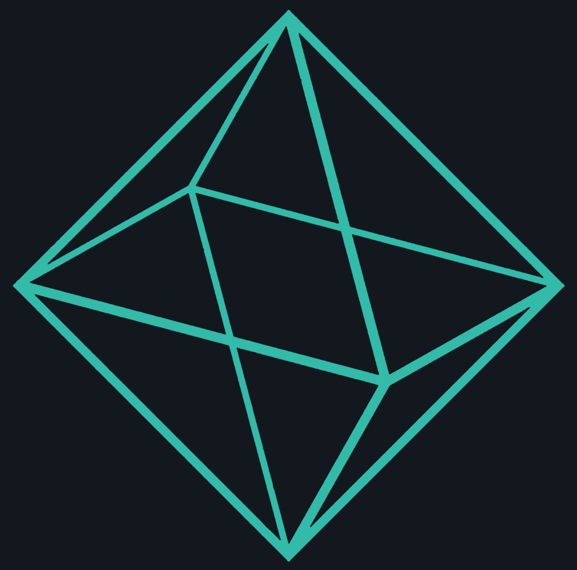


BilleGateCoin - BGC

Scattered Corporation

RAPPORT FINAL

réalisé par

aurele.oules • leo.gervoson • raphael.brenn

Projet EPITA 2020

Table des matières

1 Introduction **4**

2 Tâches **4**
- 2.1 Worldstate . . . 4
- 2.2 Relayage d'informations . . . 4
- 2.3 Parties en cours . . . 5
- 2.4 Site web et documentation . . . 5

3 CLI **5**

4 Blockchain **6**
- 4.1 Porte-feuille . . . 6
  - 4.1.1 Clé privée . . . 6
  - 4.1.2 Clé publique . . . 7
  - 4.1.3 Adresse . . . 8
- 4.2 Nodes . . . 8
  - 4.2.1 Consensus . . . 8
  - 4.2.2 Registre . . . 9
  - 4.2.3 Distribution . . . 9
  - 4.2.4 World state . . . 9
- 4.3 Indexation . . . 10
  - 4.3.1 Réstitution des parties . . . 10
- 4.4 Mempool . . . 10
- 4.5 Proof of Work . . . 11
  - 4.5.1 Block hash . . . 11
  - 4.5.2 Ajustement de la difficulté . . . 12
- 4.6 Contrats . . . 13
  - 4.6.1 StartContract . . . 13
  - 4.6.2 TransactionContract . . . 13
  - 4.6.3 ThrowContract . . . 13
- 4.7 Encodage sous forme d'octets . . . 13
  - 4.7.1 Placement de billes . . . 13
  - 4.7.2 StartContract . . . 14
  - 4.7.3 ThrowContract . . . 16
  - 4.7.4 TransactionContract . . . 17
  - 4.7.5 Block header . . . 17
  - 4.7.6 Block . . . 17
- 4.8 Réseau TCP/IP . . . 17
  - 4.8.1 Architecture . . . 18
  - 4.8.2 Peer to peer (P2P) . . . 18
  - 4.8.3 Supernodes . . . 18

4.8.4 Messages . . . 19
4.9 Économie . . . 19

5 Graphismes **20**
5.1 Terrain de jeu . . . 20
5.1.1 Première version (9.90 Mo) . . . 20
5.1.2 Nouvelle version (447,45 Ko) . . . 21
5.2 Billes . . . 21
5.2.1 Modèles 3D . . . 22
5.2.2 Icônes . . . 22
5.3 Lancé de la bille . . . 23
5.3.1 Caméra . . . 23
5.3.2 Force du tir . . . 23
5.3.3 Frottements . . . 24
5.4 Menus . . . 24
5.4.1 Menu principal . . . 25
5.4.2 Inventaire . . . 25
5.4.3 Créer une nouvelle partie . . . 26

6 Identité visuelle **27**

7 Site web **28**

8 Logiciels utilisés **33**
8.1 VS Code . . . 33
8.2 Unity . . . 33
8.3 Discord . . . 33
8.4 Git + GitHub . . . 34
8.5 Photoshop . . . 34
8.6 Google Chrome . . . 34
8.7 LaTeX . . . 34
8.8 Google Sketchup . . . 34
8.9 IntelliJ Rider . . . 34
8.10 Double Commander + 3DBrowser . . . 34

9 Ressources **34**
9.1 Bitcoin . . . 35
9.2 Ethereum . . . 35
9.3 GitHub . . . 35
9.4 Undraw.co . . . 35

10 Limites du projet et problèmes rencontrés **36**
10.1 Déterminisme . . . 36
10.2 Triche . . . 36
10.3 Hole punching . . . 37

11 Améliorations **37**
11.1 Scalability . . . 37
11.2 Signatures avec Schnorr . . . 38
11.3 Performances . . . 38
11.4 Vulnérable aux attaques DoS . . . 39
11.5 Vulnérable à une attaque 51% . . . 39
11.6 Vulnérable à une attaque Ian Ternier . . . 40
11.7 Remplacer le Proof of Work . . . 40
11.8 Décentralisation . . . 41

12 Ressentis **41**
12.1 Aurèle . . . 41
12.2 Raphaël . . . 42
12.3 Léo . . . 42

13 Données personnelles **42**

14 Développement **43**

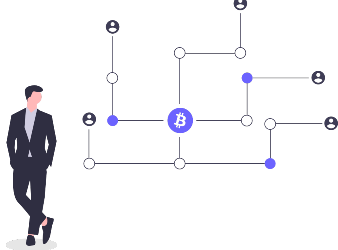
15 License **43**

16 Conclusion **44**

1 Introduction

BilleGateCoin est une plateforme décentralisée pour un jeu de billes. Cette plateforme utilisera la technologie de la Blockchain pour sécuriser les parties. En 2008, Satoshi Nakamoto est le premier à résoudre le problème de la "double dépense" dans un système décentralisé : le Bitcoin. Cette technologie révolutionnaire et surprenante n'a cessé de prendre de plus en plus de place dans notre société. Nous avons décidé d'aborder cette technologie de la Blockchain de manière divertissante en l'appliquant à un jeu-vidéo dont les règles sont simples et les objets sont collectionnables : le jeu de billes.

Nous avons terminé notre jeu BilleGateCoin décentralisé. Nous présonterons donc le projet dans sa globalité ainsi que les problèmes rencontrés et des solutions que nous aurions pu implémenter avec plus de temps.



2 Tâches

Depuis la dernière soutenance, nous avons surtout relié tous les composants que nous avons développés individuellement.

2.1 Worldstate

Nous avons finalisé l'indexation des parties et des inventaires des joueurs. N'importe quel joueur peut récupérer son propre inventaire, ainsi que celui d'un autre joueur lorsqu'il crée une nouvelle partie. Le worldstate permet également de sauvegarder la liste des parties en cours du joueur, sans avoir à parcourir la liste de block et chercher dans la liste des contrats si l'adresse du joueur est présente ou non.

2.2 Relayage d'informations

Lorsqu'un contrat arrive dans la memory pool, celui-ci est relayé sur tout le réseau, sur le même principe que le relayage de blocks. Si celui-ci existe déjà dans la memory pool lorsque la node le reçoit, il ne sera pas relayé.

```
public static void PropagateBlock(Block block) {
    byte[] message = Utils.CreateMessage(Message.COMMAND.SendBlock, block.Serialize());
    ReturnCode WeDontCare = ReturnCode.Pending;
    Console.WriteLine(nodes);
    foreach (IPEndPoint node in nodes)
    {
        SendData(node, message, ref WeDontCare);
    }
}
```

2.3 Parties en cours

Le menu des parties en cours affiche désormais la liste des parties qui ont été créées et qui ne sont pas terminées.

2.4 Site web et documentation

Nous avons continué le développement du site web. La liste des membres est disponible en bas de la page principale. On retrouve les membres Aurèle Oulès, Léo Gervoson et Raphäel Brenn avec nos liens GitHub respectifs.

Nous avons également ajouté des liens pour télécharger le cahier des charges ainsi que les différents rapports de soutenance.

3 CLI

Voici la liste des commandes implémentées :

- **init** : Permet d'initialiser la chaîne avec un block **Genesis**
- **showblocks** : Permet d'affichier la liste des hash SHA-256 des blocks
- **mine** : Lance le processus de Proof of Work avec les contrats de la Mempool
- **signcontract** : Permet de signer un contrat encodé en hexadécimale avec une clé privée
- **decodecontract** : Permet de décoder un contrat encodé en hexadécimale
- **createwallet** : Permet de générer une nouvelle clé privée ainsi qu'une clé publique correspondante
- **getwallet** : Permet d'afficher l'adresse publique du porte-feuille
- **deletewallet** : Supprime le porte-feuille de l'utilisateur
- **inventory** : Permet d'afficher l'inventaire d'un joueur existant sur le réseau

- **start** : Lance un serveur TCP/IP pour communiquer avec les autres nodes du réseau
- **send** : Permet d'envoyer des données arbitraire à une node
- **help** : Affiche la liste des commande disponibles
- **version** : Affiche la version actuelle de BGC

4 Blockchain

4.1 Porte-feuille

Un portefeuille consiste d'une clé privée et d'une clé publique correspondante. Celui-ci permet de signer des contrats sur la plate-forme and remplace le système basique d'email/mot de passe. La génération d'un portefeuille se fait en plusieurs étapes.

