Safe Crossing for Autonomous Cars – Ethereum (Solidity)

|  |  |
| --- | --- |
| Név: | Neptun-kód: |
| Árvay Péter | ID3FM3 |
| Kisteleki Dávid | QR19QG |

# Feladatleírás:

Unguarded level crossings will pose a particular challenge for autonomous vehicles; from the safety point of view, relying only on the onboard sensor packages and intelligence to decide whether it is safe to cross (no train is approaching) will be problematic. At the same time, the railway infrastructure – commonly partitioned into blocks6 – “knows” where the trains are; e.g., safety systems ensure that no train can enter a block already (or still) occupied by a train.

The task is to design and implement a train crossing smart contract with the following features:

1. The railroad infrastructure must periodically signal the crossing to be in a “FREE TO CROSS” state. This state has a preset validity time; if the last update is older than that, the crossing must be assumed to be in a “LOCKED” state. This can happen either on a train approaching or a failure of the infrastructure.
2. Autonomous vehicles wanting to cross must request permission to do so.
3. Permission may be granted only if the intersection is not in a “LOCKED” state.
4. Additionally, an intersection comprises one or more lanes. A single lane can accommodate a fixed number of crossing cars, predetermined by the railroad infrastructure managing authority.
5. Autonomous vehicles must explicitly release their permission after leaving the crossing. Failure to do so will later involve legal action; for this purpose, their identity must be recorded on the ledger in a privacy-preserving way (as much as possible).
6. As an additional safety measure, using a means of communication independent from the one used by the infrastructure mentioned above, approaching trains also explicitly request the crossing to get into the “LOCKED” state (and release this signal only when they have passed it).
7. However, if the intersection is still occupied, it transitions into a special state (also signaling this to the train) where the train will have priority to request crossing, i.e., no more cars are granted crossing permission until the train crosses.
8. If the train cannot gain permission in a predetermined time since the original attempt, it is assumed that there is an obstacle at the crossing, and the train can break or halt.

Note: this exercise does reflect some of the concepts used in safety-critical engineering but falls far from a full, real-life safety strategy. (i.e., do not build a real system based on this specification. If you happen to be a rail fan, https://arxiv.org/abs/1901.06236 and

https://www.deutschebahn.com/en/Digitalization/technology/NewTechnology/blockchain-3520362 are good reads.)

# Megvalósítás:

## State -ek:

**enum State { FREE\_TO\_CROSS, LOCKED, OCCUPIED, OCCUPIEd\_AND\_LOCKING };**

Ezek az állapotok fejezik ki a kereszteződést felépítő több „autó” sáv állapotát, a vonat és a sávon lévő autók függvényében.

**Magyarázat:**

* **FREE\_TO\_CROSS** : olyan állapot, amikor nincs se autó, és se vonat a kereszteződésnek a sávjában.
* **LOCKED** : olyan állapot, amikor vagy vonat érkezik a kereszteződésbe, vagy már ott van, és nincsen egy autó sem a kereszteződésnek a sávjában.
* **OCCUPIED** : olyan állapot, amikor legalább egy autó van a kereszteződésnek ennek a sávjában, és nem érkezik vonat és nincsen vonat a kereszteződésben vonat.
* **OCCUPIED\_AND\_LOCKING** : olyan állapot, amikor van legalább egy autó a kereszteződésnek ennek a sávjában, és érkezőben van a kereszteződéshez egy vonat.

A képen képernyőkép, szöveg, Betűtípus, tervezés látható

Automatikusan generált leírás

**enum registyevent { ENTERED\_CROSSING, PERMISSION\_GIVEN, PERMISSION\_REMOVED };**

Ezek az állapotok jelzik az autók által végzett műveleteket, amit az okosszerződésben tárolunk.

**Magyarázat:**

* **PERMISSION\_GIVEN**: olyan művelet, amikor megkapja az autó az engedélyt.
* **PERMISSION\_REMOVED**: olyan művelet, amikor az autó elveszti az engedélyt.
* **ENTERED\_CROSSING**: olyan művelet, amikor az autó belép a kereszteződésbe.

## Struct -ok:

**struct Lane**

Ez modellezi egy sávot a kereszteződésben, amin az autók haladnak keresztül.

Három mezője van:

* State state: A sáv jelenlegi állapotát jelöli.
* uint capacity: A sávban elfoglalt autók számát jelöli.
* uint maxCapacity: A sávban megengedett autók maximális autók számát jelöli.

**struct Permission**

Ez modellezi az autóknak adott jogot a kereszteződésen való átkeléshez.

