

Blockchain

Démonstrateur

Manuel de l’utilisateur

|  |  |
| --- | --- |
| **Team n°** | 7 - K |
| **Rédacteurs** | Yoann BILIATO, Antonin BROQUET, Lucie MERCENNE, Michaël PERRIN |
| **Date de création** | 18 décembre 2017 |
| **Statut** | Beta, en attente de validation |
| **Version** | 1.0 |

# 

# SOMMAIRE

[**SOMMAIRE**](#_suhfxzcw2kgy) **1**

[**1 - INTRODUCTION**](#_jxp0jtaei7f4) **2**

[**2 - DESCRIPTION**](#_vtbzto7xenw8) **3**

[2.1 - Architecture](#_kzeeqgxymjy8) 3

[2.2 - Application](#_d7xjonijxr0r) 4

[**3 - INSTALLATION**](#_gtk5z2sllhw6) **5**

[3.1 - Pré-requis](#_44hpmujky1lt) 5

[3.2 - Matériel](#_axdzn1j2lmyy) 5

[3.3 - Configuration PC et Raspberry PI](#_qrzbpr31m8cu) 6

[**4 - LANCEMENT DU DÉMONSTRATEUR**](#_6ol6lpgofg8v) **9**

[**5 - DÉROULEMENT DU JEU**](#_21s7m8natikn) **11**

# 1 - INTRODUCTION

Le démonstrateur permet d’appuyer l’explication de la blockchain à un public averti mais non expert dans la thématique. L’objectif principal est d’expliquer, de manière didactique, le fonctionnement d’une blockchain.

Pour cela, nos choix se sont concentrés autour de la volonté de faire participer l’auditoire afin que les concepts soient mieux appréhendés mais surtout pour souligner la décentralisation. Nous avons donc décidé de déployer un réseau Ethereum “privé”, c’est à dire isolé d’Internet, plutôt que d’utiliser le réseau public qui requiert de posséder de la cryptomonnaie.

Concernant l’application, il s’agit d’un jeu de roulette. Les utilisateurs misent sur un nombre qui va sortir de manière aléatoire et gagnent ou perdent leur mise suivant le résultat. Les utilisateurs peuvent aussi s’échanger de la cryptomonnaie et ainsi réaliser des transactions.

Le démonstrateur est composé de 2 incréments:

* incrément 1 : mise en place d’un réseau Ethereum privé de plusieurs noeuds en local sur un PC. Le manuel de l’incrément 1 est écrit dans le README.md du dépot git.
* incrément 2 : mise en place d’un réseau Ethereum privé de plusieurs noeuds sur un réseau local et déploiement d’un *smart contract* proposant un jeu de la Roulette.

Ce document ne concerne que l'incrément 2 (dossier *demo2* du dépôt git).

