# **Exercise 1**

Consider an instance of the BC4C system modelled during the lectures. The following is the ledger containing the list of transactions issued in this blockchain:

|  |  |  |
| --- | --- | --- |
| Transaction id | Transaction | Originator |
| 0 | Genesis of BC4C | BC4C |
| 1 | CreateNewGame[LukaTrae1, Trae] | Luka |
| 2 | CreateNewGame[LukaRuss1, Russ] | Luka |
| 3 | CreateNewGame[StephLuka1, Luka] | Steph |
| 4 | ConfirmGameCreation[LukaRuss1] | Russ |
| 5 | ConfirmGameCreation[LukaTrae2] | Trae |
| 6 | CreateNewGame[StephKlay1, Klay] | Steph |
| 7 | Move[LukaTrae1, 1, …, …, false] | Luka |
| 8 | ConfirmGameCreation[StephKlay1] | Klay |
| 9 | Move[StephKlay1, 1, …, …, false] | Steph |
| 10 | ConfirmMove[LukaTrae1,1] | Trae |
| 11 | ConfirmMove[StephKlay1,1] | Klay |
| 12 | Move[StephKlay1, 2, …, …, false] | Steph |
| 13 | CreateNewGame[LukaRuss2, Russ] | Luka |

~~Based on the content of the ledger, answer the following questions:~~

1. ~~Is a game between Steph and Luka currently being played?~~
2. ~~How many moves have been played and confirmed in game ‘StephKlay1’?~~
3. ~~How many games are currently confirmed?~~
4. ~~Can you find any error in the ledger above?~~

Updated after discussion in class:

1. Can you find any error in the ledger above? Where?

**Transaction #5: LukaTrae2 should be LukaTrae1**

**Transaction #7 will be invalid as a result of transaction #5**

**Transaction #10 will be invalid as a result of transaction #5**

**Transaction #12: Steph is making a consecutive move which is not possible in chess**

Now consider two cases:   
(1) The list of transactions obtained by fixing the errors in the most reasonable way) that you have identified **(in transaction #12, we would change the originator to Klay)**  
(2) The list of transactions obtained by deleting the transactions in which you have identified an error.

Answer the following questions in the two cases (1) and (2):

1. Is a game between Steph and Luka currently being played?
2. No, (2) No
3. How many moves have been played and confirmed in game ‘StephKlay1’?
4. Only one, (2) Only one
5. How many games are currently confirmed?
6. 3, (2) 2

# **Exercise 2**

2.1) Consider a modified version of the BC4C blockchain where:

1. Moves do not have to be confirmed by the counterpart, i.e., the correctness of moves according to the rules of chess is checked when a transaction is validated by a node.
2. Because of (a), checkmate moves do not have to be confirmed as well.
3. The players can pause and resume a game at any time. Pausing/resuming is done by a node issuing a transaction and can be done only by a node only when it is their turn to move.

Provide a specification of this system, i.e., type of transactions and ~~transaction validation rules~~ blockchain protocol rules (ref. Slides 31 and 32 of W4-blockchain-agnostic.pptx)

|  |  |  |
| --- | --- | --- |
| Rule ID | Rule Specification | Notes |
| 1 | A game involves 2 players. A new game is proposed by one player and must be accepted by the opposite player to start. When a game is accepted, an id is generated for it according to standard rules, e.g. “AliceBob3” for the 3rd game involving Alice and Bob. |  |
| 2 | A game starts with the standard configuration of a chess board (number/types of pieces and positions) |  |
| 3 | The players of a game take turns in making moves. The player who proposed a game moves first. A move is proposed by one player and its correctness it terms of chess rules is checked when a transaction is validated by a node. A unique id for each move can be generated following simple rules. |  |
| 4 | A game terminates when a **“valid”** checkmate move, which is checked by a node, is proposed by one player. | Termination |

|  |  |  |
| --- | --- | --- |
| Rule ID | Specification | Notes |
| 1 | CreateNewGame [gameId, counterPlayer]: this type of transaction creates a new game between the originator of the transaction and another player, identified by the parameter counterPlayer. |  |
| 2 | ConfirmGameCreation [gameId]: this type of transaction confirms the creation of a game. It is issued by the counter player. |  |
| 3 | Move[gameId, moveId, piece, new position, isCheckMate]: This type of transaction specifies a move in a game. A move involves moving a piece to a new position on the board, and it is checked for correctness in terms of chess rules when it is evaluated by a node. If it passes the check, then it considered as a valid move and appears in the transaction ledger. A Boolean flag isCheckMate identifies whether the move leads to checkmate. | If isCheckMate is true and the transaction was validated by a node, then the game terminates. |

