



BilleGateCoin - BGC
Scattered Corporation

RAPPORT DE SOUTENANCE N°2

réalisé par

aurele.oules • leo.gervoson • raphael.brenn

Projet EPITA 2020

Table des matières

1	Introduction	2
1.1	Point	2
1.2	Avancée du projet	3
2	CLI	3
3	Minage / Proof of Work	3
4	Récompenses des mineurs	4
5	Application des règles consensus	5
6	World state	5
7	Memory pool	7
7.1	Ordre de priorité (GetBestContracts)	8
8	Communication autonome des nodes	9
8.1	Enregistrement des nodes	9
9	Fonctionnement en continu de la blockchain	9
10	Communication entre Unity et la blockchain	10
10.1	Réseau local	11
11	Jeu	11
11.1	Menus	12
11.2	Plan général	12
11.2.1	Menu principal	13
11.2.2	Menu jouer	13
11.3	Création de la partie	14
11.4	Déplacement de la bille	14
11.5	Déterminisme	15
11.6	Obstacles	16
12	Site web	17
13	Conclusion	20

1 Introduction

BilleGateCoin est une plateforme décentralisée pour un jeu de billes. Dans ce deuxième rapport de soutenance nous allons évoquer les différentes fonctions et tâches implémentées dans notre projet. Nous discuterons également de ce qu'il reste à faire.

1.1 Point

Nous avons terminé la génération complète d'un porte-feuille. C'est-à-dire la génération d'une clé privée, le calcul de la clé publique ainsi que l'adresse associée. Ce porte-feuille est ensuite stocké dans un **wallet.dat** et encrypté avec un mot de passe choisi par l'utilisateur.

L'utilisateur peut exécuter trois types de contrats sur la blockchain. Dans un premier temps, le StartContract permet de démarrer une partie avec l'accord d'un deuxième joueur. Lorsque la partie commence, les billes mises en jeu sont mises dans un espace verrouillé auquel aucun joueur n'a accès tant que la partie n'est pas terminée. Ensuite, le joueur peut lancer sa bille avec un ThrowContract, celui-ci ne contient uniquement un vecteur force à trois dimensions (X, Y, Z) et sa signature (et des informations plus techniques).

Le joueur a également la possibilité d'échanger directement des billes sans avoir à les jouer, ce contrat est le TransactionContract. Sa structure est quasiment identique au StartContract, mais l'échange est instantané.

Un algorithme de minage a été implémenté avec une difficulté variable, mais nous l'avons amélioré entre-temps.

Un module de persistance a également été ajouté pour stocker les blocks et les inventaires de chaque joueur, nous utilisons la base de données clé/valeur LevelDB développée par Google.

La communication peer-to-peer entre les différentes nodes est opérationnelle et permet aux nodes d'échanger leurs blocks et transactions pour former la plus longue chaîne valide.

Une première version du terrain de jeu avait été réalisée avec quelques designs de billes. Nous avons également pris de l'avance sur la construction du site web et de la documentation.



1.2 Avancée du projet

Depuis la dernière présentation du projet nous avons rajouté beaucoup de fonctionnalités, nous allons les présenter dans ce rapport.

Nous avons établi les règles consensus du réseau, ainsi que les récompenses des mineurs, c'est-à-dire l'économie de BilleGateCoin. Un **World State** permettant de garder en mémoire l'inventaire de chaque joueur a été mis en place, fonctionnant sur le principe du World State d'Ethereum.

La Memory Pool permettant de stocker les contrats non validés est opérationnelle et fonctionne sur la couche réseau. La communication entre le coeur de BilleGateCoin (Blockchain) et Unity est établi grâce à la couche réseau également.

Nous avons pris un peu de retard sur la partie Unity car un membre de notre groupe est parti, nous avons notamment été focalisé sur cette partie du projet.

2 CLI

Nous avons rajouté quelques commandes dans le CLI de BilleGateCoin.

- *showblocks* : affiche la liste des blocks
- *mine* : lance le processus de minage de block
- *inventory* : affiche la liste des billes d'un joueur

3 Minage / Proof of Work

Nous avons amélioré l'algorithme de minage des billes. Un block doit contenir un maximum de 256 KB de contrats et doit être miné environ toutes les 2 minutes pour assurer des parties fluides. La difficulté de minage doit donc être adaptée en fonction de la puissance de calcul du réseau. Si le réseau mine trop de blocks trop rapidement alors cette difficulté augmente, dans le cas contraire elle diminue.

Tous les 1440 blocks, ce qui équivaut à environ 48h, cette difficulté est ajustée automatiquement par chaque node du réseau BilleGateCoin avec la formule suivante :

$$\text{nouvelle difficulté} = \text{difficulté} \times \frac{\text{temps attendu}}{\text{temps réel}}$$



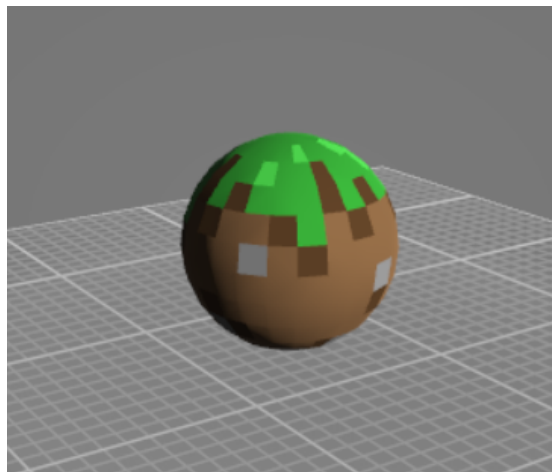
4 Récompenses des mineurs

Les mineurs sont désormais récompensés de leur travail avec des billes. Pour qu'un mineur soit récompensé il doit inclure dans le block qu'il mine un TransactionContract contenant sa récompense, le réseau validera si ce contrat est valide. Il existe 5 type de billes de 6 couleurs différentes, voir docs.scattereredcorp.tech pour plus d'informations. Pour chaque block miné, un mineur est récompensé de :

