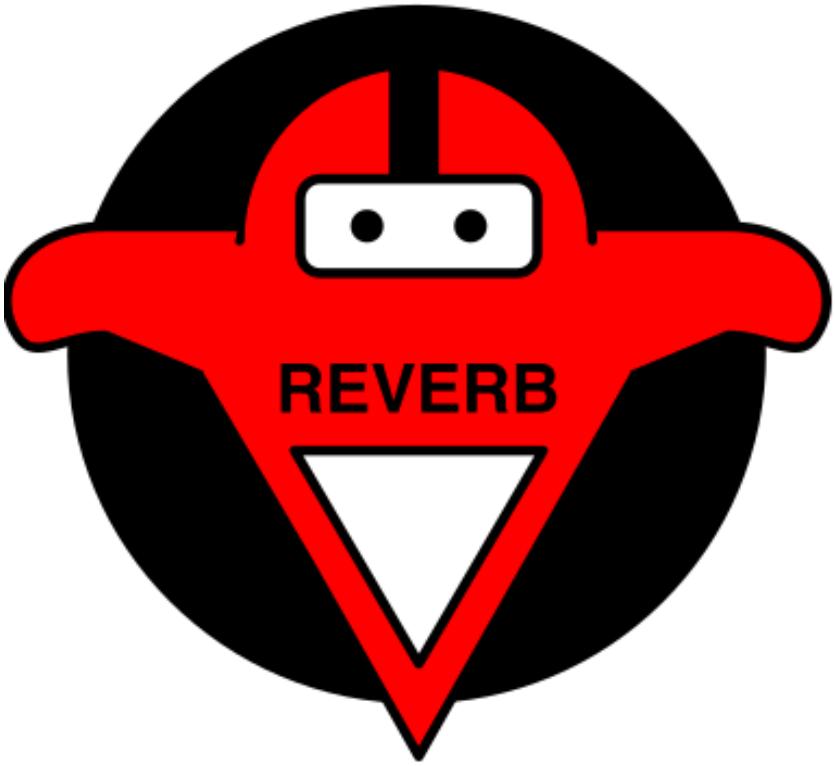
PROJET REVERB

Rapport projet final - Polytech Sorbonne



MAIN 5 - Semestre 9 - Année 2018/2019

Encadré par : François PECHEUX

Projet réalisé par :
Yassine ABBAR,
Aurélien ABEL,
Fatine BENTIRES ALJ,
Geng REN,
Alexia ZOUNIAS-SIRABELLA

Résumé

Ce rapport a pour but de présenter le projet REVERB réalisé dans le cadre de la cinquième année d'école d'ingénieur en spécialité Mathématiques Appliquées et Informatique à Polytech Sorbonne. Le projet `REVERB` (**R**enewable **E**nergy, **V**irtual **R**eality and **B**lockchain) répond a un nouveau problème sociétal qui est la consommation d'énergie. Pour cela, il prend exemple sur la modélisation d'un ensemble de lotissements auto-suffisants en énergie. Le principe est le suivant : prenons un pays chaud tel que le Mexique, les maisons du lotissement dotées de panneaux solaires vont pouvoir s'auto-alimenter en énergie. Afin de ne pas faire appel à des fournisseurs externes, des transferts entre les maisons vont également avoir lieu. Mais comment modéliser ces transferts et s'assurer une répartition équitable ? C'est ici que le projet prend tout son intérêt.

Pour commencer, nous avons du choisir un langage de programmation. Le C# s'est alors présenté comme un choix évident. La première raison est pédagogique. Notre encadrant, Monsieur François Pêcheux, voulait faire perdurer cette tradition qui est qu'un MAIN se doit de pouvoir manipuler un nouveau langage de programmation sans problème. La seconde est la multitude de bibliothèques existantes et son orientation objet très compatible avec une visualisation à l'aide d'un casque de réalité virtuelle.

Le rapport qui suit se décompose ainsi en 4 grandes parties. La première vise à présenter le principe de la `Blockchain` ainsi que l'implémentation que nous avons faite. La `Blockchain` est une technologie de stockage et de transmission d'informations qui se veut transparente, sécurisée, qui fonctionne sans organe central de contrôle et repose sur des principes de cryptologie. Habituellement utilisée pour les transferts d'argent, nous avons présenté dans cette partie notre implémentation de la `Blockchain` visant à transférer de l'électricité.

La seconde partie aborde quant à elle les modèles de comportement humain et météo qu'il nous a fallu mettre en place afin de rendre le modèle plus réaliste. Les fonctions développées dans cette partie prennent ainsi comme entrée les données extraites des sondages réalisés en 2017 par des élèves de la classe MAIN pour le projet industriel Tomaro. A travers cette partie nous ajoutons une connotation réaliste au projet en lui permettant de se baser sur des faits et attentes réels.

La troisième partie réfère à une méthode de gestion des tâches : la `Smartgrid`. Connue en France sous le nom de "réseau électrique intelligent", la `Smartgrid` est un réseau de distribution permettant de traquer l'ensemble des échanges de chaque maille en temps réel. Dans notre modélisation, une maille correspond à une maison. Ce réseau va alors permettre de gérer les transferts d'énergie entre les maisons données.

La dernière partie que nous abordons porte sur la fameuse application sans laquelle de nombreux jeux n'auraient jamais pu prendre vie : Unity. Développée par Unity Technologies, ce logiciel principalement utilisé pour les jeux vidéos permet de modéliser des scènes en 2 ou 3 dimensions. En utilisant ce logiciel, nous espérons permettre non seulement une meilleure compréhension de notre projet mais également se familiariser avec le monde du virtuel.

Remerciements

Nous tenons à remercier toutes les personnes qui ont contribué au bon déroulement de notre projet.

Tout d'abord, nous adressons nos remerciements à notre encadrant et professeur de l'école d'ingénieur Polytech Sorbonne, Monsieur François Pêcheux sans qui ce projet n'aurait pas eu lieu.

Nous tenons également à remercier Monsieur Sylvain Viateur qui s'est tenu à notre disposition de nombreuses fois pour des questions de logistiques concernant le casque de réalité virtuelle.

Finalement, merci à notre directrice, Madame Myriam Comte, pour avoir permis d'avoir un tout nouvel ordinateur et casque de réalité virtuelle Oculus Rift et merci à Madame Frédérique Charles, Madame Cécile Braunstein, Madame Fanny Villers et Monsieur Xavier Tannier pour nous avoir accompagnés et guidés tout au long de notre cursus.

Table des matières

Première partie
Introduction

”Dans son scénario de référence, l'EIA estime que la consommation mondiale d'énergie pourrait fortement croître dans les prochaines décennies : elle pourrait passer de 549 milliards de MBtu (British Thermal Unit) en 2012 à 629 milliards de MBtu en 2020 et 815 milliards de MBtu en 2040, soit une hausse de 48% en moins de trois décennies.”. Une telle constatation demande une meilleure gestion de l'énergie.

