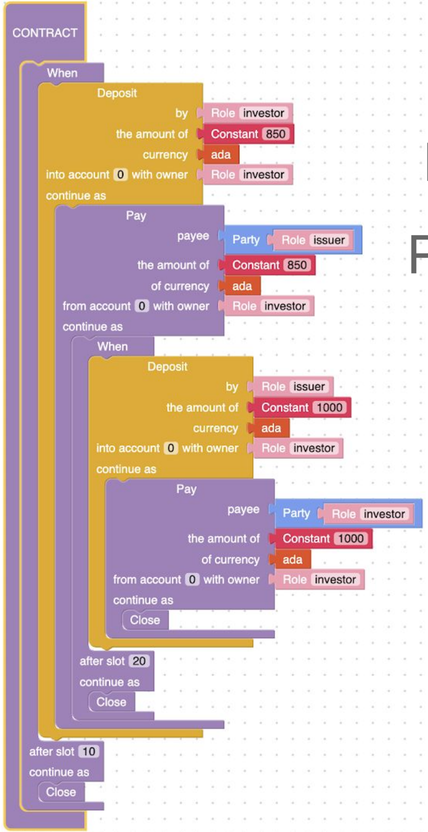
Cardano challenge: **Actus in Marlowe**



```
CONTRACT
  When
    Deposit
      by  Role investor
      the amount of  Constant 850
      currency  ada
      into account 0 with owner  Role investor
      continue as
        Pay
          payee  Party  Role issuer
          the amount of  Constant 850
          of currency  ada
          from account 0 with owner  Role investor
          continue as
            When
              Deposit
                by  Role issuer
                the amount of  Constant 1000
                currency  ada
                into account 0 with owner  Role investor
                continue as
                  Pay
                    payee  Party  Role investor
                    the amount of  Constant 1000
                    of currency  ada
                    from account 0 with owner  Role investor
                    continue as
                      Close
            after slot 20
            continue as
              Close
  after slot 10
  continue as
    Close
```

Project Title: Marlowe EasyRead

Project Goal: Enhance Actus in Marlowe by visualizing how Actus contracts operate, utilizing easy to read on and off chain contracts.

Cofounder of CryptoBounty LLC
SPO on Cardano
Stake Pool Ticker “307”

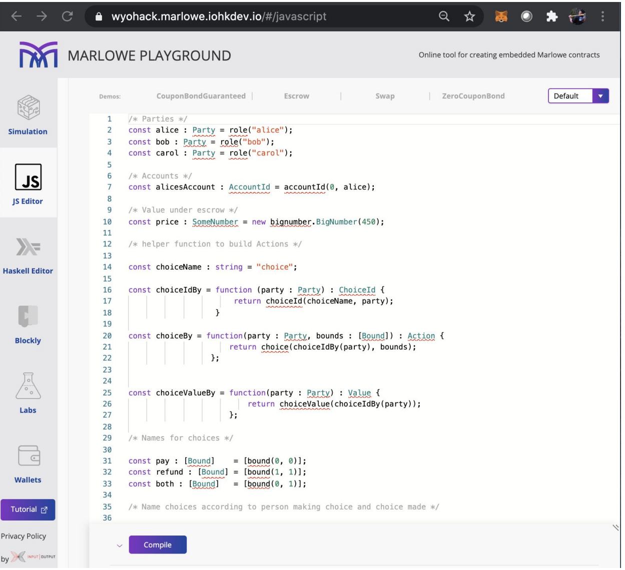
Tony Olson “Tony From Shoshoni”

@ShoshoniTony MCSE, A+, Network+

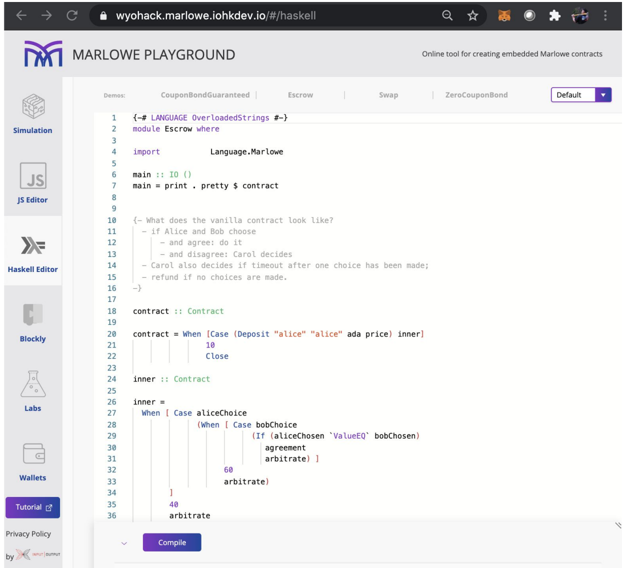
Technology Coordinator / Technology Instructor
Fremont County School District #24
Shoshoni, Wyoming

