Rapport de TPEA : Scrabblos - Jeu de mots sous la forme d'une blockchain

Réalisé par :

Benalla ANIS
George MATHIEU

13 décembre 2020

1 Introduction

Le but de ce projet est d'implémenter un jeu de mots semblable au scrabble via l'utilisation d'une blockchain. Le jeu est composé de deux entités majeures : les auteurs et les politiciens.

Les auteurs produisent les lettres. Dans une version du jeu en tour par tour, chaque auteur envoie une lettre par tour. Les politiciens récupèrent ensuite ces lettres et fabriquent des mots. On considère également un serveur central qui régule les échanges et transmet les lettres et les mots, selon la version du jeu implantée. Un algorithme de consensus détermine quel mot est insérer à la chaîne ou non.

Chaque mot inséré possède un score selon la somme des scores des lettres qu'il contient. Un politicien gagne des points lorsqu'un des mots qu'il a générés intègre la chaîne. Un auteur gagne des points lorsqu'une des lettres qu'il a envoyées est présente dans un mot intégré dans la chaîne.

On a développé deux versions pour le projet :
— Une version en roue libre dans laquelle les politiciens peuvent envoyer un mot n'importe quand
— Une version en tour par tour dans laquelle le serveur explicite clairement le tour actuel de la partie ce qui aide grandement à la synchronisation des entités

Les deux versions sont disponibles sur le dépot du projet.

Dans ces versions le jeu dure jusqu'au tour 50 avant de s'arrêter. A la fin les scores de chaque auteurs et politiciens sont affichés dans leur terminal respectifs.

1.1 Installation

Il est préférable d'être sous Linux pour exécuter le projet. Pour installer opam :
— `sudo apt-get install opam`
— `opam init`
— `eval $(opam env)`

Pour installer avec opam :
— Télécharger une des versions du projet et se mettre dans le répertoire `TPEA_Scrabblos`
— `opam pin add scrabblos ./`
— `opam install scrabblos`

Ensuite pour exécuter le projet :
— Compiler le programme : `make install`
— Il est nécessaire d'avoir un terminal d'ouvert par entité (serveur / politicien / auteur)
— Lancer le serveur : `scrabblos-server`, il est nécessaire de le lancer en premier
— Lancer un politicien : `scrabblos-politicien`
— Lancer un auteur : `scrabblos-author`

2 Questions

— **Question 0** : Comment s'assurer que l'auteur n'injecte pas plusieurs lettres dans un block

Pour un niveau donné, le politicien récupère la letterpool avec toutes les lettres envoyées dedans. Avant de créer son mot, il effectue un tri au préalable en vérifiant qu'au maximum une seule lettre a été envoyée par un auteur. Si un auteur en a envoyé plusieurs sur un même tour, on ne garde que la première. Cependant, on n'empêche pas un auteur d'en envoyer plusieurs, elles ne seront juste pas comptabilisées lors de la création d'un mot.

```
(* fonction auxiliaire pour check_letters_one_per_author *)
let rec check_letters_one_per_author_aux (lStore : letter list)
(author : Id.author_id) : letter list =
  match lStore with
  | [] -> []
  | x::xs -> if (x.author = author) then check_letters_one_per_author_aux xs author
  else x::( check_letters_one_per_author_aux xs author)

(* Supprime les lettres qui appartiennent au meme auteur *)
let rec check_letters_one_per_author (lStore : letter list) : letter list =
  match lStore with
  | [] -> []
  | x::xs -> x::( check_letters_one_per_author ( check_letters_one_per_author_aux xs x.author)
)
```

— **Question 1** : Implanter un auteur et un politicien pour la version tour par tour

L'implémentation tour par tour a été réalisée. Il est possible de lancer plusieurs politiciens et plusieurs auteurs dans n'importe quel ordre. L'implantation de la méthode run du politicien suit globalement la même structure que celle de l'auteur.