Le projet `REVERB` (**R**enewable **E**nergy, **V**irtual **R**eality and **B**lockchain) vise à simuler un transfert d'énergie plus efficace entre un ensemble de lotissements dotés de panneaux solaires. Considérons alors un pays chaud tel que le Mexique. Le principe est le suivant : la production énergétique d'une maison inoccupée va être transmise à une autre selon les besoins de ses habitants. Ces échanges vont alors être gérés par la `Smart Grid`, un réseau de distribution d'électricité qui favorise la circulation d'information entre les fournisseurs et les consommateurs afin d'ajuster le flux d'électricité en temps réel et permettre une gestion plus efficace. Les montants des transferts d'énergie quant à eux sont enregistrés dans une technologie de stockage et de transmission d'informations, transparente et sécurisée portant le nom de `Blockchain`. Cette dernière constitue alors une base de données qui contient l'historique de tous les échanges effectués entre ses utilisateurs depuis sa création. Au final, tous les résultats obtenus reposent sur l'utilisation de deux bases de données : une première qui modélise le comportement d'habitants et une seconde qui représente un modèle météo.

Ainsi nous commençons par présenter l'architecture générale du projet. Cette partie est indispensable pour avoir une idée claire des liens entre les différents composants. Après cela la première partie du projet répond aux composantes `RE` et `B` du projet c'est à dire à la modélisation de transferts d'énergies entre un ensemble de lotissements utilisant à la fois Blockchain et SmartGrid. La seconde partie, quant à elle, représente la composante `REV` du projet : la réalité virtuelle.

Deuxième partie

Architecture générale

Une notion propre à ce projet est la présence d'une multitude de composantes (`Blockchain`, `Smartgrid` et `réalité virtuelle`). Cette notion, bien évidemment intéressante, rend néanmoins la compréhension plus compliquée. Nous consacrons ainsi cette partie à une explication approfondie du lien entre les différentes composantes du projet.

La figure ci-dessous représente les liens entre les différentes composantes/ les différents fichiers créés.

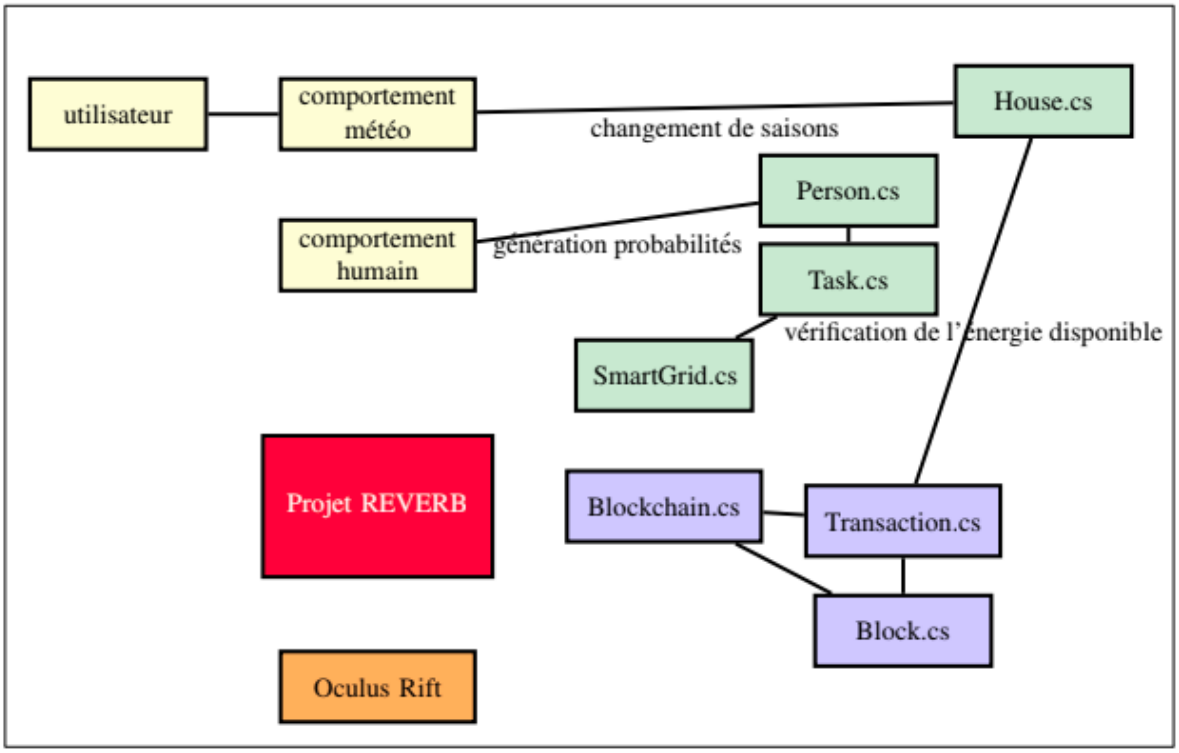


FIGURE 1 – *Représentation des liens entre les différentes classes du projet REVERB*

Le procédé est le suivant :

- `utilisateur, comportement humain et météo` : L'utilisateur va répondre à un sondage ce qui permet d'établir un comportement humain réaliste. Le comportement météo quant à lui est donné par différentes mesures physiques présentées dans la suite du rapport.

- `SmartGrid, House, Person et Task` : La base de données de comportement humain va être utilisée dans la classe `Person` pour générer des probabilités de réalisation de tâches. Ces probabilités produites suite à la lecture de la base de données vont ensuite être utilisées pour générer des tâches de la classe `Task` avec une certaine probabilité. En considérant que la probabilité pour la colonne machine à laver est de 20%, un exemple serait de générer un nombre aléatoire compris entre 0 et 100 et de produire une tâche "lancer la machine à laver" uniquement si le nombre aléatoire est inférieur à 20. La base de données météo est utilisée dans la classe `House`. Cette classe à pour but entre autre

de fournir une production d'énergie solaire différente selon les saisons. L'utilisation de ces bases de données permet ainsi d'obtenir un modèle de tâches et de production/consommation d'énergie plus réalise. Pour finir, la classe `SmartGrid` est utilisée pour trier les tâches produites par ordre de priorité (allumer la lumière la nuit est plus important que lancer une machine à laver).

- `Block, Blockchain et Transaction` : Cette partie permet de gérer une autre composante qui est les échanges d'énergies. La classe `block` va représenter une ou plusieurs transactions de la classe `Transaction` effectuées dans un laps de temps restreint. La classe `Blockchain` va permettre de fournir un historique des transactions en prenant en compte les caractéristiques des personnes (classes `Person` et `House`) et regroupe un ensemble de blocs (classe `Block`). Dans notre cas, effectuer une tâche (allumer la lumière) peut revenir à effectuer une transaction (demander de l'énergie à une autre maison). En effet, si la tâche nécessite plus d'énergie que celle captée par notre panneau solaire, effectuer la tâche revient alors à établir une transaction d'énergie. C'est ici que le lien entre la composante `Blockchain` et la composante `Smartgrid` prend son sens. Par ailleurs, la mesure de la production d'énergie par maison repose sur le modèle météo et la classe `House` présentés plus haut.

- `Oculus Rift` : Cette composante va permettre de visualiser ce qui se passe précédemment notamment la création d'une tâche telle que "lancer la machine à laver" ou sa réalisation. Elle nécessite l'adaptation du code précédent à Unity, le logiciel utilisé et Oculus Rift, le casque de réalité virtuel.

