

Levinswap, comment ça marche ..

Le document détaille le mode de fonctionnement des pools Levinswap, utilisées par RealT pour offrir un premier niveau de liquidité aux Realtokens.

Levinswap est un fork d'Uniswap version 2. Cette version, qui date de 2022, a été largement utilisée et fait l'objet de nombreux documents détaillant son algorithme. C'est souvent avec des formules mathématiques, qui peuvent en rebuter certains ;-). Pour être plus didactique, nous allons utiliser un simple tableur et analyser étape par étape, comment l'application fonctionne.

1 - Analyse à partir d'un tableur

Le tableur utilisé, se trouve à l'adresse suivante :

<https://docs.google.com/spreadsheets/d/1WnXiLUNrTD6M0bMDg0tvUcxLoM5-uUPN-lG1NG3bFU/edit?usp=sharing>

(copiez le sur votre compte, ainsi vous pourrez modifier les valeurs et voir ce qui se produit..)

Etape 1 : Création de la paire.

Suite à la création du Realtoken, RealT crée la paire correspondante.

Prenons l'hypothèse d'un Realtoken à 50 \$ et d'un USDC à 1\$.

Règle 1 de fonctionnement (pour la version 2 d'Uniswap étudié) : chacun des deux tokens de la paire doit être apporté en quantité identique. Une répartition 50 % / 50 % de la liquidité.

Ligne 6, du tableur : RealT va donc créer une paire avec : 20 Realtokens et $20 * 50 = 1000$ USDC.

La pool affichera une parité de 50 USDC / RealToken (50 \$ / token, pour simplifier l'écriture)

(Oublions les LP token à ce stade, ils seront détaillés par la suite)

Etape 2 : Exécution de swap (sans frais, pour commencer)

Ligne 8 : un « holder » (et pas un simple utilisateur, car il faut être whitelisted!) va acheter 1 Realtoken contre des USDC.

En prenant le point de vue de la pool : celle-ci va donc perdre 1 Realtoken (d'où le signe négatif) et gagner des USDC (d'où le montant positif).

Pour déterminer la quantité d'USDC, qu'il va falloir payer, il faut appliquer un algorithme.

Règle 2 : Pour stabiliser le fonctionnement des pools de liquidité, différents algorithmes sont utilisés. La version 2 d'Uniswap utilise la méthode « à produit constant » : c'est à dire que lors d'un swap, le produit des quantités des deux tokens doit rester constant.

La nouvelle d'USDC (après le swap) est donc déterminée par une simple règle de 3 : produit des tokens (avant le swap) = $20 * 1000$ divisé par la nouvelle quantité de token = 20 -1, ce qui donne 1052,63. En déduisant la quantité d'USDC avant swap on obtient le nombre d'USDC du swap. Ce mode opératoire est celui que vous retrouvez dans le tableur (cellule E8).

L'algorithme « à produit constant » sert à « décourager » l'utilisateur de vider significativement la pool. Ainsi l'acheteur du Realtoken a payé 52,63 \$ un token qui valait sur le marché 50 \$!

Le *Price Impact* (la surcote) que ça lui aura coûté est de 5,26 % ($52,63 / 50 - 1$)

Plus la quantité de token acheté augmente, plus le prix unitaire (et le *Price Impact*) augmente.

Inversement, si vous remplacez dans le tableur le -1 par -0,1 vous aurez payé le Realtoken 50,25 \$ et le *Price Impact* sera réduit à 0,5 %.

D'où la recommandation : de limiter vos swap à des fractions de Realtoken, compte tenu du peu de liquidité.

Ligne 9 : l'état de la pool après la transaction est la simple mise à jour des quantités des deux tokens et le calcul de la nouvelle parité.

Ce swap a augmenté la parité de la pool qui est passé de 50 à 55,40 \$ / token. Incitant par la même des holders à vendre ce Realtoken qui sera payé au dessus du prix de marché....

Ligne 10 : nous faisons le swap inverse où l'on vend 2 Realtokens (cellule D10 alors positive) contre 100,25 USDC (le calcul E10, étant identique à celui décrit précédemment pour E8). La parité de la pool redescend à 45,35 \$ / token, incitant des holders à acheter ce Realtoken qui est à un prix moindre que le marché.

Etape 3 : Swap avec frais

Progressons dans la « complexité » du protocol, en ajoutant les frais.