On a ajouté pour l'auteur et le politicien un système qui permet d'éviter l'arrêt du programme lors de la connexion d'une des entités au programme en cours d'exécution. Lors d'un appel au serveur demandant la letterpool ou à la wordpool, l'entité attendait une réponse concernant l'objet demandé dans la requête, cependant il pouvait recevoir n'importe quel autre message, ce qui provoquait une erreur et l'arrêt de l'entité. Dans le cas de l'auteur, étant déjà enregistré, le serveur va attendre qu'il envoie une lettre, provoquant ainsi l'arrêt du programme. Pour y remédier, on relance la réception d'un message jusqu'à obtenir un message d'un type convenable.

```
(* Get initial wordpool *)
let getpool = Messages.Get_full_wordpool in
Client_utils.send_some getpool ;
let rec wait_wordpool () : Messages.wordpool =
  match Client_utils.receive () with
  | Messages.Full_wordpool wordpool -> wordpool
  | Messages.Diff_wordpool diff_wp -> diff_wp.wordpool
  | _ -> wait_wordpool ()
in
let wordpool = wait_wordpool () in
```

Le serveur change de tour lorsque tous les auteurs enregistrés ont envoyé une lettre. Les auteurs envoient une lettre lorsqu'ils reçoivent la notification de changement de tour. Les politiciens envoient aussi un mot à la réception d'un changement de tour, ils produisent un mot selon les critères de validité suivants :

— Faisant référence (par son hash) à un mot prédécesseur valide
(fonction `make_word_on_blockletters`)

— Ne contenant que des lettres injectées avec une référence au mot prédécesseur

```
(* Supprime les lettres qui n'ont pas la meme reference au mot predecesseur *)
val check_letters_same_hash : word -> letter list -> letter list ;
```

— Ne contenant pas plus d'une lettre venant d'un même client (voir Question 0)

— Existant dans un dictionnaire

```
(* Choisis le dictionnaire a utiliser pour construire un mot a partir du letter
store *)
val get_word_from_dict letter list -> letter list option

(* Recupere un mot en cherchant dans tous les dico inferieurs si aucun mot est
trouve dans un dico superieur *)
val get_dictionnary_aux : int -> letter list -> letter list option

(* Recupere un mot dans le dictionnaire en utilisant le contenu du letter store *)
val get_word_from_dict_aux : string list -> letter list -> letter list -> letter
list

(* Verifie que le mot courant a toutes ses lettres dans le letter store *)
val check_dict_word_valid :char list -> letter list -> letter list -> list option

(* Verifie que la lettre courante est dans le letter store *)
val check_letter_in_store : char -> letter list -> letter option
```

On a aussi ajouté qu'un mot ne contient que des lettres envoyées lors du même tour.

On note que lors de la recherche dans un dictionnaire, le politicien cherche à trouver le mot le plus long possible selon les lettres qu'il a à sa disposition. Si un mot comprenant toutes les lettres du letter_store est trouvé, il est immédiatement renvoyé. Selon la taille du letter_store, on commence à chercher dans le dictionnaire contenant les mot les plus grands (par exemple, si on a 40 lettres, on commence à chercher dans `dict_100000_25_75.txt`. Si aucun mot n'est trouvé dedans, on cherche dans le dictionnaire immédiatement inférieur (`dict_100000_5_15.txt` dans notre exemple). Si aucun mot n'est trouvé dans aucun dictionnaire, on renvoie `None` et l'auteur n'ajoute pas de mot pour ce tour.

L'algorithme utilisé n'a pas d'élément aléatoire. Les politiciens ayant le même algorithme de recherche et accès aux mêmes lettres, ils enverront le même mot. Cependant seul un des mots sera gardé dans la chaîne, c'est le premier qu'on reçoit.

Le score d'une entité est calculé lorsqu'elle reçoit un mot. Pour l'auteur, si le mot reçu appartient à la blockchain (nouvelle tête) et contient une lettre ayant le même identifiant que l'auteur, son score est incrémenté de la valeur de la lettre (voir Question 2).

Pour le politicien, on a modifié l'instruction du serveur lors de l'envoie d'un message `Inject_word` dans `answerer` : un politicien reçoit également les mots qu'il envoie. Ainsi, si le mot reçu appartient à la blockchain et a été envoyé par ses soins, son score est incrémenté de la valeur du mot.

Lorsqu'une des entités atteint la limite de tour, elle envoie un message `Fin_de_Partie` au serveur qui l'envoie aux autres entités. Lorsque ce message est reçu, on affiche le score final de l'entité et on quitte la boucle de jeu provoquant la fin de la partie.

