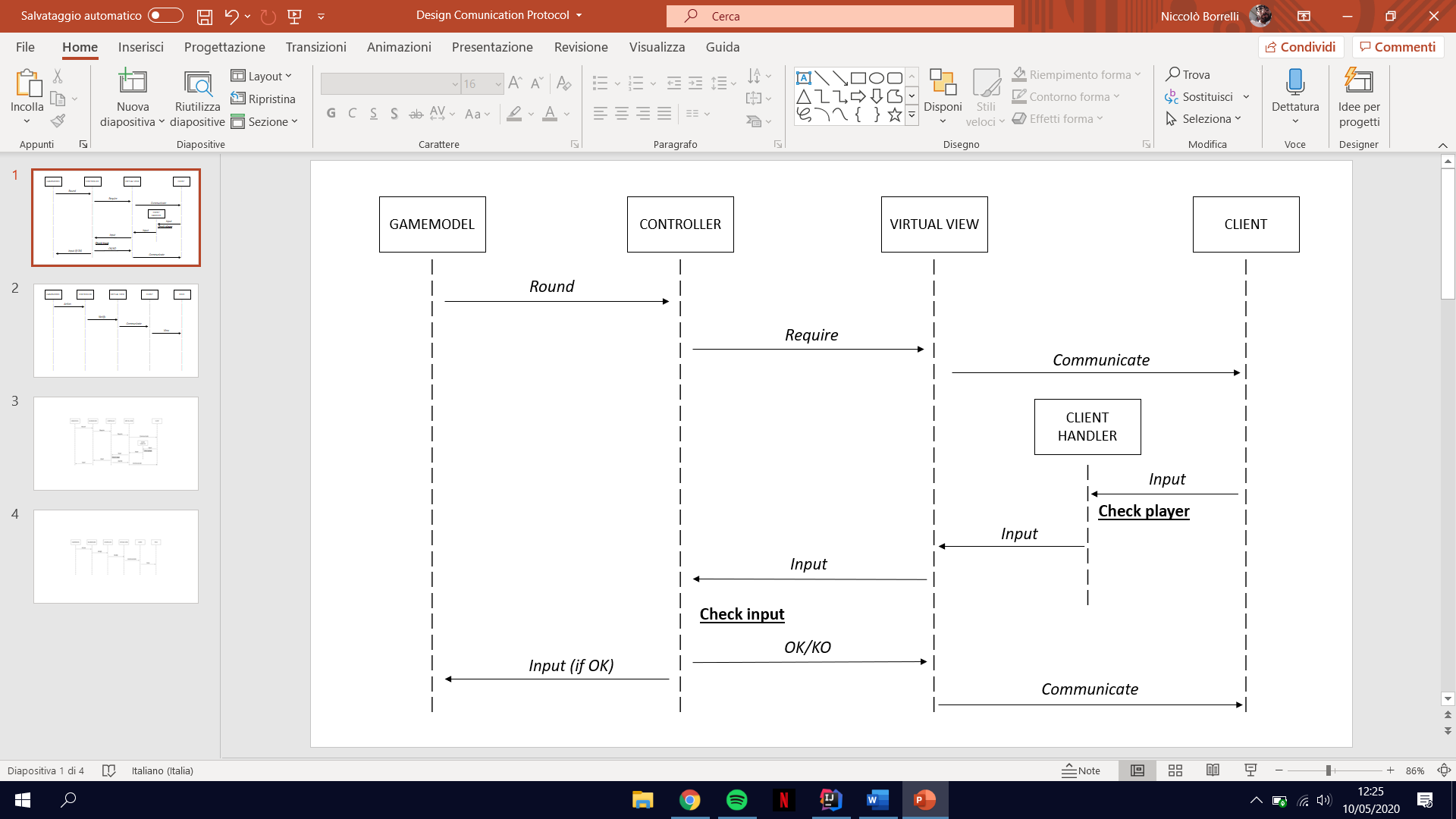
**COMUNICATION PROTOCOL CODES**

The server and client speak each other via a communication protocol, every message has an XML tag <code> that manage the client behavior: the code is a number between 0 and 10:

* 0 -> the client print on screen the element in the XML tag <body>
* 1 -> the server asks the client an integer input
* 2 -> the server asks the client a string input
* 3 -> the server asks the client two integer input
* 4 -> the server notify the client the game is over
* 5 -> the server notify the client that the input received is valid
* 6 -> the server sends the client the possible space to move or to build for, respectively, the movement phase and the building phase
* 7 -> the server notify the client a worker movement so that the view get an update
* 8 -> the server notify the client a worker build so that the view get an update
* 9 -> the server notify the client a worker remove so that the view get an update
* 10 -> the server notify the client a worker insert so that the view get an update

**ROUND PHASE and INITIAL CONDITIONS**



GAMEMODEL:  
Round is one of the method that require an input from the client. These methods are: workerToMove, getMovementPosition, getBuildingPosition, all the methods inside move/build decorators and the initial god assignation and name selection

* workerToMove(Player activePlayer, List<Space>[] possibleMovements)
* getMovementPosition(Worker movingWorker, List <Space> possibleMovements)
* getBuildingPosition(Worker movingWorker ,List <Space> possibleBuilding)
* godAssignation(List <String> availableGods)

CONTROLLER:  
Require is one between requiredInt, requiredSpace, requiredPosition, requiredString or requiredName

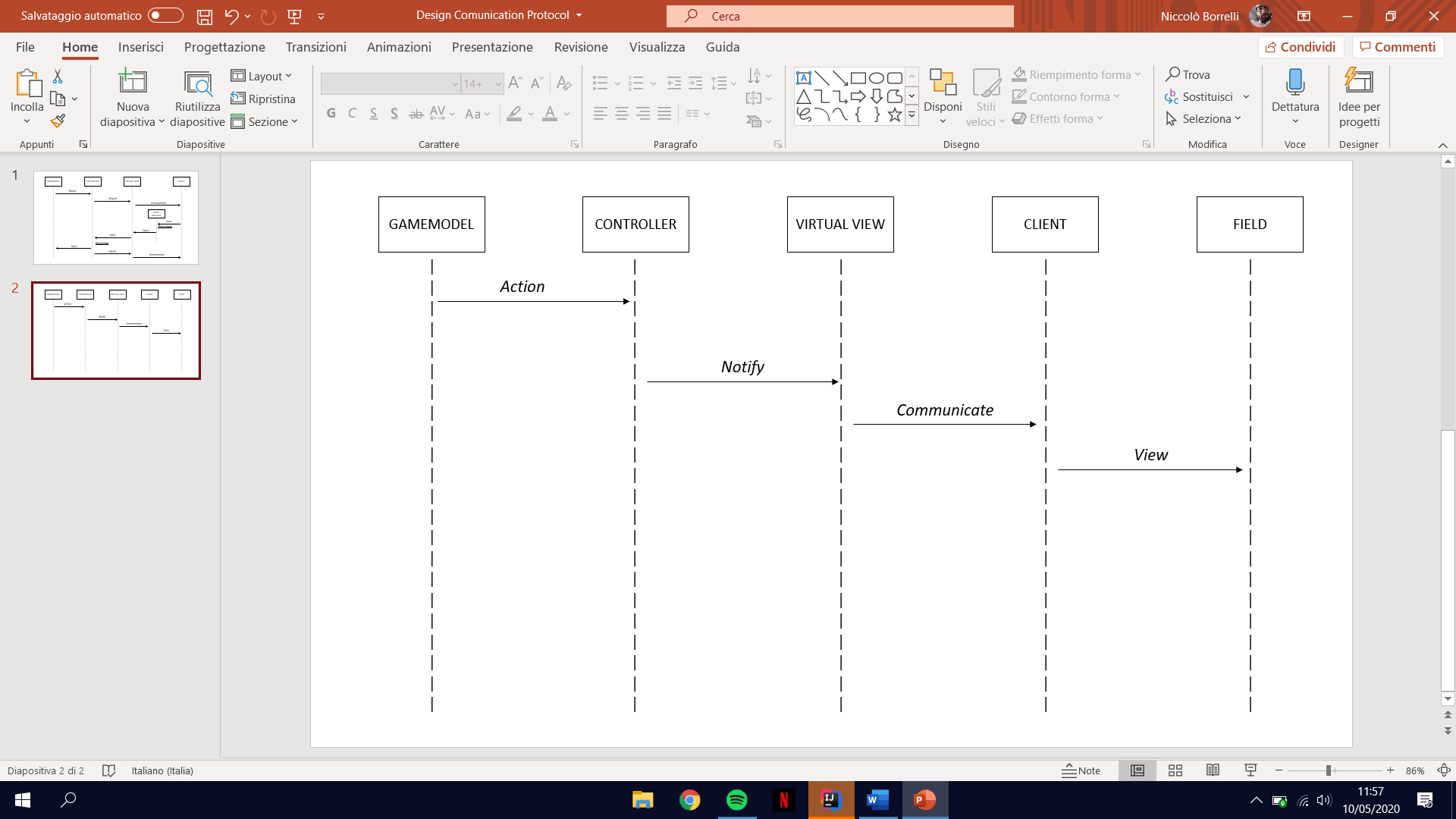
* requiredInt(Socket sc, String message, List<Integer> available)
* requiredSpace(Socket sc, String message, List<Space> available)
* requiredString(Socket sc, String message, List<String> available)
* requiredPosition(Socket sc, String message, int[][] matrix )
* requiredName(Socket sc, String message, List<String> available)