Règle 3 : Pour récompenser les fournisseurs de liquidité, le protocole prélève des frais (0,3%) sur la somme ajoutée lors d'un swap.

Ligne 13 : le swap consiste cette fois à vendre 1 Realtoken et en tenant compte des frais.

L 14 : 0,3 % de frais sur le Realtoken ajouté sont isolé.

L15 : La quantité de token objet de la règle 2 est réduite des frais et donne le montant d'USDC.

L16 : Mise à jour des quantité des jetons de la pool et actualisation de la parité.

L 17 : Effectuons le swap inverse (achat de token) pour comprendre une petite subtilité de la règle 3. Les frais sont perçu sur les montants ajoutés à la pool lors d'un swap. Dans ce cas de swap, les montants ajoutés sont des USDC. La mode de calcul est donc un peu différent.

L18 : Par application de la règle 2, le nombre de token donne le nombre d'USDC mais sans frais.

L19 : Les frais sont alors ajoutés aux USDC de la ligne précédente.

L 20 : La quantité des tokens est augmentée, frais compris coté USDC. On voit ainsi que les frais de swap, sont accumulé directement dans les montants des tokens de la pool. Et ne seront redistribué aux holder, que lors d'un retrait de liquidité.

Etape 4, Apport de liquidité :

L 22 : Nous allons apporter 2 Realtoken et les USDC correspondants.

En application de la règle 1, les deux tokens de la paire doivent être ajoutés dans des quantités identiques. La valorisation des Realtokens est faite avec la parité de la pool (lors de la création de la pool, il n'existe pas de parité, on utilise alors le prix de marché).

La parité de la paire étant 50,38 pour 2 RealTokens il faut apporter 100,76 USDC.

En contre partie des deux tokens qui ont été transféré vers la pool, l'application nous renvoie des Liquidity Providing token (LP token), qui sont des preuves de dépôt.

Règle 4 : En contre partie d'un apport de liquidité, des LP tokens sont créés et adressés. A l'inverse, en cas de retrait de liquidité les LP token correspondants doivent être transmis et sont détruit par l'application.

Le LP token est propre à chacune des paires (c'est le token du smart contrat de la paire).

La façon de calculer le nombre de LP token dépend si il en existe ou pas. A la création de la pool (ligne 6), aucun LP n'existe : le nombre de LP créé se calcul comme la racine carré du produit du nombre de token (cf cellule G6). Lorsque des LP existe déjà (ligne 22), le nombre de LP est égale au nombre existant augmenté de la proportion des jetons ajoutés : soit $141,42 * 2 / 19 = 14,886$

Le nombre total de LP a augmenté (à 156,3) et la répartition des LP entre holder est modifiée (cellules K et L23).

Vous aurez noté que l'apport (comme le retrait) de liquidité ne modifie pas la parité de la paire.

Seuls les swap modifient la parité et font l'objet de frais.

L 25 : opération inverse, retrait de la totalité de la liquidité apporté par le holder.

L26 : la quantité de token restitué au holder est le produit du nombre disponible par la proportion de LP token du holder transmis au contrat par rapport au nombre total de LP token de la pool.

Par exemple pour les USDC : $1057,97 * 14,886 / 156,308 = 100,76$.

L24 : Une transaction de swap a été interposé entre l'apport (L22) et le retrait (L26) de liquidité, pour vous faire comprendre comment vous récupérer les frais de swap.

Lorsque le nombre de token de ce swap (D24) est à zéro, il n'y a pas de transaction entre l'apport et le retrait. L'application vous restitue exactement le même montant que vous avez apporté (cellules N 22 et 26).

Modifions le montant des token du swap (D24), en le passant par ex à 5. La somme restitué (N26) est maintenant supérieur à celle déposé (N22). C'est votre quote part des frais du swap depuis votre dépôt.

Nota : Dans le tableur, la formule de calcul des USDC ligne 24 est la concaténation des cas traités à l'étape 3 (suivant que le nombre de token est positif, nul ou négatif). Les formules ont une présentation différente, mais sont identiques. Cette présentation est celle utilisée dans les smart contrat, que nous verrons dans un prochain chapitre.

2 - Analyse à partir d'un exemple

Continuons au travers d'un exemple, pour comprendre quelques points complémentaires.