Troisième partie

Blockchain : un réseau de confiance

1 Principe de fonctionnement

La `blockchain` est une technologie de stockage et de transmission d'informations qui se veut transparente, sécurisée, qui fonctionne sans organe central de contrôle et repose sur des principes de cryptologie. Il s'agit d'une technologie qui permet d'instaurer de la confiance dans un groupe où la confiance ne réside pas. En effet, la `blockchain` permet de mettre en place un relevé des différentes opérations qui ont eu lieu entre les différents parties (membres) d'un réseau. Chaque partie du réseau ayant une copie de ses contrats, il n'est pas possible d'effectuer des manoeuvres frauduleuses puisqu'il n'est pas possible de modifier les copies des autres membres. Les champs d'applications de la `blockchain` sont très larges et vont de la finance à l'énergie en passant par la logistique.

Notre représentation de la `blockchain` s'inspire de celle qu'Anders Brownworth a implémenté en JavaScript ainsi que des notions de cryptographie.

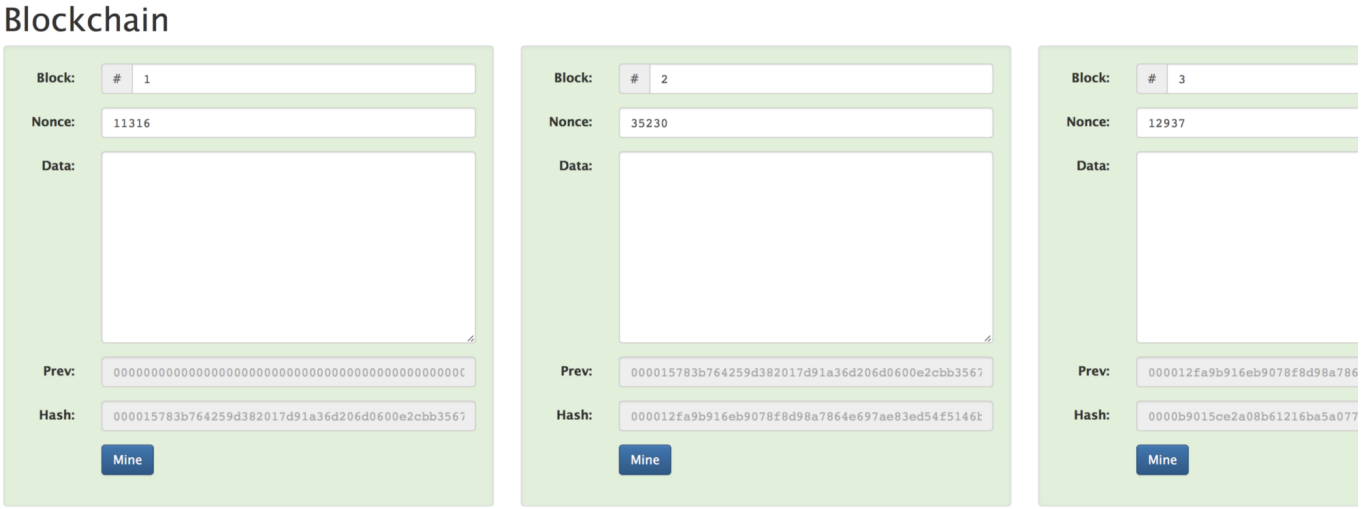


FIGURE 2 – *Blockchain tirée du site d'Anders Brownworth*

Dans cette représentation la `blockchain` est constituée d'un ensemble de blocs. Chaque bloc est ainsi décomposé en 4 champs :

- `bloc` : le numéro du bloc
- `Nonce` : permet la mise à jour de la clé de hachage de manière à ce qu'elle réponde à certains critères
- `Data` : champs vide qui contient le message que l'on veut transmettre
- `Hash` : clé de hachage associée au bloc

La clé de hachage associée à un bloc ressemble à un ensemble de nombres aléatoires, est unique et doit répondre à certains critères. Il s'agit d'une empreinte reliée à des informations. Elle sera toujours de même longueur. Elle sera toujours identique pour un message du champ `Data` donné. Une clé de hachage est

également disponible pour le message vide.

La `blockchain` peut ainsi être vue comme un alignement de blocs. En plus des champs présentés ci-dessus, les blocs seront constitués d'un champ `prev` qui correspondra à la clé de hachage du bloc précédent. Le champ `Data` correspondra également non pas à un champ vide mais à un ensemble de transactions. L'impossibilité de modifier de manière frauduleuse les données vient non seulement du fait qu'il existe de nombreuses copies d'une même `blockchain` mais repose également sur le fait qu'une clé de hachage pour un champ `Data` donné est unique. En modifiant les informations contenues dans ce champ, la clé de hachage sera donc modifiée. Étant donné qu'un bloc est composé de la clé de hachage du bloc précédent, modifier les données dans un bloc `x` va donc rendre le bloc `x + 1` obsolète. La clé de hachage du champ `prev` de ce bloc `x + 1` sera mauvaise. Une possibilité pour une personne ayant de mauvaises intentions est donc de modifier "à la main" ce champ. Ceci est possible si l'on se trouve en bout de `blockchain` mais beaucoup moins si l'on se trouve au début (une `blockchain` peut contenir des milliers de blocs).

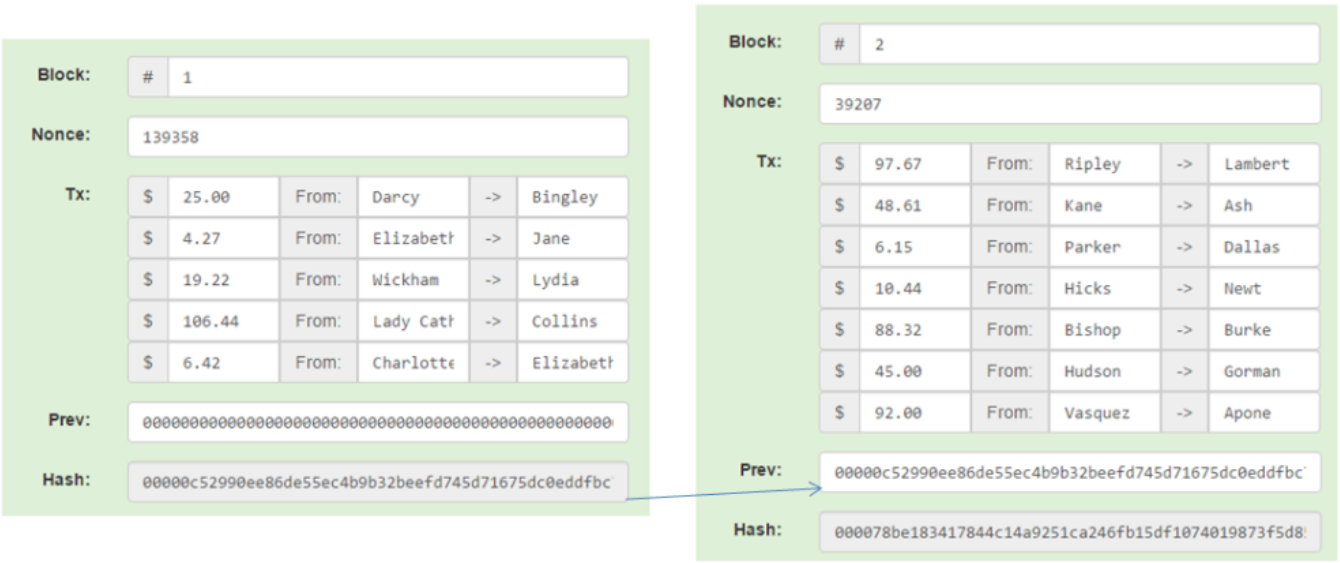


FIGURE 3 – *Blockchain avec champ Prev et Transactions tirée du site d'Anders Brownworth*

