Projet WEB2
League Of Stones

Université Toulouse 2 - Jean Jaurès 7 janvier 2019

1 Présentation

1.1 Concept

Dans ce projet, il est question de développer un *mashup* (mélange) de deux jeux vidéos. En utilisant le système de jeu de Hearthstone (HS) [1] développé par Blizzard™ nous intégrerons les données ouvertes du jeu League Of Legends (LoL) [2] développé par Riot Games™. Ce projet s'intitule "League of Stones" et il propose un jeu de cartes dont les cartes proviennent de LoL. Les deux joueurs s'affrontant possèdent un deck [3] de 20 cartes et 150 points de vie. Chaque carte décrit un champion avec une statistique d'attaque et une de défense. Les joueurs jouent chacun leur tour dans l'objectif de réduire les points de vie de l'adversaire à 0 pour gagner la partie. Au début de la partie, les deux joueurs piochent 4 cartes. Durant son tour, un joueur peut effectuer 3 actions dans l'ordre qu'il le souhaite :

- Piocher une carte (une fois par tour)
- Poser une carte sur le plateau (au maximum 5 par joueur sur le plateau)
- Attaquer (une fois par carte sur le plateau)

Une carte posée sur le plateau durant ce tour ne pourra attaquer qu'au tour suivant. Au tour suivant la carte peut attaquer un monstre sur le plateau adverse. Résolution de l'attaque :

- Si la valeur d'attaque de la carte qui attaque est supérieure à la valeur de défense de la carte adverse alors cette dernière est supprimée et la différence entre les deux valeurs de carte est retirée aux points de vie de l'adversaire.
- Si les deux cartes ont une attaque et une défense égales alors les deux cartes sont détruites.
- Si la valeur d'attaque est inférieure à la valeur de défense alors la carte attaquante est détruite.
- S'il n'y a aucune carte sur le plateau adverse alors la carte peut directement attaquer les points de vie de l'adversaire.

Une fois qu'une carte a attaqué, elle devra attendre le tour suivant pour attaquer de nouveau.

Objectifs

1. Développer une application web utilisant des requêtes AJAX pour interroger les Web Services présentés dans ce document pour réaliser un client pour le jeu League of Stones.
2. Cette application devra être développée de manière *responsive* pour qu'elle soit utilisable sur un smartphone.
3. Bonus : faire une application mobile contenant cette application (React Native, Progressive Web App, APK, IOS)

1. `http://eu.battle.net/hearthstone/fr/`
2. `http://euw.leagueoflegends.com/fr`
3. ensemble de cartes disponibles pour toute la partie

2 Composition de votre groupe

Pour ce projet vous serez (dans l'idéal) 4 ou 5 membres dans votre groupe. Chaque membre aura une responsabilité spécifique concernant le projet :

- Gestion de projet
  - Le responsable "gestion de projet" sera en charge du bon déroulement du projet dans son intégralité. Il devra vérifier que le cahier des charges est bien respecté, gérer la gestion de version (GIT, SVN, ...), faciliter la communication entre les membres, et rapporter l'état d'avancement aux enseignants.
- Responsable technique
  - Le responsable technique aura pour tâche de décider des frameworks à utiliser, de l'organisation du code et du développement des fonctionnalités de manière générale. Il portera une attention particulière aux tests ainsi qu'à la propreté du code (formattage, clarté).
- Responsable interface
  - Le responsable interface aura pour rôle de s'assurer que le client est responsive, de la mise en place du design, de l'expérience utilisateur.
- Responsable backend
  - Le responsable backend devra étudier l'architecture du backend pour connaitre les Web Services et leur implémentation. De plus il sera en charge de développer les améliorations du backend nécessaires (voir la section 4 à la fin de ce document)
- Responsable mobile (pour les groupes de 5 ou en bonus)
  - Le responsable mobile devra mettre en place une version mobile du jeu compatible Android a minima, soit en utilisant le coté responsive, soit React Native, soit les Progressive Web Apps.