Allons sur la paire 1-Holdings-Tappan :

<https://info.levinswap.realt.community/pair/0xd2f0e23155cb582ea9f59942335eda8ca7a5a41e>

et analysons les premières transactions :

Transactions			
All	Swaps	Adds	Removes
Total Value	Token Amount	Token Amount	
Swap USDC for REALTOK...	\$10.45	10.4268 USDC	0.2 REAL...
Swap USDC for REALTOK...	\$10.24	10.23642 USDC	0.2 REAL...
Add USDC and REALTOK...	\$202.90	101.12676 USDC	2 REAL...
Swap USDC for REALTOK...	\$0.51	0.5069 USDC	0.01 REAL...
Swap USDC for REALTOK...	\$2.53	2.52688 USDC	0.05 REAL...
Add USDC and REALTOK...	\$1002	1005 USDC	20 REAL...

en les reprenant avec un tableur (comme celui élaboré précédemment) :

https://docs.google.com/spreadsheets/d/1J00tjFfQ-USPK1UjU7Z6C54RrLzTr_bqDshSfwYevcs/edit?usp=sharing

Commençons par la **transaction de création de la paire** :

Dans la liste, en cliquant sur le lien bleu à gauche, l'explorateur s'ouvre sur la transaction.

② Tokens Transferred: 4	» From 0x5fc96c182bb7e... To 0xd2f0e23155cb5... For 1,005.2 (\$1,005.20) USD//C on xD... (USDC)
	» From 0x5fc96c182bb7e... To 0xd2f0e23155cb5... For 20 RealToken S ... (REALTOK...)
	» From 0x00000000000000... To 0x00000000000000... For 0.0000000000000001 Uniswap V2 (UNI-V2)
	» From 0x00000000000000... To 0x5fc96c182bb7e... For 0.000141788574997128 Uniswap V2 (UNI-V2)

Dans la partie *Tokens Transferred* :

- les deux premiers transferts correspondent aux calculs fait dans le tableur (ligne 7),
- le dernier transfert correspond aux Token LP qui ont été envoyé (à RealT) en échange de leur apport. On remarque que le chiffre correspond, mais pas les décimales...

Ce point vient du **codage des quantités de jeton**, dans les smart contrat. Le langage Solidity ne sait pas traiter les nombres décimaux (qui ont des chiffres derrière la virgule). Pour intégrer cette contrainte, on va « déplacer la virgule » vers la droite de x décimales.

Ainsi les calculs se font sur des nombres entier (sans virgule) en gardant toute leur précision. La virgule est « remise à sa place », grâce aux conversions qui sont faites dans les applications d’affichage (comme celle de *Tokens Transferred*).

Histoire d’ajouter un peu plus de complexité... tous les jetons n’ont pas le même décalage ! Dans le cas présent : nous avons trois jetons : le Realtoken, l’USDC et le LP token (nommé Uni-V2). Pour retrouver le codage des décimales de chacun, il suffit d’afficher l’adresse de leur contrat dans l’explorateur.

Pour ce faire, il vous suffit de cliquer sur le nom du token (par ex à droite de l’affichage dans *Tokens Transferred*) Vous devriez trouver les valeurs suivantes :

Profile Summary	
Contract:	0xe2fbde...
Decimals:	18

Profile Summary	
Contract:	0xddafbb505ac...
Decimals:	6

Profile Summary	
Contract:	0xd2f0e23155cb5...
Decimals:	18

Les Realtokens et LP token sont avec 18 décimales et l’USDC avec 6.

Le déplacement de virgule, évoqué précédemment consiste en fait à multiplier ou diviser le chiffre par 10 puissance le nombre de décimal. (on multiplie pour passer du codage utilisateur au codage smart contrat, et on divise pour la conversion inverse).

Dans le cas des LP token, il faut aussi intégrer, la façon dont ils sont calculés. A l’étape 4, nous avons vu que c’est la racine carré du produit des quantités de Realtoken par USDC. La quantité « brut » de LP, dans le smart contrat, sera avec 12 décimales (Racine de ($10^{18} * 10^6$)) et comme lors de l’affichage du token LP le montant brut est divisé par 10^{18} on se retrouve avec un décalage de 6. Ce que l’on observe.

Désolé pour ce parcours tortueux dans les codage décimaux, mais ainsi vous pouvez comprendre ce qui s’affiche (nous réutiliserons ce point par la suite).

