**Bot Contrat**

Table des matières

[1- Projets Utiles 1](#_Toc92963349)

[1- CreateWallet.sln 1](#_Toc92963350)

[2- BotContract.sln 2](#_Toc92963351)

[2- Préparation pour le bot, que faire ? 2](#_Toc92963352)

[1- Création et préparation de wallets et préparation des comptes 2](#_Toc92963353)

# Projets Utiles

Tout ce qui va vous servir à créer un bot est codé et utilisé dans les projets suivants :

CreateWallet.sln -> Création de portefeuille, gestion des transferts de monnaie de gouvernance et monnaie secondaire, gestion des balances

BotSoiMemeSunFlower.sln -> Exemple de bot utilisé sur le jeu SUNFLOWER (fonctionnel)

BotContract.sln -> Gestion de communication avec une des fonctions du contrat

## CreateWallet.sln

CreateWallet est un programme qui permet de préparer le nombre de Wallet que nous souhaitons utiliser avec le bot avec le nombre de token qu’il faut sur chacun des wallets.

Ce programme nous permet de faire plusieurs choses :

-Création de compte sur la blockchain grâce a la génération d’un private Key et d’une adresse, mémorise les comptes à chaque lancement dans le fichier config\Wallet.json

-Transfert de monnaie entre le Main Account et les autres comptes (pour envoyer du matic sur chacun de ses multicompte et inverse)

-Récupération des balances des comptes (monnaie gouvernance + jeton secondaire)

## BotContract.sln

BotContract est un programme qui va nous permettre d’interagir avec un contrat lié au Play to earn, c’est à dire que nous allons appeler les fonctions du contrat pour automatiser avec le bot sans passer par l’interface du jeu.

Ce programme nous permet de faire plusieurs choses :

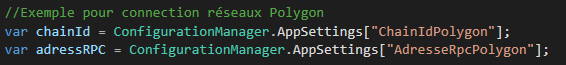
-Interagir avec un contrat provenant de la blockchain

# Préparation pour le bot, que faire ?

## Choisir son réseau et ses contrats

Cette étape consiste à savoir sur quelle blockchain nous souhaitons-nous installer pour ce bot, BSC, POLYGON, AVAX ?

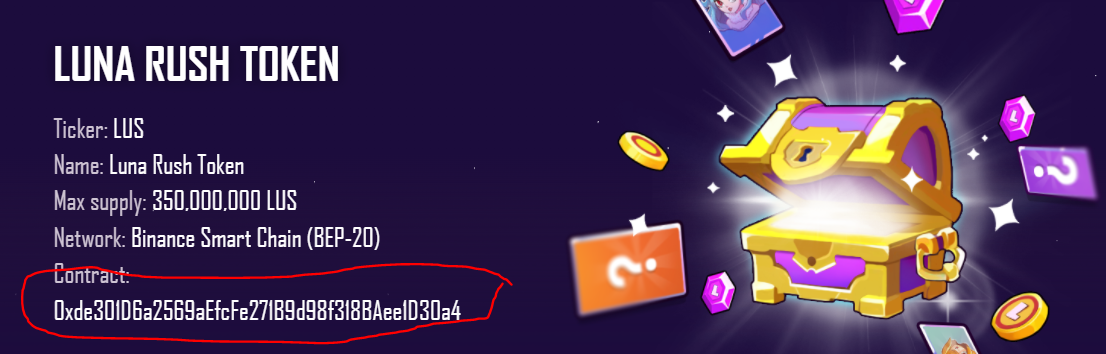
Pour cela il va falloir modifier les lignes suivante en indiquant le réseaux utilisé (voir fichier app.config)



Une fois les réseaux choisis, il nous faut récupérer le contrat du jeton secondaire que nous allons envoyer à nos différents comptes



Pour récupérer le contrat du jeton, il suffit de se rendre sur le site du Play to earn et de récupérer l’adresse du jeton, celle-ci devra être renseigner dans le fichier App.Config.



## Création de wallets

La première étape va consister à savoir combien de compte nous allons utiliser dans ce bot.