- 256 billes Earth
- 24 * 6 billes Elastic
- 8 * 6 billes Layers
- 8 * 6 billes Ribbon
- 2 * 6 billes Stripes
- 6 billes Whirlwind
- 1 bille Soccer

```
public static Marble Earth = new Marble("Earth", Type.Earth, 256, true);
public static Marble Elastic = new Marble("Elastic", Type.Elastic, 24);
public static Marble Layers = new Marble("Layers", Type.Layers, 8);
public static Marble Ribbon = new Marble("Ribbon", Type.Ribbon, 8);
public static Marble Stripes = new Marble("Stripes", Type.Stripes, 2);
public static Marble Whirlwind = new Marble("Whirlwind", Type.Whirlwind, 1);
public static Marble Soccer = new Marble("Soccer", Type.Soccer, 1, true);
```

Une nouvelle bille a été ajoutée, elle n'a pas beaucoup de valeur et sert principalement de récompense pour les mineurs. Sa texture est dérivée de celle d'un autre jeu connu Minecraft.



le visuel de la nouvelle bille "dirt"

Tous les 5000 blocks, environ une semaine, ces récompenses sont divisées par deux : **Block reward halving**, jusqu'à que la totalité des billes soient distribuées aux joueurs. C'est un système qui fonctionne sur la déflation de l'économie du jeu.



Les billes très communes étant distribuées en abondance, il y aura plus d'offre que demande ce qui lui donnera une valeur très faible, contrairement aux billes légendaires qui auront plus de demande que d'offres.

De plus, la distribution des récompenses des mineurs se fait à l'aide de TransactionContracts où un joueur inexistant, mais valide, donne la récompense au mineur, en échange de zéro billes, un peu comme lorsque la banque centrale européenne délivre un chèque à partir de fonds inexistants aux banques. Dans le modèle de BilleGateCoin en revanche, le taux d'inflation est prédéterminé et est divisé en deux toutes les semaines.

Ce mineur a la liberté de jouer ces billes, les vendre, ou les garder dans sa collection.

5 Application des règles consensus

Nous avons mis en place les règles du jeu et du protocole dans notre implémentation. Un block ne peut dépasser 256 KB, ils doivent être minés toutes les 2 minutes en moyenne. Les récompenses des mineurs sont divisées par deux tous les 5000 blocks (48h) et la difficulté de minage est ajustée toutes les 48h. Lorsqu'une node reçoit un contrat ou un block, celui-ci doit s'assurer de la validité de toutes ces règles, ainsi que la validité des signatures des joueurs.

6 World state

Lorsqu'une node reçoit un block, celle-ci va lire tous les contrats gravés dans le block. Le protocole va s'occuper d'indexer pour chaque adresse le nombre de billes qu'elle possède, gagne, ou perd. Ce processus d'indexation se nomme le **World State**. Ce principe fonctionne comme celui d'Ethereum, la quantité de billes pour chaque adresse est stocké sur le disque dur de chaque node, à ne pas confondre avec le système d'UTXO du Bitcoin. Ce système possède des avantages et des désavantages. Il est beaucoup plus simple à implémenter, plus rapide, mais ne garantit pas aux utilisateurs l'anonymat.

Nous avons implémenté ce **World State** car il serait trop long de parcourir tous les contrats de chaque block pour obtenir des informations sur un joueur. Ce **World State** n'est pas directement stocké sur la blockchain, mais il est fabriqué par chaque ordinateur du réseau individuellement au fur et à mesure du temps, il peut donc être entièrement reconstruit à partir des blocks de la chaîne, à n'importe quel moment.

Lors de la synchronisation initiale de la chaîne, le world state se génère automatiquement.



```

public static void RedindexWorldState(Block block) {
    for(uint i = 0; i < block.Contracts.Length; i++) {
        IContract contract = block.Contracts[i];

        switch(contract.Type) {
            case (byte) Contracts.ContractType.TransactionContract: {
                TransactionContract tx = (TransactionContract) contract;
                ExchangeMarbles(tx);
                break;
            }

            case (byte) Contracts.ContractType.StartContract: {
                StartContract tx = (StartContract) contract;
                // Contract is a StartContract
                // This means we need put put the player's marbles in a lock space

                LockMarbles(tx.PlayerOnePubKeyHash, tx.PlayerOnePlacement);
                LockMarbles(tx.PlayerTwoPubKeyHash, tx.PlayerTwoPlacement);
                break;
            }
        }
    }
}

```

```

+ build git:(master) ./BGC inventory --address xg3S73psPZR5MFycvrKCTiNRv4Mi8XKdps
Type: Earth
Amount: 3584
Color: None

Type: Whirlwind
Amount: 14
Color: Blue

Type: Whirlwind
Amount: 14
Color: Green

Type: Whirlwind
Amount: 14
Color: Orange

Type: Whirlwind
Amount: 14
Color: Red

Type: Whirlwind
Amount: 14
Color: Purple

Type: Whirlwind
Amount: 14
Color: Dark

Type: Elastic
Amount: 336
Color: Blue

Type: Elastic
Amount: 336
Color: Green

Type: Elastic
Amount: 336
Color: Orange

```

Lorsqu'un joueur lance une partie avec un StartContract, toutes les billes mises en jeu



sont déplacés dans un espace verrouillé auquel aucun joueur n'a accès tant que la partie n'est pas terminée. Une fois la partie terminée, toutes les billes reviennent au vainqueur de celle-ci.