Cardano enthusius, love grandkids, Wyoming, learning and teaching blockchain technologies

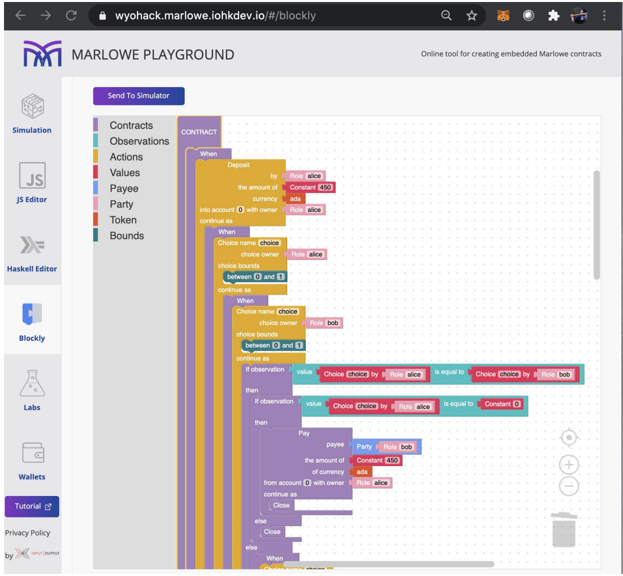
Problem Although you can create contracts in Javascript, Haskell, or Blockly within Marlowe



```
/* Parties */
const alice : Party = role("alice");
const bob : Party = role("bob");
const carol : Party = role("carol");

/* Accounts */
const alicesAccount : AccountId = accountId(0, alice);

/* Value under escrow */
const price : SomeNumber = new bignumber.BigNumber(450);

/* helper function to build Actions */

const choiceName : string = "choice";

const choiceIdBy = function (party : Party) : ChoiceId {
                        return choiceId(choiceName, party);
                    }

const choiceBy = function(party : Party, bounds : [Bound]) : Action {
                       return choice(choiceIdBy(party), bounds);
                   };


const choiceValueBy = function(party : Party) : Value {
                          return choiceValue(choiceIdBy(party));
                      };

/* Names for choices */

const pay : [Bound]    = [bound(0, 0)];
const refund : [Bound] = [bound(1, 1)];
const both : [Bound]   = [bound(0, 1)];

/* Name choices according to person making choice and choice made */

```



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

import           Language.Marlowe

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


{- What does the vanilla contract look like?
  - if Alice and Bob choose
      - and agree: do it
      - and disagree: Carol decides
  - Carol also decides if timeout after one choice has been made;
  - refund if no choices are made.
-}

contract :: Contract

contract = When [Case (Deposit "alice" "alice" ada price) inner]
                10
                Close