Pour cela il suffit de savoir le prix d’entrée du jeu pour 1 compte et son budget à investir, par exemple :

*20$ pour rentrer sur le jeu, j’ai 100$ de budget, je peux donc utiliser 5 comptes.*

Pour créer 5 comptes il suffit de modifier la variable



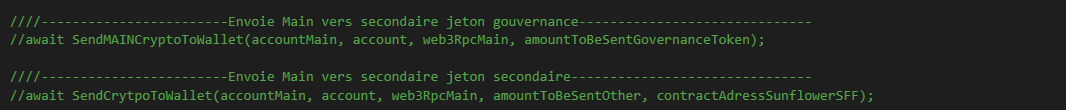
Si les comptes n’existent pas déjà alors le bot va les créer sinon il utilisera les wallets déjà générés.

## Préparation des wallets

Une fois les comptes terminés, il faut préparer les tokens à envoyer pour les frais des transactions ainsi que le cout dans le jeu.

*Dans le cas de sunflower, il fallait 1,9 SFF pour monter sa maison lvl 2 et planter ses choux et 1 MATIC pour gérer les frais des transactions par compte.*

Pour cela il va falloir utiliser ces lignes ci-dessous :



La première fonction envoie SEULEMENT le jeton main de la blockchain, c’est-à-dire MATIC pour réseaux POLYGON, BNB pour la BSC et AVAX pour le réseau avalanche.

La seconde fonction envoie le jeton secondaire tel que le SFF ect…

Nous allons devoir utiliser ses deux fonctions pour alimenter nos différents comptes.

Pour choisir les montants a envoyé il va falloir modifier ces deux variables :



## Préparation du bot contrat

Voila tous nos comptes sont créés et alimentés, il ne nous reste plus qu’a interagir correctement avec le contrat.

La première étape va être de prendre une copie de la solution BotContract.Sln.

Une fois la copie faite, nous allons voir comment récupérer les infos utiles aux contrats.

## Récupération des fonctions du contrat

La partie surement la plus difficile, ici nous allons utiliser la génération de l’utilitaire NETHERUM afin de créer notre classe et toutes les fonctions du contrat.

Deux cas possibles :

**1er cas : Le code source de l’application est donné par le développeur en open sources, et donc nous avons accès au ABI utilisé par les DEV.**

* C’est le cas le plus simple car nous avons juste à prendre le fichier JSON qui nous intéresse et le transformer avec l’outil NETHERUM en class C# que nous allons appeler dans notre programme par exemple :



Dans ce cas ci nous avons le code source du jeu SUNFLOWER (cool) et donc nous allons vouloir utiliser les fonctions de FARM, c’est-à-dire planter les choux, les récolter, level up la maison, donner aux charity (condition pour jouer).

Tout d’abord aller voir le dossier src/abis/.

Pour trouver les fonctions il va d’abord falloir fouiller dans les JSON afin de trouver le bon JSON parmi tous, c’est-à-dire celui qui nous intéresse pour notre bot, les autres JSON on s’en fou pour le moment.