En revanche, lors d'un simple échange de billes (TransactionContract), les billes sont simplement échangées entre les joueurs.

7 Memory pool

La memory pool est une liste de contrats stockés non confirmés. Les blocks sont formés à partir des contrats en attente dans la memory pool. Les contrats ajoutés à la memory pool sont, comme les nouveaux blocks, diffusés récursivement aux nodes. C'est-à-dire que chaque node va transmettre aux nodes qu'elle connaît le nouveau block, jusqu'à que tout le réseau soit synchronisé. Ce processus devrait prendre moins de 5 secondes pour synchroniser tout le réseau avec des nodes localisées tout autour du globe.




```

public static class Mempool {
    private static List<IContract> Pool { get; } = new List<IContract>();

    public static void AddContract(IContract contract) {
        Pool.Add(contract);
    }

    public static void RemoveContracts(IEnumerable<IContract> contracts) {
        foreach (IContract c in contracts) {
            Pool.Remove(c);
        }
    }

    public static List<IContract> GetBestContracts(int count) {
        // Minus: decreasing order
        Pool.Sort((contract, contract1) =>
            -contract.Fee.TotalValue().CompareTo(contract1.Fee.TotalValue()));

        List<IContract> contracts = new List<IContract>();

        for (int i = 0; i < count && i < Pool.Count; ++i)
        {
            contracts.Add(Pool[i]);
        }

        return contracts;
    }
}

```

7.1 Ordre de priorité (GetBestContracts)

Les mineurs ont pour but de s'enrichir, en plus de récupérer le **Block reward**, ils reçoivent les taxes de chaque contrat qu'ils minent. Alors ils doivent optimiser les contrats qu'ils gravent dans la blockchain. Les contrats sont triés par ordre de priorité. Cet ordre de priorité est établi à partir de la valeur de la taxe payée pour le minage du contrat. Cette taxe est calculée en faisant la somme des valeurs des billes constituant la taxe, selon la formule suivante :

$$Valeur_{bille} = \frac{Echelle}{Exemplaires_{bille}}$$

La valeur de chaque bille est donc proportionnelle à sa rareté.

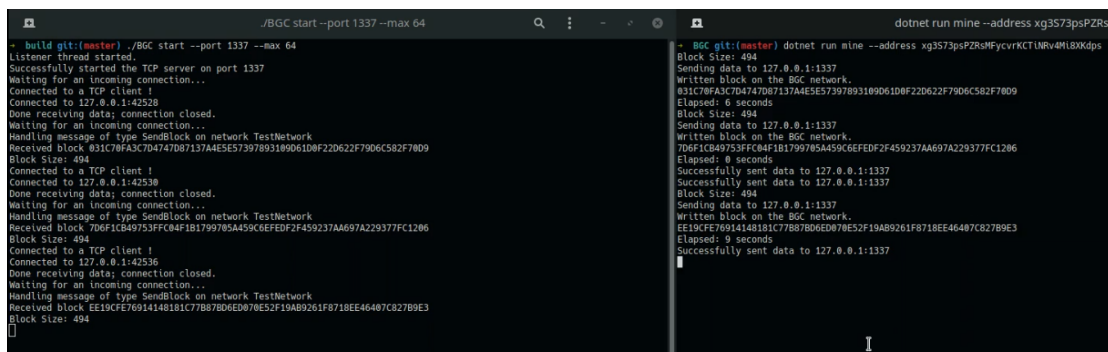


8 Communication autonome des nodes

Chaque node possède une liste de nodes dans un fichier texte. De cette manière, à chaque fois qu'une opération est effectuée (ajout de block, ajout de node, etc.), les nodes la propagent récursivement entre elles jusqu'à ce que tout le monde l'ait reçue.

8.1 Enregistrement des nodes

Le téléchargement d'une liste de nodes est nécessaire pour initialiser une connexion à la blockchain. Une fois téléchargée, il suffit de lancer une propagation d'ajout de node (RegisterNode) récursive dans le réseau pour que toutes les nodes prennent connaissance de l'existence d'une nouvelle node.



```
./BGC start --port 1337 --max 64
- build git:(master) ./BGC start --port 1337 --max 64
Listener thread started.
Successfully started the TCP server on port 1337
Waiting for an incoming connection...
Connected to a TCP client !
Connected to 127.0.0.1:42528
Done receiving data; connection closed.
Waiting for an incoming connection...
Handling message of type SendBlock on network TestNetwork
Received block 831C70FA3C7D4747D87137AAE5E57397893109D61D8F22D622F79D6C82F78D9
Block Size: 494
Connected to a TCP client !
Connected to 127.0.0.1:42539
Done receiving data; connection closed.
Waiting for an incoming connection...
Handling message of type SendBlock on network TestNetwork
Received block 706F1C849753FFC84F1B1799785A459C6EFEDF2F459237AA697A229377FC1286
Block Size: 494
Connected to a TCP client !
Connected to 127.0.0.1:42536
Done receiving data; connection closed.
Waiting for an incoming connection...
Handling message of type SendBlock on network TestNetwork
Received block EE19CFE76914148181C77887806ED078E52F19A89261F8718EE46407C827B9E3
Block Size: 494

- BGC git:(master) dotnet run mine --address xg3573psPZRsMfycvrKCT1NRv4M18XKdps
Block Size: 494
Sending data to 127.0.0.1:1337
Written block on the BGC network.
831C70FA3C7D4747D87137AAE5E57397893109D61D8F22D622F79D6C82F78D9
Elapsed: 6 seconds
Sending data to 127.0.0.1:1337
Written block on the BGC network.
706F1C849753FFC84F1B1799785A459C6EFEDF2F459237AA697A229377FC1286
Successfully sent data to 127.0.0.1:1337
Block Size: 494
Sending data to 127.0.0.1:1337
Written block on the BGC network.
EE19CFE76914148181C77887806ED078E52F19A89261F8718EE46407C827B9E3
Elapsed: 9 seconds
Successfully sent data to 127.0.0.1:1337
```

