Université de Lille 1 SCIENCES ET TECHNOLOGIES

Soutenance le 27 Mai 2016

À Villeneuve d'Ascq - 16h30

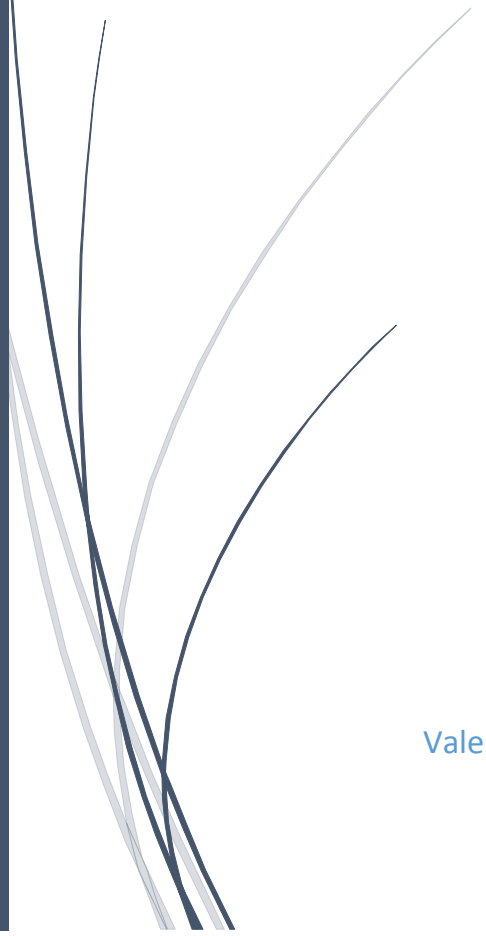
(119) Breathing Forest

Jeu de gestion de forêts par navigateur

Tuteur :

Marius Bilasco

Enseignant chercheur, Université de Lille 1



Valentin Ramecourt & Nicolas Vasseur

Table des matières

I. Présentation du projet

a. Présentation du jeu

Le projet que nous avons mené au cours du second semestre de notre première année de Master informatique se base sur un sujet que nous avons créé et présenté afin de mettre en avant le domaine web qui nous tient à cœur et le cursus que nous avons suivi n'a pas vraiment mis l'accent sur cet aspect de l'informatique qui pourtant représente une importante partie de l'avenir.

C'est pourquoi nous avons souhaité faire un site afin de pouvoir mettre en œuvre des compétences web plus approfondies que celles vues pendant nos études.

Notre projet a donc consisté à créer un jeu implémenté via une application web en utilisant les technologies HTML, CSS, PHP et JavaScript. Il s'agit d'une sorte de jeu par navigateur qui se déroule sur une seule page et qui propose à l'utilisateur de planter des forêts sur des cases d'une grille afin de répondre à des besoins en oxygène et d'obtenir un score selon comment il optimise l'oxygène produit.

Le jeu consiste donc à planter des forêts sur une grille de 144 (16x9) cases et la façon dont sont répartis les forêts détermine un score qu'il faudra maximiser, celui-ci est calculé selon différents facteurs. Il faut d'abord comprendre les différents éléments composant la grille de jeu initiale, il y a trois types de cases différentes :

- Cases vides : Ce sont les cases sur lesquelles on peut planter une forêt, elles fournissent de l'eau uniquement à la forêt qui est posée dessus et ont un besoin en oxygène.

- Cases villes : Ces cases ont un besoin en oxygène important qui dépend de la population et qu'il faudra satisfaire à l'aide des forêts alentours.

- Cases rivières : Ces cases offrent un bonus d'eau fournie aux 8 cases vides alentours.

On dispose d'un montant d'argent de départ de 1 000 000 €. Il faudra en utiliser le moins possible afin d'avoir un meilleur score.

