Safe Crossings for Autonomous Cars

(Persons of interest: Hajdú Patrik Zsolt and Scholtz Bálint András)

The Design Decision

Addresses:

- infrastructure: The address of the entity that manages the infrastructure and has special privileges, such as updating the crossing state and authorizing trains.

Units:

- crossingValidity: The time period for which the "FREE_TO_CROSS" status remains valid. This ensures that if no updates are received within this period, the crossing is considered "LOCKED" or "PRE-LOCKED".
- maxCarsPerLane: The maximum number of cars that can be allowed to cross at any given time.
- preLockedTime: The time period for which the crossing can be in a "PRE_LOCKED" state. After that the train, which requested crossing permission must start breaking, if the crossing is not in "LOCKED" state.
- lastUpdate: Stores the timestamp of the last state update for the crossing.
- currentCrossingCarNumber: Keeps track of the number of cars currently crossing.
- trainCrossingRequestNumber: Keeps track of the number of train crossing requests currently active.

Enum variable:

- crossingState: Stores the current state of the crossing (FREE_TO_CROSS, LOCKED, or PRE_LOCKED).

Maps:

- crossingVehicles: Maps addresses of cars to a boolean indicating whether they have permission to cross.
- trainCrossingRequests: Maps addresses of trains to a boolean indicating whether they have requested to cross.
- trainRequestTimes: Maps addresses of trains to the timestamp of when they first requested to cross.
- authorizedTrains: Maps addresses of trains to a boolean indicating whether they are authorized as trains. After that they can use the train functions.

The Data Model Your Contract Stores and Manipulates

Modifiers:

- onlyInfrastructure: Ensures that only the infrastructure manager can call certain functions.
- onlyAuthorizedTrain: Ensures that only authorized trains can call certain functions.
- onlyFreeToCross: Checks if the crossing is in the "FREE_TO_CROSS" state and has not timed out.
- onlyWhenCrossingNotFull: Checks if the crossing is not.

Constructor:

- Initializes the contract with the validity period for the crossing status, maximum number of cars per lane, and the pre-locked time. Sets the initial state of the crossing to "LOCKED" and assigns the deployer as the infrastructure manager.

The API of the Smart Contract

Functions for Cars:

- requestCarPermission: Allows a car to request permission to cross if the crossing is free and not full. It updates the state to indicate the car is crossing and emits an event.
- releaseCarPermission: Allows a car to release its crossing permission once it has crossed. It updates the state and emits an event.

Functions for Trains:

- requestTrainCrossing: Allows an authorized train to request crossing permission. It updates the state to "PRE_LOCKED" or "LOCKED" depending on the number of cars currently crossing, and emits relevant events based on the crossing state and request status.
- releaseTrainCrossing: Allows an authorized train to release its crossing permission after crossing. It updates the state and emits an event.

Administrative Functions:

- authorizeTrain: Allows the infrastructure manager to authorize a train.
- deauthorizeTrain: Allows the infrastructure manager to deauthorize a train.
- updateFreeToCrossState: Allows the infrastructure manager to update the crossing state to "FREE_TO_CROSS" and reset the timer.

The Essential Implementation Details

Time Management in the Program (requestTrainCrossing function):

- The trainRequestTimes map stores the timestamp when each train requests crossing permission.
- The checkFreeToCrossTimer internal function checks if the current time exceeds the crossingValidity period since the last update, and if so, it transitions the crossing state to "LOCKED".

How the Train Crossing is Handled (lockCrossing function):

- The lockCrossing internal function transitions the crossing state to "PRE_LOCKED" if there are cars currently crossing.
- If no cars are crossing, the state transitions directly to "LOCKED".
- If it was the first request of the train, it sets the trainCrossingRequests map to true at the corresponding key address, and marks the trainRequestTimes map with the current time at the corresponding key address.
- If the difference between the first request and the current time exceeds the preLockedTime the train stops and the StopTrain event is emitted.

TrainCrossingTest Contract

The TrainCrossingTest contract is a test helper contract that inherits from the TrainCrossing contract. It is designed to expose the internal functions of the TrainCrossing contract for testing purposes. This allows us to test the internal logic and state transitions of the contract without modifying the original TrainCrossing contract.

The TrainCrossingTest contract inherits from the TrainCrossing contract. This means it inherits all the state variables, functions, and modifiers of the TrainCrossing contract.

The TrainCrossingTest contract defines two public functions, publicCheckFreeToCrossTimer and publicLockCrossing, which internally call the respective internal functions checkFreeToCrossTimer and lockCrossing from the TrainCrossing contract. These functions allow us to test the internal logic of the TrainCrossing contract.

Definition of Test Cases

The following are detailed explanations of each test case, including what they are testing and how they are implemented.