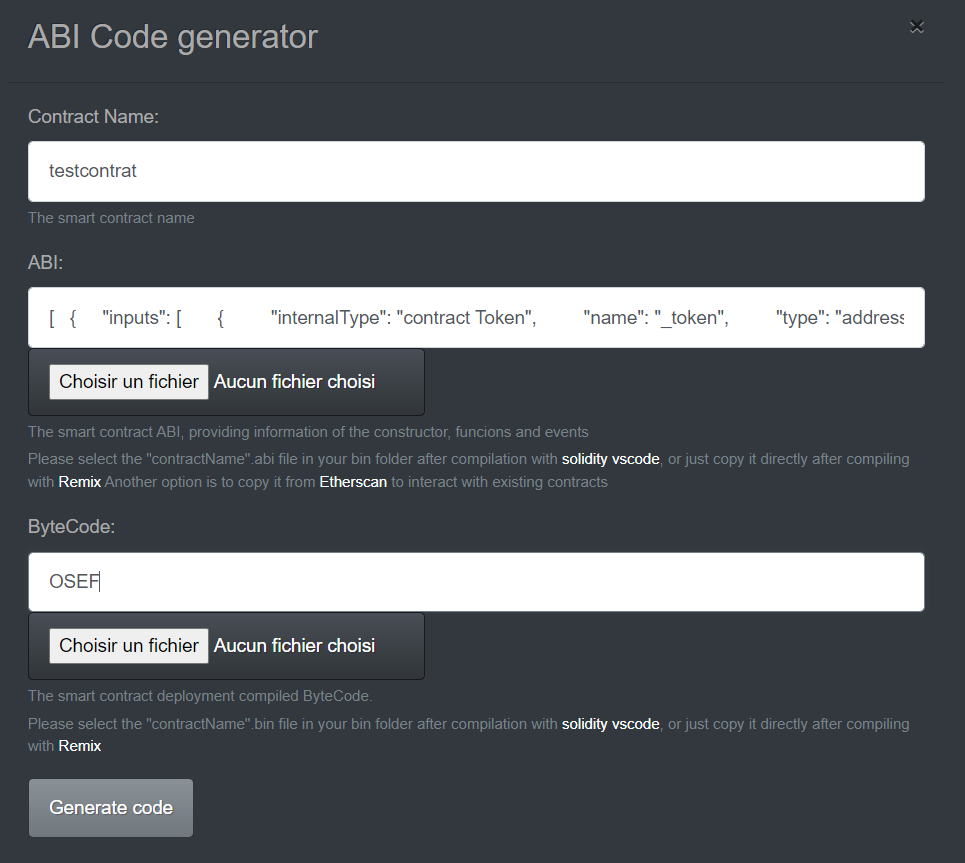
Celui qui nous intéresse est le fichier Farm.JSON car il contient tout ce que nous voulons.

## Génération du code Source C# avec NETHERUM

Une fois le fichier JSON identifié, rdv sur le site suivant : <http://codegen.nethereum.com/>

Nous allons utiliser comme le dis le site, le générateur de code à partir de l’ABI.

Pour cela, renseigner les champs suivants :



Et cliquer sur GENERATE CODE.

Voilà votre classe est généré, maintenant vous pouvez dégager la main et ne laisser que les fonctions, vous pouvez la rajouter à votre programme.

## Utilisation des fonctions du contrats dans notre code.

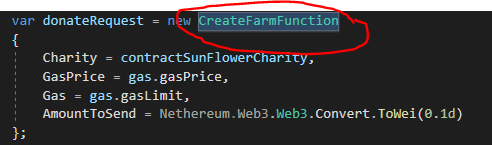
Maintenant que notre classe avec les différentes fonctions est prête, nous allons pouvoir les appeler directement, maintenant il va falloir deviner ce qu’attend la fonction que nous appelons comme paramètre.

Ici dans cet exemple nous allons essayer de donner 0.1 matic au charity (condition obligatoire pour jouer a sunflower)

La première étape est déjà de déterminer les frais de gaz que nous allons devoir payer :



Ensuite nous allons appeler la fonction du contrat CreateFarmFunction, qui va nous permettre de faire le don CHARITY :



Comme on peut le voir ici, la fonction utilise 4 paramètres, le contrat de charity, le prix du gaz, le gaz que nous devons utiliser et le nombre de token que nous envoyé au contrat de charity.

Comment trouver quels paramètres la fonction a besoin ? à part voir dans le code sources actuellement JE NE SAIS PAS

Une fois la request alimenté, nous allons l’exécuter, pour cela :



Nous allons définir une variable utilisant le WEB3 et le contrat et nous allons envoyer une requête vers ce contrat avec notre requête créer précédemment.

Et voila vous avez réussi à interagir avec une fonction du contrat, maintenant vous pouvez faire de même avec les autres fonctions.

## Pas d’open sources, que faire

**2eme cas : Le code source de l’application n’est pas donné par les développeurs en open sources.**

Dans ce cas-ci, j’ai quelques idées que je n’ai pas encore testés

1-il falloir surveiller les requêtes réseaux voir ce que l’on envoie et ce que l’on reçoit, voir s’il y a des logs oubliés qui nous amènerais à la page de code ou toutes les fonctions sont présentes.