Une forêt a un coût lorsqu'on veut la planter, un type comme chêne, pin, hêtre... qui détermine son coût, l'oxygène qu'elle produit et l'eau qu'elle a besoin. Chaque forêt donne de l'oxygène aux 8 cases alentours, l'oxygène que produit une forêt dépend directement de l'eau qui lui est fournie, si une forêt possède la moitié de l'eau qui lui est nécessaire alors elle produira uniquement la moitié de l'oxygène qu'elle devrait produire normalement. La même chose se produit si la forêt a deux fois trop d'eau.

Il faut donc disposer les forêts le mieux possible selon les différents facteurs cités plus haut (eau fournie et eau nécessaire, oxygène produit et besoin en oxygène) pour optimiser la production d'oxygène ce qui va maximiser le score. En clair, il faut essayer d'offrir la valeur juste d'oxygène aux cases vides et villes tout en minimisant le coût de plantation des forêts.

Il y a la possibilité de voir le score des autres utilisateurs via une page de classement.

b. Architecture de l'application

Le site requiert une authentification afin de pouvoir jouer ainsi chaque utilisateur doit se créer un compte personnel qui lui permet d'accéder à sa propre grille sur laquelle le jeu se déroule.

Lorsque l'on n'est pas connecté, il nous est possible de se connecter, s'inscrire, voir le classement ou voir la page 'A propos'. Une fois connecté, il nous est possible en plus d'accéder à la page du plateau de jeu, de voir notre profil et de nous déconnecter.

Sur chaque page de l'application, un bas de page est présent avec comme informations le nom du jeu 'Breathing Forests' et le temps qu'il a fallu pour générer la page. Il existe aussi un menu en haut qui permet d'accéder aux différentes page du site (qui est différent selon que l'on soit connecté ou non). Voici les différentes pages du menu :

- Page d'accueil :

La première page sur laquelle on accède au site, il y a deux formulaires sur cette page, un nous permettant de nous connecter et l'autre de nous inscrire.

- Page de classement :

Cette page permet de voir la liste des utilisateurs et leur score trié de façon descendante afin de voir qui a trouvé le meilleur score.

- Page 'A propos' :

On présente ici le contexte de développement du site et il y a des liens vers le git du projet ainsi que le wiki décrivant les règles du jeu.

- Page 'Mon compte' :

Uniquement accessible si l'on est connecté, elle permet de voir les informations de notre compte (nom d'utilisateur et date d'inscription), de changer de mot de passe ou de supprimer son compte.

- Page plateau :

La page principale du site qui permet de jouer, elle est uniquement accessible une fois l'utilisateur connecté.

Il y a deux éléments principaux sur cette page :

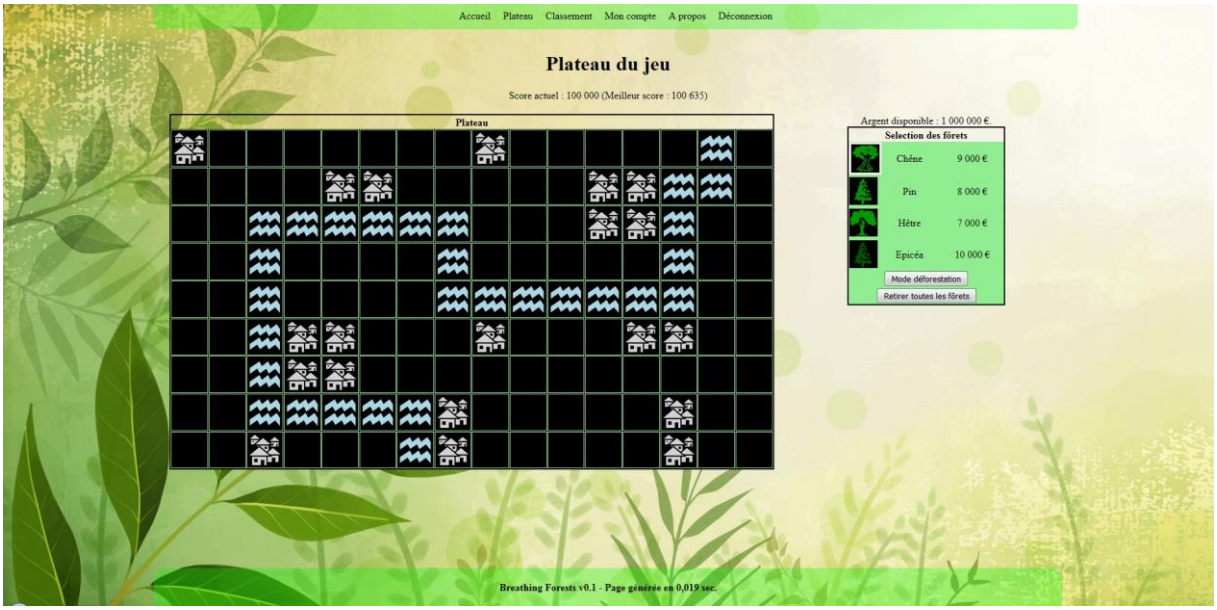
- Grille du jeu à gauche de 16x9 soit 144 cases

