WEEK 1 PLUTUS PIONEER PROGRAM

Contents

The (E)UTxO Model

Overview

One of the most important things you need to understand in order to write Plutus smart contracts is the accounting model that Cardano uses, **the Extended Unspent Transaction Output model.**

The **UTxO model**, without the (E) is the one that was introduced by Bitcoin, but there are other models.
Ethereum for example, uses the so-called **account-based model**, which is what you are used to from normal banking, where everybody has an account, each account has a balance and if you transfer money from one account to another then the balances get updated accordingly.

That is not how the UTxO model works.
Unspent transaction outputs UTxO are exactly what the name says.

- They are transaction outputs from previous transactions that have happened on the blockchain that have not yet been spent.
- A transaction is something that contains an arbitrary number of inputs and an arbitrary number of outputs.
- The effect of a transaction is to consume inputs and produce new outputs.

The important thing is that you can only ever use complete UTxOs as input. Alice cannot simply split her existing 100 ADA into a 90 and a 10, she has to use the full 100 ADA as the input to a transaction.Once consumed by the transaction, Alice's input is no longer a UTxO (an unspent transaction). It will have been spent as an input to Tx 1. So, she needs to create outputs for her transaction.

She wants to pay 10 ADA to Bob, so one output will be 10 ADA (to Bob). She then wants her change back, so she creates a second output of 90 ADA (to herself).
The full UTxO of 100 ADA has been spent, with Bob receiving a new transaction of 10 ADA, and Alice receiving the change of 90 ADA.

In any transaction, **the sum of the output values must match the sum of the input values**. Although, strictly speaking, this is not true. **There are two exceptions.**

1. Transaction fees. In a real blockchain, you have to pay fees for each transactions.
2. Native Tokens. It's possible for transactions to create new tokens, or to burn tokens, in which case the inputs will be lower or higher than the outputs, depending on the scenario.

When Is Spending Allowed?

The way it works is by adding signatures to transactions.

- In transaction 1, Alice's signature has to be added to the transaction.
- In transaction 2, both Alice and Bob need to sign the transaction.

Everything explained so far is just about the UTxO model, not the (E)UTxO model.

The extended part comes in when we talk about smart contracts, so in order to understand that, let's concentrate on the consumption of Alice's UTxO of 100 ADA.

In the UTxO model, the validation that decides whether the transaction that this input belongs to is allowed to consume the UTxO, **relies on digital signatures**.
In this case, that means that Alice has to sign the transaction in order for the consumption of the UTxO to be valid.

The idea of the (E)UTxO model is to make this more general.

Instead of having just one condition, namely that the appropriate signature is present in the transaction, we replace this with arbitrary logic.

This is where Plutus comes in.
Instead of just having an address that corresponds to a public key that can be verified by a signature that is added to the transaction, we have more general addresses, not based on public keys or the hashes of public keys, but instead contain arbitrary logic which decides under which conditions a particular UTxO can be spent by a particular transaction.

So, instead of an input being validated simply by its public key,

- the input will justify that it is allowed to consume this output with
- some arbitrary piece of data that we call the ***Redeemer***.

- We replace the public key address (Alice's in our example), with a **script**, and
- we replace the digital signature with a ***Redeemer***.

What exactly does that mean? What do we mean by *arbitrary logic*?
It is important to consider the context that the script has. There are several options.

Script Context

The Cardano approach

In Plutus,

- the script cannot see the whole blockchain, but it can see the whole transaction that is being validated.
- In contrast to Bitcoin, it can't see only the redeemer of the one input, but it can also see all the inputs and outputs of the transaction, and the transaction itself.

The Plutus script can use this information to decide whether it is ok to consume the output.
There is one last ingredient that Plutus scripts need in order to be as powerful and expressive as Ethereum scripts.

- That is the so-called **Datum**. That is a piece of data that can be associated with a **UTxO** along with the **UTxO value**.

With this it is possible to prove mathematically that Plutus is at least as powerful as the Ethereum model - any logic you can express in Ethereum you can also it can express using the (E)UTxO model.

Cardano approach has a lot of advantages compared to the Ethereum model.
For example, in Plutus,

- it is possible to check whether a transaction will validate in your wallet, before you ever send it to the chain.