9 Fonctionnement en continu de la blockchain

Le coeur de BilleGateCoin mine en permanence des blocks, même s'ils sont vides. Dès qu'une node possède de nouvelles données, elle doit immédiatement les distribuer à toutes les autres nodes qu'elle connaît. La blockchain fonctionne ainsi en continu, comme un serveur. La boucle de fonctionnement est assez simple :



```

public class Callback {
    public static bool exitRequested = false;

    public static void Run(int port, int maxClients) {
        Listener listener = new Listener(port, maxClients);

        listener.StartListening();

        while (!exitRequested)
        {
            listener.QueueMutex.WaitOne();
            if (listener.IncomingQueue.Count == 0)
            {
                listener.QueueMutex.ReleaseMutex();
                Thread.Sleep(50);
                continue;
            }

            NetworkMessage msg = listener.IncomingQueue.Dequeue();
            listener.QueueMutex.ReleaseMutex();

            MessageHandler.Handle(msg, ref exitRequested);
        }
    }
}

```

Le serveur listener fonctionnant en parallèle afin que le réseau reste réactif, un mutex est utilisé pour protéger la file de messages entrants. Les messages dans la file sont traités un à un par MessageHandler, qui est en fait un grand switch sur la commande du message reçu.

10 Communication entre Unity et la blockchain

Le jeu développé en Unity est l'interface utilisateur qui communique avec le protocole BilleGateCoin implémenté en C# également. Nous avons réfléchi longtemps à la manière de procéder. Nous avons trouvé plusieurs solutions mais elles étaient soit trop complexes, soit ne fonctionnait uniquement sur Linux ou sur Windows. Nous avons décidé d'utiliser le serveur TCP/IP en local développé par Léo.

Lorsqu'un joueur souhaite créer une nouvelle partie, il choisit les billes qu'il veut jouer



sur l'interface développée avec Unity puis envoie toutes ces données au coeur de BilleGateCoin, le contrat est signé puis diffusé sur le réseau entier. Le joueur peut également récupérer des informations sur son inventaire en communiquant avec le World State en TCP/IP local.

Le coeur de BilleGateCoin communique avec tous les ordinateurs du réseau, quant au processus Unity, il ne sert uniquement à afficher les données.

10.1 Réseau local

Le jeu Unity communique avec le coeur (la blockchain) par le réseau local. Le module réseau du jeu est une version modifiée de celui du coeur, en gardant le serveur "Listener" et le modèle de messages COMMAND / SIZE / MAGIC / PAYLOAD.

Les commandes utilisées par le jeu sont spécifiques et commencent toutes par Unity pour les distinguer (ex : UnityGetInventory), car leurs formats et utilisations sont différentes. Les ports utilisés par Unity et la blockchain pour communiquer entre eux sont hardcodés à l'avance, sur des valeurs arbitraires.

On utilise un contrôleur global pour qu'une seule instance du serveur Listener serve à toutes les scènes Unity :

```
public class GlobalController : MonoBehaviour {
    private static Listener listener;

    internal static Listener Listener { get => listener; set => listener = value; }

    // Start is called before the first frame update
    void Awake()
    {
        DontDestroyOnLoad(this.gameObject);

        Listener = new Listener(27496, 2);
        Listener.StartListening();
    }

    private void OnDestroy()
    {
        listener.StopListening();
    }
}
```

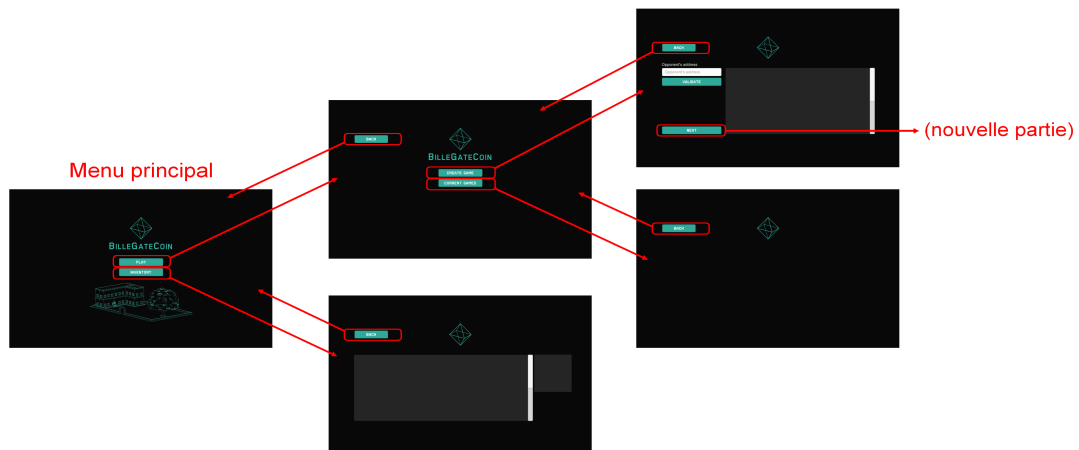
11 Jeu



11.1 Menus

11.2 Plan général

Les menus du jeu sont organisés selon l'architecture suivante :



