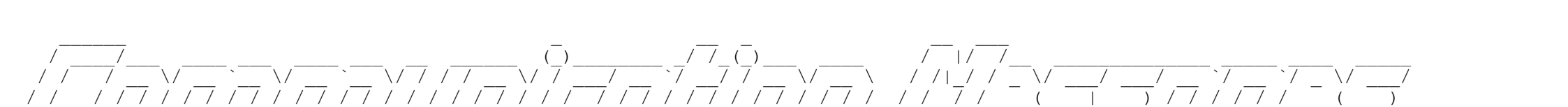

--- MATCH CREATION

1

{ -addPlayer: {"action":"addPlayer","data":{"playerNickname":"Bill","lobbySize":2}}

communicates to the server the intention to join the game,
giving the server the (unique) nickname and the the lobby size

-ServerResponse:
{"action":"addPlayerResponse","data":{"playerNickname":"Bill","lobbySize":2,"lobbyState":true,"validNick":true,"fullLobby":false}}

-the server checks if it is necessary to open a new lobby or if the chosen lobby already exists (lobby Size indicates the type of lobby desired)
-if it exists and it's the player's choice, check if the nickname is unique in the whole server
then sends the response

-playerNickname: unique in the whole server
-lobbySize: type of lobby chosen
-validNick: result of the check made by the server, on the validity of the nickname
-fullLobby: if false, it indicates that the server is occupied by a game and the player cannot be entered in a lobby
}

2

{ -setPickedCards: {"action":"setPickedCards","data":{"playerNickname":"Bill"}} BROADCAST MESSAGE

God Player whose nickname will be in the message, choose 3 or 2 cards according to the lobby size,
the other players will receive the message will unlock the thread
but will not choose the cards because it will not be their nickname

{ -getDeck: {"action":"getDeck"}
All players call this function which requires the deck to the server

-ServerResponse: {"action":"getDeckResponse","data":{"deck":[DivinityCard.class]}}
Server sends to the players, the deck from which he can choose the cards (skimmed according to the lobby size)
}

-ClientResponse: {"action":"setPickedCards","data":{"cards":["ATHENA","APOLLO"]}}

the client's task (God Player) will be to select the cards from the complete deck based on the number of players allowed in the lobby
(the server can recheck)
}

3

{   -setPlayerCard: {"action":"setPlayerCard","data":{"cards":["ATHENA","APOLLO"]}}

The player chooses a card from the deck created by the "god" player (that will be the last)

-ClientResponse: {"action":"setPlayerCard","data":{"playerNickname":"Bill","card":"APOLLO"}}

the server receives the chosen card, binds it to the player together with a color,
removes the card from the list of cards to be sent to the next player
}

4

{   -setWorkersPosition: {"action":"setWorkersPosition","data":{"workersID":[2,3]}}
the server sent to the player his workersID

4a   {   -getPlayers: {"action":"getPlayers"}

the client asks the server for the PlayersList to be sent to him   (nick+color+card)

-ServerResponse: {"action":"getPlayersResponse","data":{"players":[PlayerInterface,PlayerInterface]}}
forces all clients to update the Players in match

-PlayerInterface = {"playerNickname":"Bill","color":"BLUE","card":"APOLLO"}
}

4b   {   -getBattlefield: {"action":"getBattlefield"}

the client asks the server for the battlefield to be sent to him

-ServerResponse: {"action":"getBattlefieldResponse","data":{"cellMatrix":CellInterface[][]}}
}

-ClientResponse:

{"action":"setWorkersPosition","data":{"playerNickname":"Bill","workersPosition":[{"workerID":0,"x":4,"y":4},{"workerID":1,"x":4,"y
":3}]}}

the player based on the battlefield received, places workers in an allowed position and send to the server this message
}

5

{   -battlefieldUpdate: {"action":"battlefieldUpdate","data":{"cellMatrix":CellInterface[][]}} BROADCAST MESSAGE

forces all clients to update the battlefield, as soon as it is changed
}

---   START MATCH

1

{   -actualPlayer: {"action":"actualPlayer","data":{"playerNickname":"Bill"}} BROADCAST MESSAGE

the server notifies clients who is the current player who can take the turn
}

2

{ -setStartTurn: {"action":"setStartTurn","data":{"playerNickname":"Bill","basicTurn":true}}

the player notifies the server what type of turn he wants to perform (true = no effects, false = with card effects)

-ServerResponse: {"action":"setStartTurnResponse","data":{"playerNickname":"Bill","basicTurn":true,"currentStep":"MOVE"}}

the server responds by confirming the choice and indicating the first step that the player can take
}

3

{ -selectWorker: {"action":"selectWorker","data":{"playerNickname":"Bill","x":4,"y":4}}

having a valid step, the player selects one of his workers, returning to the server its position on the battlefield

-ServerResponse: {"action":"workerViewUpdate","data":{"workerView":boolean[][]}}

-the server will check if the workerView of any worker is null (impossible move) if it were the player lost and is eliminated
-otherwise the server responds by sending the workerView of the selected player (even if all false)
(it will be up to the client to force you to choose another worker)
}

4 (OPTIONAL)
{ -playStep: {"action":"playStep","data":{"x":3,"y":4}}

the player chooses to perform the step, indicating the battlefield row and column for the step (the step can be move, build or remove)
-the player in particular steps can choose whether to skip them (not playing them)

-ServerResponse: {"action":"playStepResponse","data":{"x":4,"y":4,"nextStep":"END"}}

the server responds with the action made and with the next step
}

5 (OPTIONAL)
{ -skipStep: {"action":"skipStep"}

the player chooses to skip the current step and move on to the next
-

-ServerResponse: {"action":"skipStepResponse","data":{"currentStep":"END"}}

the server responds with the next step, that the player will have to perform
}

6

{ -battlefieldUpdate: {"action":"battlefieldUpdate","data":{"cellMatrix":CellInterface[][]}} BROADCAST MESSAGE

forces all clients to update the battlefield, as soon as it is changed
}

FINISH MATCH

1

{   -notifyWinner/Loser: (Json to be defined)

-the server notifies the player or everyone, who has lost and who has won,
the game must end and the client and server side will reset everything

-N.B: this messages can arrive at any time, moreover,
a player can lose but the game does not end for everyone (the client is not aware of it, the server does)
}