— **Question 2** : Description du consensus

Le consensus est concentré autour d'une fonction `head` qui permet à un auteur ou un politicien de récupérer la tête de la chaîne.

```
val head : int -> Store.word_store -> word option
```

Elle prend en paramètre le niveau de la chaîne et le wordstore. Si aucun niveau est donné (c'est un paramètre optionnel), on renvoie `None`. Si le wordstore est vide (au début), on renvoie le `genesis word` qui est un mot prédéfini dans le programme comme étant la première tête.

Si les deux paramètres sont présents, on va déterminer le niveau le plus récent parmi les mots du wordstore.

```
val get_latest_period : word list -> int -> int
```

Ensuite, on détermine les mots du wordstore ayant ce niveau le plus récent et on garde celui qui a le meilleur score. Le score est déterminé en faisait la somme des valeurs des lettres composant un mot selon les règles du Scrabble anglais.

```
val word_score : word -> int
```

On a également ajouté une fonction qui vérifie la signature des lettres au cas où le politicien envoie un mot avec des lettres qui n'ont pas été émises véritablement par un auteur.

```
val check_letters_signature : letter list -> bool -> bool
```

— **Question 3** : Implémentation en roue libre

Pour paramétrer le serveur en roue libre, on change le paramètre `turn_by_turn` dans `main_server`. Ainsi aucun message de type `Next_turn` ne sera envoyé par le serveur.
Le point important à implanter était de synchroniser les niveaux des entités sans avoir à utiliser les tours. Les auteurs et politiciens ont leur niveau mis à jour par rapport à un mot reçu lorsque celui-ci est inséré dans la blockchain.
De plus, les mots et lettres étaient envoyés par rapport aux tours ce qui n'est plus possible. Désormais ils peuvent être envoyés à n'importe quel moment. Pour l'auteur on envoie une lettre lorsqu'on reçoit un mot, peu importe s'il est inséré dans la chaîne ou non. Pour le politicien on envoie un mot à chaque fois qu'on reçoit une lettre.

C'est pourquoi dans notre implémentation en roue libre, tous les mots ont une taille d'une lettre. Ceci est corrélé au choix qu'on a fait concernant le moment où le politicien envoie son mot.

Il a également été nécessaire de modifier le concensus afin de vérifier que les lettres du mot retenu avaient bien le bon niveau

```
val check_letters_level : int -> letter list -> bool -> bool
```

— **Question 4** : Résistance aux attaques

On a posé beaucoup de responsabilité sur le politicien supposé honnête, concernant la vérification du mot qu'il propose. On pourrait imaginer un scenario ou le politicien est malhonnête et voudrait maximiser ses chances de gagner, dans ce cas il va pouvoir proposer un mot très long composé de plusieurs fois de la même lettre du letter pool ayant le score le plus élevé.

Dans notre consensus actuel on vérifie que les lettres sont d'un niveau cohérent et que les signatures sont correctes, comme décrit précédemment.

La solution consisterait à avoir plus de vérification dans le consensus.

— **Question 5** : Modifier le système pour ne plus utiliser de serveur centrale

Il faudrait faire fonctionner les acteurs sans avoir à passer par les messages qui sont distribués par le serveur et fonctionner avec un réseau pair-à-pair.

En proof of work, la solution consisterait à ajouter pour les politiciens une difficulté sur le minage d'un mot, cela correspond à la longueur du mot à miner, le premier politicien qui réussit aura son mot retenu et ajouté à la blockchain, la difficulté sera variable en fonction du temps qu'auront mis les politiciens à trouver un mot au niveau précédant.

Lorsqu'un politicien propose un mot valide, il deviendrait le leader. Le prochain à proposer un mot par-dessus deviendra le nouveau leader. En cas de fork le consensus devrait rediriger le politicien vers la chaîne la plus longue qui, implicitement, a nécessité plus de travail.

— **Question 6** : Modification vers Proof of Stake

On peut envisager d'intégrer un système de vote où le coefficient du vote d'un politicien sera en proportion du nombre de blocks qu'il détient dans la blockchain, le block ayant le plus de votes sera retenu et ajouté à la blockchain. Cela est plus utilisé dans des cas de prise de décisions liées à l'avenir de la blockchain du jeu.

En cas de fork, on pourrait également utiliser la chaîne la plus longue de manière similaire au Proof of Work.