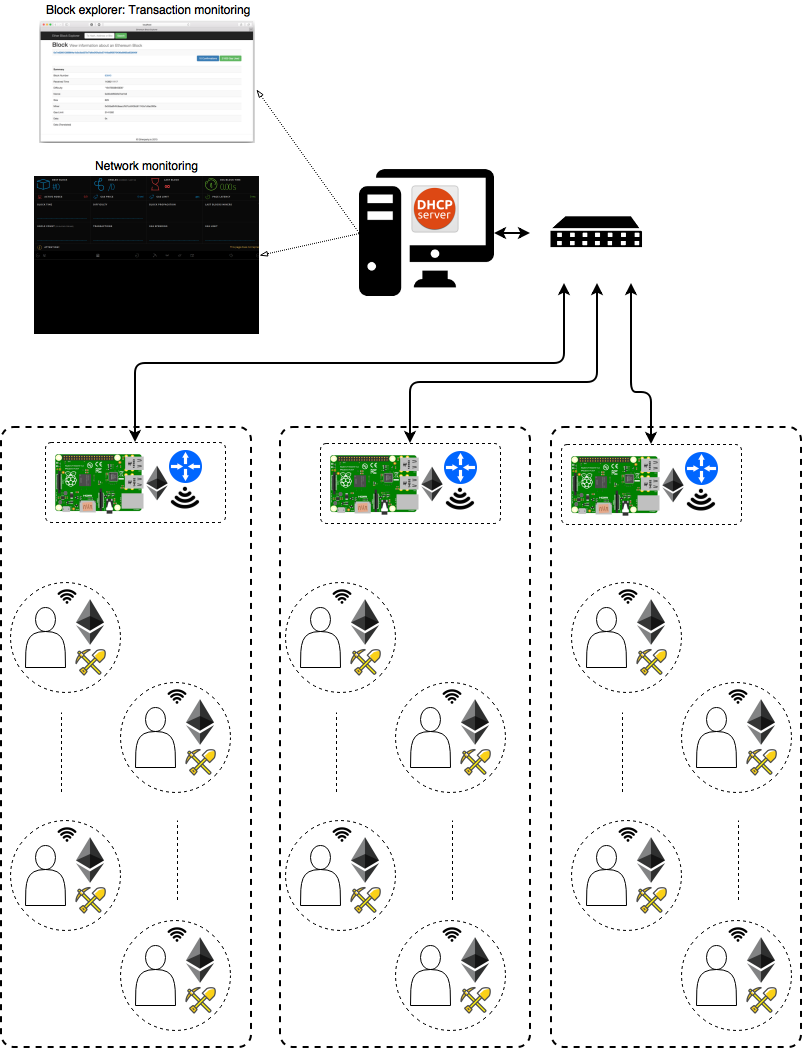
# 

# 

# 2 - DESCRIPTION

## 2.1 - Architecture

La figure 1 présente l’architecture du démonstrateur :

[](https://www.draw.io/#G1YYkEbXnYWREZo7rEnBLmEzGKvpL0UQqN)

*Figure 1 : Architecture globale du démonstrateur.*

Le réseau est constitué des éléments suivants :

* **Serveur DHCP :** permet d’assigner les adresses IP des noeuds du réseau local. Le serveur DHCP est installé sur une machine sur laquelle sont installés les outils de monitoring des statistiques du réseau et des blocs de la blockchain. Cette machine est aussi un noeud Ethereum du réseau.
* **Network switch :** permet de connecter les Raspberry Pi sur le réseau local.
* **Raspberry Pi :** elles sont utilisées en tant que Hotspot Wifi et routeur. Les 3 Raspberry Pi permettent de répartir les connections des utilisateurs sur le réseau local. L’utilisation des Raspberry Pi permet aussi d’apporter la dimension IoT au démonstrateur. Elles exécutent aussi un client Ethereum mais ne minent pas.
* **Utilisateurs :** ils se connectent au réseau local à travers une Raspberry Pi. Chaque utilisateur exécutent un client Ethereum et mine. Une interface web leur permet d’accéder à l’application et de réaliser des transactions de cryptomonnaie vers d’autres utilisateurs.

## 2.2 - Application

L’application est un jeu de roulette, où les joueurs (les utilisateurs) parient en Ether soit sur un numéro seul, soit sur la parité du nombre.

La banque (le contrat) récupère les mises et rembourse les paris gagnants.

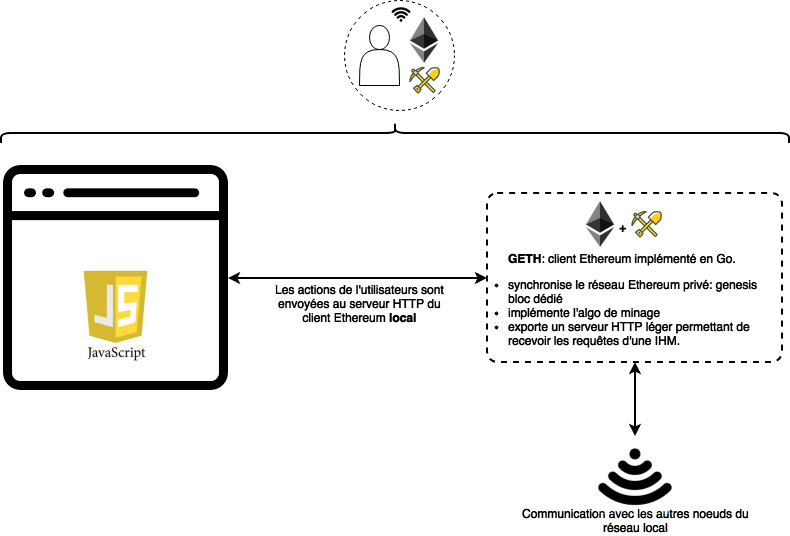
Un lancer de la roulette est effectué par la banque. Les joueurs gagnants reçoivent leur mise multipliée selon le pari utilisé (x10 nombre seul, x2 pair ou impair).