On visualise cette grille à l'aide d'images qui représentent les différentes cases possibles, il y a donc une image pour les cases vides, une pour les cases villes, une pour les cases rivières et une image pour chaque type de forêt différente (chêne, pin, hêtre...).

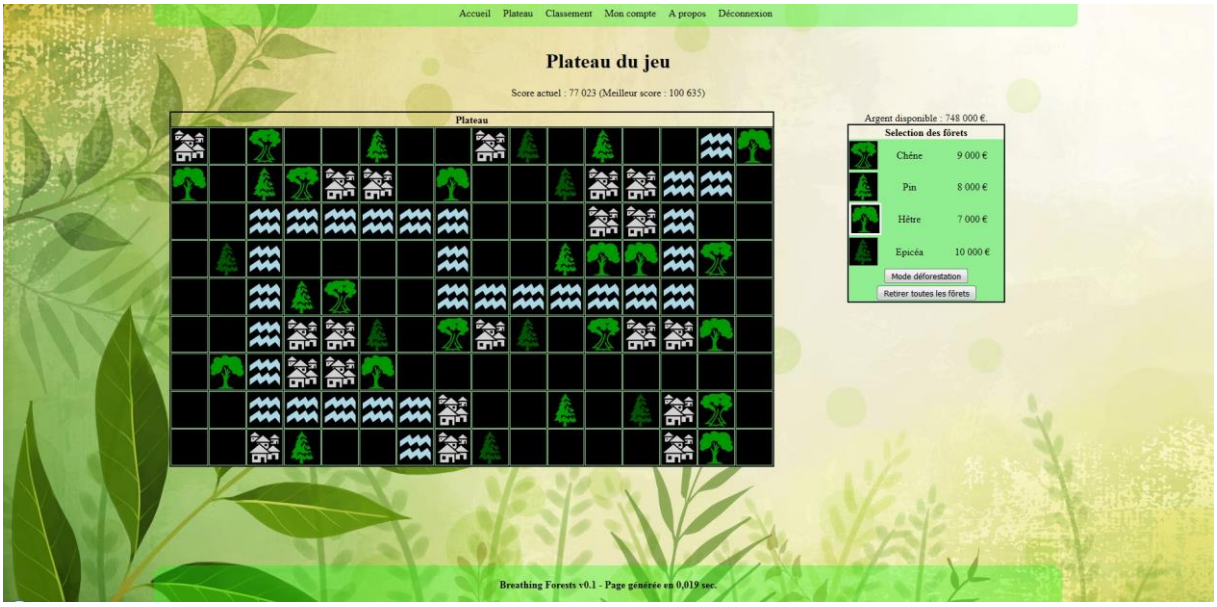
- Liste des forêts à planter

Un tableau avec pour chaque ligne, l'image de la forêt, son type d'arbre et son coût.
Pour sélectionner une forêt, il faut cliquer sur son image puis on peut cliquer sur n'importe quelle case vide pour planter la forêt sur celle-ci.