inner :: Contract

inner =
  When [ Case aliceChoice
              (When [ Case bobChoice
                          (If (aliceChosen `ValueEQ` bobChosen)
                             agreement
                             arbitrate) ]
                    60
                    arbitrate)
        ]
        40
        arbitrate
```

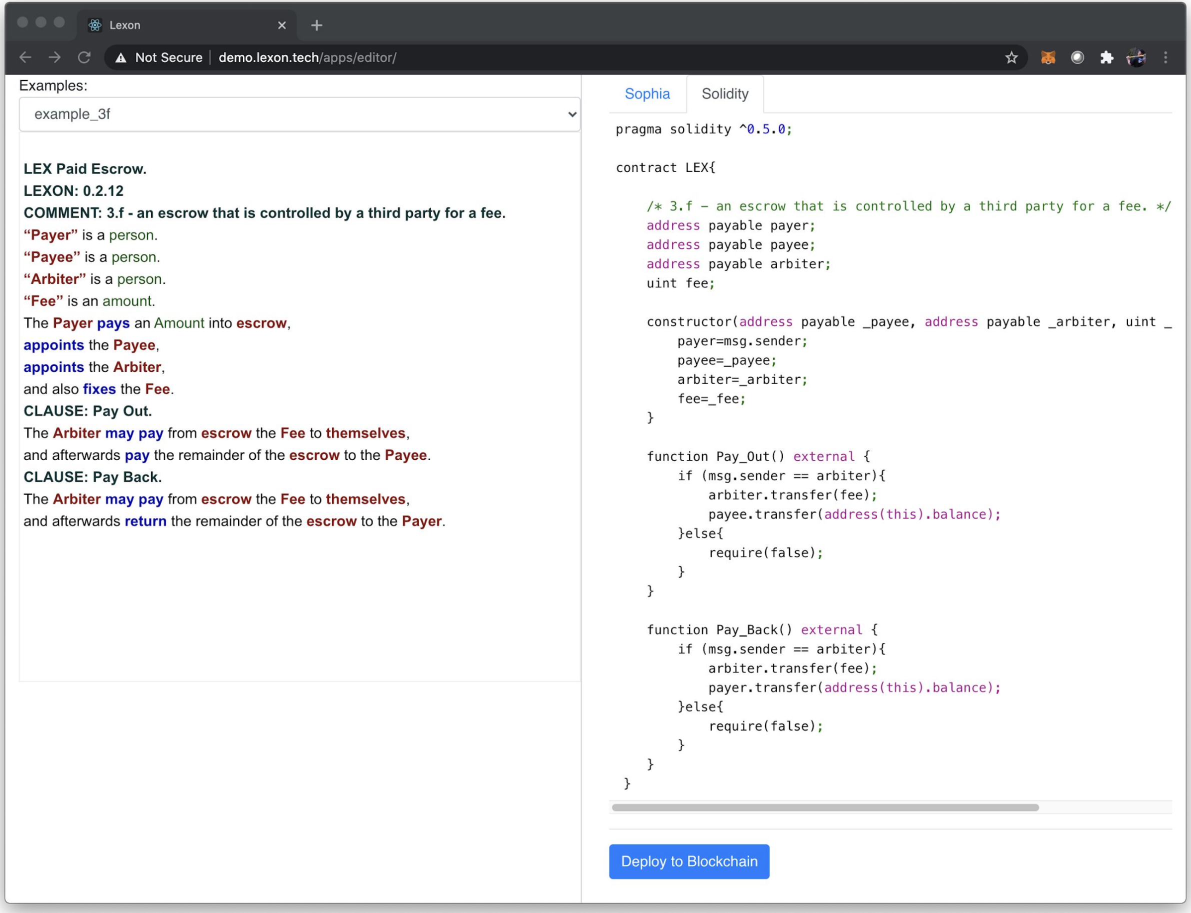


Javascript, Haskell, or even Blockly are not all that readable to a someone without a programing background.

A person entering into a “smart contract” would have a hard time verifying the intent of the contract as even blockly can be a bit hard to interpret without the assistance of a computer programer.

Solution

Integrate the open source **Lexon** language into Marlowe



Examples:

example_3f

LEX Paid Escrow.
LEXON: 0.2.12
COMMENT: 3.f - an escrow that is controlled by a third party for a fee.
"Payer" is a person.
"Payee" is a person.
"Arbiter" is a person.
"Fee" is an amount.
The **Payer pays** an Amount into **escrow**,
appoints the **Payee**,
appoints the **Arbiter**,
and also **fixes** the **Fee**.
CLAUSE: Pay Out.
The **Arbiter may pay** from **escrow** the **Fee** to **themselves**,
and afterwards **pay** the remainder of the **escrow** to the **Payee**.
CLAUSE: Pay Back.
The **Arbiter may pay** from **escrow** the **Fee** to **themselves**,
and afterwards **return** the remainder of the **escrow** to the **Payer**.

```
pragma solidity ^0.5.0;

contract LEX{

    /* 3.f - an escrow that is controlled by a third party for a fee. */
    address payable payer;
    address payable payee;
    address payable arbiter;
    uint fee;

    constructor(address payable _payee, address payable _arbiter, uint _
        payer=msg.sender;
        payee=_payee;
        arbiter=_arbiter;
        fee=_fee;
    }

    function Pay_Out() external {
        if (msg.sender == arbiter){
            arbiter.transfer(fee);
            payee.transfer(address(this).balance);
        }else{
            require(false);
        }
    }

