Date: 12-05-2022
Student: Carlos Martinez Ival

Course: CP108 Plutus/Haskell I

Problem statement: Bribes and kickbacks to obtain public tenders are common in Africa. Lack of transparency and accountability leads to a multitude of ghost contracts or unfinished projects with no traceability of the public contract. As a tool for change, blockchain can help in solving some of these governance issues plaguing Africa.

Task: A public bidding process has multiple stages, first the competition is held and then the funds are released to the winner. These stages can be carried out in multiple ways. This smart contract is intended to model the last stage once the administration releases the funds to the bidder

Possible solution: In this smart contract a contractor goverment' bids (like fix the street) to Neither trusts each other, but they both trust a mediator. The goverment pays the price into the contract account: if both the goverment and the contractor agree that the goverment has received the service, then the contractor receives the price; if there is any problem, then the mediator ensures that the goverment or the contractor gets their money.

Script: Can be used for this exam and can be tested at the Marlowe Playground;

```
{-# LANGUAGE OverloadedStrings #-}
module Escrow where

import Language.Marlowe.Extended

main :: IO ()
main = print . pretty $ contract

-- We can set explicitRefunds True to run Close refund analysis
-- but we get a shorter contract if we set it to False
explicitRefunds :: Bool
explicitRefunds = False

contractor, goverment, arbiter :: Party
goverment = Role "Goverment"
contractor = Role "Contractor"
arbiter = Role "Mediator"

price :: Value
price = ConstantParam "Price"
```

```
depositTimeout, disputeTimeout, answerTimeout, arbitrageTimeout ::
Timeout
depositTimeout = SlotParam "Payment deadline"
disputeTimeout = SlotParam "Complaint response deadline"
answerTimeout = SlotParam "Complaint deadline"
arbitrageTimeout = SlotParam "Mediation deadline"

choice :: ChoiceName -> Party -> Integer -> Contract -> Case
choice choiceName chooser choiceValue = Case (Choice (ChoiceId
choiceName chooser)
                                                    [Bound choiceValue
choiceValue])

deposit :: Timeout -> Contract -> Contract -> Contract
deposit timeout timeoutContinuation continuation =
   When [Case (Deposit contractor goverment ada price) continuation]
        timeout
        timeoutContinuation

choices :: Timeout -> Party -> Contract -> [(Integer, ChoiceName,
Contract)] -> Contract
choices timeout chooser timeoutContinuation list =
   When [choice choiceName chooser choiceValue continuation
         | (choiceValue, choiceName, continuation) <- list]
        timeout
        timeoutContinuation

contractorToGoverment, payContractor :: Contract -> Contract
contractorToGoverment = Pay contractor (Account goverment) ada price
payContractor = Pay goverment (Party contractor) ada price

refundGoverment :: Contract
refundGoverment
| explicitRefunds = Pay goverment (Party goverment) ada price Close
| otherwise = Close

refundContractor :: Contract
refundContractor
| explicitRefunds = Pay contractor (Party contractor) ada price Close
| otherwise = Close

contract :: Contract
contract = deposit depositTimeout Close $
```

```
            choices disputeTimeout goverment refundContractor
               [ (0, "Everything is alright"
                 , refundContractor
                 )
               , (1, "Report problem"
                 , contractorToGoverment $
                   choices answerTimeout contractor refundGoverment
                      [ (1, "Confirm problem"
                        , refundGoverment
                        )
                      , (0, "Dispute problem"
                        , choices arbitrageTimeout arbiter
refundGoverment
                             [ (0, "Dismiss claim"
                               , payContractor
                                 Close
                               )
                             , (1, "Confirm problem"
                               , refundGoverment
                               )
                             ]
                        )
                      ]
                 )
               ]
```

Language: Haskell and Plutus

Student: Carlos Martinez Ival Course: CP108 Plutus/Haskell