Voici le plateau à l'état initial et avec des forêts :



Plateau du jeu sans forêts



Plateau du jeu avec forêts

c. Présentation des technologies utilisées

Les langages :

- Hypertext Preprocessor : Le PHP

Le langage PHP est le langage pour le développement web que nous avons choisi. Etant un des langages les plus utilisés sur les sites internet et étant un des nombreux descendants du langage C, ce langage non typé est devenu un pilier du web. L'intégration du code HTML au sein d'une page PHP se fait très naturellement et l'inverse également.

- Le JavaScript

Le JavaScript est un langage de programmation de scripts. Un script s'exécute sur le navigateur côté client. Les scripts permettent d'effectuer des actions sur la page sans avoir à recharger la page et sans données du serveur, c'est-à-dire, sans avoir à recontacter le serveur pour obtenir une nouvelle page.

- Asynchronous JavaScript And XML : L'AJAX

L'AJAX est une combinaison du JavaScript, des CSS, du XML, du DOM et des XMLHttpRequest. En d'autres termes, c'est un langage de programmation de scripts venant compléter le JavaScript. En utilisant AJAX, le dialogue entre le navigateur et le serveur se déroule la plupart du temps de la manière suivante : un programme écrit en langage de programmation JavaScript, incorporé dans une page web, est exécuté par le navigateur. Celui-ci envoie en arrière-plan des demandes au serveur Web, puis modifie le contenu de la page actuellement affichée par le navigateur Web en fonction du résultat reçu du serveur, évitant ainsi la transmission et l'affichage d'une nouvelle page complète.

Les frameworks & plugins :

- JQuery

jQuery est un des frameworks JavaScript les plus connu. Il est multiplateforme et a été conçu pour simplifier la syntaxe des scripts côté client dans le code HTML. jQuery est libre, opensource et bon nombre de plugins sont basés sur son code.

- Prototype

Prototype est un framework JavaScript initialement créé comme un des éléments de base pour le support AJAX de Ruby on Rails. C'est, comme jQuery, une des bibliothèques les plus populaires de JavaScript. Prototype fournit des fonctions permettant la création d'objets, ce que JavaScript classique ne permet pas. Il ne faut cependant pas confondre ce framework avec la propriété prototype de JavaScript classique lors de la création d'une fonction.

- qTip

qTip est un plugin JavaScript pour le framework jQuery. Il met à disposition des outils simple pour la gestion des infobulles.

II. Structure & fonctionnalités

a. PHP

Le PHP est l'un des langages cœur de notre projet, il permet principalement de gérer l'affichage des éléments dynamiques comme le plateau de jeu et les interactions avec la base de données à l'aide de la classe PDO.

Grâce à ce langage, il a été possible de mettre en place une structure du site beaucoup moins redondante à l'aide d'inclusion de fichiers qui permet d'éviter la répétition d'une partie du code commun à chaque page comme le fichier de configuration, l'élément HTML head, le menu ou encore le pied de page. Ainsi, si une modification doit avoir lieu dans un de ces fichiers, il n'est pas nécessaire de répéter celle-ci sur chaque page du site, il suffira de modifier le fichier en question.

Le système de création de compte et de connexion est implémenté via ce langage en traitant les requêtes POST des formulaires d'inscription et de connexion. Dans le cas de l'inscription, on vérifie les données saisies par l'utilisateur (on regarde par exemple si le nom d'utilisateur n'est pas trop long, s'il contient uniquement des caractères alphanumériques...) avant d'insérer une ligne dans la base de données pour stocker les informations du compte, on pense d'ailleurs à crypter le mot de passe à l'aide d'une fonction de hachage pour une question de sécurité. Pour la connexion, il faut vérifier si l'utilisateur existe et si le mot de passe est correct, si c'est le cas, nous créons alors une session qui permet d'identifier l'utilisateur et de le connecter à son compte.