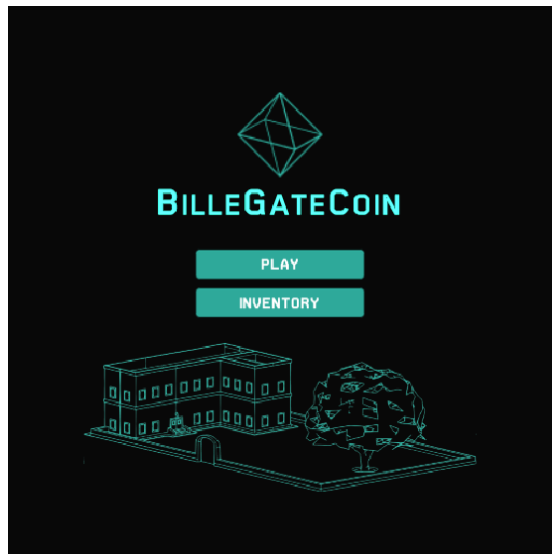
Depuis la dernière soutenance, les principaux menus (page principale – pages de création / reprise de parties – inventaire) ont été créés et sont fonctionnels, dans la limite de l'avancement du reste du projet bien sûr.

Chaque page du menu est construite selon le même schéma :

- - les visuels sont à dominante turquoise sur fond noir, avec des détails supplémentaires en blanc si nécessaire (ce sont les principales couleurs de la palette qu'on a choisi pour le jeu, on les retrouve notamment sur le site internet)
- - si la page est un menu vers d'autres pages de ce menu, alors les boutons qui mènent à chacune de ces pages sont disposés à la verticale au centre de l'écran, et on retrouve au-dessus de ceux-ci le logo et le nom du jeu
- - sinon, la page n'est surmontée que du logo (sans le nom), et le contenu de la page dépend de la(les) fonctionnalité(s) qu'elle remplit
- - toutes les pages sauf la page principale ont un bouton « retour » vers la page précédente.



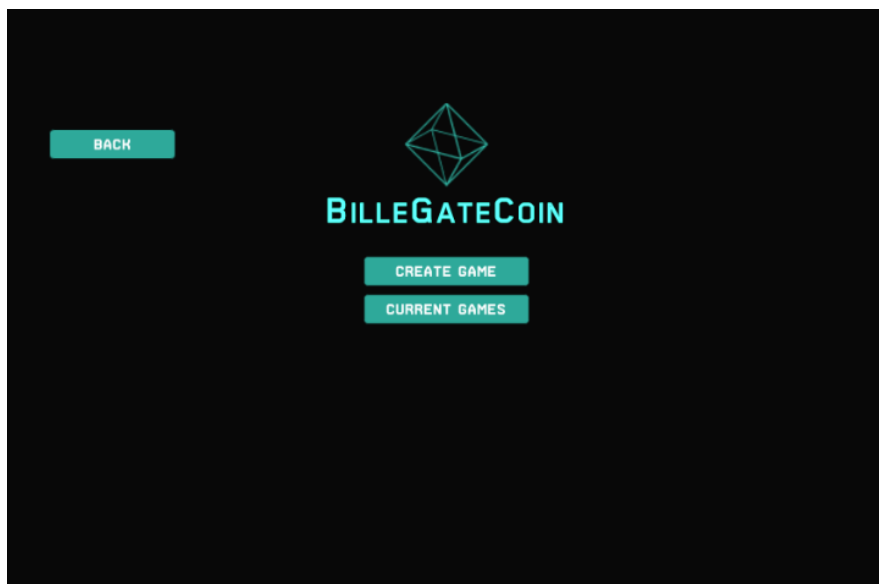
11.2.1 Menu principal



Le bouton **PLAY** amène à un nouveau menu, pour créer une partie, ou continuer les parties en cours.

Le bouton **INVENTORY** permet de visualiser l'inventaire du joueur.

11.2.2 Menu jouer



Ce menu permet de créer une nouvelle partie ou continuer celles en cours.



11.3 Création de la partie

Lorsqu'un joueur démarre une nouvelle partie, un **StartContract**, celle-ci doit être minée et incluse dans un block. Une fois le **StartContract** miné, la partie démarre avec un **seed**, qui est le hash du block dans lequel le contrat est, qui permet de générer la position de tous les obstacles et les positions d'origine des billes.

