Blockchain HF1 - Safe crossing of autonomous cars

using Ethereum Solidity
by Németh Ágoston, Raisz Olivér
2021. 05. 12.

CONTENT

IMPLEMENTATION SKELETON

Authority

- Can issue new crossings
- Trains and cars can request clearance

Crossing

- Keeps track of occupied lanes

Car

- Can ask for crossing permission
- Must release the given permission, otherwise a ticket must be paid

Train

- Can ask for crossing permission
- Can explicitly lock the crossing

DESIGN DECISIONS

We implemented the task so that every participant (trains, cars, crossings) is an address on the network. Trains and crossings should be registered by the admin first (which is the address of the railway company).

As the skeleton above implies *(no, not that one)*, the cars can request crossing permission (sometimes referred to as pass).
If they don't return the given pass within a time frame, their address is stored and they must pay a penalty.
Trains can request to lock a crossing, but only one train can lock a train at the same time. They must release this lock after passing.

Since there can be no precisely triggered events on the blockchain (recurring every ten seconds is the scale), we chose not to implement time critical validations, such as pass validity and free states of a crossing. We provide a backend for these events to be checked, and the users (i.e. the trains, crossings and the cars) are responsible for safety validation. This also reduces unnecessary calls, thus eliminates a great portion of the overall operation costs.

API CALLS

getAuthority():
    Parameters:
        - None.
    Requires:
        - Nothing. Anyone can call it.
    Returns:
        - The user who deployed the smart contract.
        That address is the singular admin address,
        the permission issuing authority.

IsCrossingFree(address):

Parameters:

- Address of the crossing of interest.

Requires:

- The given parameter should be a crossing.

Returns:

- Boolean - whether there are any cars holding a pass permission for the given crossing.

RegisterCrossing(address, int, int):

Parameters:

- Address of the crossing to be registered
- Number of available lanes in given crossing
- Car capacity per lane

Requires:

- Admin permissions.

Returns:

- Nothing.

RegisterTrain(address):

Parameters:

- Address of the train to be registered

Requires:

- Admin permissions.

Returns:

- Nothing.

LockCrossing(address):

Parameters:

- Address of the crossing to be locked

Requires:

- Only trains can call this function
- Must get a crossing as parameter

Returns:

- An enum. The return value can indicate one of the following things:
    - Another lock is active: The lock can not be set, because another train has active control over the crossing. The train must halt and continue to poll the crossing's availability.

- Lock successful: The lock has been set, and the train is allowed to pass.
- Lock requested: The lock has been activated, but there are cars in the crossing. No more crossing permissions are issued, and until the last car has passed, the train must not enter the crossing. See Halt.
- Halt: the train must halt, because the safety timeout is over. This occurs when there are still cars on the rails after a crossing-specifically predefined delay.

ReleaseLock(address):

Parameters:
- The crossing, which the train wants to unlock.

Requires:
- Only trains can perform this action.
- The given parameter must be registered as a crossing.

Returns:
- Nothing.

RegisterCar():

Parameters:
- None.

Requires:
- Not to be registered as another entity.
- Otherwise anyone can perform this action.

Returns:
- Nothing.

RequestPass(address, int):

Parameters:
- Address of the crossing.
- Line ID, in which the car wishes to enter the crossing ahead.

Requires:
- Only cars can perform this action.
- The specified address must be registered as a crossing.

```
        - The specified line of the crossing must
        not have reached the maximum car capacity.
    Returns:
        - A timestamp until which the pass permission
        is valid.

ReleasePass():
    Parameters:
        - None.
    Requires:
        - Only cars can perform this action.
    Returns:
        - Nothing.

IsPassValid():
    Parameters:
        - None.
    Requires:
        - Only cars can perform this action.
    Returns:
        - Boolean, whether the issuer car has a valid pass at the moment.

CheckIfPassIsReleased():
    Parameters:
        - None.
    Requires:
        - Only cars can perform this action.
    Returns:
        - Nothing.

IsCar(address):
    Parameters:
        - Address to be identified as a car
    Requires:
        - Nothing.
    Returns:
        - Boolean, whether the given address belongs to a car

IsTrain(address):
    Parameters:
```

- Address to be identified as a train

Requires:

- Nothing.