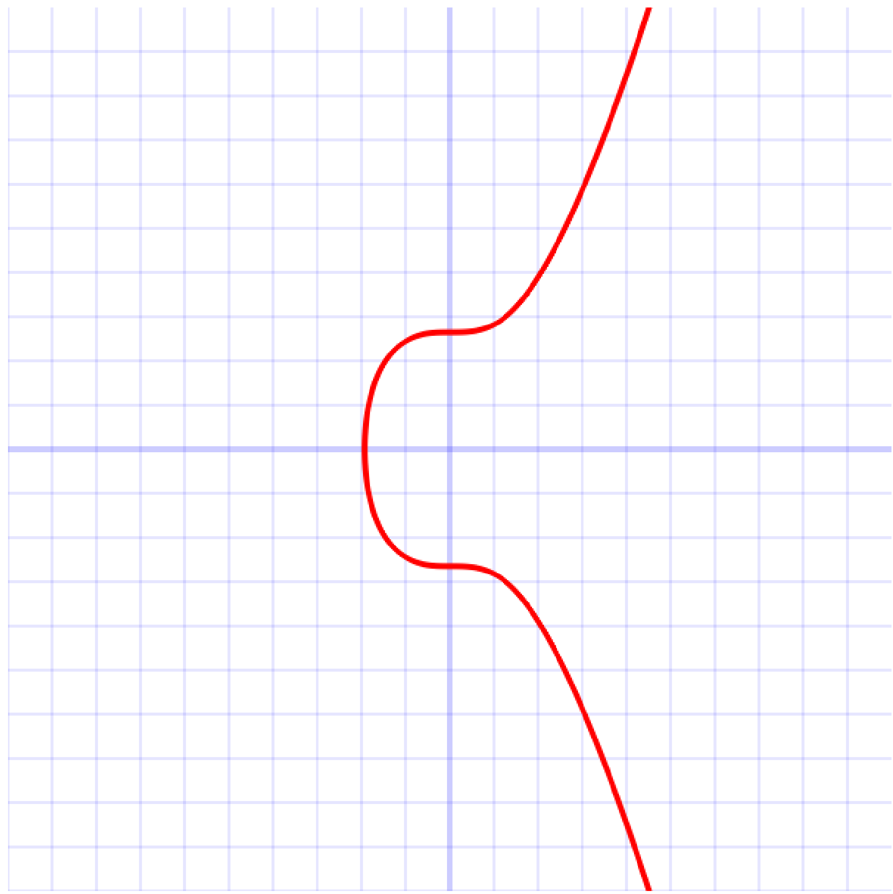
ECDSA est l'algorithme cryptographique utilisé par BGC pour signer les contrats. BilleGateCoin utilise la courbe elliptique **secp256k1**. Nous avons décidé d'utiliser le même algorithme de signature ainsi que la même courbe elliptique que le Bitcoin afin de pouvoir comparer nos résultats et également utiliser les normes déjà établies par la communauté.

4.1.1 Clé privée

Une clé privée est un nombre aléatoire compris entre 1 et $2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$. Il faut noter qu'il existe environ 2^{272} atomes dans l'univers observable, il serait donc difficile de tomber par hasard sur la clé privée de quelqu'un d'autre, même avec toute l'énergie de notre système solaire. Voici à quel point les billes du réseau sont sécurisées. Ce nombre doit donc être stocké de manière très sécurisée. L'utilisateur doit chiffrer cette clé privée avec un mot de passe grâce au système de chiffrement AES 256 bits.

Cette clé privée doit être sur la courbe elliptique **secp256k1** définie par

$$y^2 = x^3 + 7$$



Étant donné que les humains sont trop prédictibles, ce nombre aléatoire est généré grâce un à générateur de nombres aléatoires cryptographiques.

4.1.2 Clé publique

Pour générer une clé publique à partir d'un clé privée il suffit de multiplier la clé privée par le point générateur G de la courbe secp256k1 tel que

$$P = xG$$

.

Les étapes pour multiplier et additionner des points sur une courbe elliptique se trouvent ici : **https ://en.wikipedia.org/wiki/Elliptic_curve_point_multiplication#Point_operations**.

P est un point sur la courbe, pour encoder sa valeur décompressée sous forme d'octets il suffit de concaténer son X et son Y ensemble. Pour encoder sa valeur compressée, il suffit simplement d'encoder son X car le Y peut se recalculer facilement grâce à l'équation $y^2 = x^3 + 7$.

4.1.3 Adresse

Dans BilleGateCoin, nous utilisons des adresses plutôt que des clés publiques pour référencer les joueurs. Une clé publique pèse 64 octets contrairement à une adresse qui pèse seulement 25 octets. Le poids des données est très important dans une blockchain alors nous faisons le maximum pour éviter de stocker des données inutiles. Une adresse est en réalité un hash de la clé publique. Voici les étapes de construction d'une adresse.

- Générer une clé privée
- Dériver la clé publique
- Réaliser un hash SHA-256 sur la clé publique
- Réaliser un hash RIPE-MD sur le hash SHA-256
- Ajouter l'octet de version devant le hash RIPE-MD (0x8A)
- Réaliser le hash double hash SHA-256 du hash RIPE-MD versionné
- Ajouter les 4 premiers octets du double hash (checksum) à la fin du hash RIPE-MD versionné

4.2 Nodes

Une *node* est un ordinateur connecté à tout le réseau de la blockchain. Elle a un rôle majeur sur la sécurité du réseau.

4.2.1 Consensus

La node possède le rôle d'arbitre. Lorsque une partie est créée ou un joueur lance sa bille, la node va vérifier si le joueur possède bien cette bille, si c'est bien au tour du joueur de jouer, s'il n'essaye pas de lancer sa bille plus fort que la limite, validité des signatures, s'il ne lance pas plusieurs fois la bille. Si le contrat est valide, celui-ci rentre dans la *memory pool* de la node, sinon il est rejeté. Une node maintient également l'intégrité de la chaine, elle doit s'assurer que chaque block est lié avec le précédent, que les récompenses que les mineurs s'attribuent sont valides, que la solution trouvée par les mineurs est valide, que les contrats de chaque block sont valide, que la taille du block ne dépasse pas 256 Ko, etc.

4.2.2 Registre

La node s'occupe de stocker l'historique entier de toutes les parties et transactions de tous les joueurs, c'est-à-dire la blockchain.

L'ordinateur réserve également un emplacement pour stocker les transactions non confirmées par les mineurs, c'est-à-dire les lancés de billes et créations de parties qui viennent d'être diffusées par les joueurs. Cet emplacement s'appelle la *memory pool*.

4.2.3 Distribution

Le rôle principal de la node est de partager la blockchain à toutes les autres nodes du réseau. La node qui possède la plus grande chaîne est celle qui possède la chaîne la plus valide, elle doit donc la partager au réseau de manière décentralisée. N'importe quelle node peut demander des informations manquantes à une autre node, puis les distribuer à son tour. Cela fonctionne comme un torrent mais au lieu d'avoir un tracker principal, chaque node envoie sa propre liste d'ip.

4.2.4 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.

4.3 Indexation

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 chaine, à n'importe quel moment.

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

4.3.1 Réstitution des parties

Grâce au Worldstate, lorsqu'un joueur continue une partie, tous les obstacles et la position des billes sont rétablie. Le joueur peut ainsi continuer sa partie.

4.4 Mempool

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;
    }
}
```

4.5 Proof of Work

Les mineurs assurent l'immuabilité de la blockchain. Un block contient des contrats et des lancés de billes. Pour ajouter un block à la blockchain, un puzzle constitué exclusivement des données du block doit être résolu. Ce puzzle n'est solvable uniquement en essayant des milliards de combinaisons par seconde.

4.5.1 Block hash

Un hash [1] SHA-256 est un nombre hexadécimal aléatoire compris entre 1 et $2^{256} - 1$.

1. Une fonction hash est une fonction qui prend une entrée telle que du texte ou un nombre, et retourne un nombre qui apparaît aléatoire mais est déterministe. Pour la même entrée, la fonction retournera toujours la même sortie. C'est une fonction à sens unique, le hash ne permet pas de retrouver l'entrée.

La solution du puzzle que les mineurs doivent résoudre est en réalité un hash du block qui doit être compris entre 1 et $\frac{2^{256}-1}{mining\ difficulty}$.

Les mineurs forment un block à partir des contrats de la *memory pool* des nodes.

Si le hash n'est pas compris dans cet intervalle, les mineurs calculent à nouveau un hash du block en concatenant un *nonce* différent. C'est un nombre arbitraire qui permet uniquement de générer un hash différent en conservant les valeurs du block.

Ce hash sécurise la blockchain car il est facile de le résoudre si tout le réseau travail sur ce problème, mais il est impossible de résoudre ce problème seul. Ce système assume que plus de 50% du réseau est honnête.

4.5.2 Ajustement de la difficulté

La difficulté de minage est une variable qui assure que les mineurs ajoutent des blocks à la blockchain de manière régulière même si le nombre de mineurs, donc de puissance de calcul, augmente.

Cette difficulté permet aux nodes de prévoir l'espace disque nécessaire pour stocker la blockchain et assure une économie stable (voir section suivante). Elle doit donc être ajustée régulièrement.

Le *block time* est une constante qui définit le temps qu'un mineur met pour ajouter un block à la chaîne.

Sur la plateforme de jeu de billes, le block time doit être assez court afin de permettre des exécutions de contrats (créations de parties, et lancés de billes) rapides. Par exemple, le Bitcoin utilise un block time de 10 minutes, et Ethereum un block time de 20 secondes.

Un intervalle trop court peut causer des problèmes sur le délai de distribution des nouveaux blocks entre les nodes.

La difficulté de minage est au départ 1. Celle-ci est ajustée tous les X blocks. Chaque node va calculer le temps que X block prennent pour être minés en théorie, divisé par la moyenne de temps que les X derniers blocks ont pris pour être minés. La difficulté est ainsi multipliée par ce coefficient.

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

4.6 Contrats

Il existe trois types de contrats que les joueurs peuvent publier sur BilleGateCoin.

4.6.1 StartContract

Ce contrat permet de lancer une nouvelle partie. Un joueur choisis son adversaire, les billes que l'adversaire doit mettre en jeu, ses billes, puis les deux joueurs doivent ajouter leur signature au contrat et le diffuser sur le réseau.

4.6.2 TransactionContract

Ce contrat permet d'échanger instantanément des billes entre deux joueurs. Les joueurs doivent également ajouter leur signature au contrat.

4.6.3 ThrowContract

Ce contrat doit être créé par le joueur lançant sa bille. Il contient le vecteur de lancé du joueur, le hash de la partie, et sa signature.

4.7 Encodage sous forme d'octets

Afin de transférer les données des contrats ainsi que les blocks, les nodes doivent encoder ces données sous forme d'octets avec un protocole bien définie pour qu'ils puissent être lus par tout le monde.

4.7.1 Placement de billes

Un placement de billes est une structure de données réutilisée dans plusieurs contrats. Un placement permet à un joueur de placer une ou plusieurs de ses billes dans un contrat pour les jouer, ou simplement les échanger.

Octets	Valeur
1	Nombre de types
1	Type
4	Quantité
...	...
1	Type
4	Quantité

4.7.2 StartContract

Ce contrat doit être signé par les deux joueurs afin de créer une nouvelle partie. Celui-ci contient les billes mises en jeu des deux joueurs, leur adresse respective, et leur signature. La première bille du placement est celle jouée par le joueur.

Octets	Valeur
1	Version
1	Type = 0
?	Fee (Placement)
?	Placement du joueur 1
?	Placement du joueur 2
25	Adresse du joueur 1
25	Adresse du joueur 2
4	Nonce du joueur 1
65	Signature du joueur 1
4	Nonce du joueur 2
65	Signature du joueur 2

Un contrat sérialisé puis converti en héxadécimal ressemble à ceci :

010002000C000000010F00000002024000000000420000000002057A00000008020000008A9B9E6AD7633E9
F498D044D22B8F6466EEA73DCD727E966458A4BA642E726EAB822B91C6FEA82802466AC6BA4E00115D2C2

04000000D59C2D1DA1F26883303C2541F35F406ADCB7EAB78DAA5B1A15975B0DBE3DB0C62B149E7B93F83
430173FEFABC8F3A2F6E680F5A236F07424B5BA86F293B7747000080000007A95F34FA2752E1588139BEB
68042D24A7B42D75EB012BD0A9A6F67984C92F4647B42F8B9D2EC0C94CD997CDA463C5067E488EFB39163
1C8EEB8820F6CCF5A0D00

Celui-ci peut être converti en QR Code pour pouvoir être envoyé à un joueur facilement.

Ce contrat peut ensuite facilement être lu par le protocole.