Pour le classement, on fait une requête SQL qui permet de récupérer un jeu de données contenant les différents utilisateurs et leur score trié en ordre décroissant, il suffit ensuite d'afficher le résultat avec une simple boucle et une mise en forme via l'élément HTML table.

Sur la page 'Mon compte', le PHP est utilisé pour afficher les informations du compte comme le nom d'utilisateur ou la date d'inscription qui sont récupérées à l'aide de l'objet PDO pour faire une requête SQL, de la même manière que les formulaires d'inscription et de connexion. Lorsqu'on appuie sur un bouton 'Valider', une requête POST est envoyée sur la même page et le PHP traite les informations. Il est ainsi possible de changer de mot de passe, avec toutes les vérifications nécessaires ou de supprimer son compte.

C'est sur la page du plateau que le PHP est le plus utilisé, principalement par le biais d'Ajax. Lorsqu'on arrive sur la page, un attribut onload sur l'élement HTML body permet d'invoquer une fonction JavaScript qui va s'occuper de faire l'appel Ajax pour charger le contenu de la page. De cette façon, chaque fois qu'une modification aura lieu sur le plateau, il ne sera pas nécessaire de recharger toute la page mais seulement son contenu via un appel Ajax.
Quand on charge le contenu de la page du plateau, on envoie un tableau de données récupéré en base de données via un autre appel Ajax à un fichier PHP qui va générer l'affichage de la grille du jeu. Le tableau lui permet alors de connaître les différentes forêts à afficher à la place des cases vides. Il y a aussi la sélection des forêts à planter, à droite de la grille de jeu, qui précise l'argent encore disponible et la forêt sélectionnée via un contour blanc si c'est le cas. Ainsi, chaque fois qu'une forêt est sélectionnée, qu'une forêt est plantée ou qu'une forêt est retirée (quelque-soit le moyen utilisé), on fait un appel Ajax pour enregistrer ce changement en base de données et on charge à nouveau le contenu de la page avec les nouvelles données à prendre en compte comme la nouvelle forêt à afficher ou à ne plus afficher ou encore ajouter un contour blanc sur la forêt qui vient d'être sélectionnée. Il y a donc une véritable association du PHP et JavaScript via l'Ajax sur cette page, ce qui permet de la rendre un peu plus dynamique.

Maintenant que l'on a expliqué comment le PHP a été mis à profit pour notre projet, nous allons parler du deuxième langage cœur de l'application : le JavaScript.

b. JavaScript : fonctions.js & case.js

Notre projet est structuré de sorte à effectuer la partie calculatoire et logique du côté du JavaScript (JS).
Pour cela nous avons deux principaux fichiers :

- fonctions.js
- case.js

Le JavaScript nous a permis de modéliser notre jeu et ainsi d'alléger la charge sur le serveur car il s'effectue sur le navigateur de l'utilisateur. Par l'utilisation d'une librairie JS du nom de Prototype, nous avons pu modéliser la classe case, qui représente chaque case de la grille de jeu. Ainsi tous les traitements se feront en JavaScript tandis que le PHP s'occupera principalement des accès à la base de données.

Le premier regroupe principalement les fonctions pouvant être appelés à partir du PHP et permet une mise à jour de la base de données à la suite d'une action (placer un arbre, enlever un arbre, ..), il passe ensuite la main au deuxième fichier regroupant la logique concernant les cases de la grille.
Le deuxième fichier est une classe représentant une case du tableau. Elle possède des paramètres essentiels au bon fonctionnement de la grille que ce soit du côté du modèle, de la vue ou du contrôleur.