- Things can still go wrong with off-chain validation, however.
    - For example in the situation where you submit a transaction that has been validated in the wallet but gets rejected when it attempts to consume an output on-chain that has already been consumed by another transaction. **In this case, your transaction will fail without you having to pay any fees.**
    - But if all the inputs are still there that your transaction expects, then you can be sure that the transaction will validate and will have the predicted effect.
    - **This is not the case with Ethereum.** In Ethereum, the time between you constructing a transaction and it being incorporated into the blockchain, a lot of stuff can happen concurrently, and that is unpredictable and can have unpredictable effects on what will happen when your script finally executes.

- In Ethereum it is always possible that **you have to pay gas fees for a transaction even if the transaction eventually fails with an error**. And that is guaranteed to never happen with Cardano.

- In addition to that, it is also easier to analyse a Plutus script and **to check, or even prove, that it is secure**, because you don't have to consider the whole state of the blockchain, which is unknowable.
    - You can concentrate on this context that just consists of the spending transaction.
    - So you have a much more limited scope and that makes it much easier to understand what a script is actually doing and what can possibly go wrong.

Who is responsible for providing the datum, redeemer and the validator?

- The rule in Plutus is that **the spending transaction has to do that** whereas the producing transaction only has to provide hashes.
    - That means that if I produce an output that sits at a script address then this producing transaction only **has to include the hash of the script and the hash of the datum that belongs to the output**.
    - Optionally it can include the datum and the script as well.
- If a transaction wants to consume such an output then *that* transaction has to provide the datum, the redeemer and the script.
    - Which means that in order to spend a given input, you need to know the datum, because only the hash is publicly visible on the blockchain.

This is sometimes a problem and not what you want and **that's why you have the option to include the datum in the producing transaction**.
If this were not possible, only people that knew the datum by some means other than looking at the blockchain would ever be able to spend such an output.

The (E)UTxO model is not tied to a particular programming language. What we have is Plutus, which is Haskell, but in principal you could use the same model with a completely different programming language, and we intend to write compilers for other programming languages to Plutus Script which is the "assembly" language underlying Plutus.

On-chain and Off-chain code

The important thing to realise about Plutus is that there is on-chain and off-chain code.

On-chain

On-chain code is

- the scripts from the UTxO model.
- In addition to public key addresses we have script address and outputs can sit at such an address, and if a transaction tries to consume such an output, the script is executed, and the transaction is only valid if the script succeeds.
- If a node receives a new transaction, it validates it before accepting it into its mempool and eventually into a block. For each input of the transaction, if that input happens to be a script address, the corresponding script is executed. If the script does not succeed, the transaction is invalid.

The task of a script is to say yes or no to whether a transaction can consume an output.

Off-chain

- In order to unlock a UTxO, you must be able to construct a transaction that will pass validation and that is the responsibility of the off-chain part of Plutus.

- This is the part that runs on the wallet and not on the blockchain and will construct suitable transactions.

Windows Plutus Setup με WSL2 και Nix

1) Installation of Windows Subsystem Linux 2:

https://docs.microsoft.com/en-us/wind...

- Follow the above documentation if you are a Windows User. It's one way where you can set up Plutus Playground. Otherwise, you can use Virtual Box or to set up Ubuntu on your Windows. I suggest this solution it will take some steps but it's faster at the end. When installation of WSL is ready,
- Download from Microsoft Store the Linux Distribution of your choice/
- When everything is set up, you need to set as a default Ubuntu (the version that you have downloaded)

```
wsl --set-default Ubuntu-20.04
wsl --list
```

With the above commands you can to set WSL 2 as the default version when installing a new Linux distribution:

- Install and set up Windows Terminal (it helps)
- Open a Windows Terminal for Ubuntu(wsl2) and press this command:

```
sudo apt-get update
```

If you have any issue or Fetching errors with the above command probably you need to run the below command first:

```
echo "nameserver 8.8.8.8" | sudo tee /etc/resolv.conf > /dev/null
```

Cabal Installation:

```
sudo apt install cabal-install

cabal --version
```

And check the version of it to be sure about the installation.

Nix Installation:

https://nixos.org/download.html

```
curl -L https://nixos.org/nix/install | sh

nix --version
```

And check the version of it to be sure about the installation.