```
Version: 1
Type: 0
Fee:
        Type: 0
        Amount: 12

        Type: 1
        Amount: 15

Player One Placement:
        Type: 2
        Amount: 64

        Type: 4
        Amount: 32

Player Two Placement:
        Type: 5
        Amount: 122
```

```
        Type: 8
        Amount: 2

Player One Address: xmq5PkTAp21ajwLsM6VEuofwUjrNXeix6p
Player Two Address: xeYEjQwU3Q35zCp4GCxHDPHWphHRh1FB1j
Player One Nonce: 4
Player One Sig: 213 156 45 29 161 242 104 131 48 60 37 65 243 95 64 106 220 183 234
183 141 170 91 26 21 151 91 13 190 61 176 198 43 20 158 123 147 248 52 48 23 63 239
171 200 243 162 246 230 128 245 162 54 240 116 36 181 186 134 242 147 183 116 112 0
Player Two Nonce: 8
Player Two Sig: 122 149 243 79 162 117 46 21 136 19 155 235 104 4 45 36 167 180 45 117
235 1 43 208 169 166 246 121 132 201 47 70 71 180 47 139 157 46 192 201 76 217 151 205
164 99 197 6 126 72 142 251 57 22 49 200 238 184 130 15 108 207 90 13 0
```

4.7.3 ThrowContract

Ce contrat doit être créé dès qu'un joueur lance sa bille. Il contient les informations du joueur (sa clé publique), son vecteur de lancé (X, Y, Z), le hash de la partie en cours, et sa signature.

Octets	Valeur
1	Version
1	Type = 1
?	Fee (Placement)
1	Vec.X
1	Vec.Y
1	Vec.Z
32	Hash de la partie
4	Nonce de la partie
4	Nonce du joueur
65	Signature

4.7.4 TransactionContract

Ce contrat doit être signé par les deux joueurs lorsqu'ils souhaitent échanger instantanément des billes. Il possède la même structure que le **StartContract** sauf que son octet de version est 2.

4.7.5 Block header

Le Block Header contient des méta-données du block en lui même.

Octets	Valeur
1	Version
32	Hash précédent
32	Merkle root
4	Date
4	Difficulté (target)
4	Nonce

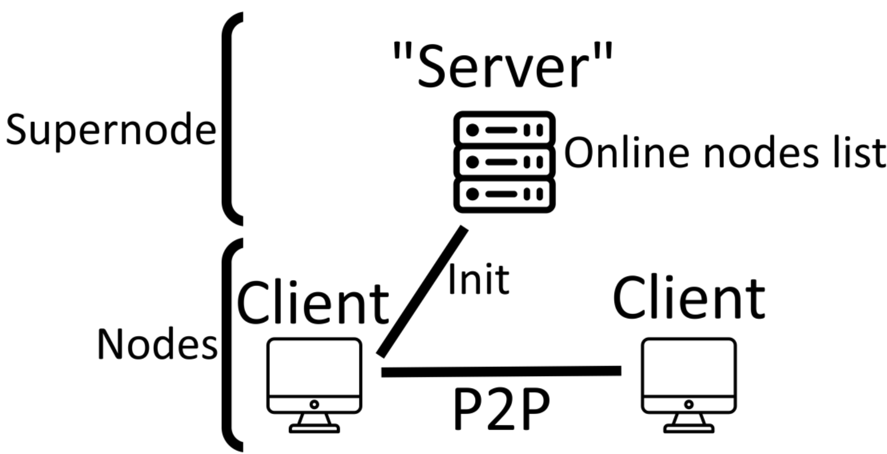
4.7.6 Block

Octets	Valeur
77	Block header
4	Nombre de contrats
?	Contrats sérialisés

4.8 Réseau TCP/IP

L'implémentation d'un protocole de communication est majeur dans une blockchain. **Léo** s'est occupé de la partie réseau et de la construction des messages via TCP/IP.

4.8.1 Architecture



Le réseau est constitué de :

- Supernodes : serveurs (ports ouverts) permettant aux nodes de communiquer entre elles.
- Nodes : clients

4.8.2 Peer to peer (P2P)

Une blockchain est basée sur la communication P2P entre les nodes. Cependant, les clients utilisent la plupart du temps un routeur NAT dont les ports sont fermés. Il faut donc des supernodes intermediaires afin d'initialiser une connection P2P. On utilisera la méthode du hole punching pour ouvrir les ports des clients.

Le fonctionnement du hole punching est assez simple : un client voulant devenir une node du réseau initie une connexion à une supernode. Son routeur ouvre un port et le lui attribue pour communiquer avec la supernode. Il garde ensuite ce port indéfiniment ouvert et signale aux autres nodes de l'enregistrer (message RegisterNode).

4.8.3 Supernodes

Les supernodes auront des adresses IP fixes, hardcodées dans la blockchain. Il faut au minimum une supernode en ligne pour garantir le fonctionnement du réseau. Les supernodes recensent toutes les nodes en ligne. Toute initiation de connexion P2P entre nodes se fait en passant par une supernode :

- La node initiant la connection contacte la supernode pour obtenir l'addresse IP de la node visée
- Elle envoie ensuite un paquet à la cible, qui sera rejeté par le pare-feu. Cette étape a pour but d'ouvrir un port acceptant les paquets venant de la cible
- Elle envoie l'ID du port nouvellement ouvert à la supernode, qui le communique à la cible

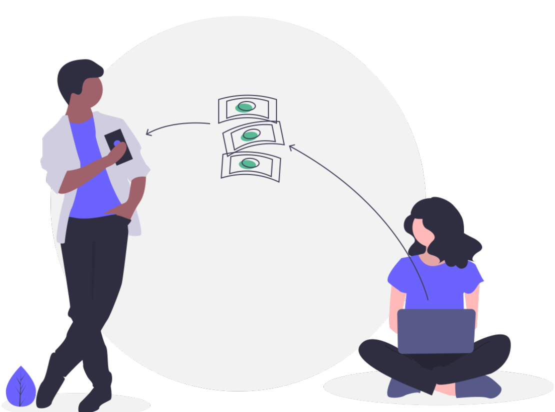
— La cible envoie un paquet au port ouvert, qui cette fois est accepté
— Les 2 nodes ont à présent chacune un port ouvert acceptant les paquets venant de l'autre

4.8.4 Messages

Les paquets envoyés sur le réseau, appelés messages, possèdent une architecture prédéfinie. Ils suivent l'architecture suivante :

Bytes	Valeur
1	Magic (réseau)
1	Commande
4	Taille du contenu
4	Checksum du contenu
Variable	Contenu

4.9 Économie



Dans ce jeu de billes, il n'y a pas d'autorité centrale qui distribue les billes. Un challenge de ce projet a été de définir la manière de distribuer ces billes de manière décentralisée. L'économie de ce jeu ressemble à celle de toutes les crypto-monnaies actuelles, sauf que au lieu d'y avoir qu'un seul **token**, il y en a un par type de bille, qui a un nombre total fixe.

Par exemple, pour le Bitcoin, nous savons qu'il y aura 21 millions de bitcoins distribués, alors que dans ce projet, nous savons combien de billes seront distribuées pour chaque type de bille. Cela rend l'économie plus compliqué, mais plus diversifiée.

Comme la plupart des crypto-monnaies, les billes sont distribuées sous forme de récompenses aux mineurs lorsqu'ils sécurisent le réseau en minant. Ces billes peuvent être vendues pour rembourser le coût d'électricité du minage, conservées pour être collectionnées, ou les jouer pour en gagner encore plus.

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

Plus il y a de billes en circulation, moins elles sont rares. C'est pour ça que la bille **Soccer** est beaucoup plus rare que la bille **Earth**. C'est une manière de définir la rareté d'une bille, sans que les prix soient définis sur une plateforme centralisée.

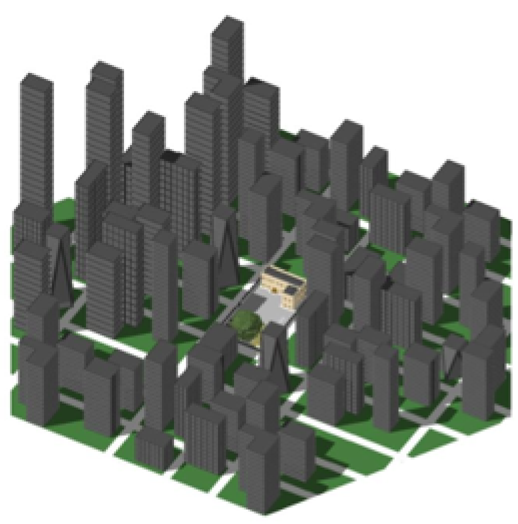
5 Graphismes

5.1 Terrain de jeu

Le terrain de jeu représente une cour de récréation, nous avons choisi ce thème car il est en adéquation avec celui du jeu de billes – traditionnellement un jeu pour enfants.

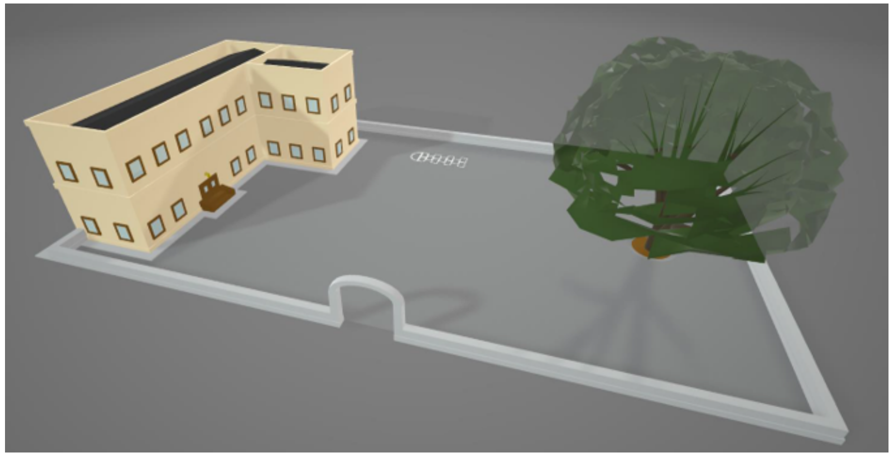
5.1.1 Première version (9.90 Mo)

La première version du terrain de jeu était la plus ambitieuse, trop ambitieuse d'ailleurs puisque elle s'avérera trop imposante et trop détaillée pour pouvoir offrir des performances graphiques correctes : cette première version était bien sûr constituée de la cour de récréation, mais également d'une école, d'un arbre, de barrières, d'un bac à sable et d'autres jeux divers et variés, tous très détaillés… le tout au milieu d'un quartier entier de 70 buildings d'une dizaine d'étages chacun, eux aussi très détaillés. Cela consommait énormément (trop, en tous cas) de ressources. Une refonte quasi-totale était donc nécessaire.



5.1.2 Nouvelle version (447,45 Ko)

Cette refonte a pris la forme d'une deuxième version beaucoup plus légère, qui est la version que le jeu utilise actuellement et qui ne se compose que de versions simplifiées de la cour, l'école et l'arbre. Elle fut finie début février, environ deux semaines après la première, et ne faisait plus que 4,8% de sa taille originale , ce qui a engendré un gain considérable en performances tout en conservant les éléments principaux.



5.2 Billes

Le jeu compte sept types de billes, correspondant chacune à sept modèles différents. Chaque modèle est lui même disponible en six couleurs différentes, sauf pour deux billes spéciales (il y a donc 5 * 6 + 2 = 32 billes existantes dans le jeu). Raphaël s'occupe du

design et de la définition des billes. Ci-dessous un tableau contenant les billes classiques, en chacune de leurs 6 versions :

Name	Rarity	Blue	Green	Orange	Red	Purple	Dark