Objectif

A la fin du projet il vous sera demandé de fournir :

- un rapport contenant une section par responsable,
- le code source de votre application (repo GIT ou SVN),
- l'application utilisable [4]

4. Déployée sur un serveur et/ou l'application mobile

3 Développement du jeu

Le développement du jeu s'appuiera sur un ensemble de *Web Services* déjà disponibles [5]. Votre objectif ici sera donc de créer un client capable d'exploiter ces Web Services de façon à pouvoir jouer avec d'autres joueurs qui utiliseront potentiellement un client différent du vôtre. De plus il vous sera demandé de créer un client *responsive*, c'est à dire utilisable convenablement depuis un smartphone. L'objectif est d'utiliser des technologies existantes permettant de créer une application mobile à partir d'un client web. Le développement se déroulera en 3 étapes :

1. création de la partie utilisateur (création de compte, connexion, ...)
2. mise en relation des joueurs désirant faire une partie et de commencer un match ("match-making")
3. gestion d'un match (piocher carte, jouer une carte, attaquer, ...).

Le client web sera développé en React. Un début d'implémentation du client web contenant l'authentification est disponible sur `https://github.com/blandine/los-ui`. Vous pouvez forker ce repository pour vous aider à démarrer.

3.1 Généralités

Tous les Web Services ont un fonctionnement similaire. Ils sont tous accessibles à partir du serveur :
`http://ling.westeurope.cloudapp.azure.com:3000/`
Le code du serveur est accessible sur le Git suivant : `https://github.com/blandine/League-Of-Stones`. Une fois le serveur Express lancé (exemple : locahost :3000) et la base de données MongoDB créée, vous pourrez utiliser ce code backend pour votre développement local. Toute modification du code originale est prohibée. Les Web Services sont tous accessibles à partir de la racine du serveur "/" : `localhost:3000/`

Le nom des Web Services doit être précisé à la fin de cette URL. Tous les Web Services sont de type GET, les paramètres étant des paramètres GET. Les Web Services se trouvent dans le dossier "modules" du Git. Les Web Services sont regroupés en 4 catégories précisées avec le début de l'URL :

- Utilisateurs : `/users/...`
- Recherche de match : `/matchmaking/...`
- Match : `/match/...`
- Cartes : `/cards/...`

Ces Web Services retournent toujours un object JSON contenant 2 champs. Le premier est "status" qui permet de savoir si le Web Service s'est bien déroulé ou si une erreur est survenue. Le champ "status" contient "ok" si ça s'est bien passé, "error" sinon. Dans le cas où le Web Service s'est bien déroulé, un champ "data" contient la donnée utile retourné par le Web Service. Dans le cas d'une erreur un champ "message" contient un message précisant l'erreur

5. Tous les Web Services sont définis dans ce document

survenue. Voici deux exemples de résultat :

Listing 1 – "Exemple de résultat pour un Web Service réussi"

```
{
  "status" : "ok",
  "data" : ...
}
```

Listing 2 – "Exemple de résultat pour un Web Service avec une erreur"

```
{
  "status" : "error",
  "message" : "Message de l'erreur"
}
```

Dans la suite de ce document, on ne présentera que le contenu du champ "data" d'un appel réussi de chaque Web Service.

3.2 Gestion utilisateur

La première étape du développement sera de mettre en place la gestion de l'utilisateur. Un utilisateur peut effectuer les actions suivantes :

- Créer son compte
- Se désinscrire
- Se connecter
- Se déconnecter

3.2.1 Créer son compte

La première étape étant la création du compte pour pouvoir accéder au jeu. Le Web Service est le suivant :

`/users/subscribe`

Ce Web Service nécessite 3 paramètres : l'email (qui servira de login de connexion), le mot de passe et le nom d'utilisateur.