VIRTUAL VIEW:  
Communicate method send to the client a string in XML with all the info, it keeps sending the same XML to the client unless the controller notify

* communicate(DataOutputStream dataOutputStream, String message, int typeMessage)

CLIENT HANDLER:  
It checks if the player is the same player the server sent the requires, if so it sends the client input to the virtual that sends it to the controller, otherwise it deletes it  
  
CONTROLLER:  
It checks if the input is valid, if so it sends the input back to the islandboard and notifies the virtual view, otherwise he waits for another input.

**INTERFACE UPDATE**



GAMEMODEL:  
Action is one of the method that modify the field. These methods are: move, build, checkHasWon, deletePlayer or setupWorker

* move(Worker worker, Space finishSpace, IslandBoard islandBoard)
* build(Worker worker, Space buildSpace, IslandBoard islandBoard)
* deletePlayer(Player player)
* setupWorkers(Player activePlayer)
* checkHasWon(Worker worker, int startLevel, IslandBoard islandBoard)
* checkHasWon(List<Player> players)

CONTROLLER:  
Notify is one between updateMovement, updateBuilding, updateWin, updateSomeoneLose or updateSetUp

* updateSetUp(Space space, String color)
* updateLose(Socket sc, Space spaceWorker1, Space spaceWorker2)
* updateWin(Socket sc)
* updateBuilding(Space buildSpace)
* updateMovement(Space startPlace, Space finishPlace, String color)