Elastic	Common						
Layers	Uncommon						
Ribbon	Uncommon						
Stripes	Rare						
Whirlwind	Extra rare						

5.2.1 Modèles 3D

Les modèles 3D des billes ont tous étés créés pour l'occasion, à l'aide du logiciel Google Sketchup. Ces modèles, comme pour le plateau de jeu, ont étés étudiés pour affecter le moins possible les performances graphiques du jeu tout en ayant chacune un aspect à la fois original et esthétique, et même si certaines d'entre elles (notamment deux) semblent nettement plus complexes que les autres cela n'affecte pas significativement les performances du jeu, nous les avons donc gardées ainsi.

5.2.2 Icônes

L'algorithme de conversion du groupe de modèles 3D en deux groupes de fichiers d'icônes (un avec les icônes en .png et un avec les icônes en .gif) est le suivant : pour chaque modèle 3D, d'abord le fichier est renommé selon la forme « [modèle] - [couleur] », et ensuite ce fichier est transformé en deux miniatures, une en .png (sur fond blanc) et l'autre en .gif (animation rotative antihoraire – 24 fps – fond blanc). On applique cet algorithme à la chaîne sur tous les modèles de billes à l'aide du logiciel 3DBrowser, et on obtient bien deux groupes de fichiers d'icônes en .gif et .png, tous avec leurs noms sous la même forme pour faciliter leur manipulation.

5.3 Lancé de la bille

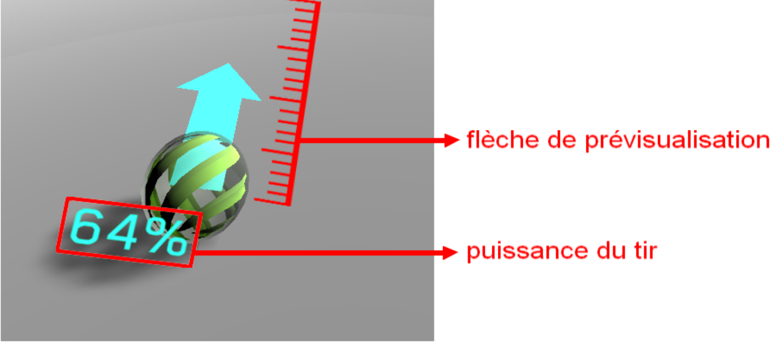
Lors d'une partie, le but est de lancer une de ses billes quelque part sur le terrain, en essayant de percuter une ou plusieurs des billes de l'adversaire an de les remporter. La mécanique de tir est simple : on pivote la caméra pout viser, puis on tire et on admire la bille s'envoler, avec un peu de chance vers la(les) bille(s) de quelqu'un d'autre.

5.3.1 Caméra

La caméra pivote autour de l'axe vertical uniquement, et le tir sera toujours effectué dans le sens dans lequel la caméra regarde (elle sert donc à regarder autour de soi, mais aussi et surtout à viser). On peut la faire pivoter en déplaçant la souris horizontalement avec le click droit enclenché.

5.3.2 Force du tir

Pour tirer, on déplace d'abord 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 spécique 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. Quand on est satisfait de la puissance choisie, on relâche le click gauche et le tir s'effectue.



5.3.3 Frottements

Les frottements sont gérés en trois parties : celle qui fait décélérer la bille quand elle est en l'air (imite les frottements de l'air), celle qui fait décélérer la bille quand elle est au sol (imite la rugosité du sol) et celle qui stoppe la bille quand elle roule à faible vitesse, pour éviter qu'elle ne se déplace pendant trop longtemps et que la partie ne soit pas assez dynamique.

- les fonctions qui gèrent la première et la deuxième partie sont déjà implémentées dans les outils Unity, il ne nous restait plus qu'à ajuster les différentes variables de rugosité du sol et d'aérodynamisme de la bille
- la fonction qui gère la troisième partie a dû être créée, c'est une simple fonction qui arrête les mouvements vectoriels et rotatifs de la bille quand la vitesse de celle-ci passe en-dessous d'une certaine limite, assez basse pour être quasi-imperceptible et assez élevée pour faire effet le plus rapidement possible. Ci-dessous la fonction concernée :

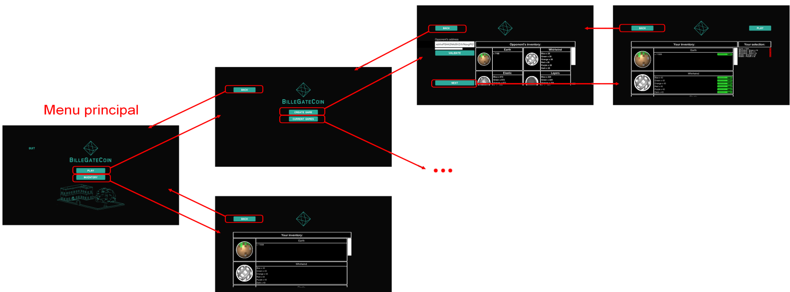
```
public class GroundDrag : MonoBehaviour
{
    private void FixedUpdate()
    {
        if (Math.Abs(GetComponent<Rigidbody>().velocity.x) + Math.Abs(GetComponent<Rigidbody>().velocity.z) < 0.1)
        {
            GetComponent<Rigidbody>().velocity = Vector3.zero;
        }
    }
}
```

5.4 Menus

Chaque page du menu est construite selon le même schéma :

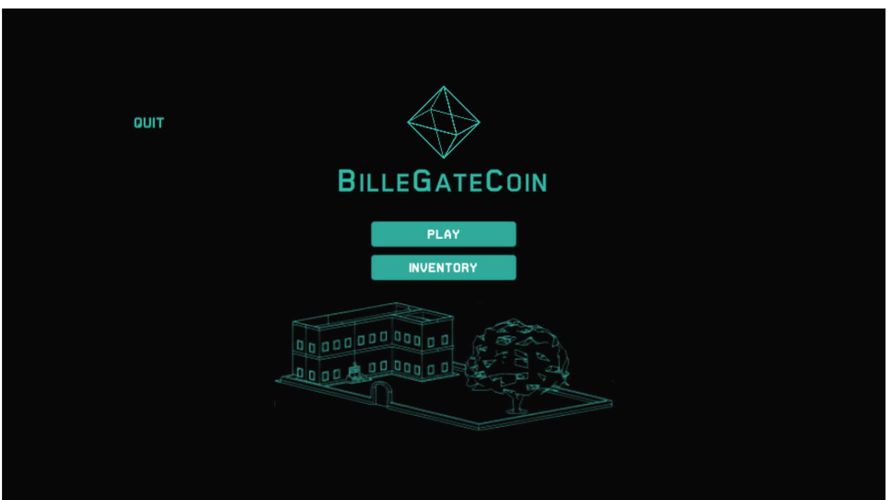
- les visuels sont à dominante turquoise sur fond noir, avec des détails supplémentaires en blanc si nécessaire (ce sont les principales couleurs de la palette qu'on a choisi pour le jeu, on les retrouve notamment sur le site internet)
- si la page est un menu vers d'autres pages de ce menu, alors les 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
- si la page comporte un tableau inventaire, celui-ci est blanc avec les cases noires, à l'intérieur desquelles de trouve soit un texte (en blanc), soit un autre tableau qui suit les mêmes règles.
- toutes les pages sauf la page principale ont un bouton « retour » vers la page précédente.

La navigation dans le menu suit celle décrite dans le schéma suivant :



5.4.1 Menu principal

La page principale comporte un aperçu stylisé du plateau de jeu, deux boutons en son centre (le bouton PLAY qui amène à la page suivante, pour créer une partie ou continuer les parties en cours, et le bouton INVENTORY permet de visualiser l'inventaire du joueur) et un bouton QUIT pour quitter le jeu.

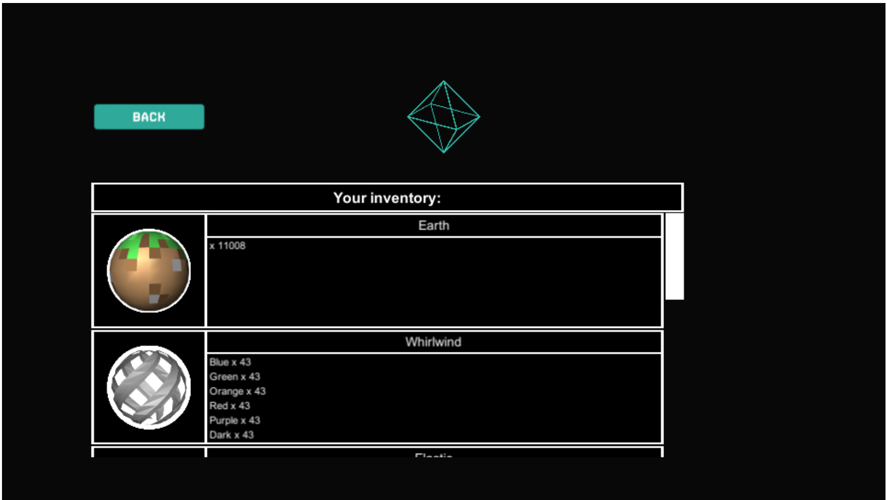


5.4.2 Inventaire

La page de l'inventaire contient un tableau géré par un algorithme qui est lancé à chaque ouverture de la page (on peut donc l'actualiser en quittant et en revenant dessus), chaque case de ce tableau concerne un type de bille. Une case contient donc :

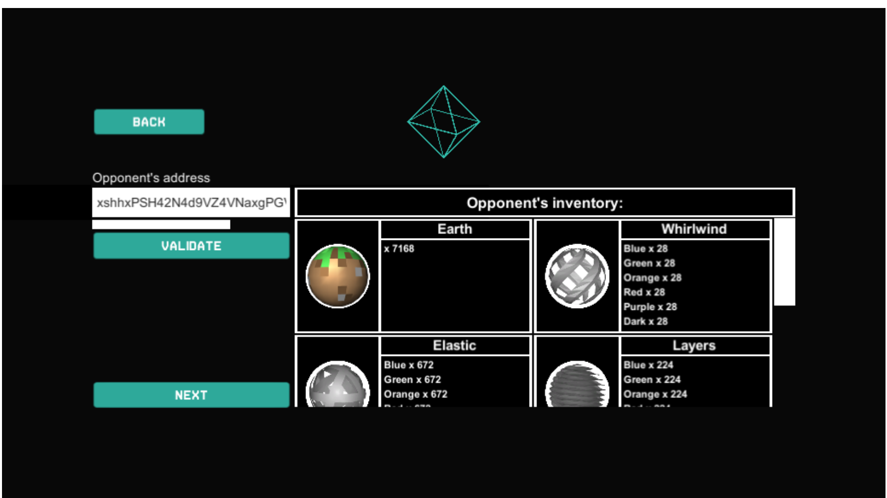
- le nom de la bille
- l'icône associée
- une liste des variantes que l'on possède, chaque élément de la liste est de la forme « [couleur] x [nombre] »

L'algorithme qui gère cette page utilise l'inventaire du joueur, qu'il « demande » à chaque lancement.

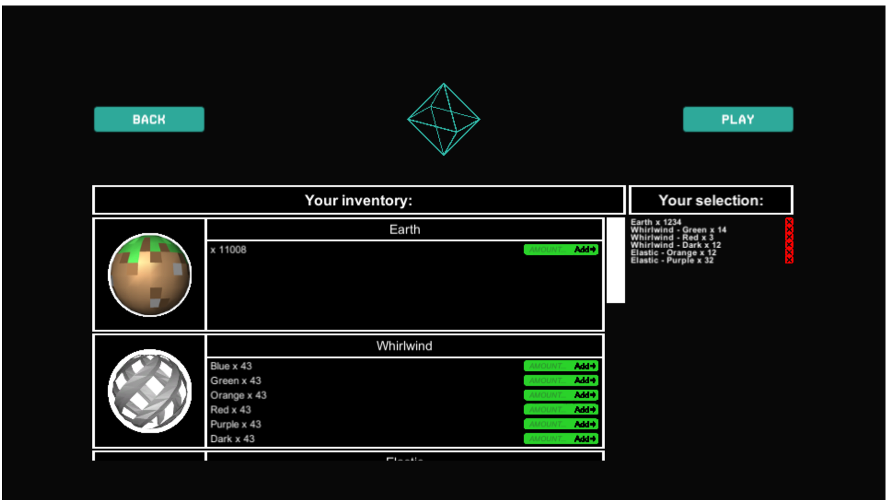


5.4.3 Créer une nouvelle partie

D'abord, on entre l'adresse de l'adversaire de notre choix. Si l'adresse est valide, on peut alors visualiser l'inventaire de cet adversaire et valider notre choix. Ci-dessous un aperçu de cette interface, lorsque l'on renseigne une adresse valide :



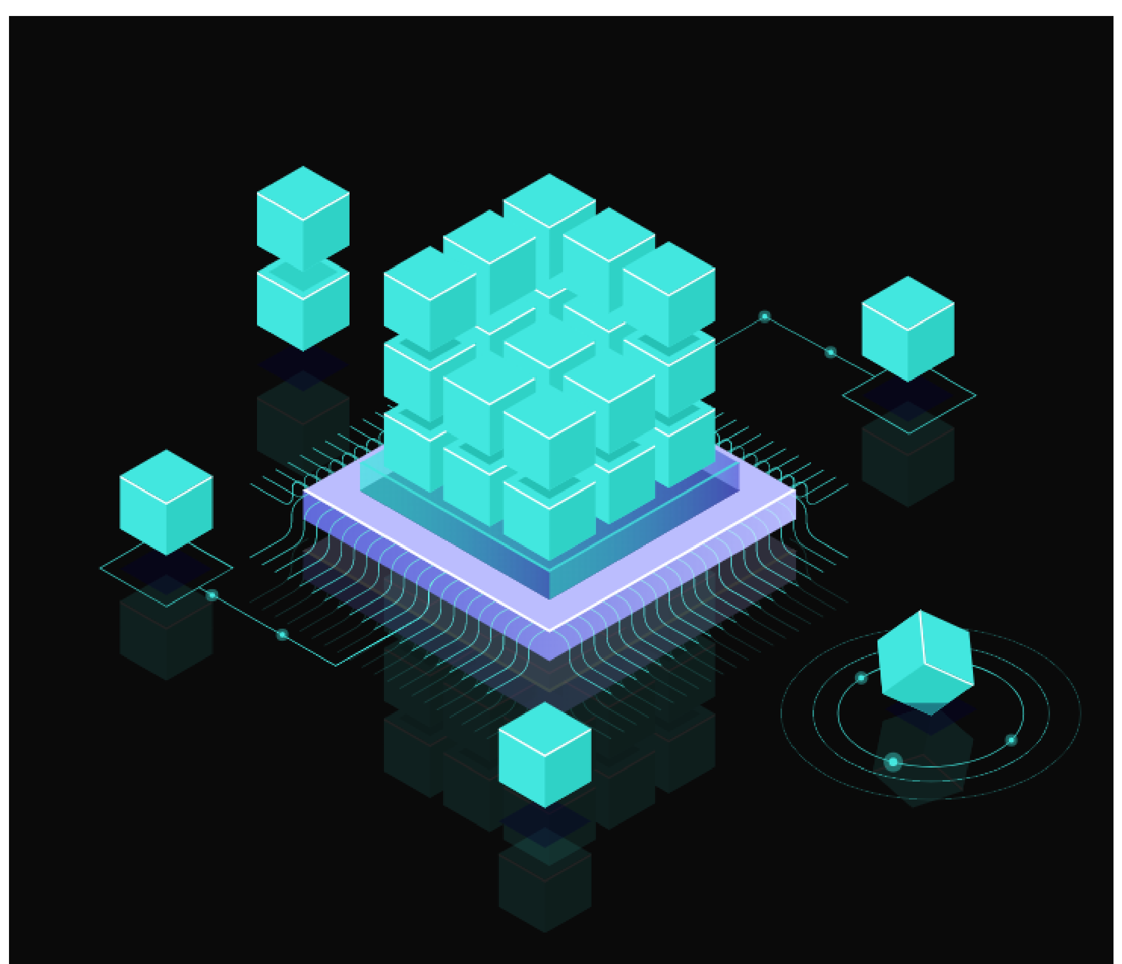
Ensuite, on a accès à notre propre inventaire, avec à droite de celui-ci une liste exhaustive des billes que l'on souhaite « miser » pour cette partie. Cette liste est vide par défaut, il faut donc la remplir en cliquant sur les cases de l'inventaire qui correspondent aux billes que l'on veut jouer, après avoir renseigné le montant désiré bien sûr (qui ne pourra ni être négatif, ni être supérieur au nombre de billes de ce type que l'on possède).



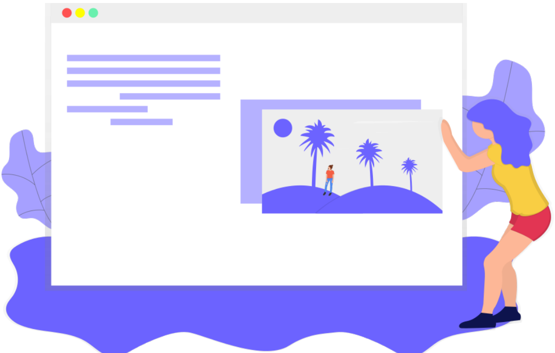
Une fois la liste remplie, on lance la partie. Lorsqu'un joueur démarre une nouvelle partie, un StartContract, celle-ci doit être minée et incluse dans un block. Une fois le StartContract miné, la partie démarre avec un seed, qui est le hash du block dans lequel le contrat est, qui permet de générer 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.

6 Identité visuelle

Notre identité visuelle est très apparente sur l'ensemble du projet. Nous avons opté pour nu thème sombre bleu foncé-vert. Ce thème est plutôt futuristique.

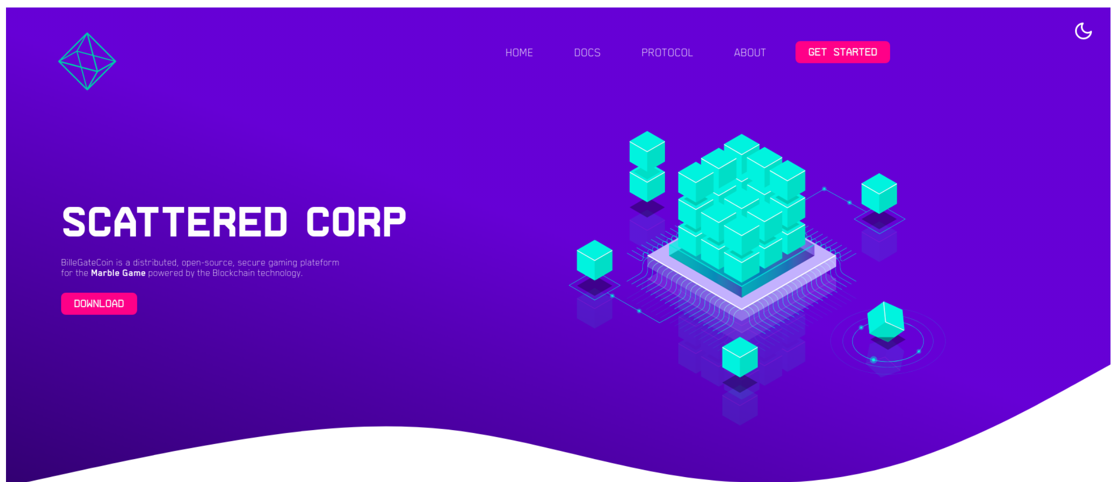


7 Site web

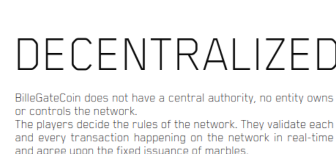


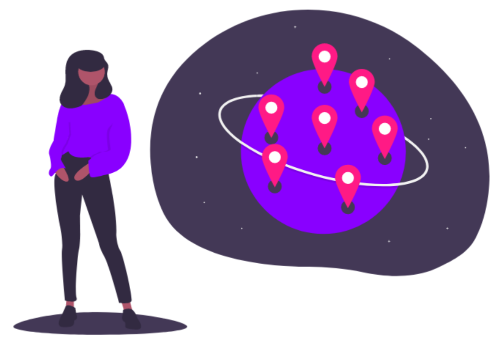
Le site web est réalisé entièrement à la main en JavaScript (React.js) et Sass (équivalent de CSS).

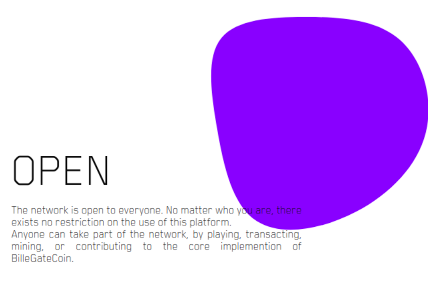
```
return <div className="section home">

