# Un exemple de contrat Ethereum

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 13 septembre 2015. Dernière mise à jour le 9 mars 2016

https://www.bortzmeyer.org/contrat-ethereum.html

---

J'ai déjà parlé d'Ethereum dans un précédent article <https://www.bortzmeyer.org/ethereum.html>, avec une description de ses possibilités. Je suis passé plutôt vite sur la notion de **contrat** alors qu'elle est essentielle dans Ethereum, et qu'elle est sa principale innovation par rapport à Bitcoin et tous ses dérivés. Je comptais me mettre à l'écriture de contrats mais je manque de temps donc, en attendant que j'apprenne sérieusement, je vais décrire un exemple que je n'ai pas réalisé moi-même mais qui a l'avantage pédagogique d'être assez simple à comprendre et assez compliqué pour faire quelque chose de réel : la Pyramide <https://www.ethereumpyramid.com/>.

Le nom fait référence à une accusation stupide, mais récurrente, faite contre Bitcoin et ses dérivés, qui seraient soi-disant une pyramide de Ponzi. Mais avant de décrire ce que fait la Pyramide Ethereum, revenons aux contrats.

Un contrat est un programme informatique, stocké sous forme de code pour une machine virtuelle, l'EVM ("*Ethereum Virtual Machine*"). Il va s'exécuter sur les machines du réseau qui minent, c'est-à-dire qui vérifient les transactions. Le langage de l'EVM (qu'on appelle parfois, lui aussi, EVM) est un langage de Turing, ce qui, en français, veut dire que tout ce qui peut être programmé dans un autre langage de Turing (C, PHP, Haskell, Go...) peut être programmé pour l'EVM. Un contrat est aussi une entité stockée dans le livre des opérations (la "*blockchain*") et a donc une adresse : on peut lui écrire, plus précisement lui envoyer des ethers (l'unité de compte Ethereum) pour lui faire exécuter son code. (Ethereum n'a pas de code tournant en permanence et n'a pas l'équivalent de cron : il faut explicitement appeler le contrat depuis l'extérieur.)

Les contrats sont stockés sous forme de code de bas niveau mais on les écrit en général dans un langage de plus haut niveau, le plus répandu étant Solidity.

Ces préliminaires étant posés, que fait le contrat « Pyramide » ? Eh bien, il mérite son nom, c'est une vraie pyramide de Ponzi. On envoie un ether (environ un euro au cours actuel) au contrat, et on est ajouté à la pyramide. Quand le niveau où on a été ajouté est plein, un niveau inférieur est ajouté (la pyramide grandit) et on reçoit des ethers. Comme tout bon schéma de Ponzi, cela ne marche que lorsque de nouveaux entrants arrivent en permanence. Quand ils cessent d'affluer, le schéma ne marche plus et les derniers arrivés n'ont plus qu'à pleurer.

Le code écrit en Solidity est disponible en ligne <https://www.ethereumpyramid.com/contract.html>. Les programmeurs familiers avec des langages comme C ou comme JavaScript ne devraient pas avoir trop de mal à le lire. Les points importants :

```
struct Participant {
        PayoutType payoutType;
        bytes desc;
        address etherAddress;
        bytes bitcoinAddress;
    }

Participant[] public participants;
```

Cette variable `participants` est l'**état** de la pyramide. Les transactions Ethereum peuvent en effet modifier l'état du système. (Ethereum, comme Bitcoin, est donc aussi un système de stockage.)

```
function() {
        enter(msg.data);
    }
```

Cette fonction anonyme est celle qui est appelée lorsque des ethers sont envoyés au contrat, le « réveillant ». La fonction `enter`, qu'elle appelle, fait l'essentiel du travail (je l'ai un peu simplifiée ici) :

```
function enter(bytes desc) {
        if (msg.value < 1 ether) {
            msg.sender.send(msg.value);
            return;
        }

        if (desc.length > 16) {
            msg.sender.send(msg.value);
            return;
        }

        if (msg.value > 1 ether) {
            msg.sender.send(msg.value - 1 ether);
         }
```