Creation of /etc/nix directory if it doesn't exist:

```
mkdir -p /etc/nix
```

Set up IOHK binary caches -prerequisite for Plutus Playground-

Adding the IOHK binary cache to your Nix configuration will speed up builds a lot, since many things will have been built already by our CI.

If you find you are building packages that are not defined in this repository, or if the build seems to take a very long time then you may not have this set up properly.

To set up the cache:

Create file nix.conf inside directory /etc/nix/ and add the following:

```
sudo vim /etc/nix/nix.conf

where you should put the below:



sandbox = false

use-sqlite-wal = false

substituters = https://hydra.iohk.io https://iohk.cachix.org https://cache.nixos.org/

trusted-public-keys = hydra.iohk.io:f/Ea+s+dFdN+3Y/G+FDgSq+a5NEWhJGzdjvKNGv0/EQ=
iohk.cachix.org-1:DpRUyj7h7V830dp/i6Nti+NEO2/nhblbov/8MW7Rqoo= cache.nixos.org-
1:6NCHdD59X431o0gWypbMrAURkbJ16ZPMQFGspcDShjY=
```

Confirm that you have set up correctly IOHK binary caches by doing this command:

```
cat /etc/nix/nix.conf
```

Clone the The Plutus repository, check out the correct commit as specified in cabal.project Plutus Repo:

https://github.com/input-output-hk/pl...

Plutus Pioneer Program Repo:
https://github.com/input-output-hk/pl...

Create a folder of **Plutus**, and follow these commands:

```
mkdir plutus
cd plutus
git clone https://github.com/input-output-hk/plutus.git
cd plutus
git checkout 3746610e53654a1167aeb4c6294c6096d16b0502
git branch
(you will see the detached HEAD branch)
```

This tag specifically matches to the branch of plutus that we want to run the local server.

In this plutus branch where you are:

Install Nix-shell

```
nix-shell
```

The installation will take more or less 30 minutes, depending on your environment. Take a cookie and when installation is completed you will see something like this at the end, which means nix-shell is installed successfully, you have enter a nix-shell.

You can rerun this command and you will see that this will take less than a minute this time.

```
nix-shell
```

In the same directory plutus:

Set up Plutus playground:

```
nix-build -A plutus-playground.client
nix-build -A plutus-playground.server
```

We continue by following the below commands. We need to build some dependencies before we start the local server:

```
nix-build -A plutus-playground.generate-purescript
nix-build -A plutus-playground.start-backend
nix-build -A plutus-pab
```

!!We need to be always in a nix-shell
nix-shell

We follow then these commands:

```
cd plutus/plutus/
```

```
cd plutus-pab
plutus-pab-generate-purs
cd ../plutus-playground-server
plutus-playground-generate-purs
cd plutus-playgorund-server

We are now ready to start the Playground Server:
plutus-playgorund-server


At this moment we need to see something like this:
```

```
[nix-shell:~/plutus/plutus]$ cd plutus-playground-server/

[nix-shell:~/plutus/plutus/plutus-playground-server]$ plutus-playground-server
plutus-playground-server: for development use only
[Info] Running: (Nothing,Webserver {_port = 8080})
Initializing Context
Initializing Context
Warning: GITHUB_CLIENT_ID not set
Warning: GITHUB_CLIENT_SECRET not set
Warning: JWT_SIGNATURE not set
Interpreter ready
```

Next step is to set up our Plutus-Client:

But first we need to be sure that we have already installed node && npm:

```
node --version
```

If node hasn't been installed yet, we install it:

```
sudo apt install node
node --version
sudo apt install npm


cd plutus
nix-shell
cd plutus-playground-client
npm run start
```

Voila! Plutus plauground!