        <img className="logo" src={Logo}/>

        <a className="theme-switch" onClick={switchTheme}>
            <Moon fill={theme === "dark" ? "#fff" : "transparent"}/>
        </a>

        <a onClick={toggleMenu} className="menu-burger">
            <Burger/>
        </a>

        <div className={["menu-container", menu ? "active" : ""].join(" ")} >
            <div className="menu">
                <a href="">Home</a>
                <a href="https://docs.scatteredcorp.tech" target="_blank">Docs</a>
                <a href="https://docs.scatteredcorp.tech/#/wallet" target="_blank">Protocol</a>
                <a href="">About</a>
                <button className="button">
                    Get started
                </button>
            </div>
        </div>

        <div className="sections">
            <div className="left slide-bottom delay">
                <h1>Scattered Corp</h1>
                <p>BilleGateCoin is a distributed, open-source, secure gaming plateform for the <a
target="_blank" href="https://simple.wikipedia.org/wiki/Marbles_(game)">Marble Game</a> powered by the
Blockchain technology.</p>
                <div>
                    <a
href="https://github.com/scatteredcorp/game/releases/download/1.0.0/BilleGateCoin.exe"
className="button">
                        Download
                    </a>
                </div>
            </div>

            <div className="right">
                <img src={Computer} className="computer slide-bottom"/>
            </div>