Három mezője van:

* address vehicleAddress: címe az autónak, ami az engedélyt kapta az átkeléshez.
* uint lane: A kereszteződésben lévő sáv számát jelöli, amelyet adunk az autónak.
* uint expirationDate: A lejárati idejét jelöli az autónak adott átkelési engedélyhez.

**struct Registry**

Ez tárolja a bejegyzés, ami egy autó által végzett műveletet tárol, a visszakövethetőség érdekében.

Öt mezője van:

* address vehicleAddress: címe az autónak.
* uint lane: A kereszteződésben lévő sáv számát jelöli.
* uint time: A művelet időpontja.
* RegistryEvent registryEvent: Az okosszerződésben tárolt művelet.
* Permission permission: Az autó joga.

## Event -ek:

A főbb történésekhez az alábbi eseményeket hoztuk létre, amik a tájékoztatják a külvilágot, a belső történésekről. Legtöbb csupán tájékoztató jellegű, a TrainCanPass() kritikus, mivel ha a vonat az érkeyése előtt egy általa meghatározott időn belül nem kapja meg ezt az eventet, úgy értelmezi, hogy az átkelő blokkolva van és elkezd lassítani.

* event TrainComing();
* event TrainCanPass();
* event TrainPassed();
* event LaneOccupied(uint laneIndex);
* event LaneFree(uint laneIndex);

## Mapping -ek:

mapping (address => bool) public carHasPermission;

* Ez adja meg a létezését az autóhoz tartozó „Permission” -nak.

mapping (address => Permission) public permissionOfAddress;

* Ez adja át az autóhoz tartozó „Permission” -t.

mapping (address => uint) public laneOfCar;

* Ez adja meg az autóhoz tartozó sáv számát a kereszteződésben.

mapping(uint => Lane) public lanes;

* Ez adja meg a sáv azonosítóhoz tartozó sáv objektumot.

## Constructor:

**constructor(uint \_numberOfLanes, uint \_maxCapacityByLane, uint \_validityTime)**

Az okosszerződés konstruktora, a hálózatra való delploy során fut le egyszer és a kritikus szerződés változók értékeit állítja be. Jelen esetben a sávok a számát,a sávok kapacitását és a hirdetések és engedélyek érvényességének intervallumát.

## Function -ok:

Get -terk, és Set -terek, amik a program alapvető működését segítik, amik a programunk javíthatóságában vesznek részt, mert a kódon nem változtathatunk miután a blockchain -re feltöltöttük:

* function getLaneCapacity(uint calldata \_laneId)
* function laneHasCapacity(uint calldata \_laneId)
* function getNumberOfLanes()
* function setValidityTime(uint \_validityTime)
* function setNumberOfLanes(uint \_numberOfLanes)
* function setMaxCapacityOfLane(uint \_laneId, uint \_maxCapacity)

function checkTrainCanPass()

* A függvény a sávok ellenőrzést hajtja végre, ugyanis ha bármely sáv a vonat érkezése előtt nem vált LOCKED-ba, az baleset veszélyt idéz elő, a vonatnak pedig el kell kezdenie lassítani. Egyszerű bool-al tér vissza, ezt fogjuk ellenőrizni a TrainCanPass() event emit előtt.

function trainComing() external onlyOwner

* Csak az infrastruktúra hívhatja meg ezt a függvényt, jelez a vonat érkezéséről, és frissíti a sávok állapotát a fennt mellékelt ábra alapján, ezután leellenőrzi a szabad áthaladást a vonatnak, ami ha biztosítva van kiküldi a TrainCanPass eventet

function trainGone(uint \_laneId)

* Csak az infrastruktúra hívhatja meg a függvényt, jelez a vonat távozásáról, és frissíti a sávok státuszát.

function noTrainUpdate()

* Az infrastruktúra hívja meg, frissíti a sávok státuszát, nem jön vonat a kerszteződés felé.

function tryToEnterLane(uint \_laneId)

* Ezt a függvényt az autók hívják meg, amikor be szeretnének jutni a kereszteződés egyik sávjába.

function requestPermission()

* Ezt a függvényt az autók hívják meg, amikor jogot kérnek a kereszteződésen való átkeléshez.

function carEnter(uint \_laneId)

* Ezt a függvényt az autók hívják meg, és a feladata az autók engedélyeinek a megfelelő felhasználása és elleőrzése.

function carLeave(uint \_laneId, address \_vehicleAddress)

* Az infrastruktúra hívja meg ezt a függvényt, amikor egy autó elhagyja a kereszteződést, és frissíti az infrastruktúra státuszát.