- "email" : email de l'utilisateur (identifiant)
- "name" : nom de l'utilisateur (qui sera affiché sur l'interface)
- "password" : mot de passe (de préférence crypté)

Ce Web Service retourne l'ID de l'utilisateur qui a été créé. Exemple d'utilisation du Web Service :

`/users/subscribe?email=loutre@murloc.fr&name=Loutre&password=02166e5243dc7270996`

Listing 3 – "Résultat du Web Service d'inscription"

```
{
  "id" : "123456789ABCD"
}
```

Ce Web Service retourne une erreur si l'utilisateur est déjà inscrit (même email).

3.2.2 Supprimer son compte

De la même manière, il est possible de supprimer son compte en appelant le Web Service suivant :
`/users/unsubscribe`

Ce Web Service nécessite 2 paramètres :
— "email" : l'email de l'utilisateur à supprimer
— "password" : le mot de passe de l'utilisateur à supprimer

3.2.3 Connexion

Une fois son compte créé, l'utilisateur peut se connecter en appelant le Web Service suivant :
`/users/connect`

Ce Web Service nécessite 2 paramètres :
— "email" : l'email de l'utilisateur qui souhaite se connecter
— "password" : le mot de passe de l'utilisateur
Il retourne 4 champs :
— "id" : identifiant de l'utilisateur connecté
— "email" : email (identifiant) de l'utilisateur connecté
— "name" : le nom d'utilisateur à afficher
— "token" : le token d'identification de la session
Une session est créée automatiquement à la connexion de l'utilisateur. Cette session est identifiée par le token retourné par ce Web Service. Ce token devra être envoyé à tous les Web Service qui nécessite une connexion. Si ce token n'est pas envoyé alors l'utilisateur sera considéré comme non connecté. Tous les Web Service qui suivent nécessitent une connexion. Il sera alors nécessaire d'ajouter le paramètre token de la manière suivante :
`/api/ws?param=...&token=[token]`
Le paramètre a ajouté a donc la clef "token" et doit contenir la valeur retournée dans le champ "token" du Web Service de connexion.
Elle lui donne accès aux Web Service de "matchmaking".

3.2.4 Déconnexion

Pour se déconnecter le Web Service est le suivant :
`/users/disconnect`

Ce Web Service ne requiert aucun paramètre, il faut néanmoins être déjà connecté. A la suite de l'appel de ce Web Service, la session de l'utilisateur est supprimée.

3.3 Recherche de match

Une fois que l'utilisateur est connecté, il peut chercher un adversaire pour lancer une partie. Pour cela le système de "matchmaking" fonctionne avec la mise à disposition d'une liste de joueurs désirant faire une partie. Il faut donc d'abord que le joueur indique qu'il souhaite rejoindre une partie (de cette manière il apparaît dans la liste des joueurs désirant jouer). Il peut ensuite envoyer une demande à un autre joueur de la liste. Le second joueur peut alors ignorer la requête, ou l'accepter et commencer la partie. Les Web Services "matchmaking" sont les suivants :

3.3.1 Participer

L'utilisateur peut se rajouter à la liste des joueurs voulant faire une partie en appelant le Web Service suivant :
`/matchmaking/participate`

Ce Web Service ne nécessite aucun paramètre. Une fois appelé, ce Web Service met l'utilisateur dans la liste des joueurs désirant jouer.

Ce Web Service retourne 2 champs :

- "matchmakingId" : l'identifiant du "matchmaking" qui correspond à l'entrée du joueur dans la liste des joueurs désirant jouer
- "request" : un tableau contenant l'ensemble des demandes reçues

Le champ "request" se met à jour chaque fois qu'une nouvelle requête est reçue. Il faut donc envoyer une requête vers ce Web Service régulièrement [6] pour mettre à jour les requêtes reçues. Une requête contient les champs suivants :

- "userId" : l'identifiant de l'utilisateur qui a envoyé la requête
- "matchmakingId" : l'identifiant du "matchmaking" de l'utilisateur qui a envoyé la requête
- "name" : le nom de l'utilisateur qui a envoyé la requête

Il est donc possible de connaître quel joueur souhaite jouer avec l'utilisateur connecté.