            <Wave className="wave"/>
        </div>
    </div>
```

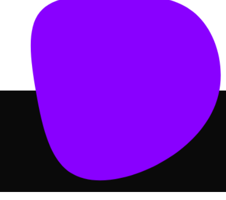


Nous affichons les valeurs du projet avec des illustrations flat-design et une courte description.

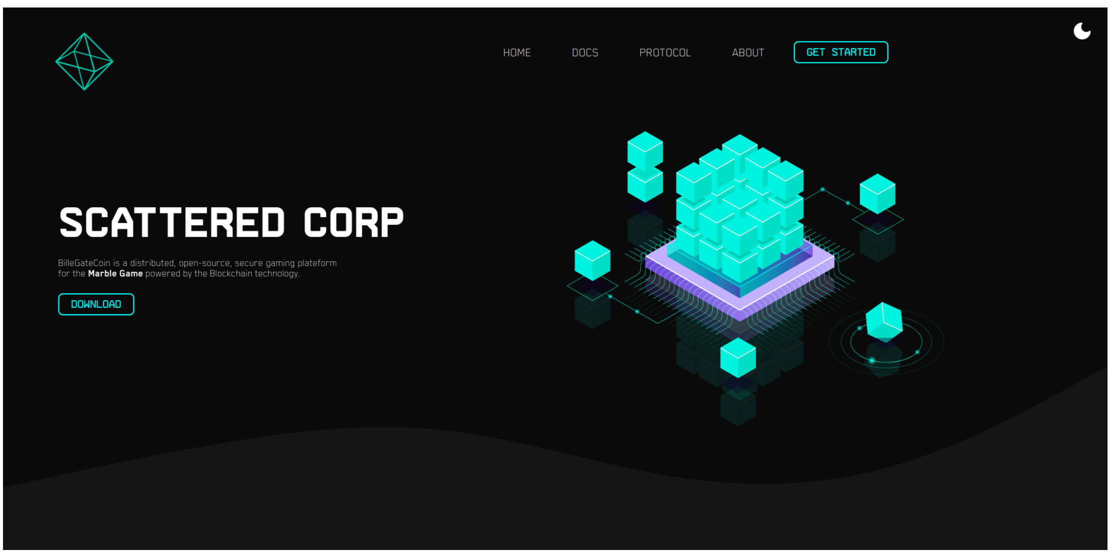




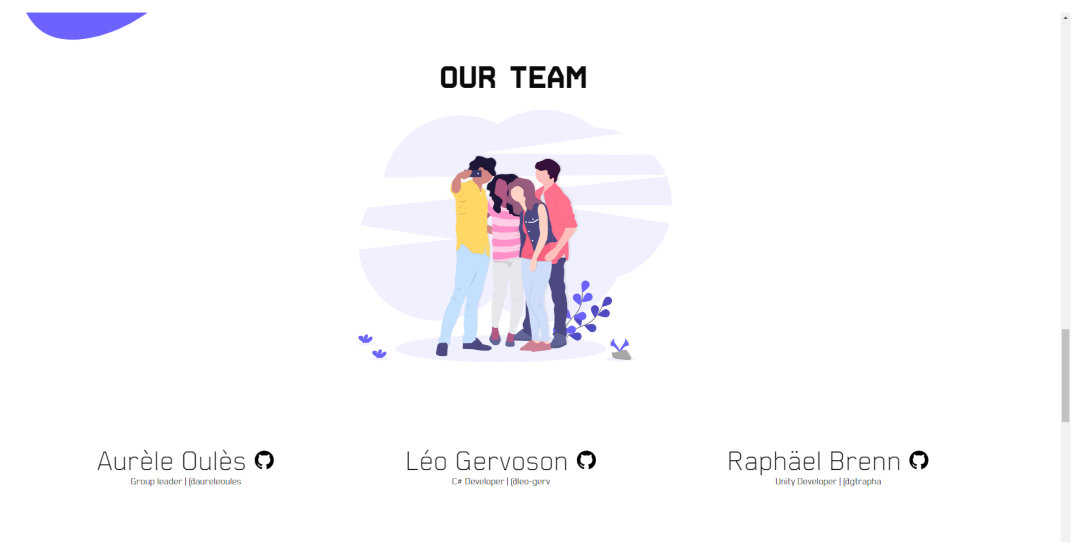




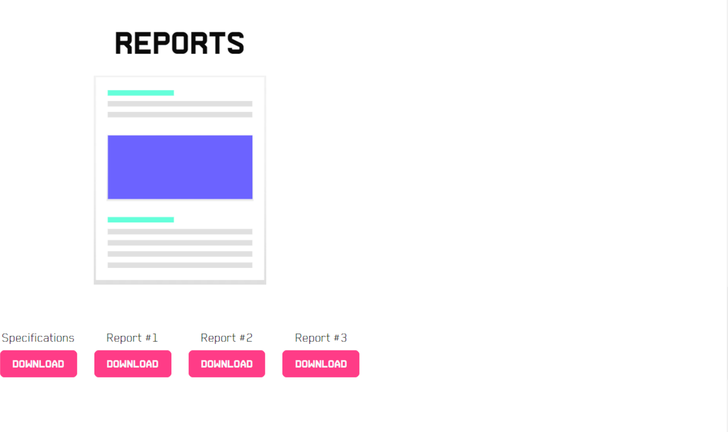
Un thème sombre a également été design afin d'apaiser les yeux des joueurs visitant le site. Il suffit de clicker sur la lune en haut à droite pour changer de thème.



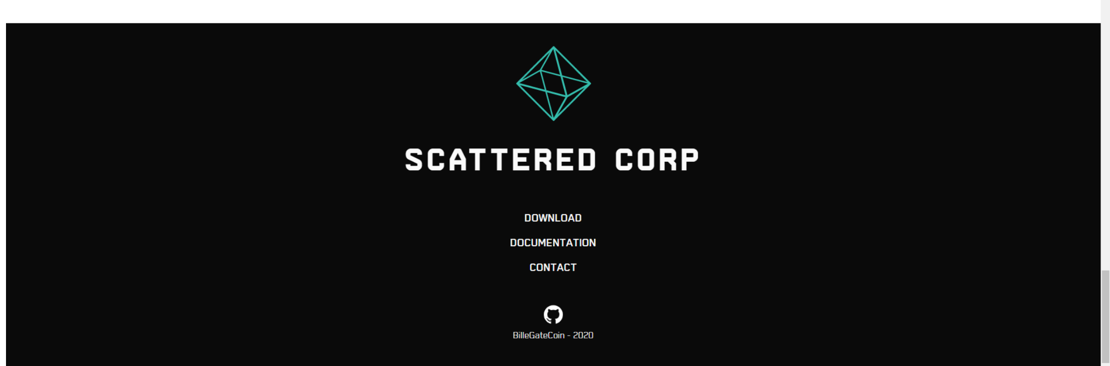
Nous avons rajouté une section qui présente notre groupe.



Cette section permet de télécharger nos rapports de soutenance ainsi que la cahier des charges.



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



8 Logiciels utilisés

8.1 VS Code

VS Code a été notre éditeur de code le plus utilisé durant ce projet.

8.2 Unity

Le jeu en lui même a été réalisé avec Unity, le logiciel qui nous a été imposé.

8.3 Discord

Nous avons créé un serveur Discord dédié au développement de BilleGateCoin. Nous avons créé un channel par tâche pour pouvoir visualiser simplement l'avancée du projet.

8.4 Git + GitHub

Nous utilisons l'outil de versionnage de code Git pour pouvoir collaborer sur notre code et revenir en arrière dans le temps quand il est nécessaire.

8.5 Photoshop

Le logo du projet et du groupe Scattered Corp ont été design sur Photoshop.

8.6 Google Chrome

Le site web a été developpé en utilisant majoritairement le navigateur Google Chrome en plus de ses outils de développeur.

8.7 LaTeX

Tous nos documents ont été rédigés en LaTeXavec un style personnalisé.

8.8 Google Sketchup

le logiciel de modélisation 3D Sketchup a été utilisé pour créer les modèles de billes ainsi que celui du terrain de jeu.

8.9 IntelliJ Rider

Rider a été utilisé pour l'édition et la gestion des fichiers contenant les différentes fonctions utilisées par le jeu.

8.10 Double Commander + 3DBrowser

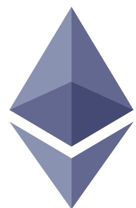
Ces deux logiciels ont étés utilisés en parallèle pour créer toute la collection d'icônes des billes du jeu.

9 Ressources

9.1 Bitcoin

https ://en.bitcoin.it/wiki/Main_Page
La documentation du Bitcoin nous a considérablement aidé durant le développement de ce projet. La plupart des fonctionnalités implémentées dans notre blockchain ont été inspirées du Bitcoin, notamment le porte-feuille, les signatures, le minage, et le réseau.

9.2 Ethereum



https ://ethereum.org/developers
La documentation d'Ethereum nous a également aidé pour l'indexation des inventaires des joueurs. Le système de transactions est très différent entre celui du Bitcoin et d'Ethereum. Nous avons préféré implémenter celui d'Ethereum car il est plus simple et plus adapté pour notre projet.

9.3 GitHub

La plateforme de collaboration GitHub nous a permis de trouver des projets open-source nous aidant à la réalisation de BilleGateCoin.

En particulier le repository Secp256k1.Net : **https ://github.com/MeadowSuite/Secp256k1.Net**. Ce projet nous permis d'effectuer des opérations sur la courbe elliptique secp256k1.

9.4 Undraw.co

Nous avons utilisés des illustrations au cours du projet, sur le site web et dans nos rapports de soutenances. Ces illustrations sont open-source et libre d'utilisation. Je tiens tout de même à remercier Katerina Limpitsouni pour son travail formidable.

10 Limites du projet et problèmes rencontrés

10.1 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-aléatoirements 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.

10.2 Triche

En effet, il n'y a pas de serveur central qui vérifie si les joueurs trichent, donc il n'y a aucun moyen de savoir si un joueur triche ou non. Un joueur pourrait tricher de plusieurs manières.

Si la bille du joueur est proche de celle de l'adversaire, qu'il lance sa bille et qu'il rate, il pourrait se déconnecter d'internet pour ne pas publier son lancé, et instantanément réessayer son coup. C'est un problème en effet, mais pas fun de la part du joueur.

Le code source et le protocole du jeu étant libre et ouvert, le joueur pourrait également développer son propre client de jeu qui lui permet de visualiser et effectuer les trajectoires les plus optimales pour toucher la bille de l'adversaire, sauf que ce joueur ne peut pas prédire le coup du joueur adverse, et peut donc complètement fausser ses calculs.

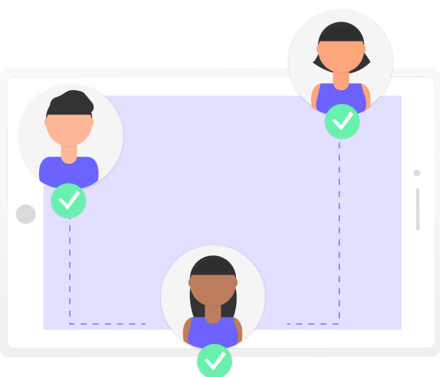
Deux joueurs pourraient également développer deux intelligences artificielles pour jouer à leur place. Le jeu deviendrait alors une compétition de celui qui a la meilleure intelligence artificielle. Peut-être que ce jeu est plutôt destiné aux intelligences artificielles réalisées par des humains, qu'un jeu pour humains.

L'historique des parties de tous les joueurs étant publique, si un joueur suspecte un autre joueur de tricher en regardant son historique de partie, celui-ci peut décider de ne pas jouer avec lui tout simplement.

10.3 Hole punching

Le fonctionnement des routeurs grand public ne permet pas d'ouvrir de ports au reste du réseau internet. En ouvrir manuellement est long, fastidieux, requiert un accès administrateur au routeur et est très difficilement possible d'automatiser.

On utilise donc en général une méthode peu orthodoxe pour ouvrir des ports, qui consiste à premièrement envoyer une requête à un serveur, ce qui ouvrira un port du routeur, puis garder ce port ouvert et l'utiliser comme port pour un serveur.



11 Améliorations

Notre jeu est très loin d'être parfait. Pour le rendre fonctionnel à grande échelle nous devrions implémenter bien d'autres fonctionnalités mais évidemment nous n'avons pas le temps et ce projet est un **Proof of Concept** avant tout.

11.1 Scalability

La question de la **scalability** se pose tout le temps dans un système décentralisé. Dans BilleGateCoin, nous devons pouvoir effectuer des parties rapides, donc il faut que les blocks de la chaine soient minés rapidement. Ici, nous avons un block miné toutes les deux minutes. Cela signifie qu'un block ne doit pas peser trop d'octets pour ne pas remplir le disque dur des utilisateurs en quelques jours. Mais limiter l'espace disponible d'un block limite également le nombre de parties que le réseau peut effectuer en deux minutes. Si nous augmentions la taille d'un block, peu de gens pourrait se permettre de devenir une node car la blockchain serait trop lourde, et donc de permettre au jeu de se décentraliser, mais si nous baissons la taille d'un block, de moins en moins de personnes peuvent jouer [2]. Ce dilemme n'est résolvable uniquement en optimisant la quantité de données des contrats.

2. Par exemple, Bitcoin Cash est un fork du Bitcoin. La taille des blocks de Bitcoin Cash font 8 Mo, comparé au Bitcoin : 1 Mo. Peu de gens peuvent se permettre stocker 1 Go par jour, ce qui rend Bitcoin Cash très centralisé.

11.2 Signatures avec Schnorr

L'espace disque est très restreint dans un environnement décentralisé comme le notre. Il serait possible d'optimiser la taille des contrats en octets en utilisant un nouveau système de signature.

En effet, aujourd'hui nous utilisons le système de signatures digitale ECDSA. Ces signatures pèsent 64 octets chacune. Dans deux contrats disponible sur BilleGateCoin, deux joueurs doivent ajouter leur signature au contrat, ce qui pèse 128 octets. Les signatures représentent un poids majeur dans le contrat.

Un mathématicien du nom de Claus Schnorr a travaillé sur un nouvel algorithme de signatures digitale : Schnorr signatures. Cet algorithme permet de générer unique signature de 64 octet pour plusieurs personnes sans révéler une seule clé privée. Les signatures Schnorr nous permettrait de gagner 64 octets par contrat demandant deux signatures. Les signatures Schnorr requièrent plusieurs échanges entre les joueurs afin de réaliser une signature complète. Cela signifie que nous devrions établir un moyen de communication **directe** en les joueurs de manière centralisée ou non [3].

