Protocollo di rete

Basic Packet
{“type”:””, “content”:{PACCHETTO}}

tutti i pacchetti sottostanti saranno contenuti nel “content”

Server:

ACK / N-ACK
{ “ACK”:{ “errorMSG”:(0-N)} }
-0 -> ok
-1 -> NotEnoughPlayers
-2 -> PlayerListFull
-3 -> NickameAlreadyTaken
-4 -> IllegalSwap
-5 -> IllegalStorageExtraction
-6 -> IllegalProduction
-7 -> LeaderActivationPrerequisitUssoddisfied
-(1-N)-> errori vari mappati in una classe apposita}

DisplayCardUpdate
{“displayCardUpdate”: {“cardType” : type ,
“level” : int ,
“obtainedMaterials” : [lista di risorse] ,
“rawMat”:[listarisorse]}
}
//invia al client le carte in testa ai mazzetti da mostrare

PapalSpaceWaning
{“PapalSpaceWaning”: {“position” : int ,
“outcome” : bool}
}
//avvisa i client che qualcuno ha raggiunto un papal space e lo
//informa se ha ottenuto i punti o no

UpdatePosition
{“UpdatePosition”: {“position” : int ,
“player” : int }

}
//invia ai giocatori la posizione corrente di uno specifico giocatore

DrawLeaders
{“DrawLeaders”: {[4 leader]}
//il server invia al client 4 leader da selezionare

client:

Login
{“login”:{“nickname”:string}}

PendingCosts //risorse da togliere
{“pendingCosts”:[lista di risorse]}
//contiene un elenco di risorse da estrarre (se ricevuto dal client esso chiederà all’utente dove vuole estrarre le singole risorse)

PendingGains //da aggiungere allo storage
{“pendingGain”:{“resource”:[lista di risorse],”whiteinfo--TODO---}
//Similarmente alla PendingCost è una lista di risorse che invece voglio inserire nello storage (quindi chederò all’utente dove inserirli)

Market
{“marketExtraction”:{“direction”:”row/col”,”pos”:int}}
-Se ricevuto dal server esegue un estrazione dal market
-Se ricevuto dal client aggiorna la gui del market

Turn
{“turnTypeRequest”:{“type”:int}}
//Indica il tipo di turno (1,2,3)

ChoseLeaders
{“ChoseLeaders”: {[2 leader]}
//il client invia al server i 2 leader da selezionati

BasicProduction
{“basicProduction”:{“rawMat”:[listarisorse],
”obtainedMat”:[lista risorse]}}
//Indica che voglio effettuare una produzione base, contiene le risorse che si vuole spendere e quelle che si vuole ottenere

Production
{“production”:{“pos”:int}
//indica che si vuole produrre con la carta in posizione “pos”
//se pos è maggiore uguale di 3 NACK

BonusProduction
{“bonusProduction”:{“pos”:int,”ResoucreType”:String}}
//Se non ci sono bonus ritorna NACK altrimenti esegue la produzione bonus

SwapDeposit
{“swap”:{“pos1”:int,”pos2”:int}}
//Esegue lo swap fra 2 depositi (permette all’utente di riordinare le risorse a piacimento)

EndTurn
{“endturn”:{“endstring” : string}}
//avvisa il server che il player ha terminato il turno

StorageMassExtraction
{“extractions”:
[
{“type”:chest/storage,“pos”:int,”quantity”:int},
{“type”:chest/storage,“pos”:int,”quantity”:int},
]}
//Una volta che **PendingCost//PendingGain** viene eseguito con successo viene inviato un pacchetto di questo tipo che contiene tutte le informazioni necessarie al server per eseguire l’inserimento/estrazione delle risorse (se fallice il server invia un NACK con un codice adeguato e il client rieseguirà la richiesa all’utente).