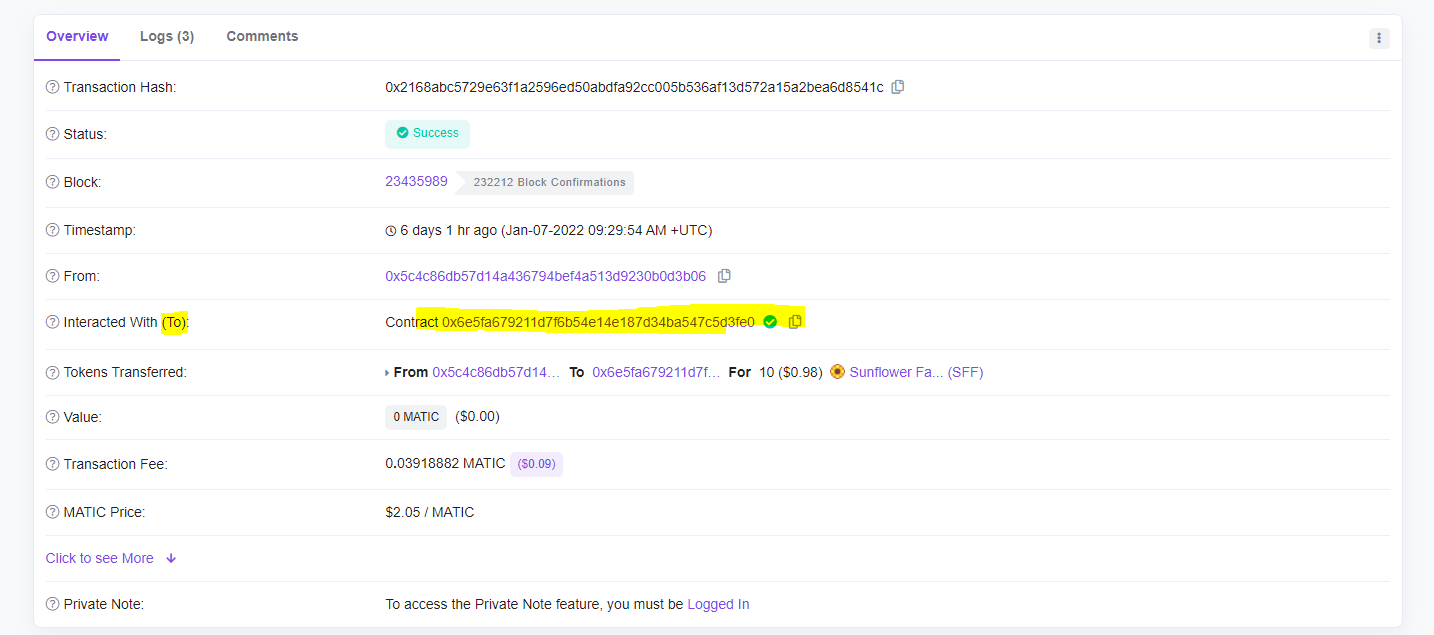
2-grace aux transactions faites, on peut voir quel contrat est utilisé pour les fonctions que nous recherchons, par exemple

Sur le réseau polygon, pour sunflower, le jeu utilise actuellement 3 contrats, il y ‘a un contrat visible de tous pour des fonctions basiques qui ne nous intéresse pas, et les 2 autres contrats sont plus techniques et ont les fonctions qui nous intéresse MAIS les DEV ont caché le contrat.