11.3 Performances

3. Le routage type onion pourrait être utilisé, très similaire au Lightning Network développé sur le Bitcoin

Nous avons été contraint à utiliser le C# qui n'est pas un langage très performant pour effectuer des millions de vérifications de signatures, règles consensus, envoie de données par TCP/IP par secondes. Une implémentation du protocole de BilleGateCoin pourrait se faire dans un autre langage afin d'obtenir de meilleures performances.

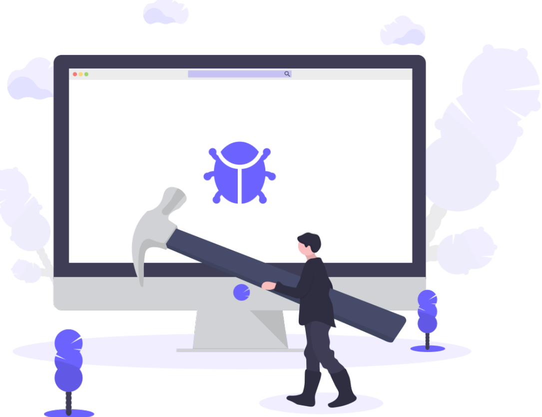
11.4 Vulnérable aux attaques DoS

Une attaque **DoS** (Denial of Service) est une attaque qui fait fonctionner au maximum les ressources d'un ordinateur pour le rendre indisponible. Notre implémentation de BilleGateCoin n'étant pas parfaite, laisse place à ce type d'attaque. Un attaquant pourrait demander à n'importe quelle node un block en continue et la node lui enverrai 256 Ko à chaque reqûete sans jamais le bannir. Cela saturerait le réseau, le disque, et le processeur de l'utilisateur.

11.5 Vulnérable à une attaque 51%

Une attaque 51% est propre aux blockchains. Lorsque la majorité de la puissance du réseau est dans les mains d'un individu, celui-ci peut altérer le bon fonctionnement du réseau en annulant les quelques derniers blocks ajoutés à la chaîne et donc peut essayer de relancer sa bille ou annuler une transaction. Dans une blockchain comme Bitcoin ou Ethereum cela n'est plus un problème car il serait extrèmement difficile de posséder 50% de la puissance du réseau vu la quantité d'énergie dépensée par les mineurs. Mais dans une blockchain à petite échelle comme la notre, cela est bien plus simple. De plus, 50% de la puissance du réseau donnerait aux mineurs seulement 50% de chance de miner un block, ce qui reste une faible probabilté pour la quantité d'enérgie investie. La seule solution à cette attaque est de rajouter des mineurs honnêtes sur le réseau.

11.6 Vulnérable à une attaque Ian Ternier



Due à la complexité du projet, il est fortement possible que si le jeu est amené à être testé, celui contienne des bugs non découverts ou résolus. Nous avons essayé de réaliser une blockchain entièrement fonctionnelle **from scratch**, mais nous avons du prendre des raccourcis à certains moments pour gagner du temps.

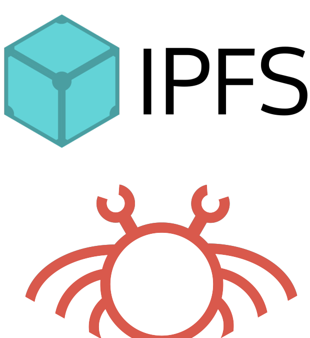
11.7 Remplacer le Proof of Work

Dans la plupart des blockchains, pour miner des blocks, le système utilisé est celui du Proof of Work. C'est-à-dire que le premier qui trouve un hash du block correcte, possède le droit de diffuser son propre block sur le réseau avec les transactions qu'il choisit à partir de la mempool. Le Proof of Work est plutôt facile à implémenter, mais celui-ci fait consommer beaucoup d'électricité aux mineurs. Aujourd'hui, environ 75% de l'énergie dépensée sur le Bitcoin est renouvelable, mais des ingénieurs cherchent à remplacer complètement le Proof of Work par un autre système pour ne pas consommer autant d'energie. Le Proof of Stake est donc une alternative, utilisé notamment par Ethereum

(en plus du Proof of Work). Le Proof of Stake est un type d'algorithme de consensus qui est conçu pour miner des blocks de manière probabiliste, comme sur du Proof of Work, mais en fonction du nombre de **tokens** que le mineur possède, et non la quantité d'énergie dépensée. Plus le mineur possède de tokens, plus il a de chance de miner le prochain block. Cet algorithme fonctionne sur le principe que si un mineur tente d'attaquer le réseau, la valeur des tokens du réseau baissera, et donc les tokens qu'il possède perdront également en valeur. Les mineurs sont donc plus motivé à protéger le réseau que de l'abîmer.

Le Proof of Stake pose de nouveaux problèmes tels que problème du capitaliste. Plus le mineur est riche, plus il s'enrichie car il a de plus en plus de chances de miner le prochain block. Ce problème est également comparable au Proof of Work : plus les mineurs minent des blocks valides, plus ils ont d'argent pour racheter du materiel de minage.

11.8 Décentralisation



Le code source de notre projet est accessible sur GitHub, qui est une plateforme centralisée. Pour atteindre une décentralisation complète de notre projet, nous pourrions distribuer notre code sur un protocole de partage de fichiers décentralisé tel que BitTorrent, IPFS, ou ORA [4]. Cela mitigerait les risques de point de défaillance unique.

12 Ressentis

L'ensemble du groupe est resté motivé jusqu'à la dernière soutenance du projet, et nous sommes très heureux du résultat final.

12.1 Aurèle

Je suis passionné par la décentralisation et le monde des blockchains. Ce projet a été l'occasion de renforcer ma connaissance sur le fonctionnement des blockchains et

4. Projet de second semestre développé par CrabWave à EPITA

de la cryptographie. Je prends peu de plaisir à développer des jeux vidéos, mais j'ai tout de même beaucoup apprécié participer au développement de ce projet ambitieux car c'est avant tout un réseau plus qu'un jeu. J'ai également beaucoup appris sur le protocol TCP/IP et les problèmes d'ouvertures de port.l J'aurais souhaité implémenter BilleGate-Coin dans un autre langage et un autre plateforme de jeu que Unity pour optimiser la vérification consensus, mais nous aurions sûrement perdu trop de temps.

12.2 Raphaël

Avant ce projet, même si j'avais déjà une petite expérience de codage, je n'avais jamais pris part à un projet aussi important (plusieurs personnes + une durée aussi importante), et je n'avais d'ailleurs jamais touché à Unity non plus, c'était donc une première pour moi, pour de multiples raisons. Le fait de travailler ainsi sur ce jeu m'a finalement extrêmement plu, durant toute la durée du projet, et m'a permis d'en apprendre énormément sur la création de jeu vidéo avec Unity et le langage C# ce qui fait que même si la gestion du temps s'est avéré être un problème qui nous empêchera de finir tout ce que l'on avait prévu de faire, je ressors très content de cette expérience qui m'a vraiment beaucoup appris.

12.3 Léo

Ce projet a été l'occasion pour moi d'approfondir mes connaissances en blockchain, cryptographie et en réseau TCP/IP, jusqu'à présent assez superficielles. J'ai aussi appris la complexité d'un réseau décentralisé et les problèmes posés par les dispositifs de sécurité chez les utilisateurs lambda (pare-feu, ports fermés...). Même si l'aspect gameplay est très pauvre à mon goût, le projet reste très intéressant dans l'ensemble et enrichissant.

13 Données personnelles

BilleGateCoin, contrairement à quasiment la totalité des autres jeux existant, ne stock aucune donnée personnelle du joueur sur un serveur, ni même sur la blockchain. La seul information propre au joueur est son adresse publique. Celle-ci est par défaut anonyme mais peut être associée à un humain, ou un ordinateur avec beaucoup de volonté et de recherche.

Ce projet est donc RGPD friendly.

14 Développement

Durant le développement de notre projet BilleGateCoin, nous avons utilisé plusieurs outils, notammenent Git en fusion avec GitHub. Cela nous a permis d'intégrer de manière continue notre projet avec des tests unitaires automatiques.

La fusion de code s'est faite sans difficulté entre les différents membres du projet.

Pour compiler notre projet rapidement nous utilisons Docker. Il suffit de taper **docker build -t code .** pour compiler le noyau de BilleGateCoin.

Cela va télécharger une image contenant **.NET Core 3.1** basée sur Ubuntu.

Puis cela va télécharger les dépendances nécessaires à la compilation du projet tels que **make et cmake**. L'algorithme de signature ECDSA sur la courbe secp256k1 est téléchargé depuis l'organisation du Bitcoin depuis GitHub puis compilé dans le processus Docker isolé.

L'image docker possède toutes les dépendances nécessaires à la compilation du projet, on peut enfin le compiler.

Pour le lancer, il suffit de lancer **docker run -it –rm core createwallet password123**.