Examinons ces fichiers plus en détail :
Nous pouvons catégoriser les fonctionnalités de fonctions.js en trois types différents :

- Fonctions récursives :
  - loadContentBoard qui permet de rafraîchir la grille de jeu avec les modifications apportés. Cette fonction peut être appelée à différents endroits du code. Elle permet de mettre à jour la grille à l'arrivée sur la page mais également à la suite des actions principales de notre jeu, à savoir : Placer un arbre, retirer un arbre, retirer tous les arbres.
  - processTooltip qui permet de mettre à jour les infobulles en fonctions des modifications sur la grille. Cette fonction est appelée à chaque appel de loadContentBoard.

- Fonctions qui ne sont appelées qu'une fois :
  - sendInitBoard qui est appelé au chargement de la page de jeu. Elle permet de créer la grille de jeu grâce aux informations récupérées de la base de données, à savoir :
    - La grille de base (Cases vides, villes et rivières).
    - Les arbres placés par l'utilisateur en question.

- Fonctions appelées à la suite d'une action précise :
  - plantCurrentTree qui permet de planter l'arbre pré-sélectionné à la case renseignée
  - removeTree qui permet d'enlever l'arbre à la case sélectionné, ou tous les arbres de la grille. Cette fonction affiche une boite de confirmation car la fonction reste sensible. En mode déforestation cette boite de confirmation ne s'affiche pas.
  - gridScore qui calcule le score de la grille. Cette fonction est appelée à la suite de la création de la grille initiale.

Le deuxième fichier représentant une case de la grille a été construit de telle sorte :

	oxygen_need	oxygen_give	oxygen_received	default_oxygen_give
Empty				
Town				
River				
Tree		AOE + himself		

	water_need	water_give	water_received
Empty		himself	
Town			
River		AOE	
Tree			

	cost	score	tree_type
Empty			
Town			
River			
Tree			

Nous pouvons remarquer dans ce tableau les différents paramètres qu'une case peut avoir et à quels types de cases ils s'appliquent.
Les paramètres de notre case sont les suivants : le type, l'abscisse, l'ordonnée, le score et les données. Le tableau ci-dessus affiche les données présentes dans case.data, qui contient différentes informations sur l'oxygène, l'eau, ... Plus généralement, toute information n'étant pas générique aux 4 types de cases.
Les cases vertes correspondent aux paramètres que les cases en question sont susceptibles d'avoir. AOE signifie que la case diffuse aux 8 cases l'entourant et himself à elle-même.

Dans notre implémentation, une case Arbre va hériter des propriétés d'une case Vide. Les propriétés des deux cases, étant compatibles, vont donc cohabiter dans la même instance de case. Si jamais nous enlevons l'arbre en question la case vide récupérera son état initial, comme si les deux cases étaient superposées.

Le fichier case.js contient tous les outils permettant de gérer les cases : la création d'une case quelconque avec l'initialisation de ces paramètres, l'ajout d'un arbre dans la grille et la suppression d'un arbre ou de tous les arbres de la grille. Ce fichier contient également des fonctions annexes permettant de faciliter le traitement des autres fonctions et d'alléger le code: get_in_grid_bounds qui permet de ne pas dépasser les limites du tableau lors de la manipulation de celui-ci et case_score qui calcule le score d'une case ce qui permet le calcul du score total.

c. Architecture MVC

Lors de la modélisation de notre projet nous avons directement pensé à l'architecture MVC pour la structuration. Cette architecture nous permet une meilleure compréhension des éléments composant nos pages et cela permet également de reprendre le projet plus simplement, par la suite ou par d'autres personnes.