Ces trois tests vérifient certaines pré-conditions : que le paiement était bien de un ether pile, et que le message envoyé par l'utilisateur (`msg.data`) fait bien moins de seize octets. Si un des deux premiers tests échoue, on est remboursé (attention donc en testant : vous aurez l'impression que rien ne s'est passé, alors qu'en fait vous avez été silencieusement remboursé). Si vous envoyez plus d'un ether, vous serez remboursé du surplus mais le contrat continuera.

```
uint idx = participants.length;
participants.length += 1;
participants[idx].desc = desc;
// for every three new participants we can
// pay out to an earlier participant
if (idx != 0 && idx % 3 == 0) {
        // payout is triple, minus 10 % fee
        uint amount = 3 ether - 300 finney;
        participants[payoutIdx].etherAddress.send(amount);
        payoutIdx += 1;
}
```

Ce code paie (avec intérêts) un ancien participant lorsque de nouveaux participants sont arrivés.

J'ai moi-même envoyé mon ether (sous le nom "Stephane B.") ainsi, depuis la console de geth (cf. mon premier article <https://www.bortzmeyer.org/ethereum.html>) :

```
> eth.sendTransaction({from: eth.accounts[0], value: web3.toWei(1, 'ether'),
    to: '0x7011f3edc7fa43c81440f9f43a6458174113b162', gas: 500000,
    data: web3.fromAscii('Stephane B.')})
```

Ethereum étant transparent, on peut voir sur un explorateur public du livre des transactions tous les paiements reçus par la pyramide, en donnant son adresse (`0x7011f3edc7fa43c81440f9f43a6458174113b162`) : . L'argent envoyé est bien débité du compte utilisé (`eth.accounts[0]`) :

```
> eth.getBalance(eth.accounts[0])
2980411500000000000

[Minage de la transaction]

> eth.getBalance(eth.accounts[0])
1975400100000000000
```

Et hop, je suis ajouté à la pyramide, en attendant un profit (quand d'autres gogos viennent, suite à cet article). C'est ainsi que, deux mois après, j'ai bien touché mes ethers <https://etherchain.org/tx/0x509eef37cad2c1e95bf2a86811c785dbdb782e3977e75cba67f33eee7840b936>. De l'argent facilement gagné ! Le code JavaScript de la page Web du projet affiche alors la pyramide modifiée. (Le programmeur aurait pu interroger dynamiquement en JSON-RPC un nœud Ethereum pour récupérer les données mais, apparemment, cette récupération est faite entièrement côté serveur et ce code n'est pas publié. Mais vous pouvez faire pareil. Comment récupère-t-on des données d'un contrat ? On note (dans le source) que le contrat a des méthodes publiques comme `getNumberOfParticipants`. Il suffit de créer un objet JavaScript dans la console ayant la bonne ABI et l'adresse du contrat. On pourrait faire l'ABI à la main à partir du source mais c'est plus facile de demander au compilateur Solidity de le faire :

```
%  solc --json-abi file pyramid.sol
```

Et l'ABI en JSON est dans le fichier `Pyramid.abi`. Plus qu'à y ajouter un peu de code Javascript au début et à la fin (déclaration au début et adresse du contrat à la fin) et on a :

```
var pyramid = eth.contract(
[{"constant":false,"inputs":[{"name":"_owner","type":"address"}],"name":"setOwner","outputs":[],"type":"function
).at('0x7011f3edc7fa43c81440f9f43a6458174113b162');
```

On demande à geth de charger tout ça :

```
> loadScript("load-abi.js");
true
```

Et on a un objet JavaScript dont on peut appeler les méthodes, récupérant ainsi les données du contrat :

```
> pyramid
{
  address: "0x7011f3edc7fa43c81440f9f43a6458174113b162",
  ...
  getNumberOfParticipants: function(),
  ...
}

>  pyramid.getNumberOfParticipants()
65
```

On voit dans cet exemple qu'il est très simple d'écrire un contrat, et de faire donc calculer ses programmes par les machines d'Ethereum.

Le code machine du contrat est également public et visible, à l'adresse du contrat : .