Toutes les actions de ce jeu (pari, mise, récupération des mises, remboursement des gains) sont des fonctions du contrat.

L’application a été packagée de manière à la déployer facilement pour la démonstration. Le package comprend :

* un script de lancement
* un client Ethereum (Geth)
* une IHM (page web)
* un front-end JavaScript faisant le lien entre l’IHM et le client Ethereum

La figure 2 détaille l’architecture logicielle de l’application :

[](https://www.draw.io/#G15Re7lzzTtBh6UVFlD7A-R5mW0i46cfx1)

*Figure 2 : Architecture logicielle de l’application*

# 3 - INSTALLATION

Le développement du démonstrateur a été réalisé sous Linux avec la distribution Ubuntu 17.04

Les instructions d’installations sont donc propres à cette distribution mais devraient fonctionner sous une autre distribution de la famille Ubuntu/Debian et peuvent être facilement adaptées à une autre distribution Linux.

## 3.1 - Pré-requis

Les logiciels suivant doivent être installés :

* **Geth & Tools** : le client Ethereum. Geth peut être installé en compilant les sources, depuis le gestionnaire de paquets ou en téléchargeant les binaires. Suivre les [instructions d’installations du wiki](https://github.com/ethereum/go-ethereum/wiki/Installing-Geth). Attention à bien choisir l’archive contenant les outils (bootnode, evm, …) dans le cas de la page de téléchargement.
* **Nodejs** : la version 6.x est nécessaire. Sur Ubuntu 17.04, exécuter les commandes suivantes :

|  |
| --- |
| curl -sL https:*//deb.nodesource.com/setup\_6.x | sudo bash -* sudo apt-get install nodejs |

* **npm**
* **git**

Le dépôt git peut être cloné ou téléchargé à partir de l’URL suivant :

<https://github.com/tonitchek/px504.git>

Dans ce manuel, les fichiers et les commandes sont relatifs à la racine du dépôt cloné ou téléchargé. Exemple :

Si le document fait référence au fichier :

|  |
| --- |
| <repo\_root>/demo2/deployment/init\_blockchain.sh |

c’est en réalité le fichier suivant :

/home/toto/px504/demo2/deployment/init\_blockchain.sh si l’utilisateur toto a cloné le dépôt dans son home.

## 3.2 - Matériel

Le matériel requis pour le démonstrateur est constitué des éléments suivants:

* **1x PC** avec une installation Ubuntu 17.04 (validée, mais peut fonctionner avec une autre distribution),
* **1x switch** **réseau**
* **3x Raspberry PI**

Les cartes SD des Raspberry PI sont prêtes à l’utilisation. Les Raspberry PI sont utilisés avec un Raspbian *lite* de base (officiel) avec des modifications liées au point d'accès WiFi et à l’installation du client Ethereum Geth.

**Raspberry PI connexion**

**login : pi**

**mdp : PX504bc**

## 3.3 - Configuration PC et Raspberry PI

Même si l’application est décentralisée, pour les besoins du démonstrateur, le PC est un point central pour le fonctionnement et la configuration de la démo. Ceci peut être amélioré par la suite.

La configuration du PC est donnée étape par étape.

**Etape 1 - Copie locale du dépôt git**

Le dépôt ne contient qu’une seule branche :

|  |
| --- |
| *git clone https://github.com/tonitchek/px504.git cd px504/demo2 git submodule update --init* |

Contenu :

* **app/packaging** : contient les sources nécessaires au déploiement de l’application sur les machines des utilisateurs (linux x64, windows x64 et macOS). Un script *bundle.sh* permet de générer une archive .zip pour chaque plateforme.
* **app/src** : sources de l’interface web et du back-end JavaScript / client Geth.
* **deployment** : scripts et sources nécessaires à l’initialisation de la blockchain, au déploiement du smart contract et à son interaction.
* **netgui** : scripts et sources permettant de démarrer le/les serveur(s) HTTP nécessaires aux utilisateurs (téléchargement archive application, monitoring reseau, blocs exploreur…).

**Etape 2 - Installation et configuration serveur DHCP**

Configurer l’adresse IP static de l’interface Ethernet. Supposons que l’interface Ethernet soit nommée *eth0*.