3.3.2 Liste des utilisateurs désirant jouer

Une fois que l'utilisateur est dans la liste de "matchmaking", il peut la récupérer et ainsi accèder à tous les joueurs désirant jouer (sauf lui-même). Le Web Service pour obtenir cette liste est le suivant :
`/matchmaking/getAll`

Ce Web Service retourne la liste directement, chaque élément de la liste contient les champs suivant :

- "email" : email du joueur désirant jouer

6. Dans la limite du raisonnable. Ce n'est pas nécessaire d'être réactif à la seconde.

— "name" : nom du joueur désirant jouer
— "matchmakingId" : l'identifiant du matchmaking du joueur désirant jouer

3.3.3 Envoyer une requête

Il est possible d'envoyer une demande pour jouer avec un joueur présent dans la liste. Pour cela il faut l'identifiant du matchmaking du joueur à qui on veut envoyer la requête. Ce Web Service est le suivant :
`/matchmaking/request`

Ce Web Service nécessite 1 paramètre :
— "matchmakingId" : l'identifiant du matchmaking du joueur à qui on souhaite envoyer la requête
Ce Web Service retourne juste un message précisant que la requête est envoyée.

3.3.4 Accepter une requête

Lorsque le joueur a reçu une requête pour jouer, il peut l'accepter. Pour cela il faut utiliser le Web Service suivant :
`/matchmaking/acceptRequest`

Ce Web Service nécessite 1 paramètre :
— "matchmakingId" : l'identifiant du matchmaking du joueur ayant envoyé la requête
Cet identifiant doit nécessairement être associé à un joueur qui vous a envoyé une requête. Dans ce cas un match est créé, et les informations concernant ce match sont retournées :

— "player1" :
  — "name" : nom du premier joueur (celui ayant envoyé la requête)
  — "id" : identifiant du premier joueur
— "player2" : idem mais pour le joueur acceptant la requête

Dès qu'une requête est acceptée, les deux joueurs n'apparaissent plus dans la liste de matchmaking (désirant jouer). Il n'est possible de jouer qu'un seul match à la fois.

3.4 Jouer un match

Une fois un match créé il est possible de jouer la partie.

3.4.1 Récupération des informations du match

Le Web Service principal sur la gestion d'un match est celui permettant de récupérer les informations du match en cours :
`/match/getMatch`

Ce Web Service ne nécessite pas de paramètre et retourne toutes les informations nécessaires pour suivre l'évolution du match. Le contenu du résultat de ce Web Service est le suivant :

- "status" : état actuel du match
- "player1" :
  - "hp" : nombre de points de vie du joueur
  - "hand" : la liste des cartes en main du joueur
  - "board" : la liste des cartes sur le plateau du joueur
  - "deck" : la liste des cartes dans le deck du joueur
  - "cardPicked" : si le joueur a déjà pioché ce tour-ci
  - "turn" : vaut "true" si c'est à ce joueur de jouer
- "player2" : mêmes informations mais pour le joueur 2

Les informations envoyées sont différentes suivant le joueur qui regarde les informations du match. En effet, seul le joueur concerné verra les cartes dans sa main ("hand") et aucun des joueurs ne pourra voir les cartes dans les decks. Lorsque le contenu n'est pas visible (main du joueur adverse, deck, ...), ces champs prennent pour valeur le nombre de cartes présentes. Par exemple, le joueur 1 verra l'ensemble des cartes qu'il a dans sa main mais ne verra pas le contenu de la main de son adversaire. Il verra uniquement le nombre de cartes qu'il lui reste dans la main.

Ce Web Service devra être appelé régulièrement afin de connaître l'état du match en cours, notamment lorsque c'est au tour de l'adversaire pour voir ce qu'il a joué et si c'est à votre tour. [7]

Lorsque les cartes sont visibles (la main du joueur ou sur les plateaux) ces cartes contiennent un certain nombre d'informations les concernant. Ces informations proviennent directement de l'API proposée par RIOT [8] (proches de celles utilisées lors des précédents TPs). **Vous n'avez pas à interroger cette API** dans ce projet puisque un Web Service pour récupérer toutes les cartes existe.