https://localhost:8009/

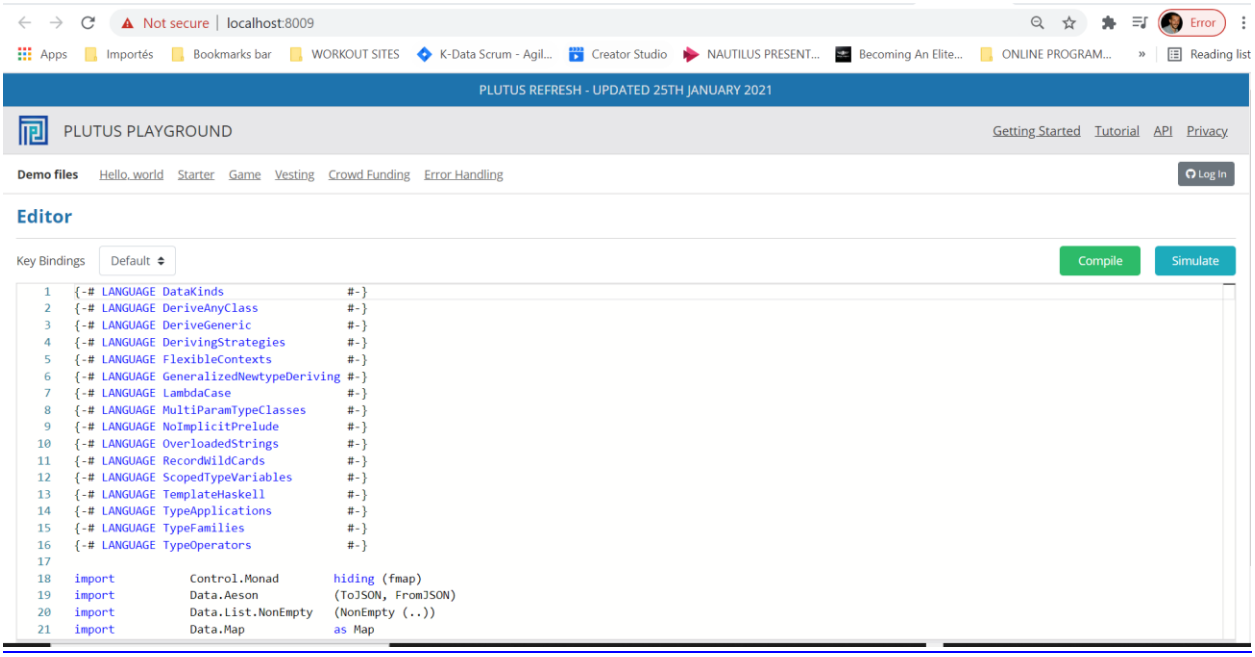
```
[info] Build succeeded.

> plutus-playground-client@1.0.0 webpack:server /home/angelo/plutus/plutus/plutus-playground-client
> webpack-dev-server --progress --inline --hot --mode=development

10% building 1/1 modules 0 active ｢wds｣: Project is running at https://localhost:8009/
 ｢wds｣: webpack output is served from /
 ｢wds｣: Content not from webpack is served from /home/angelo/plutus/plutus/plutus-playground-client/dist
 ｢wdm｣: Hash: e611d94749d61fb39a36
Version: webpack 4.44.2
Time: 20812ms
Built at: 06/05/2021 8:23:15 PM
                                  Asset       Size  Chunks                    Chunk Names
          0.app.e611d94749d61fb39a36.js    27 KiB       0  [emitted] [immutable]
  01996e11c306e5e0b1d54390a6626667.ttf  82 bytes          [emitted]
          1.app.e611d94749d61fb39a36.js  3.75 MiB       1  [emitted] [immutable]
         10.app.e611d94749d61fb39a36.js  12.1 KiB      10  [emitted] [immutable]
```

PLUTUS SERVER-CLIENT READY:

https://localhost:8009



Simulation of different auction scenarios for #week1 of Plutus-Pioneer-Program:

2) Copy and paste the code from the directory of `plutus-pioneer-program/code/week01/src/Week01/EnglishAuction.hs`
3) Remove the below lines:

```
module Week01.EnglishAuction
    ( Auction (..)
    , StartParams (..), BidParams (..), CloseParams (..)
    , AuctionSchema
    , start, bid, close
    , endpoints
    , schemas
    , ensureKnownCurrencies
    , printJson
    , printSchemas
    , registeredKnownCurrencies
    , stage
    ) where
```