Ouvrir le fichier */etc/network/interfaces* et ajouter les lignes suivantes :

|  |
| --- |
| *auto eth0 iface eth0 inet static  address 192.168.1.1  netmask 255.255.255.0  network 192.168.1.0* Installer le serveur DHCP : *sudo apt-get install isc-dhcp-server sudo cp <repo\_root>/dhcpd.conf /etc/dhcp/* |

Ouvrir le fichier */etc/default/isc-dhcp-server* et chercher la ligne contenant INTERFACESv4= afin de la remplacer avec le nom de l’interface Ethernet utilisée :

|  |
| --- |
| INTERFACESv4="eth0" |

Redémarrer le serveur DHCP :

|  |
| --- |
| *sudo service isc-dhcp-serveur restart* |

**Etape 3 - Connexion du réseau local**

Il faut maintenant connecter les Raspberry PI au PC et vérifier que le PC leur assigne bien leur adresse IP.

* Réaliser la connection des câbles Ethernet du switch au PC et aux Raspberry PI (comme indiqué sur la figure 1 de l’architecture).
* Démarrer les Raspberry PI et attendre qu’elles bootent (minimum 30s…)
* Vérifier que la connexion (rappel, mdp : PX504bc) avec un ssh sur chaque adresse fonctionne :
  + ssh pi@192.168.1.11
  + ssh pi@192.168.1.12
  + ssh pi@192.168.1.13
* Vérifier l’apparition des 3 points d’acces WiFi :
  + PX504\_BLOCKCHAIN\_AP1
  + PX504\_BLOCKCHAIN\_AP2
  + PX504\_BLOCKCHAIN\_AP3

Miscellaneous

Problème connexion ssh :

* vérifier la configuration du serveur DHCP.
* se connecter sur la console série de la Raspberry PI et debugger localement. [Tutorial interface série](http://www.framboise314.fr/le-port-serie-du-raspberry-pi-3-pas-simple/).

Problème points d'accès WiFi :

* suivre le [tutorial](https://www.raspberrypi.org/documentation/configuration/wireless/access-point.md#internet-sharing) sur la configuration d’un point d'accès WiFi sur un réseau local existant (bridge).

**Etape 4 - Initialisation de la blockchain**

|  |
| --- |
| *cd <repo\_root>/demo2/deployment ./init\_blockchain.sh ./startnode.sh* |

Le client Geth doit démarrer, le DAG doit ensuite commencer (cela peut prendre quelques minutes) et enfin le minage démarre. Laisser le noeud miner un moment, ca doit aller une fois le DAG terminé.

Le minage peut être arrêté en tapant la commande *miner.stop()* dans la console JavaScript du client Geth pendant l’affichage des opérations de minage.

Redémarrer le minage avec *miner.start()*.

Arrêter le client avec la commande exit.

Récupérer l’URL du noeud avec la commande suivante :

|  |
| --- |
| *bootnode --nodekey data/bootkey\_$HOSTNAME.key -writeaddress* |

Copier le nombre hexadécimal affiché sur la console. Il s’agit de l’*enode* du PC.

**Etape 5 - Initialisation et synchronisation de la blockchain sur les Raspberry PI**

Se connecter sur chaque Raspberry PI :

|  |
| --- |
| ssh pi@<adresse ip> |

Puis taper les commandes suivantes :

|  |
| --- |
| *rm -rf data geth --datadir /home/pi/data/ init genesis.json geth --datadir /home/pi/data/ account new* |

Choisir un mot de passe.

|  |
| --- |
| *bootnode --genkey /home/pi/data/bootkey\_$HOSTNAME.key* |

Ouvrir le script *startnode.sh* et modifier la commande geth :

* remplacer le champ “enode://b814….” par “enode://<enode PC>” avec <enode PC> correspondant au nombre hexadécimal copié dans l’étape précédente.

Démarrer le client et vérifier que la Raspberry PI synchronise la blockchain démarrée sur le PC. Un texte “import ….” devrait apparaître et le numéro de bloc doit correspondre au dernier bloc miné sur le PC.

Répéter l'opération sur chaque Raspberry PI.

**Etape 6 - Mise à jour du fichier static-nodes.json**

Ce fichier permet aux clients démarrés sur les machines des utilisateurs de se connecter à un noeud (le PC ou l’une des 3 Raspberry PI) afin de se connecter au réseau pair à pair.

Il faut donc ajouter l’*enode* du PC et des 3 Raspberry PI dans ce fichier. L’*enode* du PC a déjà été récupéré dans l'étape 4.