3.4.2 Récupération des données de toutes les cartes

Ce Web Service permet de récupérer toutes les cartes et leurs informations afin de pouvoir (notamment) constituer son deck.

`/cards/getAll`

Ce Web Service retourne un tableau contenant l'ensemble des cartes disponibles dans le jeu (pour l'instant uniquement des champions). Un exemple de résultat :

Listing 4 – "Exemple de résultat pour le Web Service cards/getAll"

```
{   "status": "ok",
    "data": [{
                "_id": "58b56c9fd52c5035d8fba571",
                "id": 24,
                "key": "Jax",
```

7. Attention à ne pas surcharger le serveur ! Restez raisonnable. Un jeu de cartes n'a pas besoin d'être réactif à la seconde ...

8. `https://developer.riotgames.com/`

```
                "name": "Jax",
                "title": "Maitre d'armes",
                "image": {
                        "full": "Jax.png",
                        "sprite": "champion1.png",
                        "group": "champion",
                        "x": 144, "y": 48, "w": 48, "h": 48
                },
                "skins": [{
                        "id": 24000,
                        "name": "default",
                        "num": 0
                        }, {
                        "id": 24001,
                        "name": "Le Grand Jax",
                        "num": 1
                        },...
                ],
                "stats": {
                        "armor": 27.04,
                        "armorperlevel": 3,
                        "attackdamage": 61.97,
                        ...
                }},
                {
                "_id": "58b56c9fd52c5035d8fba572",
                "id": 37,
                "key": "Sona",
                "name": "Sona",
                ...
        }]
}
```


Ces informations sont aussi récupéré lors de l'appel du Web Service "getMatch". Effectivement les cartes présentent sur les plateaux et dans sa main sont visibles et donc toutes ces informations sont disponibles pour chacune d'entre elles.

3.4.3 Constitution du deck

Au tout début d'un match son statut est à "Deck is pending". Ce statut signifie que l'un des deux joueurs n'a pas défini son deck. Pour cela un Web Service est disponible pour indiquer les cartes que l'on souhaite utiliser lors de ce match :
`/match/initDeck`

Ce Web Service prend en paramètre ("deck") un tableau (au format JSON) contenant des objets ayant le champ "key" défini avec la clef d'un champion que l'on souhaite utiliser dans son deck. Par exemple une requête peut être :
`/match/initDeck?deck=[{key:"Jax"},{key:"Ivern"}]`[9]

Le deck ne peut contenir au maximum que 20 cartes et elles doivent toutes être différentes. Les données des cartes sont récupérées cotés serveur, il n'est donc pas nécessaire d'envoyer

9. Pensez aux URL Encode `https://www.w3schools.com/jsref/jsref_encodeuri.asp`

toutes les informations des cartes dans ce Web Service. Le deck est ensuite mélangé et vous est attribué comme deck pour le match en cours.

3.4.4 Piocher une carte

Une fois les decks des deux joueurs définis, le match peut commencer. Chaque joueur commence avec 4 cartes dans sa main et aucune carte sur son board. Chaque joueur jouant chacun son tour (le champ "turn" du Web Service "getMatch" permet de savoir à qui est le tour), plusieurs actions sont disponibles pour chaque tour. La première est la possibilité de piocher une carte. Cette action n'est disponible qu'une seule fois par tour (le champ "cardPicked" permet de savoir si une carte a déjà été piochée ou non). Dans le cas où une carte n'a pas déjà été piochée le joueur peut décider (quand il le souhaite pendant son tour) de piocher une carte de son deck pour l'ajouter dans sa main. Pour cela le Web Service suivant permet d'effectuer cette action :
`/match/pickCard`
Ce Web Service ne nécessite pas de paramètres. Si c'est votre tour et que vous n'avez pas déjà pioché une carte, il enlève une carte de votre deck et la met dans votre main. Ce Web Service retourne toutes les informations de la carte piochée. Lorsque vous faites un "getMatch" après avoir appelé ce Web Service vous pouvez observer que votre deck contient une carte en moins et que votre main en contient une en plus.