## **Exercise 3: True or False**

3.1) Every node in a blockchain network must have access to the code of all the smart contracts and must be able to execute them whenever needed (True)

3.2) The execution of smart contract must yield consistent updates of the ledger across all the nodes of a blockchain system (True)

3.3) The code of a smart contract cannot contain loops or any other conditional statement (False)

3.4) A smart contract cannot have internal variables, i.e., it can only use the input parameters and the variables describing the blockchain state. (False)

## **Exercise 4**

Consider an instance of the BC4C-CGM system for which a list of consecutive transactions recorded in the ledger is shown below:

|  |  |  |
| --- | --- | --- |
| Transaction id | Transaction | Originator |
| … | … | … |
| T789 | Move[CrisLeo21, 56, …, …., false] | Cris |
| T790 | CreateNewGame[SungLeo92, Leo] | Sung |
| T791 | ConfirmGameCreation[SungLeo92] | Leo |
| T792 | PayDeposit[LeoSung7] (SC call) | Leo |
| T793 | PayDeposit[LeoMin5] (SC call) | Min |
| T794 | ConfirmMove[CrisLeo21, 56] | Leo |
| T795 | Move[DiegoMin4, 45, …, …, true] | Min |
| T796 | PayDeposit[SungLeo92] (SC call) | Sung |
| T797 | Move[SungCris23, 78, …, …, true] | Sung |
| T798 | ConfirmMove[DiegoMin4,45] | Diego |
| T799 | ConfirmMove[SungCris23, 78] | Cris |
| T800 | CreateNewGame[MinCris4, Cris] | Min |
| … | … | … |

ASSUMPTIONS

1. deposits are fixed to 5 CGM tokens for all games.
2. the smart contract SC handling the deposits is a bit ‘smarter’ than the one introduced during the lecture. It has only one function “payDeposit[gameID]”. The originator of the transaction calling this function uses this call to pay the deposit for the game “gameID”.
3. the SC monitors the state of the blockchain and automatically knows when a new game is confirmed and, most importantly, when a new game terminates, i.e., after a checkmate move is confirmed by the counterpart. When a game terminates, the SC immediately pays back the deposits to the winner.)

[in other words, see slide 20 in the slides, assume that the SC does not have the functions register() and recordOutcome()]

STATE OF THE SYSTEM BEFORE T789 IS RECORDED IN THE LEDGER

CGM balance of users:

* Cris: 78
* Leo: 45
* Diego: 90
* Min: 45
* Sung: 89

ID of games currently played (i.e., confirmed and deposits paid):

* CrisLeo21
* DiegoMin4
* SungCris23

ID of games confirmed, but waiting for the deposit to be paid:

* CrisLeo22 (no deposit paid)
* LeoSung7 (only Leo paid the deposit)
* LeoMin5 (only Leo paid the deposit)

ID of games proposed, but not confirmed by counterpart:

* SungCris87

**QUESTIONS**

Q1) One transaction has one error, i.e., it registers something that has already happened, which one?

Answer: T792 (Leo has already paid the deposit)

Q2) Ignoring the transaction with an error that you have identified at Q1, what is the new state of this –blockchain after transaction T800?

**New state after transaction T800:**

**CGM balances of users:**

* Cris: 78
* Leo: 45
* Diego: 90
* Min: 50 (-5, +10)
* Sung: 94 (-5, +10)

ID of games currently played (i.e., confirmed and deposits paid):

* CrisLeo21
* LeoMin5

ID of games confirmed, but waiting for the deposit to be paid:

* CrisLeo22 (no deposit paid)
* LeoSung7 (only Leo paid the deposit)
* SungLeo92 (only Sung paid the deposit)

ID of games proposed, but not confirmed by counterpart:

* SungCris87
* MinCris4