    function Pay_Back() external {
        if (msg.sender == arbiter){
            arbiter.transfer(fee);
            payer.transfer(address(this).balance);
        }else{
            require(false);
        }
    }
}
```

Value Proposition

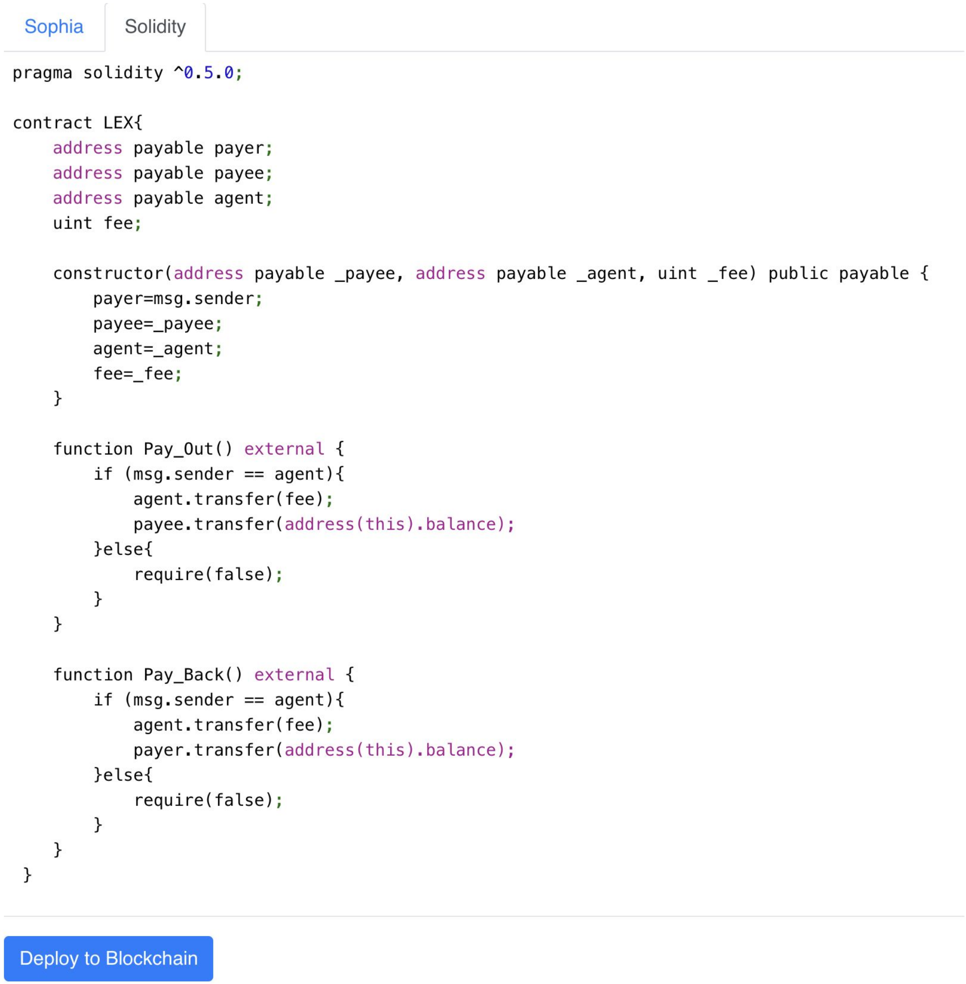
LEX Escrow.

“Payer” is a person.
“Payee” is a person.
“Agent” is a person.
“Fee” is an amount.

The Payer pays an Amount into escrow, appoints the Payee, appoints the Agent, and also fixes the Fee.

CLAUSE: Pay Out.
The Agent may pay from escrow the Fee to themselves, and afterwards pay the remainder of the escrow to the Payee.

CLAUSE: Pay Back.
The Agent may pay from escrow the Fee to themselves, and afterwards return the remainder of the escrow to the Payer.



```
pragma solidity ^0.5.0;

contract LEX{
    address payable payer;
    address payable payee;
    address payable agent;
    uint fee;

    constructor(address payable _payee, address payable _agent, uint _fee) public payable {
        payer=msg.sender;
        payee=_payee;
        agent=_agent;
        fee=_fee;
    }

    function Pay_Out() external {
        if (msg.sender == agent){
            agent.transfer(fee);
            payee.transfer(address(this).balance);
        }else{
            require(false);
        }
    }

    function Pay_Back() external {
        if (msg.sender == agent){
            agent.transfer(fee);
            payer.transfer(address(this).balance);
        }else{
            require(false);
        }
    }
}
```

Benefit

Advantage of using human readable language for code

Text can serve as a legal contract that a contract party, lawer, or judge could understand.

Text can compile directly to an executable smart contract. Languages **Lexon** currently exports to are “Sophia” and “Solidity”. Support for Plutus would be a game changer.

Participation in a DAO (decentralized autonomous organization) would be much more comprehensible if the DAO rules where written out in human readable code.

LEX Escrow.

“Payer” is a person.
“Payee” is a person.
“Agent” is a person.
“Fee” is an amount.

The Payer pays an Amount into escrow, appoints the Payee, appoints the Agent, and also fixes the Fee.

CLAUSE: Pay Out.
The Agent may pay from escrow the Fee to themselves, and afterwards pay the remainder of the escrow to the Payee.

CLAUSE: Pay Back.
The Agent may pay from escrow the Fee to themselves, and afterwards return the remainder of the escrow to the Payer.