ActivateLeader
{“LeaderAction”:{“pos”:int,”activate”:bool}}
// Se false scarta il leader in pos
// Se true attiva il leader in pos (NACK se non soddisco i requisiti)
PapalToken
{“PapalToken”:{“pos”:int,”activate”:bool}}
//Se false lo scarta dalla gui (toglie il segnalino)
//Se true lo scopre nella gui (ruota il segnalino/rimuove la X)

IncrementPoition
{“incrementPos”:{“pos”:int}}
//cambia la posizione del player di “pos”

BuyCard
{“BuyCard”:{“x”:int,”y”:int,”position”:int}}
//Invia una richiesta di compra carta
//Se ACK il client riceve un messaggio “pendingCost”
//Se NACK il client riceve errore (vedi esempio)

DiscardResource
{“discardResource”:{“quantity”:int}}
//Scarta una risorsa

UpdateWhiteLeaderInfo
//aggiorna le info del minimodel per estrazione palline bianche

MarketResult
{“MarketResult”:{“ResourceBall”:[],”WhiteBalls”:{“quantity”:int}]}

ESEMPI E SIMULAZIONI:

LOGIN
c:ping (per startare la connessione)
s:ACK
c:invia JSON di tipo login contenente nickname
s:verifica disponibilità di spazio e unicità nickname
    *c: se ACK STOP
    *c: se NACK richiedi un nuovo nickname, comunica errore (es “spazio pieno..”)

RICHIESTA TURNO
s: invia JSON di tipo “” chiedendo che tipo di turno vuole
c: chiede all’utente che turno vuole e invio JSON di tipo turnTypeRequest
s: attende un comando di quel tipo (darà NACK per ogni comando illegale per quel tipo di turno)

SWAP DEPOSIT

c: richiesta pacchetto JSON di tipo swap (posizione 1, posizione 2)
s: prova ad eseguire lo swap (se lancia eccezione NACK)
    *c:se ACK aggiorna la gui/minimodel
    *c:se NACK cominico errore al player

Market

c:client richiede estrazione da market(JSON marketExtraction)
s: server esegue l’estrazione e genera un pacchetto di tipo “marketResult”
c: chiede all’utente dove estrarre le risorse (ed eventualmente come gestire le palline bianche)
genera un pacchetto JSON di tipo “StorageMassInsertion”
s:elabora il pacchetto e prova ad inserire le risorse (se c’è eccezzione NACK)
    *c:se Ack stop
    *c:se Nack riesegui

BuyCard/Production/BasicProd/BonusProd:

-C: tramite view “compra” una carta (invia JSON buyCard/production/basicP...)
-S: controlla risorse utente,posizione dashboard etc.

*se soddisfa requisiti
S:-aggiunge il costo a PENDINGCOST
-aggiunge la carta nel model e invia un JSON pending cost al client*
C:chiede all’utente dove prendere le risorse
(invia JSON StorageMassExtraction)
S: prova ad estrarre le risorse, se “eccezione” invia indietro un NACK
C:Se Nack ripete/Se ack STOP

se non soddisfa invia un NACK
S: Invia NACK
C: riceve nack e Avvisa utente dell’errore

Scelta leader

-S: iniziato il game invia a tutti e 4 i client un pacchetto di tipo “drawLeaders”
-C: tramite view “sceglie” i lead che desidera ed invia un json “choseLeaders”
-S: Setta al player model i leader scelti

CONTENUTO MINIMODEL

L’idea è quella di tenere salvato nel minimodel (aka model ridotto presente in ogni client) i dati che andranno poi visualizzati dall’utente, dunque:

-Dashboard:
    -storage
    -chest
    -carte produzione acquistate dal giocatore
-Carte in primo piano dei mazzetti da cui si acquista
-Percorso della fede
-le sue info
    -nickname
    -score
    -position

```
//RES
activate(marketHelper)
{
        market.addRes(res);
}
//WHITE
activate(marketHelper)
{
        market.whiteAdd(res);
}
market.reset()
{
        this.res = null;
        white    = 0;
}
```