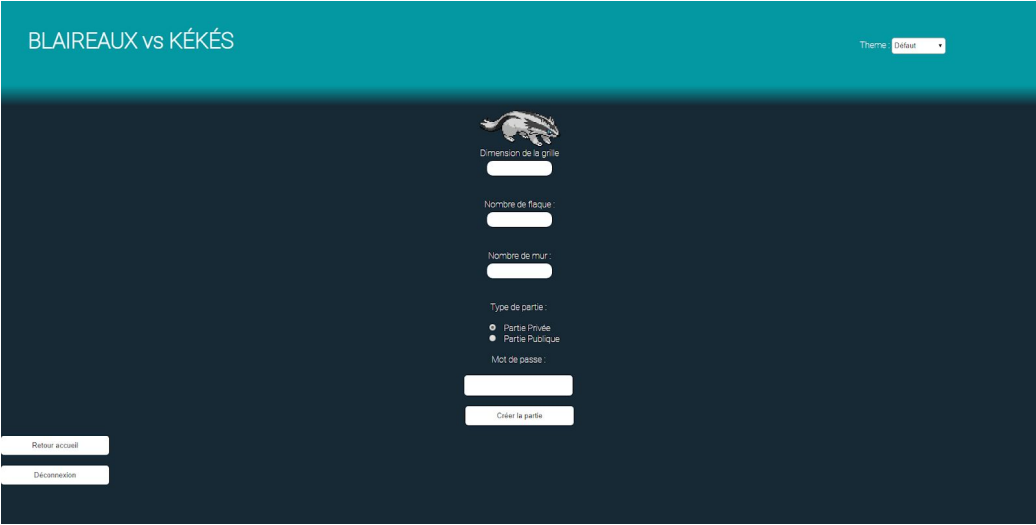
empêchent l'utilisation de fichier externe. Ceci n'étant qu'à but esthétique, nous n'avons pas voulu perdre plus de temps, mais nous étions au courant de cette possibilité. Certaines erreurs provenant du javascript nous a fait perdre énormément de temps. Nous avons découvert que javascript avait des problèmes de précisions, engendrant des fautes lors des calculs de diverses fonctions, notamment au niveau des déplacements et vérification de position. (Dans la console : $0.1 + 0.2 = 0.30000000000000004$, créant des erreurs de calcul improbables qui s'accumulent de plus en plus).

IV- Gestion des parties

- *Structure des données
- *Map dans BD à chaque partie, expliquer la table
- *Partie dans la BD
- *Moteurs ajax pour actualisation en continu

Création de partie

Le clic sur le bouton "Créer une partie" emmène l'utilisateur vers une nouvelle page. Il sera alors possible pour l'utilisateur de choisir les dimensions de la grille, le nombre de flagues ainsi que le nombre de mur présent sur la map. (Schéma 1). Enfin il pourra choisir de mettre sa partie en publique ou en privée, qui débloquera ou non la possibilité de mettre un mot de passe dans le cas où la partie sera privée. (Schéma 1). Le blocage du mot de passe se fait via une fonction JavaScript. (Schéma 2).



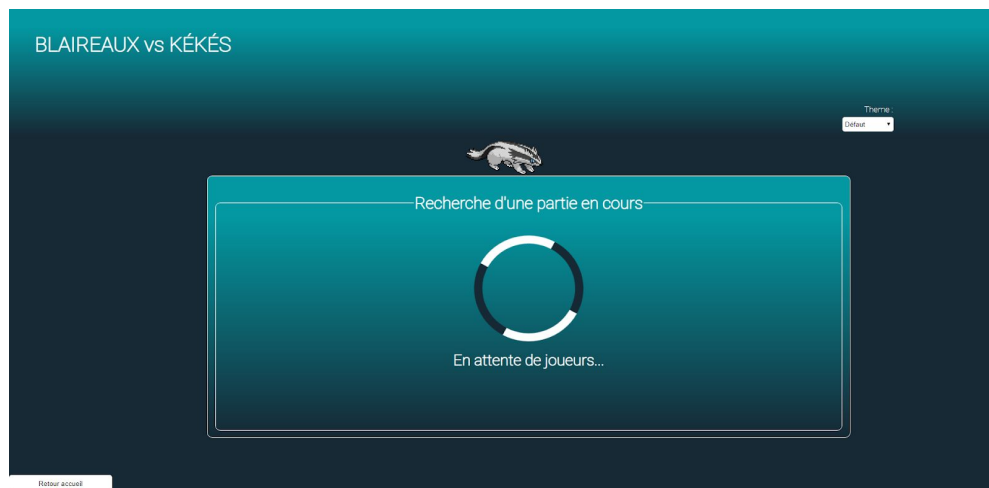
(Schéma 1)

```
function typePartie()
{
  if (document.getElementById('Public').checked)
    document.getElementById('bloqué').innerHTML="";
  else
    document.getElementById('bloqué').innerHTML="<label for=\"mdp\"> Mot de passe :</label> <br> <input type=\"text\" id=\"mdp\" name=\"mdpPartie\" />";
}
```