Pour récupérer l’enode d’une Raspberry PI :

Se connecter à l’aide d’un ssh et taper la commande :

|  |
| --- |
| *bootnode --nodekey data/bootkey\_$HOSTNAME.key -writeaddress* |

Copier le nombre hexadécimal affiché sur la console. Il s’agit de l’*enode* du PC.

Une fois que les 3 enode ont été copiés, ouvrir le fichier *<repo\_root>/demo2/app/packaging/static-nodes.json* et remplacer les *enode*. La syntaxe du fichier est la suivante :

|  |
| --- |
| [  "enode://<enode PC>@192.168.1.1:30303",  "enode://<enode PI1>@192.168.1.11:30303",  "enode://<enode PI2>@192.168.1.12:30303",  "enode://<enode PI3>@192.168.1.13:30303" ] |

# 4 - LANCEMENT DU DÉMONSTRATEUR

Le lancement du démonstrateur est donné étape par étape.

**Etape 1 - Compilation du smart contract**

|  |
| --- |
| *cd <repo\_root>/demo2/deployment ./contract\_compile.sh* |

**Etape 2 - Création des packages de l’application**

Créer les archives qui seront téléchargées par les utilisateurs :

|  |
| --- |
| *cd <repo\_root>/demo2/app/packaging ./bundle.sh lin64 ./bundle.sh win64 ./bundle.sh macos* |

**Etape 3 - Démarrer le client Ethereum du PC**

|  |
| --- |
| *cd <repo\_root>/demo2/deployment ./startnode.sh* |

**Etape 4 - Démarrer les clients Ethereum des Raspberry PI**

Le démarrage “automatique” du client Ethereum sur les Raspberry PI n’a pas encore été implémenté.

|  |
| --- |
| *cd <repo\_root>/demo2/ ./startup\_pi.sh* |

Ce script met d’abord à jour l’heure sur chaque Raspberry PI (**très important pour la synchronisation de la blockchain**), il faut taper le mot de passe pour chaque Raspberry PI. La synchronisation par NTP n’a pas encore été implémentée.

Ensuite, le script se connecte sur chaque Raspberry PI et ou le démarrage du client Ethereum se fait manuellement :

* attendre le prompt demandant le mot de passe
* entrer le mot de passe
* taper la commande *./startnode.sh*
* vérifier que le noeud démarre
* taper sur la touche *Entrée/Enter*
* taper la commande *exit*
* le script quitte la Raspberry PI et se connecte sur la suivante. Répéter l'opération.

**Etape 5 - Déploiement du smart contract**

|  |
| --- |
| *cd <repo\_root>/demo2/deployment ./contract\_gui\_launcher.sh* |

Cliquer sur le bouton *Deploy* et attendre que la création du smart contract soit minée par le PC.

Copier l’adresse du smart contract affichée dans le fichier suivant :

|  |
| --- |
| *<repo\_root>/demo2/netgui/www/address.json* |

**Etape 6 - Démarrer les services web**

Le démarrage du monitoring du réseau et de l’explorateur de blocs requiert des installations de sous modules expliquées dans l'incrément 1 du démonstrateur.

Suivre la procédure d’installation nommée “Further installation” dans le *README.md* du dépot (Attention : la commande git submodule update *--init* doit avoir été exécutée) :

|  |
| --- |
| *cd <repo\_root>/demo1/explorer npm install cd ../monitor/eth-netstats npm install sudo npm install -g grunt-cli grunt cd ../eth-net-intelligence-api sudo npm install sudo npm install -g pm2* |

Démarrer les serveurs HTTP :

|  |
| --- |
| cd <repo\_root>/demo2/netgui ./explorer.sh start ./monitor.sh start ./app\_server.sh start |

# 5 - DÉROULEMENT DU JEU

Le déroulement du jeu est donné étape par étape.

Dans cette partie, on distingue 2 types d’utilisateur :

* développeur/administrateur : il s’agit de l’utilisateur qui met en place le démonstrateur et qui a besoin d’un certain nombre d’outils et logiciels.
* joueurs : il s’agit des utilisateurs qui se connectent au réseau privé déployé par l’admin et qui participent au jeu de la Roulette.

Pour différencier ces 2 types d’utilisateurs dans la description des instructions/opérations à réaliser, les balises suivantes seront utilisées :

* [admin] : identifie une instruction dédiée à l’administrateur.
* [joueur] : identifie les utilisateurs qui se connectent au réseau privé pour jouer à la Roulette.