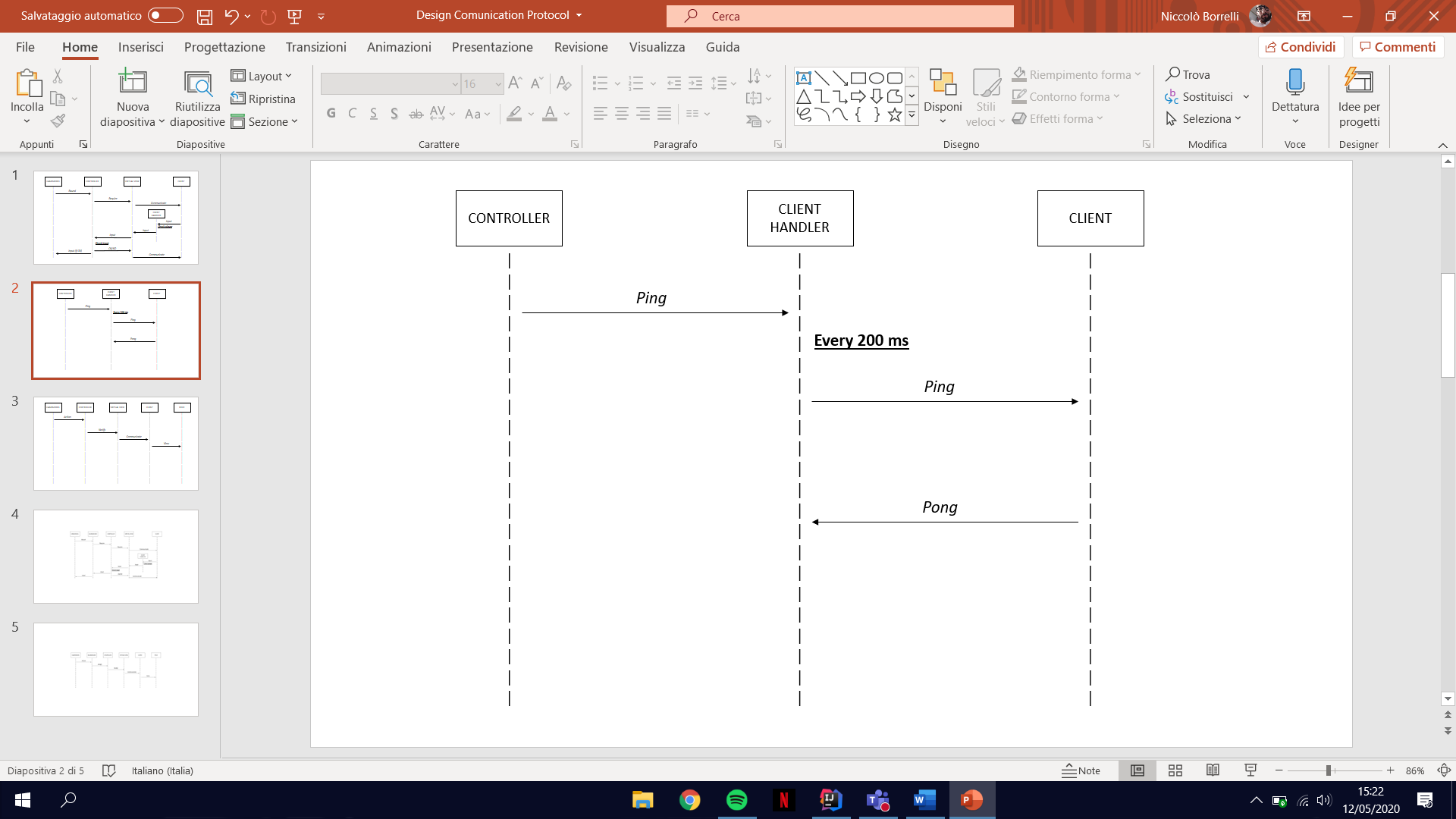
VIRTUAL VIEW:  
Communicate method send to the client a string in XML with all the info

* communicate(DataOutputStream dataOutputStream, String message, int typeMessage)

CLIENT:  
View is one between viewMove, viewBuild, viewRemoveWorker or viewSetup

* viewBuild(int row, int column, int level, int hasDome)
* viewMove(int oldRow, int oldColumn, int newRow, int newColumn, String color)
* viewSetup(int row, int column, String color)
* viewRemoveWorker(int row, int column)

**PING**



CONTROLLER:  
Ping is called after the creation of the client handler

* clientHandler.ping( )

CLIENT HANDLER:  
While the game is not finished, every 200 ms it calls the method isReachable on its socket IP address, if the method returns to the controller and the game is not finished yet the controller closes all socket in the game and the game is over

* inetAddress.isReachable(2000)