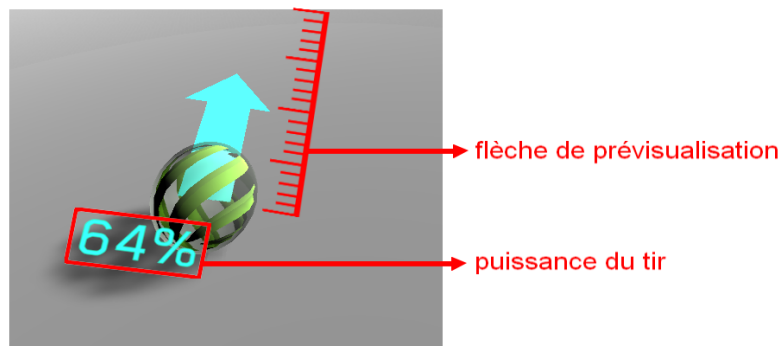
Pour un même seed donné, les positions des obstacles et des billes seront les mêmes. Cela permet de conserver l'état de la partie sans avoir à sauvegarder chaque position dans la blockchain.

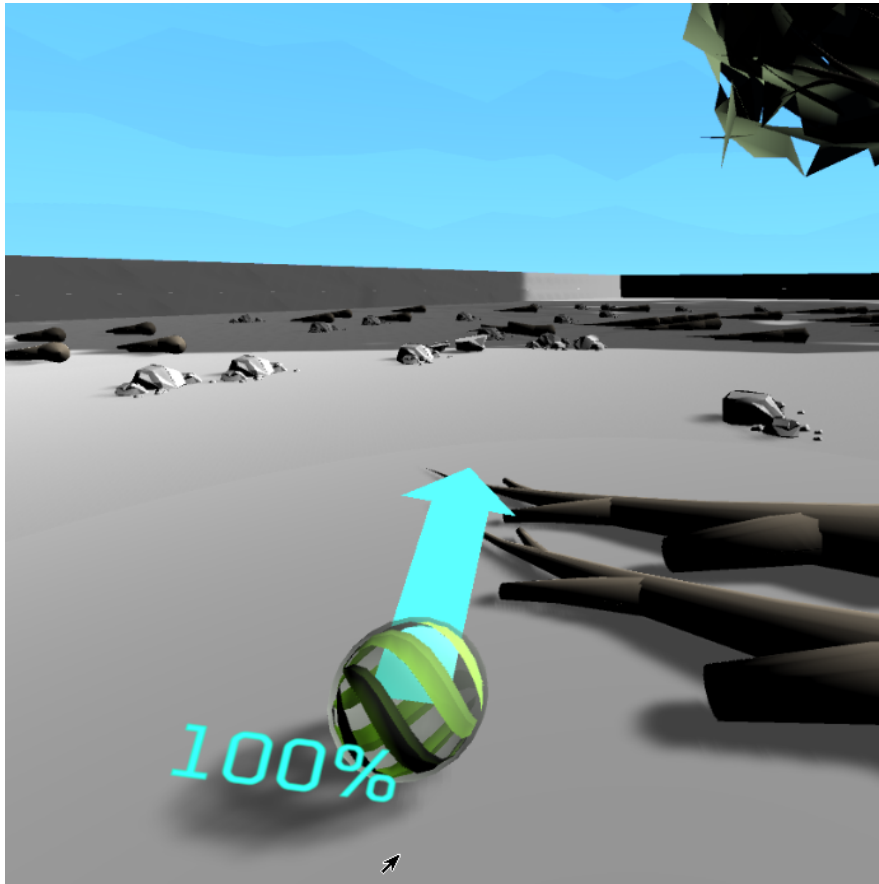
11.4 Déplacement de la bille

Lors d'une partie, le but est de lancer une de ses billes quelque part sur le terrain, en essayant de percuter une ou plusieurs des billes de l'adversaire afin de les remporter.

La mécanique de tir est simple : on pivote la caméra (qui sert à regarder autour de soi, mais également à viser) en déplaçant la souris horizontalement avec le **click droit** enclenché, et lorsque l'on est satisfait de la direction choisie on déplace la souris verticalement, vers le bas, avec le **click gauche** enclenché, afin de régler la puissance avec laquelle on voudra lancer la bille. Une interface spécifique apparaît alors, composée d'un nombre (pourcentage correspondant à la puissance actuelle par rapport à la puissance maximale disponible) et d'une flèche (dont la taille dépend elle aussi de la puissance actuelle) qui pointe en direction du tir.

Voilà à quoi ressemble cette interface :





Pour lancer la bille il suffit de cliquer dessus, la tirer avec une certaine force et la lâcher.

11.5 Déterminisme

Nous avons passé plusieurs heures de recherches sur le déterminisme du mouvement de la bille. Pour un lancé de bille précis, la bille doit toujours atterrir à la même position, cela permet de reconstituer la partie en sauvegardant uniquement les vecteurs de lancés dans la blockchain au lieu de sauvegarder la position de chaque objet. Les obstacles sont également placés semi-aléatoirement sur le plateau, c'est-à-dire que la position de chaque obstacle est prédéfini à partir du seed de la partie (le hash du block dans lequel la partie est contenue) mais sa position apparaît aléatoire. Nous avons réussi à obtenir des résultats constants pour un même ordinateur, mais ça se complique lorsqu'on change de processeur ou même de compilateur...

Chaque ordinateur gère différemment les nombres flottants, en particulier les processeurs de type x86 et ARM, car ils arrondissent différemment ces nombres. Au bout de plusieurs lancés de billes, la position de la bille se décale de quelques pixels ce qui peut causer de grands problèmes pour de longues parties.