4) Press «Compile» button
5) Press «Simulate» button

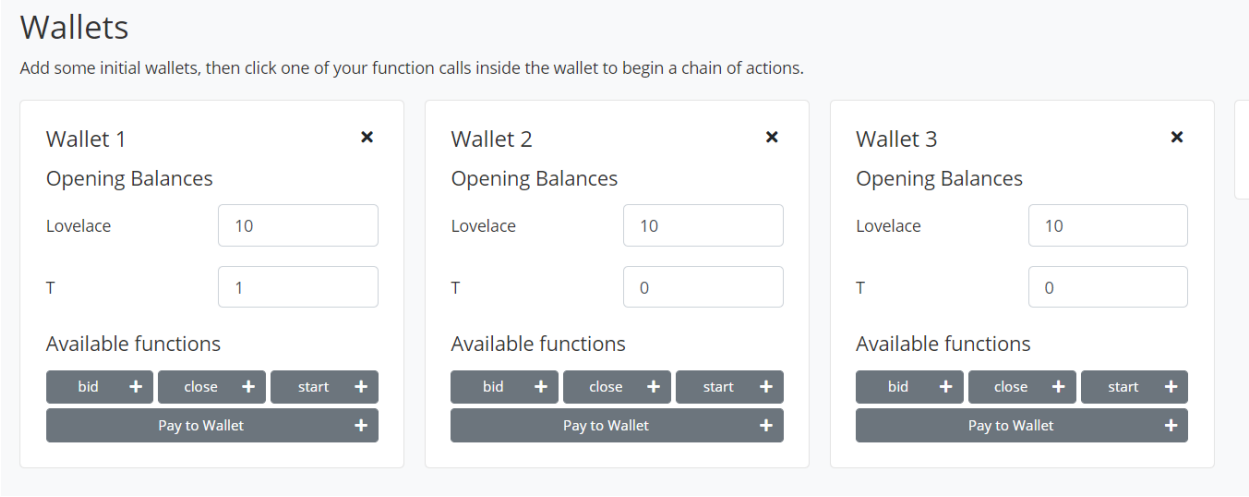
Here you can do different scenarios based of the smart contract which is written. Below you will see the simulation of the example that is explained during the lecture with its transactions. Let's have some fun.

Some things you need to know for this scenario:

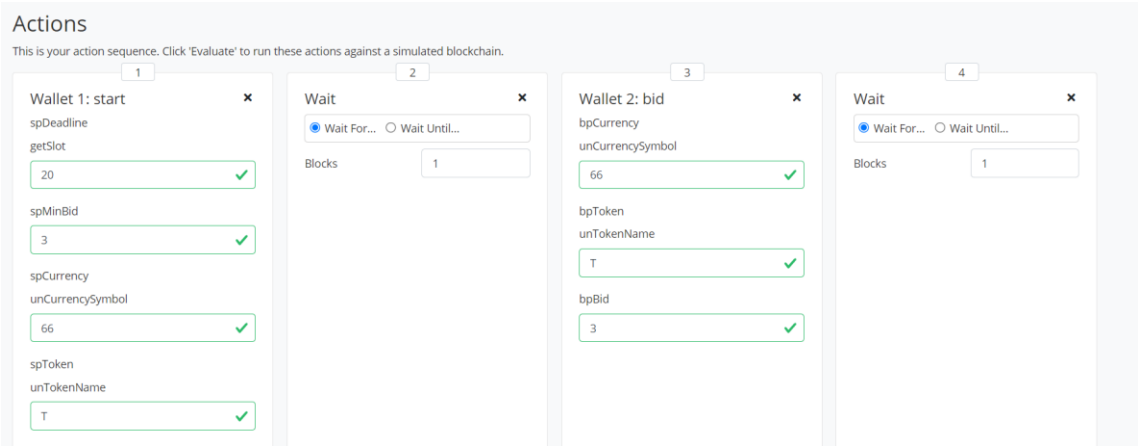
The reason of the creation of this contract is the auction of a NFT. The default option for each wallet are 10 Lovelaces, and 10 tokens. As we know for NFTs, we can't really have here more than one tokens. So, we change the default values based on our scenario.