Le principe actuel de la `blockchain` veut que le début de la clé de hachage soit constitué par la répétition d'un même élément. Un exemple est `'000aeb8bbc'`. On dira ainsi qu'une clé est de difficulté `i` si elle est constituée `i` fois du même élément en début de chaîne. Tant qu'une clé de hachage n'a pas atteint la difficulté requise, l'utilisateur doit la mettre à jour en appuyant sur la bouton `Nonce`. Plus une chaîne sera difficile à obtenir et plus il sera compliqué pour une tierce personne d'avoir accès au message alors qu'elle ne devrait pas. Ainsi, ce processus de mise à jour de la clé de hachage est connu sous le nom de `minage` et une personne effectuant ce processus s'appelle un `miner`. Dans le monde de la Finance, le temps est considéré comme de l'argent, le `miner` a donc le droit à une récompense (bitcoin) pour avoir pris le temps de miner le bloc.

L'algorithme de cryptographie de création de la clé de hachage utilisé par Anders Brownworth est le `SHA-256`. Le `SHA-2` est une famille de fonctions de hachage qui ont été conçues par la National Security Agency des États-Unis. Cette famille de fonctions est constituée d'une fonction en particulier, la `SHA-256`. Cette fonction prend en entrée un message de taille arbitraire et produit un résultat (`hash` ou empreinte) de taille fixe. Dans le cas de l'algorithme `SHA-256`, la clé de hachage obtenue est de taille 256 bits.

Cependant, pour effectuer des transactions, les membres d'un réseau ont également besoin d'une clé publique et d'une clé privée. En cryptographie asymétrique (branche de la cryptographie) une clé publique associée à un membre est un ensemble de nombres aléatoires visibles par tous tandis que la clé privée doit elle être privée. Le principe est le suivant :

- **chiffrement du message à envoyer** : l'expéditeur utilise la clef publique du destinataire pour coder son message. Le destinataire utilise sa clef privée pour décoder le message de l'expéditeur, garantissant la confidentialité du contenu.
- **vérification de l'authenticité de l'envoyeur** : l'expéditeur utilise sa clef privée pour coder un message que le destinataire peut décoder avec la clef publique de l'expéditeur ; c'est le mécanisme utilisé par la `signature numérique` pour authentifier l'auteur d'un message.

L'objectif de la `blockchain` que nous sommes en train d'implémenter est de simuler non pas des transferts d'argent mais plutôt des transferts d'électricité entre un ensemble de lotissements dotés de panneaux solaires situés sur une même propriété. Le choix d'un tel modèle résulte du fait que l'énergie solaire présente plusieurs avantages écologiques. En effet elle permet de produire de l'électricité sans matière première et sans impact sur l'environnement. Cependant, elle présente également des limites comme la plupart des énergies renouvelables. L'efficacité de la production dépend fortement des conditions météorologiques et géographiques ce qui rend nécessaire l'utilisation de systèmes intelligents pour un meilleur ajustement de la production et de la consommation d'électricité.

2 Implémentation

Cette partie repose sur l'implémentation de deux fichiers : `block.cs` et `blockchain.cs`.

Le fichier `block.cs` contient l'implémentation d'un bloc décrit dans la section précédente ainsi que de nombreuses fonctions. Nous ne présenterons dans la suite que les principales fonctions.

La fonction `CalculateHash()` a pour but de calculer la clé de hachage. Pour cela nous utilisons la fonction `SHA256.Create()` de la bibliothèque `System.Security.Cryptography`. Nous convertissons le résultat en base64 à l'aide de la fonction `Convert.ToBase64String()`.

```
public string CalculateHash()
{
    SHA256 sha256 = SHA256.Create();
    byte[] inputBytes = Encoding.ASCII.GetBytes($"{TimeStamp}-{
   PreviousHash ?? ""}-{Transactions}-{Nonce}");
    byte[] outputBytes = sha256.ComputeHash(inputBytes);
    return Convert.ToBase64String(outputBytes);
}
```

La fonction `mineBlock()` a pour objectif d'obtenir une clé de hachage pour une difficulté donnée. Elle fait ainsi appel à la fonction `CalculateHash()` présentée précédemment. Tant que le résultat obtenu n'est pas satisfaisant, nous continuons de faire appel à cette fonction.

```
public void mineBlock(int difficulty)
{
    string proof = new String('g',difficulty); //to have 3 times g at the beginning
    do
    {
        Hash = CalculateHash();
        Nonce++;
    } while(Hash.Substring(0,difficulty) != proof);
}
```

Le fichier `blockchain.cs` contient l'implémentation d'une blockchain qui correspond à une liste de blocs. Elle contient ainsi de nombreuses fonctions telles que `CreateFirstBlock()` qui permet la création d'un bloc, `AddBlock()` qui va avoir pour fonction de lier deux blocs entre eux ou encore `CreateTransaction()` qui va créer une transaction. Les principales fonctions sont néanmoins :

La fonction `ProcessPendingTransactions()`. Cette fonction a pour but d'ajouter une transaction à la liste de transactions existantes.

```
public void ProcessPendingTransactions(House minerAddress)
{
    Block block = new Block(DateTime.Now, GetLatestBlock().Hash, PendingTransactions);
    AddBlock(block);
    PendingTransactions = new List<Transaction>();
    CreateTransaction(new Transaction(new House(0,"inconnu"), minerAddress, Reward));
}
```

La fonction `IsValid()` qui va vérifier si la signature d'un bloc est valide.

```
public bool IsValid()
{
    for (int i = 1; i < _chain.Count; i++)
    {
        Block currentBlock = _chain[i];
        Block previousBlock = _chain[i - 1];

        if (currentBlock.Hash != currentBlock.CalculateHash()) // when we update transaction data
        {
            return false;
        }
        if (currentBlock.PreviousHash != previousBlock.Hash) //  link between blocks is invalid
        {
            return false;
        }
    }
     return true;
}
```

La fonction `GetBalance()` qui va mettre à jour le montant disponible dans les comptes des membres ayant fait l'objet d'une transaction.

```
public int GetBalance(House address ,SmartGrid smartgrid) // get balance
    for one address
{
    int balance = 0;
    for (int i = 0; i < _chain.Count; i++)
    {
        for (int j = 0; j < _chain[i].Transactions.Count; j++)
        {
            var transaction = _chain[i].Transactions[j];
            if (transaction.FromAddress == address && balance>transaction.
    Amount)
            {
                balance -= transaction.Amount;
                smartgrid.update_energy(transaction.Amount); //we only do
    that once
            }
            if (transaction.ToAddress == address && balance>transaction.
    Amount)
            {
                balance += transaction.Amount;
            }
        }
    }
}
```