Returns:

- Boolean, whether the given address belongs to a train

IsCrossing(address):

Parameters:

- Address to be identified as a crossing

Requires:

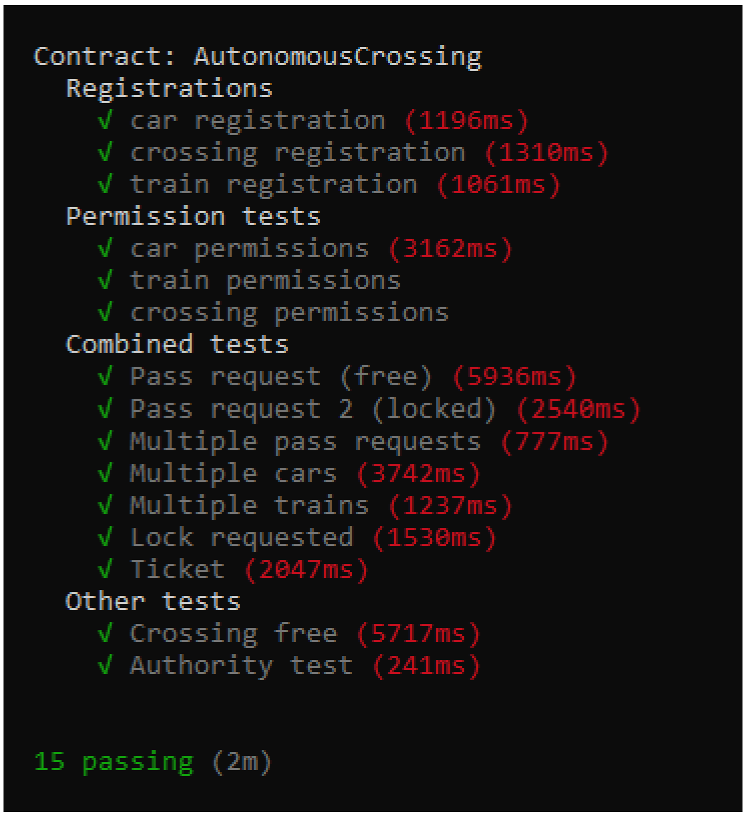
- Nothing.

Returns:

- Boolean, whether the given address belongs to a crossing

TESTING

We used Truffle and Ganache for testing, with the help of chai and truffle assertions. There are 15 test cases altogether, divided into 4 sections called Registrations, Permission tests, Combined tests and Other tests.



```
Contract: AutonomousCrossing
  Registrations
    √ car registration (1196ms)
    √ crossing registration (1310ms)
    √ train registration (1061ms)
  Permission tests
    √ car permissions (3162ms)
    √ train permissions
    √ crossing permissions
  Combined tests
    √ Pass request (free) (5936ms)
    √ Pass request 2 (locked) (2540ms)
    √ Multiple pass requests (777ms)
    √ Multiple cars (3742ms)
    √ Multiple trains (1237ms)
    √ Lock requested (1530ms)
    √ Ticket (2047ms)
  Other tests
    √ Crossing free (5717ms)
    √ Authority test (241ms)


15 passing (2m)
```

Registrations:

A car, a train and a crossing is registered and their values are checked (isSet, id, etc).

Permission tests:

We attempt to do things the given address is not allowed to, expecting an exception. This includes:
- locking crossings with the address of a car.
- registering a train without admin permissions
- registering a crossing without admin permissions

Combined tests:

This section consists of tests of complex, lifelike situations.

Pass request (free):
A car requests a pass, then releases it. We check multiple things during the process.
The main reason is to find out whether the car only has a permission after it's requested and before it's released.

Pass request 2 (locked):
A car tries to request a permission when the crossing is locked. The aim is not to give it permission.

Multiple pass requests:
A car tries to request a 2nd pass before the first has been released. It shouldn't get a second one.

Multiple cars:
Multiple cars try to request a pass. While their number is lower than the capacity of the lane (2), the passes are given,
but when their number exceeds the lane capacity (the 3rd car arrives), the new car don't get a pass.

Multiple trains:
A second train tries to lock the crossing, after it has been locked by another train.
The train should get the 'another lock is active' enum as a response.

Lock requested:
A train tries to lock the crossing while a car is in it (i.e. it has an active permission).
The train should get the 'lock requested' enum as a response.

Ticket:
A car doesn't return it's pass, so a ticket is given (it's id is stored).

Other tests

This section consists of 2 tests we failed to find a better label for. :D
Crossing free tests whether there are cars in the crossing or not, in various situations (i.e. before a lock, during a lock and after a lock).

Authority test is a simple test, it just checks whether the admin address owns the appropriate permissions.