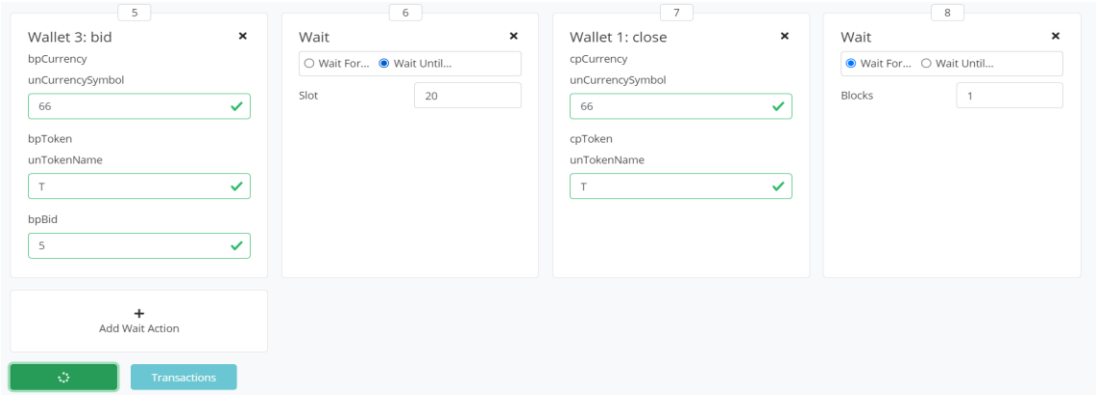
Our purpose here is to have a NFT, to create the auction between 3 wallets, the one has the token, and two other which wants to bid for this NFT. The wallet who bids the most wins. That's it.

Wallet 1 starts the auction and it closes the auction when the transaction has completed successfully.



Press “Start” button from Wallet 1 to begin the auction and we put the below values to the wallets in order to see in practice how the contract will work.

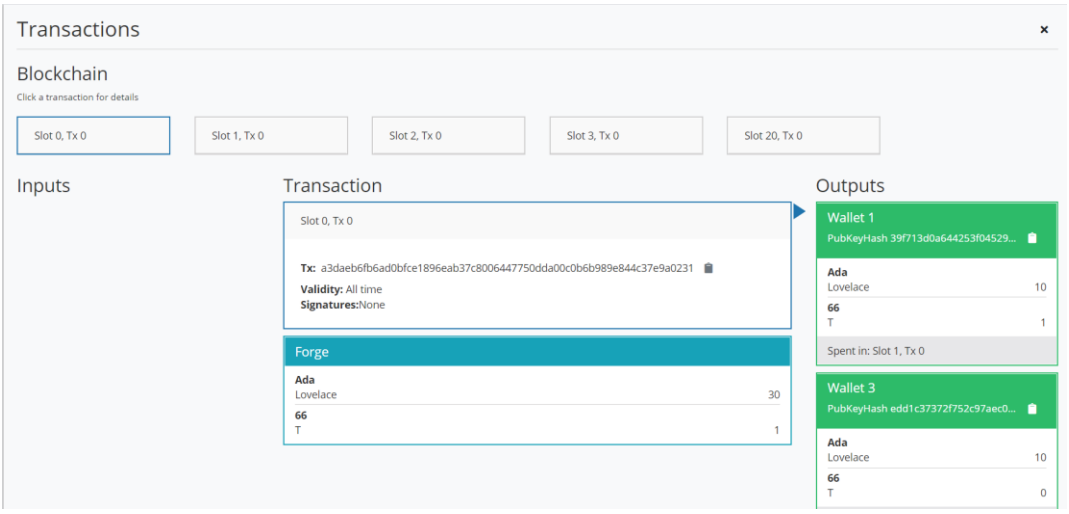




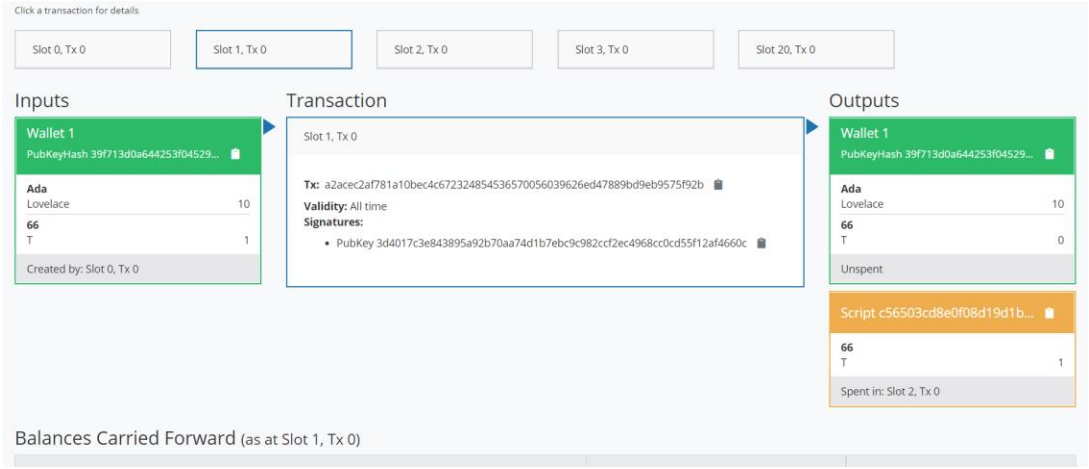
At the end we click to “Evaluate” button , the green one, and we expect to see the below transactions:

What do we see here?