Quatrième partie

Modèles de comportement humain et météo

3 Comportement humain

3.1 Architecture de la base de données

Dans cette section, la base de données utilisée est celle créée par les membres du projet **Tomaro**. Cette dernière regroupe les réponses à un sondage de 283 personnes. Les questions évoquent les habitudes des participants durant différentes tranches horaires (6-9h, 9-12h, 12-15h, 15-18h, 18-21h, 21-00h et 00-6h), une distinction est également faite entre les jours travaillés ou non :

- Temps (en minutes) d'utilisation de la télévision, jeux vidéos compris
- Temps (en minutes) d'utilisation d'un ordinateur personnel
- Temps (en minutes) d'utilisation des plaques à induction ou four
- Temps (en minutes) d'utilisation d'un appareil de cuisine de courte durée dans la journée (Machine à café, micro-onde, grille-pain, etc)
- Temps (en minutes) d'utilisation d'un appareil de salle de bain de courte durée dans la journée (Sèche-cheveux, lisseur, etc)

Ce sondage demande également le nombre d'utilisations hebdomadaires des appareils suivants :

- Machine à laver
- Lave-vaisselle
- Sèche linge

Pour finir, les participants doivent spécifier s'ils ont ou non la climatisation et/ou un chauffage d'appoint.
Il est important de noter que les différentes points évoqués correspondent à une colonne du fichier `.csv` contenant la base de données (par exemple, la tâche "Regarder la télévision entre 6-9h" correspond à la première colonne).

Bien que non exhaustive, cette base de données permet de dresser un premier modèle de comportement utilisable pour notre simulation.

3.2 Utilisation des données

Pour utiliser cette base de données, nous avons dû répondre à différents questionnements concernant la manière dont nous voulions modéliser le comportement humain et l'interaction avec le lotissement. Les fichiers correspondants à cette implémentation sont `Person.cs` et `house.cs`.

Tout d'abord, les réponses fournies par les participants au sondage permettent de calculer la probabilité que chaque évènement arrive. Par exemple, nous serons en mesure de calculer la probabilité que quelqu'un veuille utiliser la télévision entre 6h et 9h. Pour chaque évènement X, nous noterons la probabilité qu'il se produise $\mathbb{P}_X$. La fonction `probability()` de la classe `Person` prend en entrée une colonne (évènement sur une tranche horaire particulière) et calcule la probabilité d'occurrence de cet évènement.

```
public double probability(int column)
{
    int counter = 0;
    string[] habit2;
    double proba = 0.0;
    for(int line = 1; line <283; line++) //we read all the lines
    {
        habit2 = survey[line].Split(",");
        if(Int32.Parse(habit2[column]) != 0)
        {
            counter+=1;
        }
    }
    proba = (double) counter/ (double) 283;
    return proba;
}
```

La seconde étape était de pouvoir lire et exploiter ces données. Pour cela, on ouvre la base de données dans la classe `person` puis on récupère une case au hasard du fichier. Par la suite, on génère aléatoirement un chiffre représentant une probabilité (fonction `action()`). Si la probabilité ainsi générée est inférieure à la probabilité d'effectuer la tâche (*random* $\leq \mathbb{P}_{\mathbb{X}}$) alors on assigne cette tâche à une personne du lotissement.

```
event_prob = new Random().NextDouble();
if(event_prob < pc_prob_6_9)
{
    if(Int32.Parse(habit[7]) != 0)
    {
        created_task.Add(new Task(2,current_time,Int32.Parse(habit[7]),
   name));
    }
}
```

Par la suite la fonction `AddFamilyMember()` de la classe `House` permet de passer d'un système "individuel" (un objet de la classe `person` représentait une maison au départ) à un système "familial" (un ensemble de personnes représentent une maison).

```
public void AddFamilyMember(Person NewPerson)
{
  _family.Add(NewPerson);
}
```

4 Simulation Météo

4.1 Architecture de la base de données

Là encore, nous allons utiliser la même base données météo que celle du projet **Tomaro**. Cette dernière recense différentes informations sur une année complète. Ces informations sont renseignées à chaque heure de chaque jour. Pour plus de réalisme et ne pas suivre le comportement climatique d'une seule année, les mois sont pris à des années différentes, par exemple, les mois de Décembre et Novembre sont ceux de l'année 2009 alors que le mois d'Octobre est celui de 2013. Par ailleurs, les différentes informations recueillies sont les suivantes :

- Température (en Degré Celsius)
- Rayonnement solaire direct normal ($\frac{W}{m^2}$)
- Éclairement énergétique horizontal diffus ($\frac{W}{m^2}$)
- Rayonnement infrarouge vers le bas ($\frac{W}{m^2}$)
- Vitesse du vent ($\frac{m}{s}$)

4.2 Utilisation des données

L'implémentation de la simulation météo est disponible dans le fichier `house.cs`. Les principales fonctions sont les suivantes :

La fonction `ChangeSeason()` permet de modifier la saison.

```
public void ChangeSeason(Season season)
{
  _season = season;
}
```

La fonction `AddProductionPerDay()` permet de simuler une production d'énergie dépendant de la saison. Dans le modèle originel que nous avions créé, cette production d'énergie était infinie. Ce nouveau modèle est donc beaucoup plus réaliste.

```
public void AddProductionPerDay()
{
    switch(_season)
    {
        case Season.winter:
          _solar_panel_battery = 100;
        break;
      case Season.autumn:
        _solar_panel_battery = 200;
        break;
      case Season.spring:
        _solar_panel_battery = 300;
        break;
      case Season.summer:
        _solar_panel_battery = 400;
        break;
    }
}
```

Cinquième partie

SmartGrid : une méthode de gestion des tâches

5 Principe de fonctionnement

La `SmartGrid` est un réseau de distribution d'énergie intelligent permettant de traquer l'ensemble des échanges de chaque maille en temps réel. L'intérêt d'un tel réseau est avant tout d'optimiser la production, la consommation et le stockage d'énergie. En effet il n'est pas évident de stocker l'électricité avec un coût réduit, il est donc préférable de la consommer le plus rapidement possible. C'est pourquoi, la `SmartGrid` cherche à ajuster en temps réel l'utilisation de l'électricité en hiérarchisant les besoins de consommation selon leur urgence. La `SmartGrid` fonctionne avec trois modes différents :

- Surproduction, c'est à dire lorsque la production est supérieure à la consommation. Dans ce cas, on peut soit stocker l'énergie soit vendre le surplus aux réseaux extérieurs.
- Équilibre, cela correspond à l'équilibre entre la production et la consommation. C'est cette position idéale qu'une `SmartGrid` doit atteindre.
- Sous production, ce cas se produit lorsque la demande d'énergie est supérieure à la production. Afin de revenir dans l'état d'équilibre, la `SmartGrid` aura tendance à limiter la consommation des appareils énergivores voir même les arrêter si cela ne dérange pas les personnes concernées. Il pourra aussi reporter dans le futur la mise en route du fonctionnement de certaine machine.

6 Implémentation

Nous avons créé une classe `SmartGrid` possédant comme attribut la liste des tâches à ordonner ainsi que la liste des maisons représentant chacune un élément dans notre réseau. Le réseau prend connaissance de la liste tâches à effectuer toutes les minutes afin de savoir si des nouvelles tâches ont été créées. Dans le cas échéant, cette dernière sera ajoutée à la liste des tâches à ordonnancer.