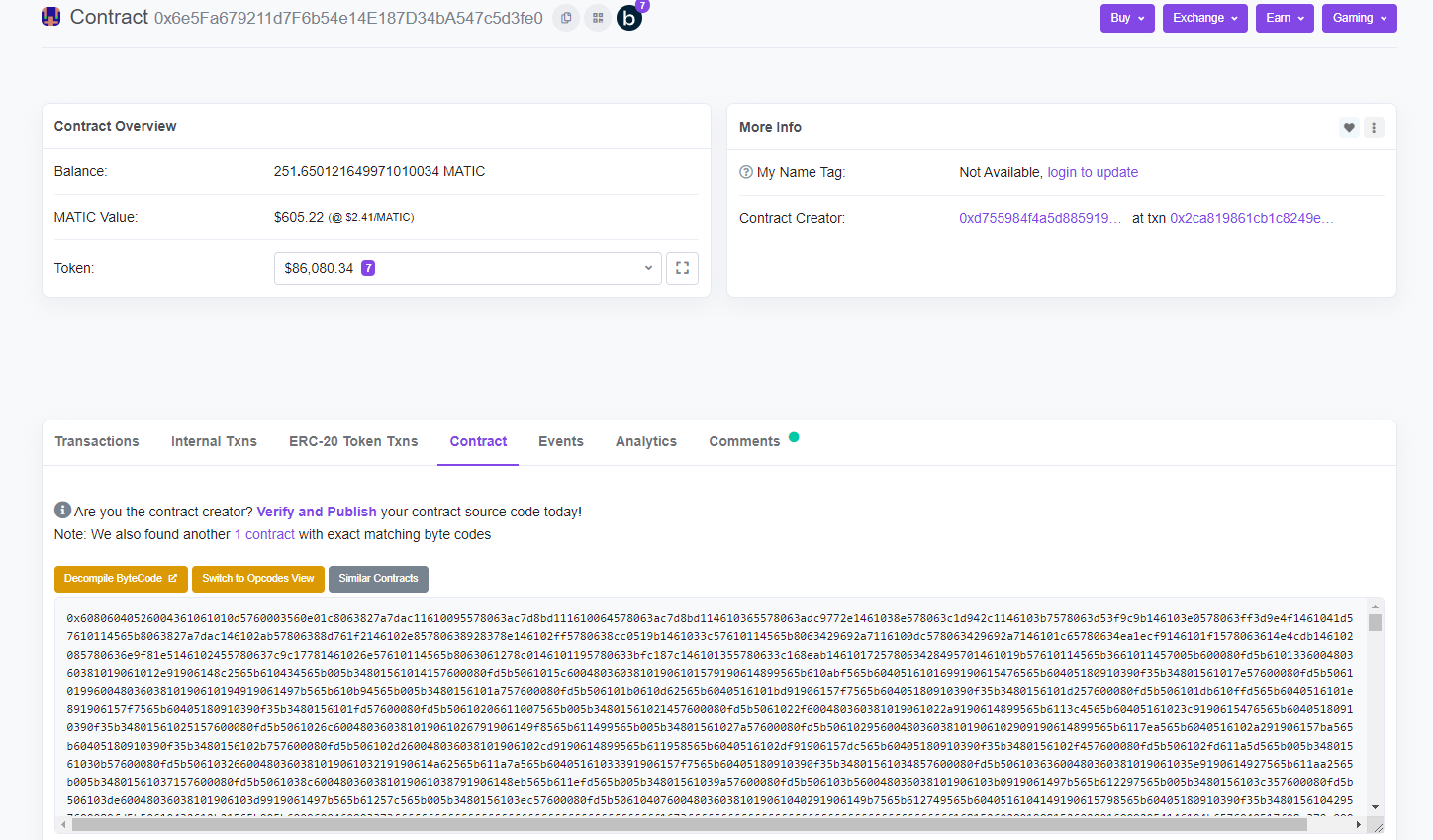
Pour cela, aller sur <https://polygonscan.com/>

Trouver une transaction de level up ou autre (une fonction qui nous intéresse)

Ici j’ai trouvé une fonction de level up :

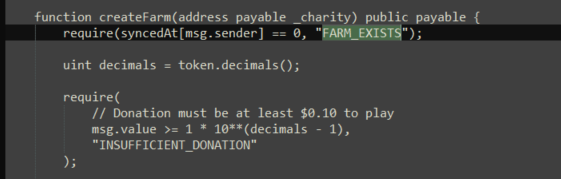


On peut voir dans le Interacted With (to) le contrat utilisé, on clique sur le contrat

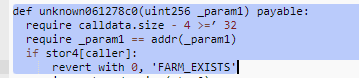


On arrive sur la page du contrat, on va dans l’onglet Contract et on peut voir que le code est compilé (caché), on clique sur le Décompile Bytecode afin de voir le code, cela nous génère un bout de code avec des fonction undefined, il va falloir fouiller dans ses fonctions pour trouver quelque chose d’intéressant.

Par exemple ce code ci dans le programme sources :



Représente ça dans le décompile :



Donc pas simple mais faisable !

A Compléter avec + d’expérience