Du point de vue des dossiers, à la racine de notre projet nous avons :

- ajax : Ce dossier renferme tous les appels AJAX comme celui rafraichissant la grille du jeu, celui qui plante un arbre, …
- includes : Ce dossier contient tous les fichiers PHP s'incluant dans des autres pages. Nous pouvons citer par exemple le header, le footer et le menu.
- utilities : Il est composé des ressources externes à notre projet comme les frameworks et plugins, mais également de nos fichiers JavaScript servant à définir le modèle de notre projet.
- styles : Ce dossier est le dossier qui est utilisé par la vue. Les fiches de styles ainsi que les images y sont rangés. La fiche de style permettant d'éditer le style des infobulles du plugin qTip y est également.

A la racine du projet nous avons toutes les pages du site tels que l'accueil, le tableau, le classement, le profil, …

Les acteurs principaux de l'architecture dans notre projet sont l'extension PDO de PHP et l'AJAX. Nous nous servons de PDO pour transmettre les informations nécessaires entre la base de données et le PHP, et de requêtes AJAX pour les transmissions entre le PHP et le JavaScript.
Nous allons analyser deux cas qui gèrent les interactions entre le Modèle, la Vue et le Contrôleur.

Premier cas : PDO, le fichier plant_current_tree.php

```php
$query_user = "SELECT money FROM user WHERE PK_user = ".$_SESSION['PK_user'];
$result_user = $bdd->query($query_user);
$row_user = $result_user->fetch(PDO::FETCH_ASSOC);

$query_tree = "SELECT * FROM tree WHERE PK_tree = ".$_SESSION['current_PK_tree'];
$result_tree = $bdd->query($query_tree);
$row_tree = $result_tree->fetch(PDO::FETCH_ASSOC);

if($row_user['money'] > $row_tree['cost']){

    $query_insert_user_tree = $bdd->prepare("INSERT INTO asso_user_tree (FK_user, nb_row, nb_column, FK_tree)
        VALUES (:FK_user, :nb_row, :nb_column, :FK_tree)");
    $query_insert_user_tree->execute(array(
        ':FK_user' => intval($_SESSION['PK_user']),
        ':nb_row' => intval($_POST['row']),
        ':nb_column' => intval($_POST['column']),
        ':FK_tree' => intval($_SESSION['current_PK_tree'])
    ));


    $query_money = "UPDATE user SET money = :money WHERE PK_user = ".$_SESSION['PK_user'];
    $result_money = $bdd->prepare($query_money);
    $result_money->execute(array(
    ':money' => $row_user['money']-$row_tree['cost']));

    // Retour le traitement en JS
    echo html_entity_decode(json_encode($row_tree));
} else {
    echo 'Vous n\'avez plus assez d\'argent pour planter cette fôret.';
}
```

Cette fonction récupère premièrement l'utilisateur courant dans la base de données, ensuite elle récupère les informations de l'arbre allant être planté.
Si l'utilisateur possède assez d'argent pour planter l'arbre, une ligne sera insérée dans la table asso_user_tree, qui permettra d'associer un utilisateur, un arbre et une case. De ce fait nous n'avons pas besoin de stocker les informations correspondantes à une case, mais simplement l'emplacement de tel type d'arbre à telle case.

Deuxième cas : AJAX, la fonction loadContentBoard() dans fonctions.js

```
function loadContentBoard(gridCase){
    if(gridCase !== undefined || gridCase != null){
        trees_score += gridCase.data.score_modif;
    }else{
        var gridCase = [];
    }
    jQuery(function ($) {
        $.post('ajax/get_current_money.php', {
        }, function(data) {
            var current_money = parseInt(data);
            var total_score = current_money * 0.1 + trees_score;
            $.post('ajax/content_board.php', {
                array_board: JSON.stringify(grid),
                total_score: total_score,
                gridCase: JSON.stringify(gridCase)
            }, function(data) {
                document.getElementById('main').innerHTML = data;
                processTooltip();
            });
        });
    });
}
```

Cette fonction permet de récupérer l'argent que possède l'utilisateur avec un appel à la base de données, elle calcule ensuite le score total de l'utilisateur en multipliant par 0.1 l'argent de l'utilisateur et en rajoutant le score que génèrent les arbres. Ensuite un appel AJAX est fait pour rafraîchir la grille de jeu en envoyant comme paramètres : le tableau, le score total et la case qui a été modifié dans le cas où la fonction loadContentBoard est appelée après l'ajout/la suppression d'un arbre.
Le DOM est ainsi modifié pour réinsérer le score et la grille de jeu. Les infobulles sont mise à jour.