3.4.5 Jouer une carte champion

Lorsque c'est votre tour vous pouvez poser une carte champion de votre main sur votre board. De cette manière cette carte devient active (mais aussi visible par l'adversaire) pour qu'elle puisse attaquer. Le Web Service permettant de poser une carte est le suivant :
`/match/playCard`

Ce Web Service nécessite un paramètre "card" qui est la valeur de l'attribut "key" de la carte que vous souhaitez jouer. Par exemple nous pouvons faire :
`/match/playCard?card=Jax`

Cet appel aura pour effet de placer la carte "Jax" qui était dans votre main sur votre board.

Ce Web Service retourne la valeur du board et de votre main, bien que ces valeurs soient accessibles aussi par le Web Service "getMatch".

Vous ne pouvez avoir que 5 cartes champions simultanément sur votre board, mais il n'y a pas de restriction par tour (vous pouvez poser 5 cartes champions dés le début si vous le souhaitez). Par contre une carte posée sur le board durant ce tour ne pourra attaquer qu'à votre prochain tour.

3.4.6 Faire attaquer un champion

Lorsque vous avez une carte sur votre board et que celle-ci n'a pas été posée pendant ce tour, cette carte peut attaquer une carte adverse. Cette carte adverse doit être présente sur le board de l'adversaire. Pour cela vous utiliserez le Web Service suivante :
`/match/attack`

Ce Web Service prend deux paramètres : "card" et "ennemyCard". Ces paramètres sont (comme pour le Web Service "playCard") les attributs "key" des cartes concernées. L'attribut "card" contient la valeur de l'attribut "key" de la carte sur votre board qui va attaquer. L'attribut "ennemyCard" contient la valeur de l'attribut "key" de la carte sur le board de l'adversaire que vous souhaitez attaquer. Une carte ne peut attaquer qu'une fois par tour (l'attribut "attack" de votre carte permet de savoir si celle-ci a déjà attaqué ou non durant ce tour). Un exemple d'utilisation de ce Web Service peut être :
`/match/attack?card=Jax&ennemyCard=Sona`

En considérant que Jax est sur notre board et Sona sur le board adverse et que Jax n'a pas encore attaqué durant ce tour.

Pour déterminer les dégâts infligés au joueur le calcul suivant est effectué :
$ennemy.hp = ennemy.hp - (card.stats.attackdamage - ennemyCard.stats.armor)$
De cette manière les points de vies de l'adversaire sont réduits de la valeur restante de la soustraction entre la valeur d'attaque de la carte attaquante et la valeur de défense de la carte attaquée. La carte attaquée est alors supprimée du board adverse.

Ce calcul est fait uniquement si
$card.stats.attackdamage > ennemyCard.stats.armor$.
Si ces valeurs sont égales alors les deux cartes sont supprimées sans modification des points de vie. Si la valeur de défense est supérieure à la valeur d'attaque alors la carte attaquante est supprimée sans aucune modification des points de vie.

Si la valeur des points de vie de l'adversaire atteint 0 (ou moins) alors vous avez gagné le match.

3.4.7 Attaquer directement les points de vie de l'adversaire

Si le board de l'adversaire est vide (il n'y a aucun champion actif) alors vous pouvez attaquer directement les points de vie de l'adversaire. Dans ce cas là le Web Service à utiliser est le suivant :
`/match/attackPlayer`

Ce Web Service prend un paramètre "card" qui est la valeur du champ "key" de la carte attaquante. Cette carte ne doit pas avoir déjà attaqué durant ce tour et le board de l'adversaire doit être vide. La valeur de l'attaque de la carte est directement déduite des points de vie de l'adversaire.

