Ultimate Tic Tac Toe Protocol – UTTT/1.0

1. Introduction
    1.1. Purpose

This protocol aims at enabling communication between two devices playing the Ultimate Tic Tac Toe game. It is divided into 3 main parts : The first part defines the start message for a first connection, the second describes all the methods in more details and finally the last part describes the various errors.

1.2. Timeout

During the PLAY phase (section 3.1), we will set a 10-second timeout: if the delay between sending PLAY and sending ACK (or between receiving PLAY and receiving ACK) exceeds 10 seconds, the connection is off and FATAL_ERROR is sent (section 4.3).
There is also a 10-second timeout during the CONNECTION (section 2) and WIN (section 3.2) phase

1.3. Hash

To send the state of play (section 3.1), we will use the hash version SHA-3 224 and the module *hashlib* on python and initiate an hash table with *hashlib.sha3_224()* and then use *update(state of play in byte format)* to hash the state of play
How to represent the state of play ? The hash should be processed like this :

Board Game	Matrix	Hash computing
OXO ... ... **XOX ... ...** **XOX ... ...** **O.. ... ...** **... .X. ...** **..X ... ...** **... ... X..** **... .X. .O.** **... ... ...**	[[[0,1,0, [.,.,., [.,.,., 1,0,1, .,.,., .,.,., 1,0,1], .,.,.], .,.,.]], [[0,.,., [.,.,., [.,.,., .,.,., .,1,., .,.,., .,.,1], .,.,.], .,.,.]], [[.,.,., [.,.,., [1,.,., .,.,., .,1,., .,0,., .,.,.], .,.,.], .,.,.]]]	*import hashlib* *m = hashlib.sha3_224()* *m.update(b"010101101-.....-...../0.....1-....1....-...../.....-....1....-1...0....")* *print(f"hash : {m.hexdigest()}")* gives : hash : 2890e6332b4c10e4fdd6ebae42684e9ac0a33161250dcec288187889 where « 1 » : 1st player (GUEST) « 0 » : 2nd player (HOST) « . » : empty square « / » is the line parser « - » is the square parser

Note: If your matrix isn't represented like this, you must keep this formatting for hash computing !!

2. UTTT Start Message

- Host player starts listening until he gets a TCP connection. Player Guest sends the TCP connection knowing the IP address of player 1.
  UTTT/1.0 CONNECTION [Pseudo]\n
- Host responds:
  UTTT/1.0 CONNECTION [Pseudo]\n
- Then **GUEST** starts to play

3. Method Definitions

3.1. PLAY

The PLAY method is used to retrieve the position at which the device sending the request has played on the board. The board is represented by a first number for the general game and a number for the small game:

GAME FORMAT:

Big game:	Small game:
0 1 2	0 1 2
3 4 5	3 4 5
6 7 8	6 7 8

It also retrieves the hash of the current state of play on the sender's board **before** this new movement, as we explained before. This maintains the level of play on both sides.

Form of the request: **UTTT/1.0 PLAY [Position] [State of play]\n**
[Position] : n°big_slot n°small_slot (without space)

After sending his move, the device should wait for a confirmation of the other device. In this confirmation should be added the state of play of the board **after** the new move sent by the first device.

Form of the new state : **UTTT/1.0 NEW_STATE [New state of play]\n**

In response to this message, the device checks whether this new game state matches its own. If so, it sends an acknowledgement and the game continues, otherwise it sends the STATE_PLAY error (section 4.1).

Form of the acknowledgement : **UTTT/1.0 ACK\n**

3.2. WIN

The WIN method is used to declare to the other device that the game is over on his board. This method is accompanied by HOST if the player host has won, GUEST if the player guest has won or nothing if there is a draw. WIN is sent by the player who **received the last** PLAY (or the **ACK**) other sends END if correct.

In response, the device receives the END method (section 3.3) or FATAL_ERROR (section 4.3) in the event of an error (the only possible is cheating).

Form of the request: **UTTT/1.0 WIN [Player who wins]\n**

3.3. END

The END method is used in response to the WIN method (section 3.2) if the end of the game is the same for both sides.

This method immediately stops communication between the two devices: if a player leaves the game and closes the connection that player sends an END message.

Form of the request: **UTTT/1.0 END\n**

4. Error Code Definitions

4.1. 404 STATE_PLAY

The state of play is checked every time the device receives one. If the device detects that it's not in the same state of play, it sends the STATE_PLAY error code to notify this. To handle this problem, it also adds the previous state of play before its move and the play it did. Then the game continues as usual.