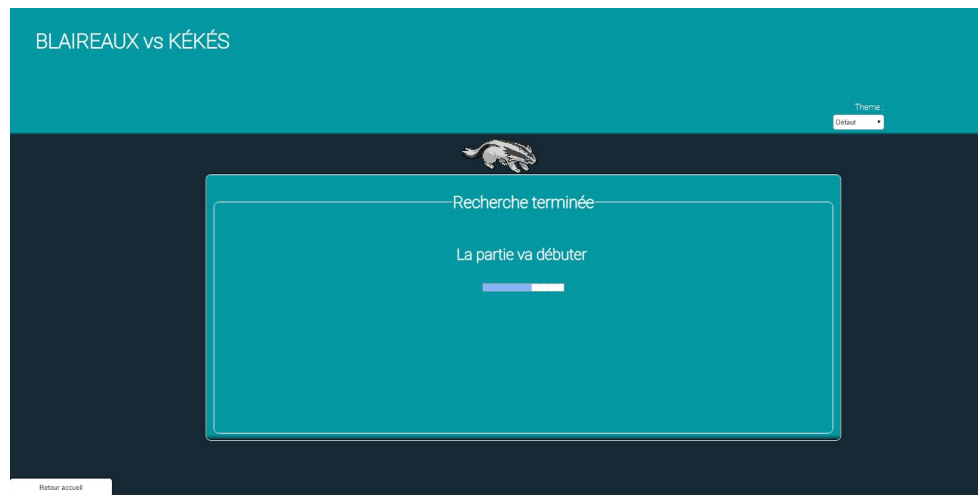
(Schéma 2)

Rejoindre une partie

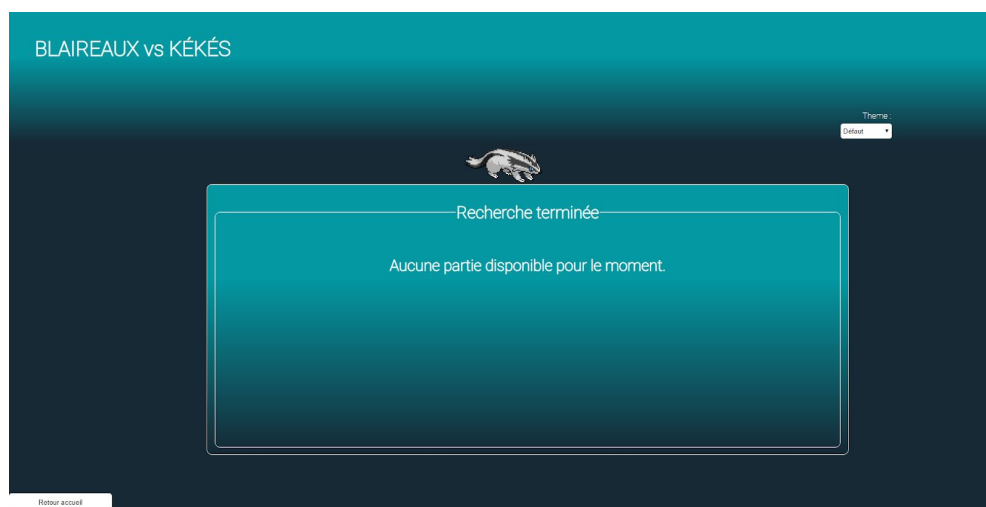
Le clic sur le bouton Rejoindre une Partie emmène l'utilisateur vers une nouvelle page en attente d'être connecté à une partie pouvant accueillir l'utilisateur. L'interface indique l'état de la recherche. Une roue de chargement tourne en continu tant que la recherche n'est pas terminée.



Lorsqu'une partie disponible est trouvée, l'interface change pour avertir l'utilisateur et une barre de progression de cinq secondes apparaît pour préparer l'utilisateur au lancement la partie.



En revanche si la recherche échoue, qu'au terme d'une certaine durée aucune partie n'est disponible, la recherche s'achève et l'interface se modifie pour en avertir l'utilisateur. Il est également possible depuis cette page de quitter la recherche par le biais d'un bouton de retour vers l'accueil.



Une partie possède différents statuts représentés par un entier dans la base de données. Elle peut être “En attente” avant d’être lancée (=0), “En cours” (=1) ou alors “Terminée” (=1).

Toutes les données concernant la partie comme la position de joueurs, des murs ou encore des objets sont stockées sur le serveur, et le client envoie des requêtes à la base de données pour les récupérer et les afficher sur la scène.

V- Limites et conclusion

Au terme du développement de ce projet, le cahier des charges pour la réalisation d'un jeu en ligne est plus ou moins respecté. Nous avons respecté le sujet dans ses grandes lignes, plus particulièrement pour les fonctionnalités attendues dans le jeu, mais nous n'avons pas implémenté complètement la partie réseau. Ainsi notre jeu ne fonctionne à ce stade qu'en local.

Afin d'étendre les subtilités de ce jeu si le développement devait être prolongé il est évidemment possible d'envisager l'implémentation d'un système de discussions instantanée entre joueurs lors des parties ou encore l'utilisation généralisée de modèles importés depuis Blender pour remplacer les primitives utilisées.

Parmi les limites de notre projet on peut aussi citer le thème que nous n'avons pas rendu persistant lors d'un changement de page.

Enfin ce projet nous a permis de découvrir de nouvelles manières de fournir un travail efficace en groupe, comme cela pourrait nous être éventuellement demandé dans une entreprise. Nous avons pu par conséquent découvrir de nouvelles méthodes de travail et nous avons pris conscience de l'effort de synchronisation qu'un projet en groupe peut demander, d'autant plus lorsque ce groupe nous est imposé. Cependant il s'avère que chacun a eu l'opportunité de travailler sur des éléments sur lesquels il se sentait relativement à l'aise.

Annexe : Diagramme des classes utilisées dans notre projet.