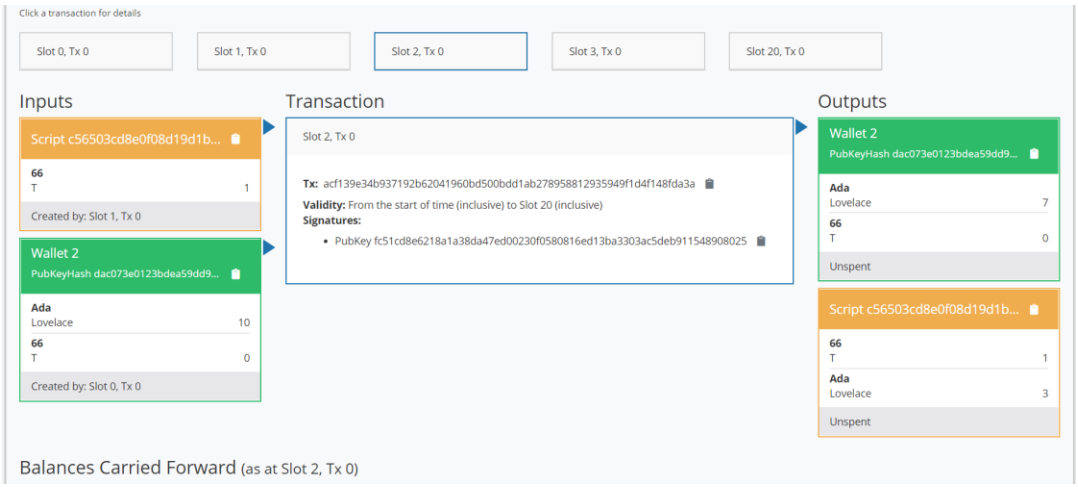
- **Slot 0, Tx 0, Creation of genesis block**



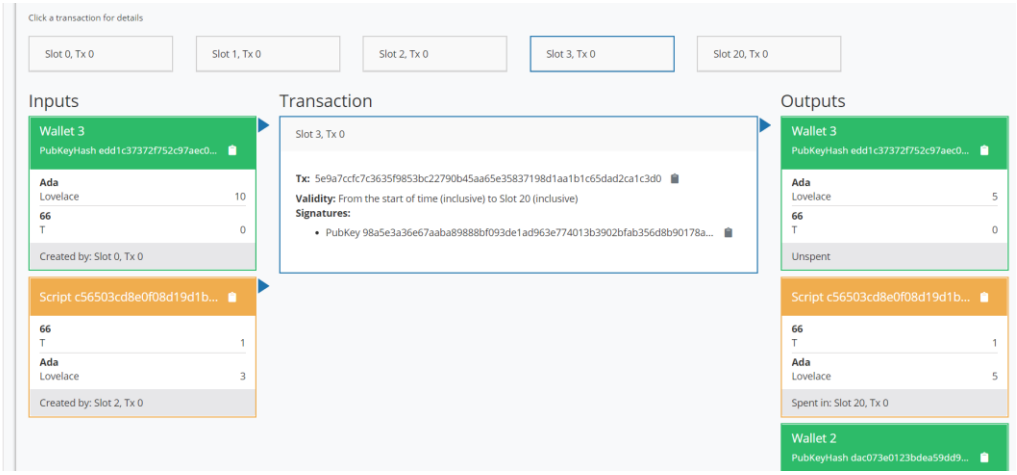
- **Slot 1, Tx 0, Token, passes to the contract**