Form of the error: **UTTT/1.0 404 STATE_PLAY [Position] [previous state of play]\n**

4.2. 405 BAD_REQUEST

The request could not be understood by the device due to malformed syntax. The other device should not repeat the request without modifications. (e.g : position already taken)

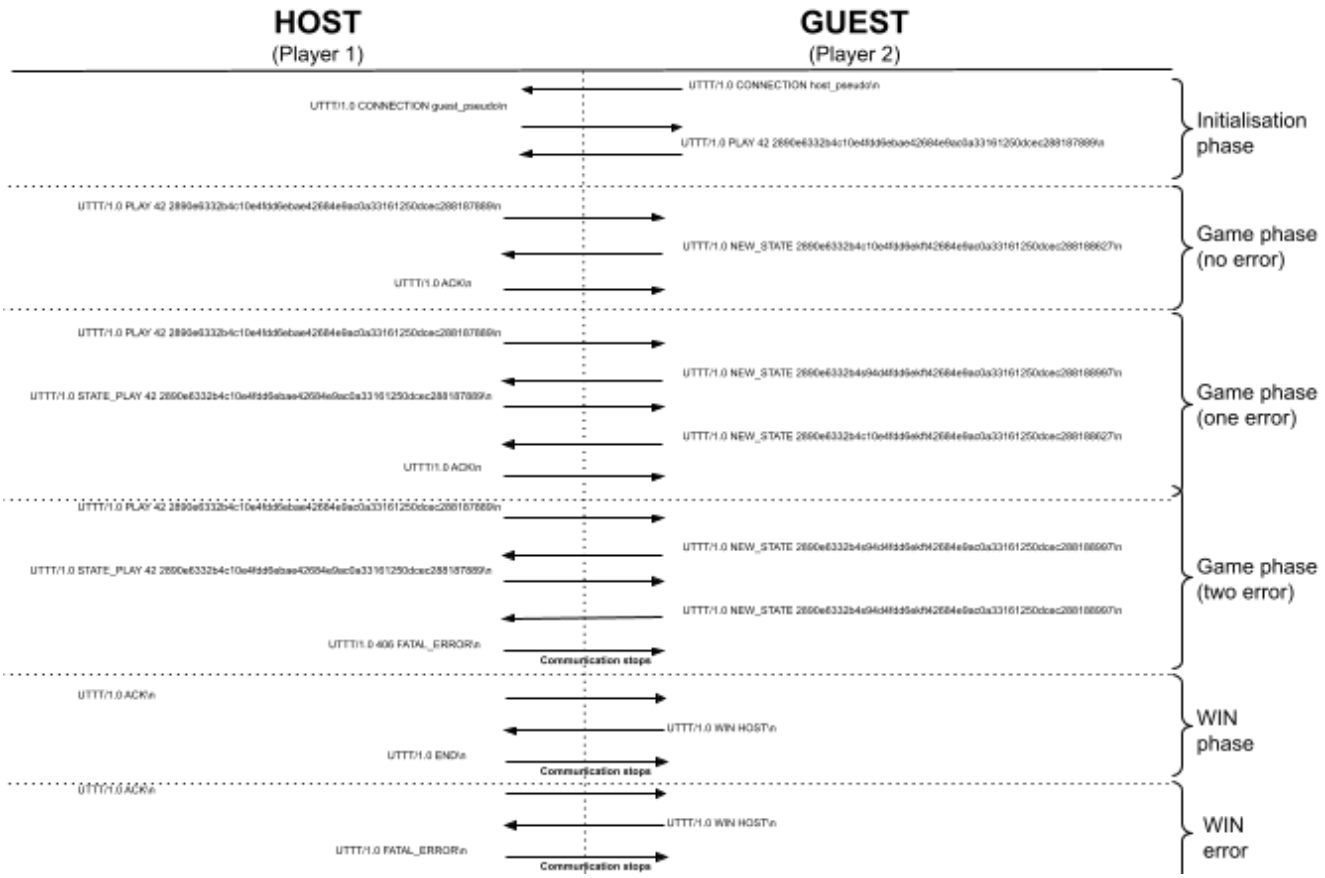
Form of the error: **UTTT/1.0 405 BAD_REQUEST\n**

4.3. 406 FATAL_ERROR

If too much error occurs in the communication (set to 2 consecutive errors), the device sends to the other device the FATAL_ERROR error code that means that the game and the communication is closed. It can happen when a device is trying to cheat or if there's a bug in the game.

Form of the error: **UTTT/1.0 406 FATAL_ERROR\n**

5. Communication diagram



6. Authors

   Network Working Group:

7. Amélioration axes

   - encrypting data
   - reconnection message and procedure