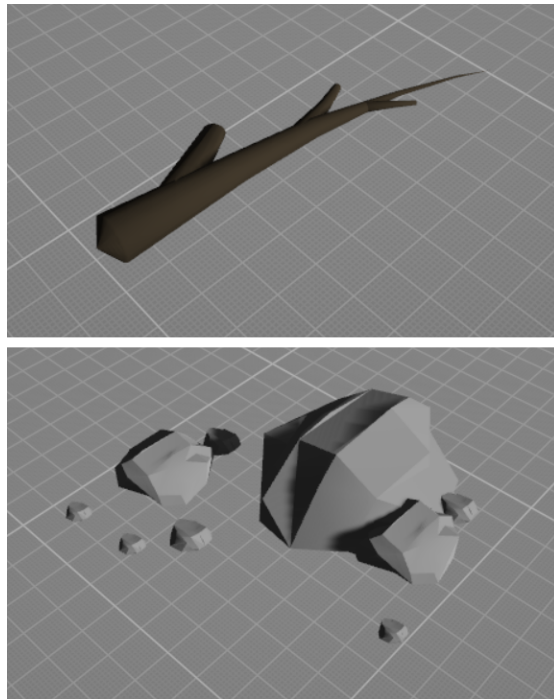


Une solution serait d'utiliser uniquement des nombres entiers, mais encore une fois cela entraîne de nouveaux problèmes tels que le calcul de la distance entre deux points qui met en jeu la fonction racine, donc des nombres flottants. Nous devrions alors remplacer la physique de Unity complètement, ce qui prendrait trop de temps, donc dans ce projet nous ne prendrons pas la peine d'implémenter ce type de déterminisme.

11.6 Obstacles

Il nous fallait trouver des petits obstacles à disposer aléatoirement sur le plateau de jeu, et celui-ci étant une cour de récréation nous avons décidé de modéliser des branches et des petites pierres qui allaient être disposées un peu partout sur le terrain afin de complexifier les parties.

Voilà à quoi ressemblent les modèles de branche et de pierres, que nous avons modélisé nous-mêmes :

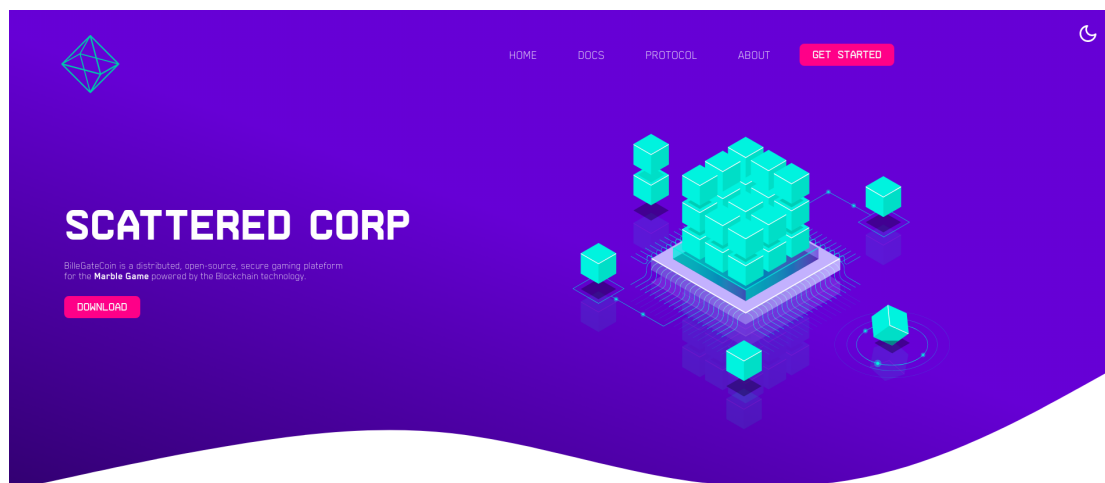


La façon dont ces obstacles sont disposés est gérée par un algorithme qui utilise la fonction « Random » initialisée avec le hash du block dans lequel le contrat de la création de la partie est contenu, toutes les parties créées dans le même block auront alors le même seed, donc même répartition d'obstacles.



12 Site web

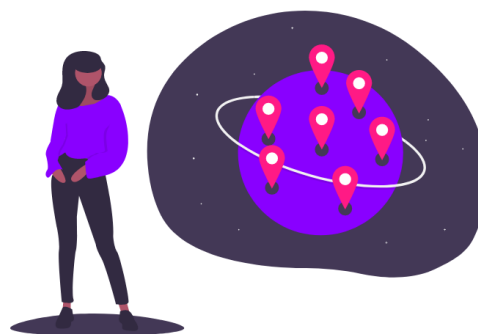
Nous avons continué le développement du site web, réalisé entièrement à la main en JavaScript (React.js) et Sass (équivalent de CSS).



Nous affichons les valeurs du projet avec des illustrations flat-design et une courte description.

DECENTRALIZED

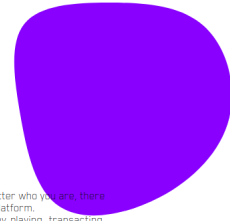
BilleGateCoin does not have a central authority, no entity owns or controls the network. The players decide the rules of the network. They validate each and every transaction happening on the network in real-time and agree upon the fixed issuance of marbles.





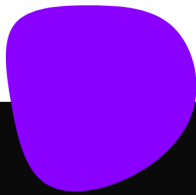
OPEN

The network is open to everyone. No matter who you are, there exists no restriction on the use of this platform. Anyone can take part of the network, by playing, transacting, mining, or contributing to the core implementation of BilleGateCoin.