Chaque tâche (fichier `Task.cs`) se voit ainsi attribuer un nombre de points en fonction de sa priorité (fonction `points_attribution()`) ainsi qu'une consommation d'énergie (fonction `calculate_consumption()`). Cette notation est bien évidemment subjective mais nous avons estimé qu'il est par exemple plus important d'allumer la lumière que de lancer une machine à laver.

Une fonction `Chosen_Task()` (objet de `Task`, fichier `Person.cs`) a également été créée afin de pouvoir ajouter une nouvelle tâche choisie à la liste des tâches à effectuer.

```
public void Chosen_Task(Person person, int lvl, DateTime current_time, int
    duration) //to add a new task into the list
{
    list_tache.Add(new Task(lvl,current_time, duration, name));
}
```

Les tâches sont finalement analysées et ordonnancées par ordre de priorité par la fonction `sort_task()` de la classe `SmartGrid`.

```
public void sort_task(DateTime current_time)
{
    _total_task.RemoveAll(item => DateTime.Compare(item.end_time,
   current_time) < 0);
    _total_task = _total_task.OrderByDescending(x => x.points).ThenBy(x =>
    x.consumption).ToList();
    for(int i = 0; i < _total_task.Count; i++)
    {
        int indice_house = _list_maison.FindIndex(x=> x._familyName ==
   _total_task[i].creator);
        if(_list_maison[indice_house]._solar_panel_battery < _total_task[i
   ].consumption)
        {
            if(_total_energy > _total_task[i].consumption)
            {
                for(int j = 0; j < _list_maison.Count; j++) // on regarde
   si y a un potentielle donneur
                {
                    if (_list_maison[j]._solar_panel_battery > _total_task
   [i].consumption - _list_maison[indice_house]._solar_panel_battery) //
   si une maison a une   nergie   plus    lev    que celle demand
                    {
                    var donneur = _list_maison[j];
                    var receveur = _list_maison[indice_house];
                    _blockchain.CreateTransaction(new Transaction(ref
   donneur, ref receveur, _total_task[i].consumption - _list_maison[
   indice_house]._solar_panel_battery));

                    _list_maison[j] = donneur;
                    _list_maison[indice_house] = receveur;
                    break;
                    }
                }
            }
            else
            {
            _total_task[i].recompute_time(30.0);
            }
            else // si la maison a assez d' nergie
            {
            _list_maison[indice_house]._solar_panel_battery -= _total_task
   [i].consumption;
            }
        }
    }
}
```

L'ordonnancement des tâches, quand à lui, repose dans un premier temps sur un système de points mis en place lors de la création des tâches. En effet, chaque tâche réfère à une action qui se voit attribuée un nombre de point allant zéro à sept. Zéro correspond alors à une tâche que l'on peut reporter dans le temps (comme par exemple l'allumage de la machine à laver) tandis que 7 sera associé à une tâche urgente qui doit être réalisée immédiatement (comme le besoin d'éclairage la nuit). Dans un second temps, si deux tâches se voient attribuer un même nombre de points, l'heure de la demande primera. Enfin suivant la consommation énergétique, certaines tâches pourrons être effectuées indépendamment des points qui leur sont attribués.

Sixième partie

Unity : donner vie au code

7 Un moteur 3D complet

Développé par Unity Technologies, ce logiciel principalement utilisé dans le jeu vidéo permet de modéliser des scènes en 2D ou 3D. Son principal avantage réside dans le multi-plateforme qui permet une utilisation autant sur PC, Mac, Navigateur, smartphone ou console de jeu. Le langage de programmation utilisé pour les scripts est le C#, il est donc aisé de visualiser les codes écrits précédemment puisqu'il s'agit du même langage de programmation.

Son approche est orientée `assets`. Ce terme désigne toutes les ressources nécessaires à la composition d'un jeu telles que les personnages, les objets, les sons ou les textures. Ces ressources peuvent être importées par l'utilisateur lui-même ou grâce à la boutique virtuelle `Asset store` disponible directement dans le logiciel. La plupart sont payantes mais les assets gratuites sont amplement suffisantes.

Ce logiciel complexe est compatible avec la réalité virtuelle grâce à des assets permettant la gestion de la caméra et des manettes qui permettent une interaction entre l'utilisateur et l'environnement virtuel.

Les deux principaux casques de réalité virtuels disponibles sur le marché sont : l'`Oculus Rift` de `Facebook` (utilisé lors de ce projet) et le HTC Vive de `HTC et Valve`. Ces deux casques demandent l'utilisation d'assets différents mais le principe d'utilisation reste le même.

7.1 Le village

7.1.1 Une première modélisation

La première étape consistait à modéliser un simple village avec des assets contenants des maisons et personnages minimalistes. Un aspect esthétique et pratique a dû être pris en compte pour obtenir le résultat final ci-dessous.

FIGURE 4 – *Modélisation d'un village de 8 maisons*

De plus, un programme en C# a également été mis au point afin de permettre de choisir le nombre de maisons à modéliser de manière à rendre le programme fonctionnel pour différentes échelles de mesure.

7.1.2 Intégration du code

La principale difficulté de cette partie a été d'intégrer le code modélisant la `Blockchain`, la `SmartGrid` ainsi que le comportement humain à notre village 3D. Pour ce faire, nous avons dû, dans un premier temps, charger sous `Unity` toutes les bibliothèques utilisées et s'assurer que la version du code C# utilisée était la même.

Dans un second temps, nous souhaitions créer des liens entre les classes du code et nos objets 3D. La première idée était de modifier le constructeur du fichier `house.cs` afin qu'il attache une maison 3D à cette classe. Le constructeur devient donc `public House(int season, string name, Transform house)` où la variable `Transform house` fait référence à une maison de notre lotissement. Ainsi, lorsque le code déterminait qu'un habitant du lotissement souhaitait effectuer une tâche, une icône représentant cette dernière apparaît au dessus de la maison de la manière suivante :