**Etape 1 - Connexion des utilisateurs**

[joueur] : connexion sur les points d'accès WiFi :

* PX504\_BLOCKCHAIN\_AP1
* PX504\_BLOCKCHAIN\_AP2
* PX504\_BLOCKCHAIN\_AP3

[joueur] : chargement de la page <http://192.168.1.1:8888>

**Etape 2 - Téléchargement et lancement de l’application**

[joueur] : télécharger l’application propre à la plateforme (linux, windows ou mac) et l’extraire.

[joueur] : aller dans le dossier extrait et cliquer/lancer le script de démarrage *setup.batch* ou *setup.sh*

Le client Ethereum se lance et le DAG commence. Ensuite le minage démarre. Attendre d’avoir miné suffisamment d’Ether pour commencer à jouer.

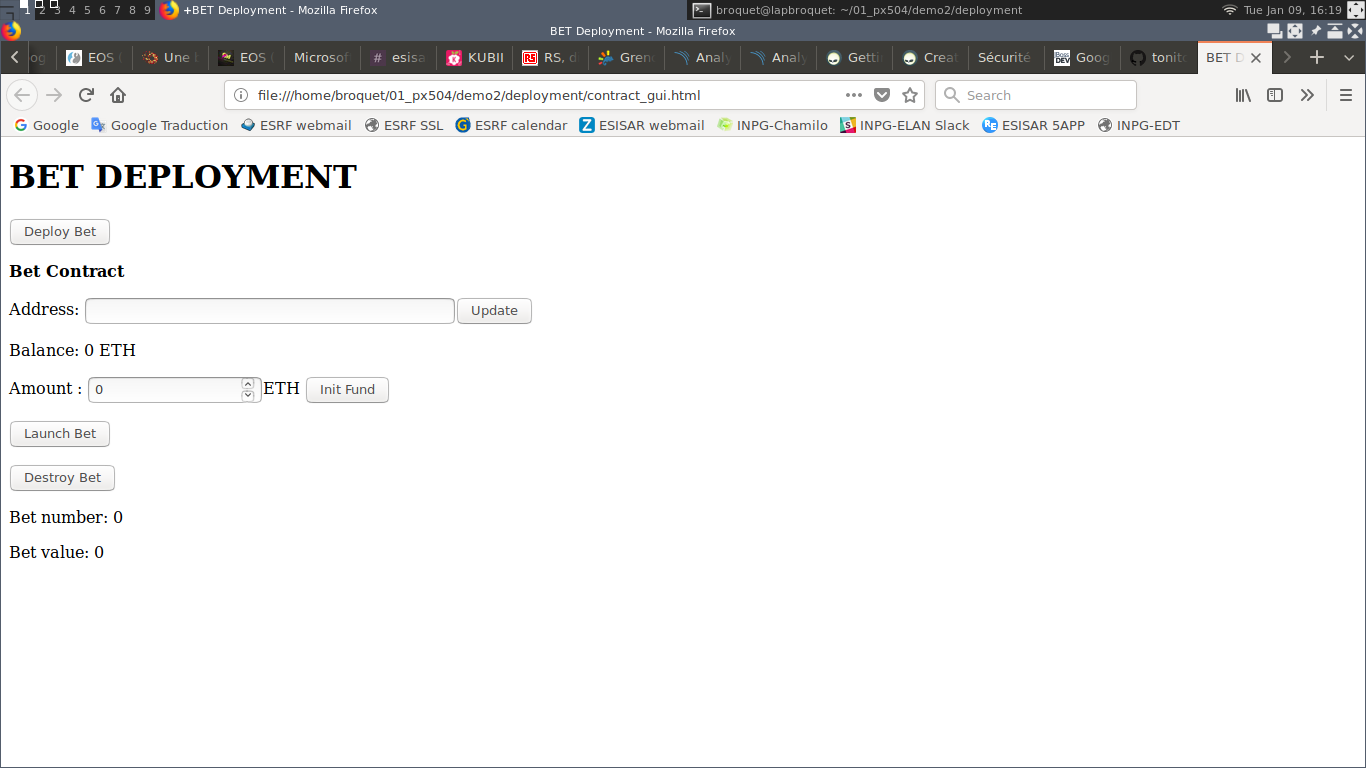
**Etape 3 - Apport d’argent sur le smart contract**

Les utilisateurs vont pouvoir commencer à parier. Pour cela, il faut apporter suffisamment d’argent au smart contract afin qu’il puisse payer les gagnants.