- Le troisième transfert fait est très faible : c’est une quantité minimum de LP token qui est crée lors de la création de la paire (pour éviter que celle-ci puisse être à zéro). La quantité minimum est de 1000 (10^3) au format brut. Et maintenant que vous savez jongler avec les codages décimaux. 10^3 brut ça fait 10 puissance -15 en décimale (puisque les LP token on 18 décimales). Dans l’affichage, vous pouvez compter, le 1 se situe bien au 15ème rang derrière la virgule !

Pour les deux **transactions de swap**, qui suivent : si vous les afficher vous constaterez que vous avez exactement les même valeurs que celles calculées dans le tableur (lignes 8 et 10). Donc vous comprenez, comment elles sont constituées.

Passons à la **transaction d’apport de liquidité**, qui nous réserve une petite surprise ...

Dans l’explorateur de la transaction, regardons les transferts :

Tokens Transferred: 4			
From	0xd9df1d931cfab5...	To	0xd2f0e23155cb5... For 2 RealToken S ... (REALTO...)
From	0xd9df1d931cfab5...	To	0xd2f0e23155cb5... For 101.126758 (\$101.08) USD/C on xD... (USDC)
From	0x00000000000000...	To	0x6d81dda24b7ff5... For 0.00000000010671529 Uniswap V2 (UNI-V2)
From	0x00000000000000...	To	0xd9df1d931cfab5... For 0.000014221532731001 Uniswap V2 (UNI-V2)

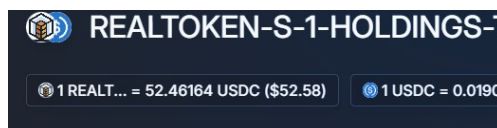
- Les deux premiers ainsi que le dernier, correspondent bien aux valeurs qu’on a dans le tableur (ligne 12),
- Le transfert de 10^3 LP token, évoqué lors de la création de la paire à bien disparue,
- Par contre, un troisième transfert apparaît. C’est une création de LP token (adresse de départ en 0x000) et envoyé sur un contrat particulier, qui sert à collecter les frais de protocole.

Règle 5 : Les 3 % de frais de swap se décomposent en 0,25 % qui sont distribués aux apporteurs de liquidité et 0,05 % qui financent le protocole.

Dans le tableur, en ligne 13 vous avez le montant des LP token correspondant ainsi que la formule de calcul. Pour en savoir plus, vous pouvez vous rendre en page 5 du whitepaper (<https://uniswap.org/whitepaper.pdf>)


Les deux transactions de swap qui suivent, correspondent aux lignes 15 et 17 du tableur.

Nous en avons terminé dans l'analyse des 5 transactions. L'état de la pool apparaît ligne 18 et les chiffres correspondent très exactement à ce qui s'affiche dans l'application. Notamment par ex la parité de la pool (identique à la cellule F18 du tableur) :



(vous verrez peut-être d'autres chiffres, si entre temps d'autres transactions se sont ajoutées).

Quand vous accédez à la répartition des holders de cette pool, (<https://youtu.be/QIK4zV7NLF4>) vous voyez ceci :

Rank	Address	Quantity	Percentage
1	0x5fc96c182bb7e0413c08e8e03e9d7efc6cf0b099	0.000141788574997128	90.8842%
2	0xd9df1d931cfab59965c1a87e1e55131632357f0d	0.000014221532731001	9.1158%
3	 0x6d81dda24b7f5b4a65039ff15d06a076e018e49	0.00000000010671529	0.0001%
4	0x00	0.0000000000000001	0.0000%

Avec les précédentes explications, vous comprenez maintenant :

- d'où proviennent les pourcentages à droite :
 - cf calculs détaillés dans le tableur,
- que sont les quantités affichées au milieu et pourquoi elles sont si petites :
 - nombre de LP token possédé par chacun des participants à la pool et affiché avec un facteur 10 puissance - 6 de décalage,
- qui est ce troisième holder, dont le montant change à chaque apport ou retrait de liquidité :
 - C'est fait un smart contrat (cf icône avant l'adresse, qui indique que c'est un smart contrat et pas un wallet). Il collecte les frais de protocole (0,05%) qui s'accumulent à chaque swap et qui sont mis à jour lors d'ajout ou retrait de liquidité (pour économiser des frais de gaz),
- qui est le dernier holder avec une adresse en 0x00..
 - Il s'agit de la quantité minimum de LP token (10 puissance - 15), créée lors de la création de la pool.

Je vous propose de voir maintenant, comment tout cela se traduit dans les smart contrats.