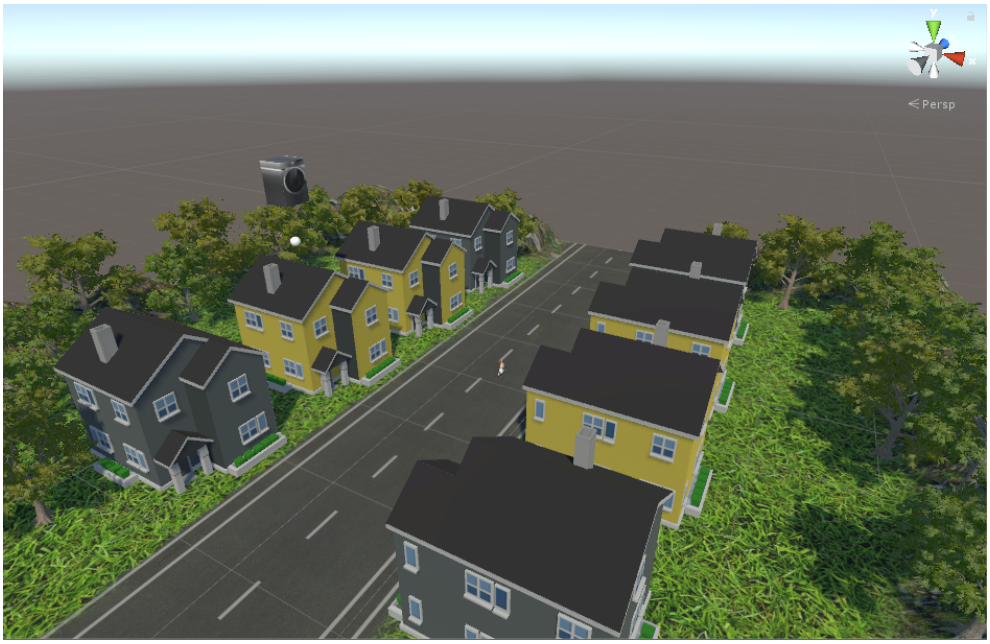


FIGURE 5 – *Modélisation de l'apparition d'un icône*

Dans l'exemple ci-dessus, l'observateur peut se rendre compte des demandes successives propres à chaque maison. Cette modélisation n'a été réalisée que pour la demande "Machine à laver" mais peut aisément être généralisée aux autres en choisissant une icône pour chaque tâche possible. La représentation sous forme de bulle a été un choix de l'équipe. L'implémentation de cette partie se fait de la manière suivante :

```
sphere = new GameObject[1];
selector = GameObject.CreatePrimitive(PrimitiveType.Sphere);
selector.transforme.localScale = new Vector3(1f, 1f, 1f);
sphere[0] = Instantiate(selector) as GameObject;
sphere[0].tranform.position = new Vector3(35f,15f,43f);

machine = new Transform[1];
machine1.transform.localScale = new Vector3(5f,5f,5f);
```

```
position = new Vector3 (35f,19f,43f);
rotation = new Vector3 (0f,-90f,0f);
machine[0] = Instantiate (machine1, position, transform.rotation);
machine[0].transform.Rotate (transform.up * (-180));

Destroy(selector);
Destroy(machine1);
```

On commence d'abord par créer la sphère (lignes.1-2) qui se trouve en dessous de l'icône puis on la place au dessus de la maison avec la taille voulue (lignes.3 à 5). Enfin, le deuxième paragraphe permet de créer l'objet "machine", lui donner la taille souhaitée et de le positionner. Ainsi, la visualisation du comportement des habitants du lotissement à la fin du projet est la suivante :

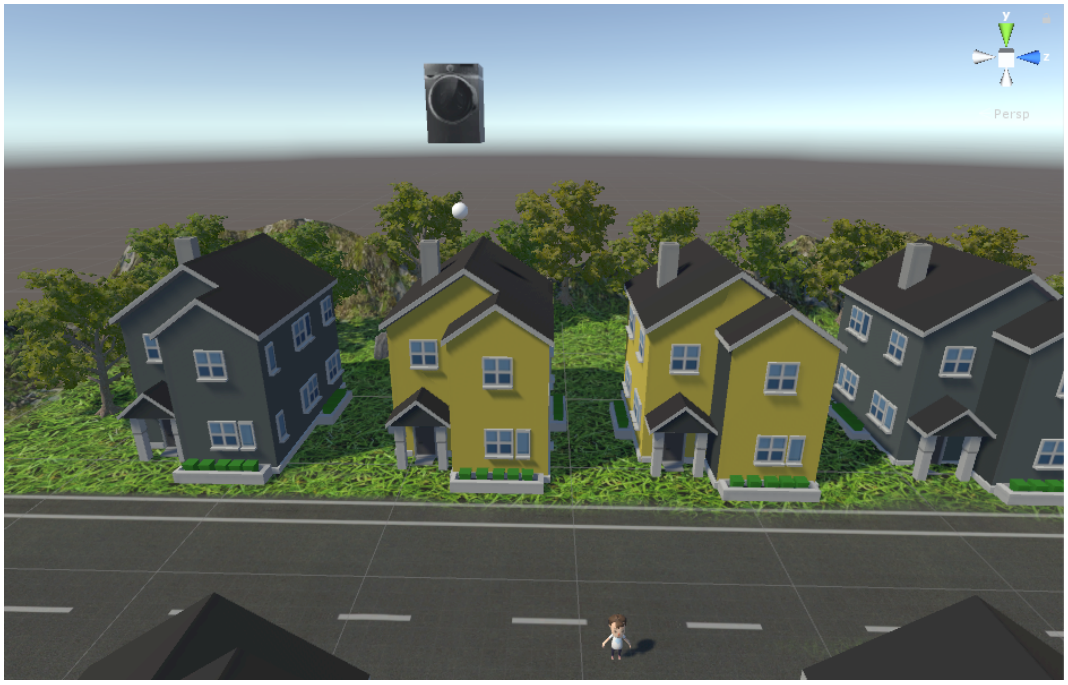


FIGURE 6 – *Représentation finale*

Dans un troisième temps, il aurait été très intéressant de modéliser, dans un but pédagogique, le fonctionnement et l'interaction de la `SmartGrid` et de la `Blockchain`. Cependant, cela n'a pu être réalisé par manque de temps.

7.2 La réalité virtuelle

La seconde étape correspond à la synchronisation du casque de réalité virtuelle et de Unity dans le but de s'assurer que le village est visible de façon claire avant d'y intégrer les éléments de `blockchain`.

Nous avions au début uniquement accès au casque HTC Vive développé par HTC et Valve. Pour utiliser le casque dans Unity, l'asset `Steam VR` était nécessaire. Celui-ci permet d'intégrer une caméra avec un champ de vision plus large que pour un écran d'ordinateur mais également 1 ou 2 manettes qui seront contrôlées par le joueur et sensible aux déplacements de la main dans l'espace. Il fallait donc insérer dans un premier temps l'objet correspondant à la caméra du nom de [CameraRig]. Cet objet est déplaçable dans l'espace de la même manière qu'une caméra classique et il est également possible de l'associer à un personnage et de

suivre ses mouvements. Associées à la caméra se trouvent les deux manettes personnalisables (possibilité de donner en jeu l'apparence de deux manettes, deux mains, deux pistolets...).

Nous avons fait par la suite l'acquisition d'un casque Oculus Rift dont la société a été rachetée par Facebook en 2014. L'utilisation est très similaire à celle de HTC Vive. L'avantage principale réside dans l'absence de capteurs fixes à placer dans la pièce où est utilisée l'appareil. Deux antennes facilement maniables en face de la zone d'utilisation suffisent.

FIGURE 7 – *Oculus Rift*

Septième partie

Critique des résultats obtenus et futurs développements

A la fin de ce semestre de projet, notre implémentation permet de répondre à la problématique principale qui est la modélisation d'un ensemble de transactions entre les maisons d'un lotissement et la mise en place d'un modèle de comportement humain et météo réaliste. Nous avons ainsi implémenté les fonctions essentielles du projet ce qui permet à notre encadrant d'effectuer une présentation lors des forums d'écoles d'ingénieurs comme annoncé.

Dans sa version actuelle, il est possible d'observer un ensemble de personnes demandant la réalisation de tâches (allumage de la télévision,...), l'ordonnancement de ces tâches et leur réalisation en temps réel. Il est également possible de voir des demandes provenant de personnes d'âges, de sexes et d'habitations variées. En plus de l'implémentation "brute", nous avons obtenus plusieurs visualisations visibles avec le casque de réalité virtuelle.

Pour aider à la transition en fin de projet et à la pérennité de l'implémentation faite, nous avons pris le temps de commenter chaque fichier créé. Ces ressources seront utiles non seulement à notre encadrant mais également à toute personne qui voudra s'y intéresser.