[admin] :

|  |
| --- |
| *cd <repo\_root>/demo2/deployment ./contract\_gui\_launcher.sh* |

La page html d’interface du smart contract (figure 3) s’affiche (comme à l'étape 5 de la partie 4-). Inscrire 1000 dans la case *Amount* et cliquer sur le bouton *Init Fund* (le PC doit avoir suffisamment miné…).

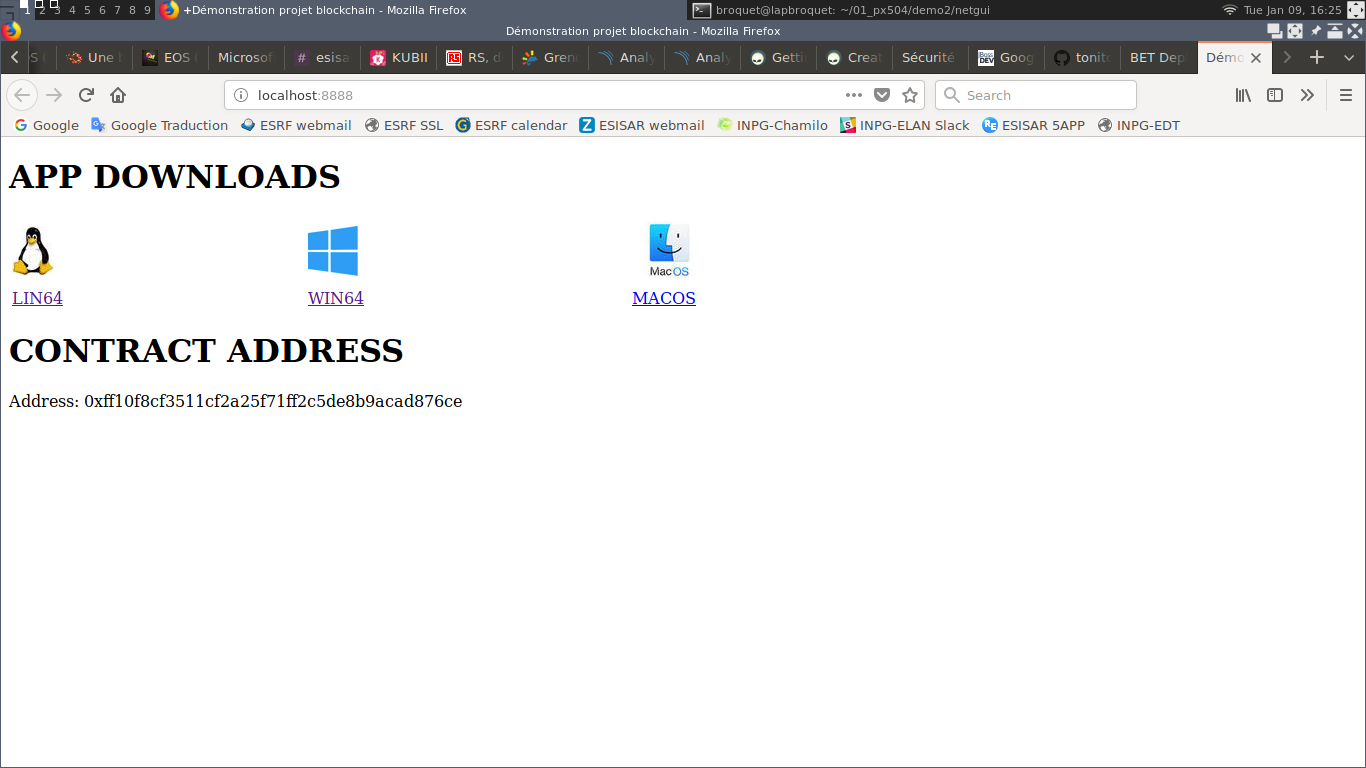


*Figure 3 : Interface graphique du smart contract.*

**Etape 4 - Lancer les paris !!**

Les utilisateurs peuvent commencer à jouer.