Comme pour le Web Service précédent, si les points de vie de l'adversaire atteignent 0 ou moins alors vous avez gagné le match.

3.4.8 Fin du tour

Lorsque vous le souhaitez vous pouvez mettre fin à votre tour. Soit parce que vous n'avez plus d'action possible (vous avez déjà pioché, toutes les cartes ont attaqué et vous ne souhaitez pas poser plus de carte sur votre board) soit parce que vous ne souhaitez pas faire d'autres actions, alors vous pouvez mettre fin à votre tour. Pour cela il vous faut appeler le Web Service suivant :
`/match/endTurn`

Ce Web Service ne demande aucun paramètre. Il est par contre nécessaire que ça soit votre tour pour y mettre fin. Lorsque vous faites un "getMatch" après le "endTurn" alors la valeur du champ "turn" vous concernant prend "false" et la valeur du champ "turn" pour l'adversaire prend "true". C'est donc à l'adversaire de jouer.

3.4.9 Fin du match

Lorsqu'un des deux joueurs voit ses points de vie atteindre 0 ou moins alors le match est fini. A ce moment, les valeurs des champs "turn" des deux joueurs sont à "false". Ce n'est donc le tour de personne puisque le match est terminé. La valeur du champ "status" du match contient un texte pour savoir qui a gagné. Vous pouvez alors mettre fin au match en utilisant le Web Service suivant :
`/match/finishMatch`

Ce Web Service ne nécessite pas de paramètre. Il est par contre nécessaire que le match en cours soit terminé (que les deux attributs "turn" soit à "false"). Ce Web Service supprime le match courant (puisque terminé) et retire les deux joueurs du "matchmaking", ce qui leur permettra de le rejoindre de nouveau pour faire un nouveau match s'ils le souhaitent. Ce Web Service n'est appelé que par l'un des joueurs, le deuxième joueur qui appellera ce Web Service recevra une erreur (puisque le match sera déjà supprimé).

4 Nouvelles idées

Le travail qui vous est demandé dans ce projet est de développer une interface web utilisant ces Web Services pour implémenter un client pour "League of Stones". Les Web Services utilisés ne sont problablement pas exempt de bugs, ou vous avez peut être déjà pensé à certaines améliorations possibles que vous souhaitez mettre en oeuvre dans ce projet. C'est pour cela que chaque groupe devra désigner un responsable backend. Ce responsable aura pour but de lister les différentes requêtes provenant de son groupe, d'étudier la faisabilité du développement de ces améliorations cotés back et ensuite de les proposer à l'ensemble des responsables backend de tous les groupes. De cette manière il vous sera proposé d'utiliser vos compétences pour améliorer le backend du jeu en parallèle du développement du client. Ce développement du backend suivra les principes d'un développement communautaire. C'est pour cela que chaque modification devra être approuvé par la communauté (de tous les responsables backend).

Le backend est développé en Node.JS (en utilisant le framework Express), en utilisant une base de données MongoDB. Le code source du backend est disponible ici :
`https://github.com/blandine/League-Of-Stones`

Chaque responsable backend a donc pour responsabilité de s'approprier le code du backend pour pouvoir non seulement répondre aux questions lors du développement du client (par exemple s'assurer du fonctionnement précis d'un Web Service ou du contenu d'un résultat ou des conditions spécifiques, etc..) mais aussi pour pouvoir proposer des améliorations. Pour cela vous pourrez faire un fork de l'application (copier le repository GIT), faire les modifications qui vous semblent appropriées, et ensuite proposer de l'implémenter dans le projet en production (si la communauté approuve).

Ces technologies n'étant pas au programme des enseignements proposés dans votre formation, ce travail sera fait en étroite collaboration avec l'enseignant. Celui-ci faisant partie des "responsables backend", il aura donc son mot à dire lors de l'ajout de modifications. Il sera aussi en charge de la mise en production de nouvelles versions.

N'hésitez pas à proposer des améliorations (réalistes et faisables) pour faire évoluer le jeu !