3 - Les smart contrats

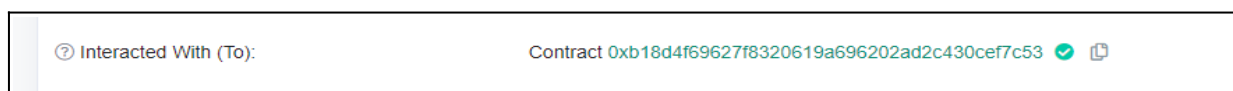
Nous n'allons pas détailler le code, juste survoler l'organisation des principaux smart contrats et identifier les principales fonctions.

Les principaux smart contrat de Levinswap sont :




- à l'adresse : <https://github.com/Levinswap/levinswap-contracts/tree/master/contracts/uniswapv2>
- *UniswapV2Factory* : Contrat « usine » qui crée les nouveaux contrats de paire et leur sert de registre,
- *UniswapV2Pair* : Contrat principal qui est responsable de la logique de base (échange, apport et retrait de liquidité). Chaque contrat ne traite qu'une seule paire.
- *UniswapV2ERC20* : Contrat pour la gestion des LP token, à partir d'une base ERC20.
- *UniswapV2Router* : Principale point d'entrée de l'interface utilisateur.
- *UniswapV2Library* : Implémente les principaux calculs.

Leurs adresses :

- Le contrat Routeur : 0xb18d4f69627f8320619a696202ad2c430cef7c53
C'est celui avec le quel votre wallet échange pour faire des transactions sur Levinswap, comme vous pouvez le voir dans l'explorateur :



- Les contrats de la paire et des jetons : ils sont accessible en bas de la page de la paire :

Pair Information			
Pair Name	Pair Address	REALTOK... Address	USDC Address
REALTOK...-USDC	0xd2f0...a41e 	0x7231...fafa 	0xddaf...7a83 

4 - Analyse de l'algorithme

Le mécanisme principale, qui équilibre la pool, est l'algorithme « à produit constant ». Analysons d'un peu plus près où et comment cela se passe.

Un peu de théorie (et de math désolé ;-), pour comprendre d'où viennent les formules codées dans les smart contrats.

Considérons deux jetons X et Y dont le produit doit être constant.

Commençons par un **swap de vente** par un holder, d'une quantité dx du jeton X (ajout positif vu du côté du pool), et donc d'une diminution dy du jeton Y.

La technique du « produit constant » conduit aux égalités suivantes :

$$(X + dx) * (Y - dy) = X * Y, \text{ constance du produit des jetons}$$

$$\begin{aligned}
(Y - dy) &= X * Y / (X + dx) \\
dy &= Y - X * Y / (X + dx) \\
dy &= (Y * (X + dx) - X * Y) / (X + dx) \\
dy &= (X * Y + Y * dx - X * Y) / (X + dx) \\
dy &= Y * dx / (X + dx)
\end{aligned}$$

Si on ajoute les frais de 0,3 % sur la quantité ajouté : donc dx

$$dy = Y * dx * 0,997 / (X + dx * 0,997)$$

Solidity ne sachant traiter les nombres à virgule, multiplions la fraction par 1000 en haut et en bas de la fraction (au numérateur et au dénominateur).

$$\begin{aligned}
dy &= 1000 * Y * dx * 0,997 / (X + dx * 0,997) * 1000 \\
dy &= Y * dx * 997 / X * 1000 + dx * 997
\end{aligned}$$

Voyons comment cela apparaît dans le smart contrat :

```

44 // given an input amount of an asset and pair reserves, returns the maximum output amount of the other asset
... 45 function getAmountOut(uint amountIn, uint reserveIn, uint reserveOut) internal pure returns (uint amountOut) {
46     require(amountIn > 0, 'UniswapV2Library: INSUFFICIENT_INPUT_AMOUNT');
47     require(reserveIn > 0 && reserveOut > 0, 'UniswapV2Library: INSUFFICIENT_LIQUIDITY');
48     uint amountInWithFee = amountIn.mul(997);
49     uint numerator = amountInWithFee.mul(reserveOut);
50     uint denominator = reserveIn.mul(1000).add(amountInWithFee);
51     amountOut = numerator / denominator;

```

<https://github.com/Levinswap/levinswap-contracts/blob/2a8223b7abc52d3679078a3429bd685226818f71/contracts/uniswapv2/libraries/UniswapV2Library.sol#L45>