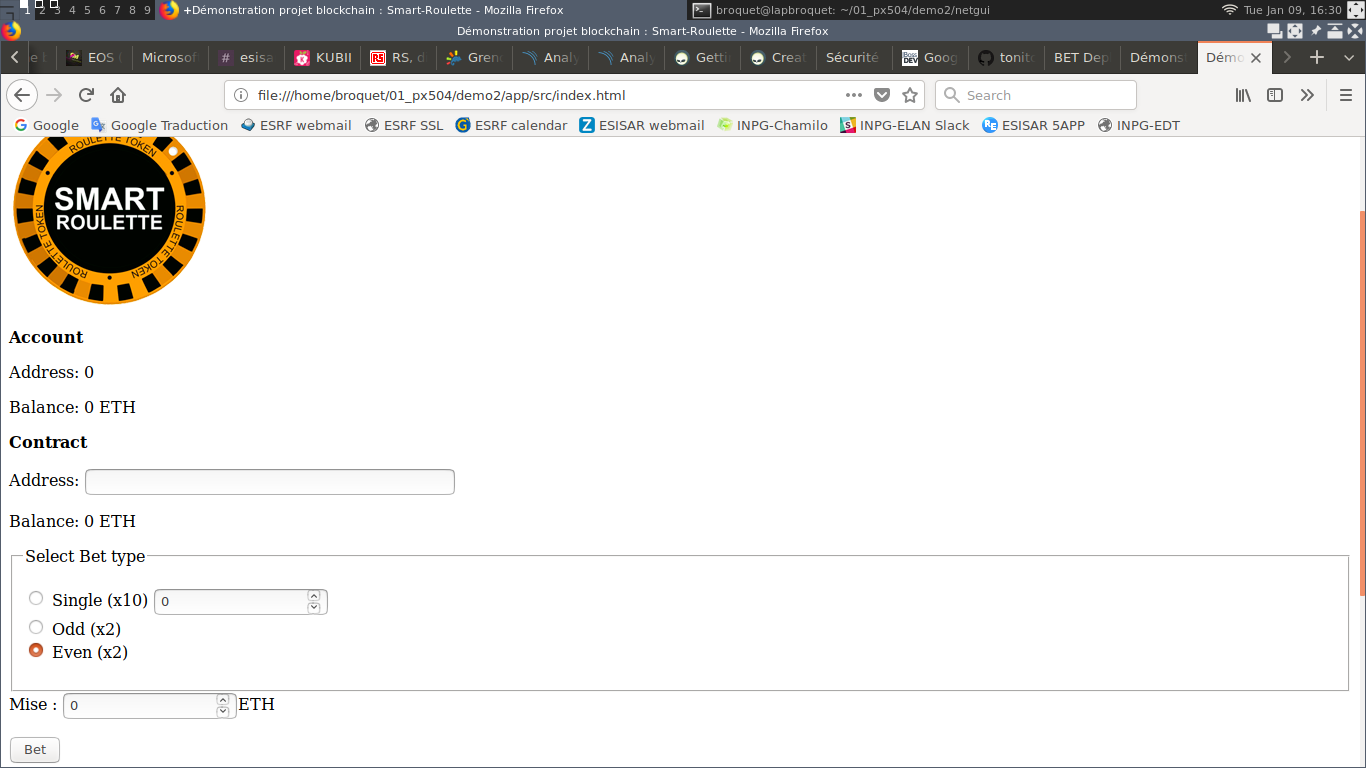
[joueur] : copier l’adresse du smart contract disponible sur la page web de téléchargement de l’application comme indiqué sur la figure 4 :



*Figure 4 : Copier l’adresse du smart contract disponible sur la page de téléchargement de l’application.*

[joueur] : insérer l’adresse du smart contract dans la case prévue à cet effet. Choisir son pari, ajouter une mise (limitée à 10ETH) et cliquer sur le bouton *Bet.*

La figure 5 montre l’interface graphique utilisateur.



*Figure 5 :Interface graphique utilisateur.*

[admin] : lancer la roulette. Retourner sur la page web d’interface du smart contract (figure 3) et appuyer sur le bouton *Launch Bet*.

**NOTE 1 :** Les mises étant faibles et les joueurs étant en train de miner, les gagnants ne verront pas leur gains incrémentés leur balance. Normalement, un message affiche qui a gagné un pari.

**NOTE 2** : Pour insister sur la sécurité et l’attention à porter sur le code d’un smart contract, il est possible de cliquer sur le bouton *Destroy Bet* au lieu de *Launch Bet* et ainsi récupérer toutes les mises des joueurs !!!