1. Test Case: Should set the right owner
   - Definition:
     - Verify that the infrastructure (owner) is set correctly upon contract deployment.

- Implementation:
    - This test fetches the infrastructure address from the contract and checks if it matches the deployer's address (owner.address).

2. Test Case: Should authorize a train and allow it to request crossing
    - Definition:
        - Verify that a train can be authorized and, once authorized, can request to cross.
    - Implementation:
        - The train (addr1.address) is authorized using authorizeTrain.
        - It checks that the train is indeed authorized.
        - Then, it simulates the train requesting crossing and verifies that the TrainCrossingRequest event is emitted.
3. Test Case: Should not allow unauthorized train to request crossing
    - Definition:
        - Ensure that an unauthorized train cannot request crossing and receives an appropriate error.
    - Implementation:
        - The test tries to make an unauthorized train (addr1) request crossing and expects a revert with the message "Not authorized train."
4. Test Case: Should allow a car to request and release crossing permission
    - Definition:
        - Test that a car can request and then release crossing permission.
    - Implementation:
        - First, the crossing state is updated to free to cross.
        - Then, a car (addr1) requests permission, and the CarCrossingPermissionGranted event is checked.
        - Finally, the car releases permission, and the CarCrossingPermissionReleased event is checked.
5. Test Case: Should not allow a car to request crossing when it's full
    - Definition:
        - Ensure that cars cannot request crossing when the crossing is full.
    - Implementation:
        - The test updates the crossing state to free to cross.
        - It then makes 5 different addresses request crossing permission.
        - Finally, it checks that a sixth address is reverted with the message "Crossing is full."
6. Test Case: Should allow authorized train to release crossing
    - Definition:
        - Verify that an authorized train can release its crossing request.
    - Implementation:
        - The train (addr1) is authorized and requests crossing.
        - The train then releases its crossing request, and the TrainCrossingPermissionReleased event is checked.
7. Test Case: Should update crossing state to free to cross
    - Definition:
        - Ensure that the crossing state can be updated to free to cross.
    - Implementation:

    - This test calls the updateFreeToCrossState function and checks that the crossingState is updated to FREE_TO_CROSS.
8. Test Case: Should lock crossing after validity period
    - Definition:
        - Verify that the crossing locks automatically after the validity period.
    - Implementation:
        - The crossing state is set to free to cross.
        - The test then increases the blockchain time by 601 seconds.
        - It tries to request crossing permission for a car and expects a revert with the message "Crossing is not free to cross," indicating that the crossing is locked.
9. Test Case: Should stop train if pre-locked time has passed
    - Definition:
        - Ensure that a train is stopped if the pre-locked time has passed.
    - Implementation:
        - The train (addr1) is authorized and requests crossing while another car is crossing.
        - The blockchain time is increased by 301 seconds.
        - The train then tries to request crossing again and the test checks that the StopTrain event is emitted, indicating that the train is stopped because the pre-locked time has passed.

10. Test Case: Should deauthorize a train and prevent it from requesting crossing

    - Definition:
        - Verify that a train cannot request a crossing after being deauthorized.
    - Implementation:
        - This test authorizes addr1
        - Deauthorizes addr1
        - Ensures it cannot request a crossing anymore.

11. Test Case: Should only allow infrastructure to update free to cross state

    - Definition:
        - Ensure that only the infrastructure can update the crossing state to FREE_TO_CROSS.
    - Implementation:
        - This test attempts to update the crossing state by a non-infrastructure address and expects it to fail, then successfully updates it with the infrastructure address.

12. Test Case: Should correctly handle checkFreeToCrossTimer behavior

    - Definition:
        - Ensure that the checkFreeToCrossTimer function works correctly, transitioning the state if necessary.
    - Implementation:
        - This test ensures that the checkFreeToCrossTimer function properly transitions the crossing state after the validity period has passed.

13. Test Case: Should lock crossing correctly when there are cars

- Definition:
    - Verify that the crossing transitions to PRE_LOCKED if there are cars crossing when locked.
- Implementation:
    - This test requests car crossing permission, then locks the crossing and checks that the state transitions to PRE_LOCKED.

14. Test Case: Should lock crossing correctly when there are no cars

- Definition:
    - Ensure that the crossing transitions to LOCKED if there are no cars crossing when locked.
- Implementation:
    - This test updates the crossing state, locks the crossing with no cars, and checks that the state transitions to LOCKED.

Running the Test Cases

To run the test cases, execute the following command in the terminal:

npx hardhat test

This command will compile the contracts and run the tests, providing output on the test results.

Deploy the Contract

To deploy the contract, run the deployment script with the following command:

npx hardhat run scripts/deploy.js --network rinkeby

This will deploy the contract to the specified network and print the contract address.