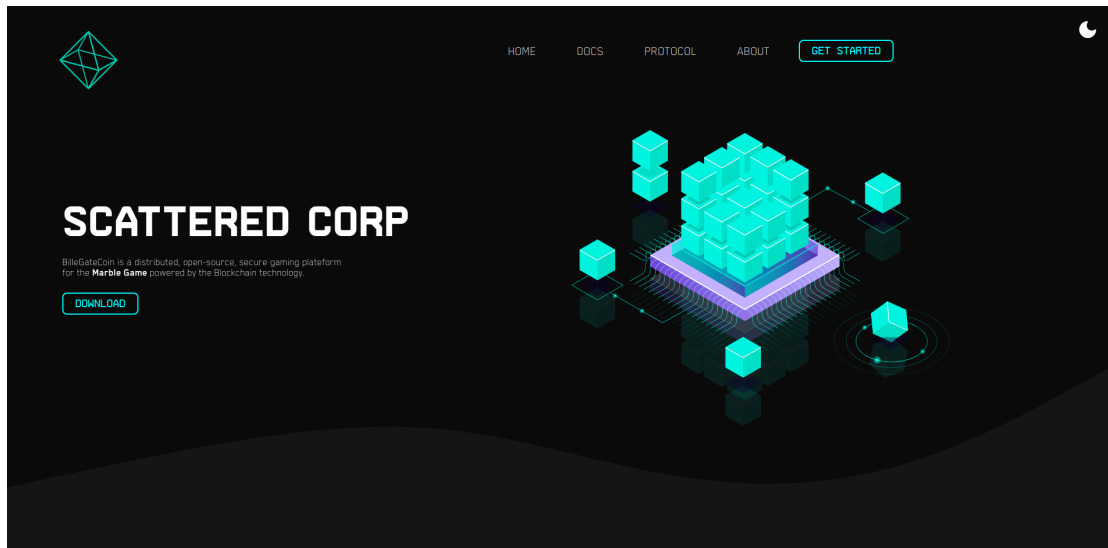
FUN

The marble game is a century-old game that has entertained generations. Earn marbles by winning duels, exchanging marbles, or securing the network by mining. Impress your friends by showing your marble collection with your **public key**.

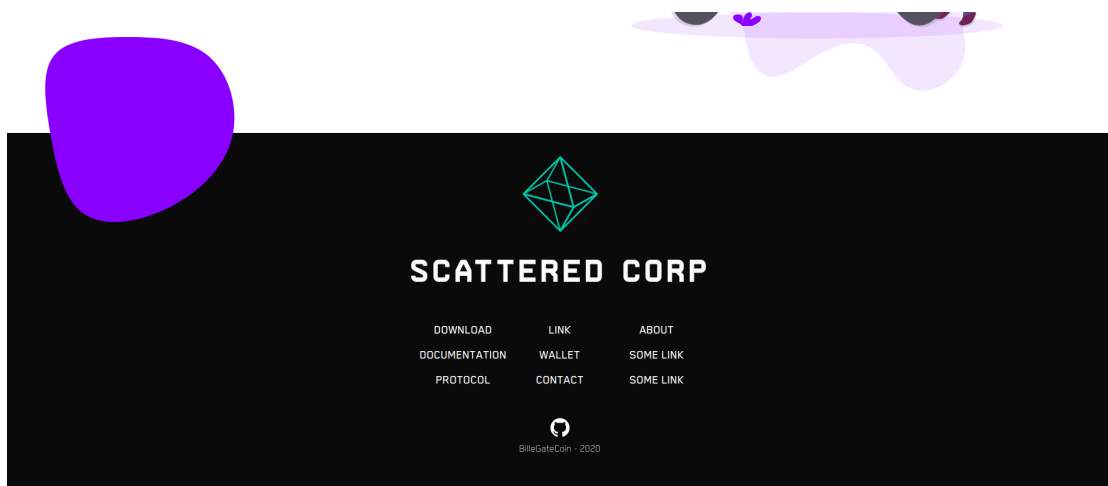


Un thème sombre a également été design afin d'apaiser les yeux des joueurs visitant le site. Il suffit de cliquer sur la lune en haut à droite pour changer de thème.





Voici le footer du site web, il permet d'afficher notre lien GitHub, et des liens qui mènent vers la documentation du protocole.



Nous avons également amélioré la documentation du protocole qui évolue au fur et à mesure de l'avancement du projet. On y retrouve les méthodes de création d'un portefeuille, comment encoder les trois différents contrats du jeu. Avec les informations de la documentation, n'importe qui pourrait être capable de créer une nouvelle implémentation de BilleGateCoin, dans un autre langage par exemple qui fonctionne en harmonie avec l'implémentation de ScatteredCorp, tant qu'elle respecte les règles consensus établies.



BilleGateCoin

- Getting started
- Presentation
- Guide
- Protocol
- Wallet
- Contracts**
- Blocks
- Network
- Marbles



Contracts

Placement

This is the structure that represents a placement of multiple marbles.

Bytes	Value
1	Number of types
1	Marble type
1	Color
4	Amount
...	...
1	Marble type
4	Amount

StartContract

This contract must be signed by both players in order to create a new game.

Bytes	Value
1	Version
1	Type = 0
?	Fee (placement)
?	Player1 placement
?	Player2 placement
25	Player1 address
25	Player2 address

13 Conclusion

En conclusion, le groupe est toujours aussi motivé pour terminer notre projet ambitieux. Nous avons bien rattrapé notre retard sur la partie Unity, nous sommes même en avance sur cette partie.

Il nous reste à finaliser le projet, à connecter toutes nos fonctions entre elles et nous aurons un jeu de billes décentralisé fonctionnel.