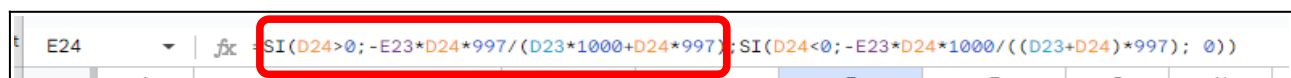
La correspondance entre le smart contrat et la formule est :

- Pour les quantités ajoutées (In) : X = reserveIn et dx = amountIn
- Pour les quantités retranchées (Out) : Y = reservOut et dy = amountOut
- amountInWithFee étant égale à $dx * 997$ ligne 48 du code ci-dessus

On retrouve bien dans le code :

- ligne 49 (pour le numérateur) : $dx * 997 * Y \Rightarrow$ amountInWithFee.mul(reserveOut)
- ligne 50 (pour le dénominateur : $X * 1000 + dx * 997 \Rightarrow$
reserveIn.mul(1000).add(amountInWithFee)
- ligne 51 : $dy =$ amountOut = numerator / denominator

Pour mémoire, nous retrouvons très exactement cette formule, dans les tableurs (par exemple dans le premier pour la cellule E24) , quand l'ajout du token x (D24) est positif



Reprenons, plus rapidement, le cas inverse d'un **swap d'achat** par un holder, d'une quantité dx du jeton X (retrait négatif vu du côté du pool), et donc d'une augmentation dy du jeton Y.

$$(X - dx) * (Y + dy) = X * Y$$

$$dy = (X * Y / (X - dx)) - Y$$

$$dy = dx * Y / (X - dx)$$

Les frais de 0,3 % étant à appliquer sur le jeton qui augmente, cette fois ce sera donc sur dy :

$$dy * 997 / 1000 = dx * Y / (X - dx)$$

$$dy = Y * dx * 1000 / ((X - dx) * 997)$$

Coté le smart contrat, on retrouve la formule, à un détail près ...

```

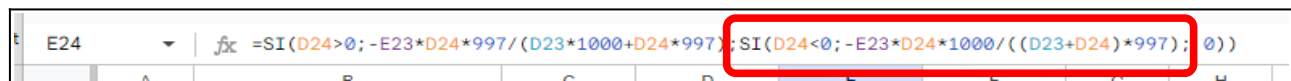
54 // given an output amount of an asset and pair reserves, returns a required input amount of the other asset
55 function getAmountIn(uint amountOut, uint reserveIn, uint reserveOut) internal pure returns (uint amountIn) {
56     require(amountOut > 0, 'UniswapV2Library: INSUFFICIENT_OUTPUT_AMOUNT');
57     require(reserveIn > 0 && reserveOut > 0, 'UniswapV2Library: INSUFFICIENT_LIQUIDITY');
58     uint numerator = reserveIn.mul(amountOut).mul(1000);
59     uint denominator = reserveOut.sub(amountOut).mul(997);
60     amountIn = (numerator / denominator).add(1);

```

<https://github.com/Levinswap/levinswap-contracts/blob/2a8223b7abc52d3679078a3429bd685226818f71/contracts/uniswapv2/libraries/UniswapV2Library.sol#L55>

Il s'agit du .add (1) à la fin de la ligne 60. La raison de cet ajout est le traitement des arrondis. Imaginons que le résultat de la division soit 7,2, l'arrondi va faire que amountIn sera 7. Hors avec cette valeur, des blocage pourrait se produire dans les calculs qui suivent. Ils ont donc préféré ajouter 1 (en fait, c'est 1 au format « brut », et 10 puissance – 18 en format réel) pour éviter tout problème. (nota : le Pb ne se produit pas dans la formule précédente d'un swap de vente, car l'arrondi joue alors en faveur des calculs qui suivent)

Dans les tableurs, quand l'ajout du token x (D24) est négatif, nous retrouvons cette formule :



The screenshot shows an Excel formula bar for cell E24. The formula is: `=SI(D24>0;-E23*D24*997/((D23*1000+D24*997);SI(D24<0;-E23*D24*1000/((D23+D24)*997);0))`. A red rectangular box highlights the part of the formula: `SI(D24<0;-E23*D24*1000/((D23+D24)*997);0))`.

Nous allons arrêter là nos analyses, mais pour ceux qui voudraient aller encore plus loin dans l'étude du code : <https://ethereum.org/fr/developers/tutorials/uniswap-v2-annotated-code/>