La première phase du projet (15 jours pendant la mois d'octobre) a été consacrée à la documentation et la prise en main du langage de programmation C#. Elle a également donné lieu à la décomposition de l'équipe en deux parties : équipe `blockchain` et équipe `Unity`.

La seconde partie du projet s'articulait ainsi autour de ces deux points avec la mise en place du code d'un côté et l'application visuelle de l'autre.

Au cours de ce travail, la difficulté principale a été la gestion du temps consacré au projet en parallèle de nos études. En effet, s'agissant de notre dernier semestre en école, la charge de travail extérieure au projet pouvait être considérable. Une autre difficulté correspond également à la mise en relation des différentes composantes du projet. Modifier le code afin de rendre compatibles les classes entre elles et rendre compatibles les fonctions créées avec Unity afin d'utiliser le casque de réalité virtuelle n'est pas toujours évident. L'encadrant a été cependant très compréhensif et satisfait du travail déjà accompli, même si certains items non traités auraient pu apporter quelques améliorations à ce prototype.

Pour le futur, il serait intéressant de développer cet aspect de réalité virtuelle en ajoutant des visualisations. Il serait également intéressant d'ajouter de nouvelles tâches qu'il serait possible d'accomplir et si possible combiner le projet avec du `Machine Learning` (demandé par l'encadrant, mais non possible vu l'état actuel des données) afin de pouvoir associer de nouvelles tâches à réaliser à des tâches pré-existantes et donc de rendre le modèle plus "intelligent".

Huitième partie
Conclusion

La finalité du projet REVERB est non seulement de simuler un ensemble de maisons qui échangent de l'énergie entre eux, mais aussi de voir cette interaction en temps réel grâce à la réalité virtuelle. Actuellement, notre implémentation modélise un comportement humain et météorologique grâce à l'utilisation d'une base de données créée par les membres du projet **Tomaro** mais aussi le fonctionnement complet d'une `BlockChain`. Il est également possible de gérer la demande en énergie du lotissement en utilisant la `SmartGrid`. Certains de ces aspects peuvent être visualisés via Unity comme l'évolution des demandes des habitants du lotissement.

Grâce à ce projet, nous avons pu apprendre un nouveau langage de programmation, le C#. Un des points les plus importants est qu'il nous a permis de monter en compétence sur de nombreuses technologies et notamment sur la `Blockchain`, une technologie utilisée dans beaucoup de domaines. De plus, ce projet est destiné à être présenté lors de forums d'école, c'est pourquoi nous nous sommes efforcés à rendre notre application la plus user-friendly et interactive possible. Nous étions également très bien encadrés grâce à la présence permanente de M. Pêcheux, ce qui nous a permis d'avancer rapidement tout au long du projet.

La multitude de tâches à effectuer nous a permis à chacun de trouver sa place dans le groupe et de s'investir dans ce qui nous intéressait le plus. C'est un facteur majeur dans l'avancée de notre projet. Le plaisir d'avancer et de trouver de nouvelles idées n'a cessé d'être de mise.

En conclusion, nous sommes satisfaits par ce projet car beaucoup de fonctionnalités sont intégrées rendant ainsi notre application interactive. Cependant, notre programme n'est pas complet, il sera intéressant d'inclure une partie intelligence artificielle afin de prédire la consommation et les échanges d'énergie dans le futur.

Annexes

Description de l'organisation du travail

Afin d'avoir tous accès aux mêmes compétences et que tous les membres aient l'occasion de s'essayer à tous les rôles au cours du projet, nous avons commencé en établissant un planning de rôles tournants dans le groupe.

Membre du groupe	Manager	Secrétaire	Scribe 1	Scribe 2	Rédacteur
Yassin Abbar	31 Déc.-fin Jan.	12 Nov.-2 Déc.	3 Déc.-30 Déc.	15 Oct.-11 Nov.	Sept.-14 Oct.
Aurélien Abel	3 Déc.-30 Déc.	15 Oct.-11 Nov.	12 Nov.-2 Déc.	Sept.-14 Oct.	31 Déc.-fin Jan.
Fatine Bentires Alj	Sept.-14 Oct.	31 Déc.-fin Jan.	15 Oct.-11 Nov.	12 Nov.-2 Déc.	3 Déc.-30 Déc.
Ren Geng	15 Oct.-11 Nov.	3 Déc.-30 Déc.	Sept.-14 Oct.	31 Déc.-fin Jan.	12 Nov.-2 Déc.
Alexia Zounias-Sirabella	12 Nov.-2 Déc.	Sept.-14 Oct.	31 Déc.-fin Jan.	3 Déc.-30 Déc.	15 Oct.-11 Nov.

Après cela nous avons décomposé le groupe en deux sous groupes : l'équipe `blockchain` et l'équipe `Unity`.

L'équipe `blockchain` comprenait les membres suivants : Yassin Abar, Fatine Bentires Alj et Geng Ren. Le but de cette équipe était d'implémenter en C# la Blockchain, la SmartGrid et les fonctions associées au comportement humain et météo. L'équipe `Unity` quant à elle était composée d'Aurélien Abel et Alexia Zounias-Sirabella. Cette équipe était en charge du transfert des informations codées par l'équipe `blockchain` sur Unity de manière à développer une simulation visible à l'aide du casque de réalité virtuelle Oculus Rift.

La séparation des tâches peut se voir globalement de la manière suivante :

- Yassin Abbar (équipe `blockchain`) : mise en place du framework .net. Implémentation des classes `house` et `transaction`.
- Aurélien Abel (équipe `Unity`) : test et paramétrage du casque de réalité virtuelle Oculus Rift. Mise en place d'un code C# compatible avec Unity et Oculus Rift. Création d'une représentation 3D d'une réalisation de tâche (lancement de la machine à laver).
- Fatine Bentires Alj (équipe `blockchain`) : création du dossier Github et Overleaf. Implémentation des classes `person` et `Block/Blockchain`. Participation à l'élaboration du logo.
- Geng Ren (ancien membre Tomaro, actuel membre de l'équipe `blockchain`) : Génération des sondages de comportements humains et météos. Implémentation des classes `SmartGrid` et `Task`.
- Alexia Zounias-Sirabella (équipe `Unity`) : test et paramétrage du casque de réalité virtuelle Oculus Rift. Mise en place d'un code C# compatible avec Unity et Oculus Rift. Création d'une représentation 3D d'une réalisation de tâche (lancement de la machine à laver).

La séparation des tâches faite précédemment est bien évidemment donnée à titre indicatif. La méthode de fonctionnement repose plutôt sur une implémentation commencée par un membre qui se voit continuée successivement par les autres.

La rédaction du rapport s'est faite de manière homogène. Chaque membre du groupe s'est vu attribuer une partie.

Bibliographie

[1] Anders Brownworth, “Site Blockchain.” (https://anders.com/blockchain/).

[2] F.Bentires Alj - M.Pêcheux, “Logo Reverb.”

[3] Polytech Sorbonne, “Logo Polytech Sorbonne.” (https://www.facebook.com/pg/PolytechParisUPMC/photos/?tab=album&album_id=156421134394814).

[4] Unity, “Site Unity.” (https://unity3d.com/fr).