III. Bilan technique

a. Perspectives

Nous n'avons pas développé ce projet pour une publication, néanmoins nous avons réfléchi aux évolutions possibles d'un tel projet, que ce soit dans le fond ou dans la forme.
Nous avons développé notre projet de telle sorte qu'il soit jouable, simple pour l'utilisateur sans trop de données superflues mais nous avons pensé à quelques idées d'amélioration :

- La mise en place d'une grille par semaine

Cette amélioration serait indispensable dans le cas d'une publication du jeu. Elle permettrait à chaque joueur de prendre le temps qu'il souhaite chaque semaine pour optimiser une grille et à contrario de ne pas "finir" le jeu en une semaine.
Une évolution possible serait de dresser différents classements : Le classement actuel prenant en compte la grille actuelle, un classement sur toutes les grilles, un classement par mois, ...

- La mise en place d'un éditeur de grille

Cette idée permettrait aux joueurs de varier le plaisir et de proposer aux développeurs des nouvelles grilles qui pourraient être utilisés par la suite. Nous pouvons imaginer un système de récompense à la personne ayant la meilleure grille, ou simplement lui attribuer des points bonus sur le classement total à raison d'une fois par mois maximum par exemple.

- L'ajout de nouvelles cases

Nous avons pensé rajouter des cases "néfastes" ou "extra-bonus". Une case usine plus ou moins néfaste pourrait être ajoutée représentant les rejets de CO2 ou simplement le rejet des voitures des personnes qui viennent y travailler et le chauffage pour chauffer les locaux. Cette case réduirait l'apport en oxygène des arbres étant à proximité et réduirait également la qualité de l'eau fournit par les rivières.
Une case "Fertile" pourrait être ajoutée, diminuant de moitié les besoins en eau de l'arbre ou doublant l'oxygène produit par l'arbre.

- L'ajout de publicité

L'ajout de publicités pourrait nous permettre de récolter des fonds pour faire des récompenses aux membres gagnants ou aux membres proposant les meilleures cartes. Nous pensons envoyer par colis un cadeau en rapport avec l'environnement (sachet de graine, pot à fleur avec indice pour arroser, ...). Ce système permettrait d'attirer des joueurs, de les fidéliser et également de sensibiliser à l'environnement.

- Proposer le jeu à une organisation environnementale

Proposer le jeu à une organisation environnementale permettrai de faire connaître le jeu, et de générer de la publicité gratuite pour l'organisation ainsi que pour le jeu. Ceci pourrait déboucher sur des opportunités professionnelles.

b. Bilan personnels

Ce PJI fut pour nous une bonne opportunité de gérer un projet entièrement, de proposer une idée, de la structurer et d'y chercher les avantages et les évolutions possibles.
La modélisation de notre projet nous a forcées à nous attarder sur des détails techniques et logistiques tels que la mise en place d'un planning de disponibilités pour le travail en commun et d'un GitHub pour partager plus simplement le projet.
Nous avons pu échanger sur notre manière d'implémenter les différentes fonctionnalités de l'application.

Ce projet nous a également ouvert l'esprit sur les possibilités et les opportunités d'un tel projet grâce à notre tuteur Mr. Bilasco qui a su nous guider et nous poser les bonnes questions.

Du côté des technologies nous avons tous deux progresser car nous n'étions pas autant à l'aise que l'autre dans tous les domaines. Le fait d'être amis hors du travail a rendu simple le travail en équipe, de plus nous avons la même vision de programmation.