```
$ docker build -t core .
$ docker run -it --rm core createwallet password123
```

15 License

Tout le code de notre projet est open-source et disponible github.com/scatteredcorp. La license choisie est **MIT**.

16 Conclusion

La technologie de la blockchain est encore très jeune et trop souvent employée pour buzzer. Ce projet BilleGateCoin a été l'occasion de montrer une application concrète de cette technologie : un jeu décentralisée où les objets sont collectionnables et non confiscables.

Notre groupe ayant déjà de l'expérience en programmation, nous souhaitions réaliser un projet complexe, qui nous permet de se démarquer, et faisable en l'espace de six mois. Le projet d'**AFIT** à EPITA nous a mis dans le bain de la cryptographie, puis le concept de cette blockchain originale a été élaboré par Aurèle.

Ce projet est à la base un **Proof of Concept** et nous ne savions pas réellement si celui-ci allait fonctionner comme nous le voulions, mais nous avons réussi. Il existe évidemment des limites et des bugs majeurs dans le projet, mais le concept fonctionne.

Nous avons tous appris beaucoup de nouvelles choses sur le fonctionnement des blockchains et les problèmes associées à celles-ci. Nous espérons également que ce projet du second semestre d'EPITA a permis aux examinateurs d'en apprendre plus sur les réseaux décentralisés tels que le Bitcoin ou Ethereum ainsi que leurs enjeux dans notre monde.