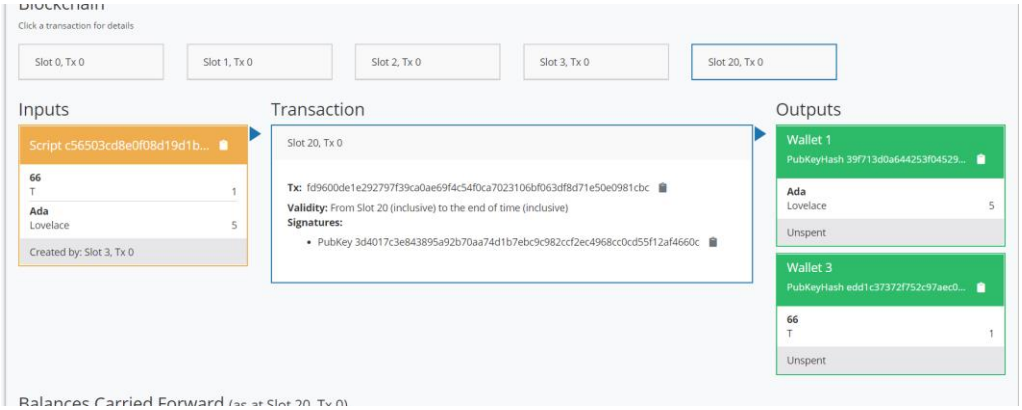
- **Slot 2, Tx 0, Wallet 2 bids 3 Lovelaces for the auction.**



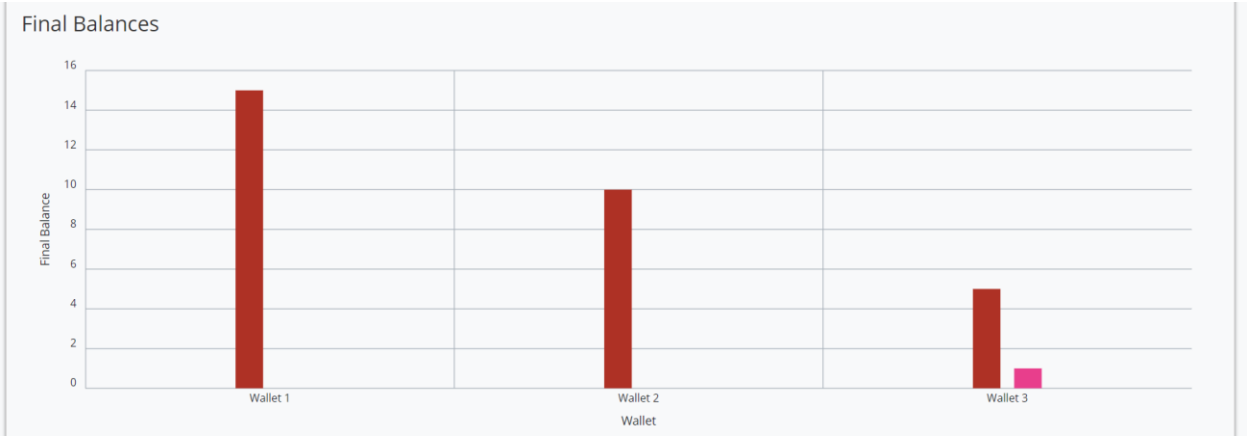
- **Slot 3, Tx 0, Wallet 3 bids 5 lovelaces.**



- **Slot 20, Tx 0, Wallet 3 wins the auction, it takes the Token and contract returns the 3 lovelaces to Wallet 2.**



Results: The result of the contract is shown below. Wallet 3 has the Token now.



The official lecture can be found here:

https://www.youtube.com/watch?v=IEn6jUo-0vU&feature=youtu.be&ab_channel=LarsBr%C3%BCnjes

of Lars Brünjes on YouTube.

The below documentation is created from Angelos Dionysios Kappos for personal use and in order to share it with Cardano "Sapiopool" community and everybody who is interested to learn the smart contracts of Cardano, and not to be sold.

SAPIOPOOL

ATTACK THE DAY
ANGELOS KAPPOS

Links/More information:

LINKEDIN: https://www.linkedin.com/in/angelos-dionysios-kappos-4b668140/

Who are SapioPool: https://sapiopool.com/

Participate in discord channel here: https://discord.com/invite/HRK9gGE9ax

Twitter accounts:

https://twitter.com/angelokappos

https://twitter.com/sapiopool

Sapiopool Youtube: https://www.youtube.com/channel/UCcPH2RMsszRGJ2awvLdMKzQ