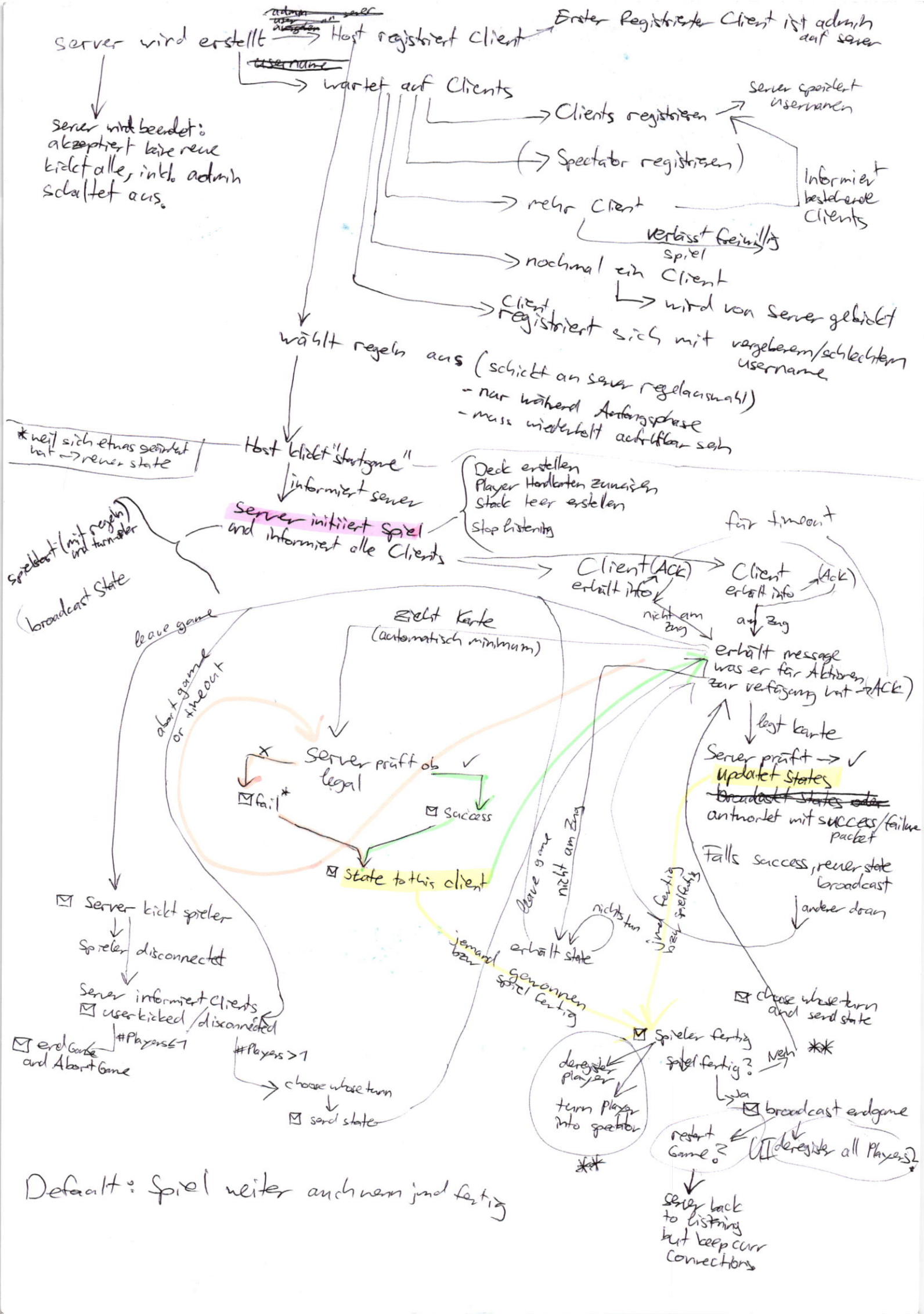
Server wird erstellt
Host registriert Client
Erster Registrierter Client ist admin auf server
wartet auf Clients
Server wird beendet: akzeptiert keine neue, kickt alle, inkl. admin schaltet aus.
Clients registrieren
Server speichert Usernamen
(Spectator registrieren)
mehr Client
Informiert bestehende Clients
verlässt freiwillig Spiel
nochmal ein Client
wird von Server gekickt
Client registriert sich mit vorgegebenem/schlechtem username
wählt regeln aus (schickt an server regelauswahl)
- nur während Anfangsphase
- muss wiederholt abrufbar sein
Host klickt "startgame"
* weil sich etwas geändert hat → neuer state
informiert server
Server initiiert Spiel und informiert alle Clients
Deck erstellen
Player Handkarten zumischen
Stack leer erstellen
Stop listening
spielbrett (mit regeln und turnorder)
broadcast State
Client (ACK) erhält info
Client erhält info (ACK)
für timeout
nicht am Zug
am Zug
erhält message was er für Aktionen zur verfügung hat (→ACK)
leave game
abort game or timeout
zieht karte (automatisch minimum)
legt karte
Server prüft ob legal
fail*
success
Server prüft → ✓
updatet states
~~broadcast states oder~~
antwortet mit success/failure packet
state to this client
Falls success, neuer state broadcast
andere dran
Server kickt spieler
Spieler disconnected
Server informiert Clients
user kicked/disconnected
#Players=1
end Game and Abort Game
#Players>1
choose whose turn
send state
leave game
nicht am Zug
erhält state
nichts tun
jmd fertig
jemand gewonnen spiel fertig
choose whose turn and send state
Spieler fertig
spiel fertig?
Nein **
ja
broadcast endgame
deregister player
turn player into spectator
***
restart Game?
deregister all Players?
Server back to listening but keep curr connections
Default: Spiel weiter auch wenn jmd fertig

Messages

Anmeldephase

Cli → register → Serv
- username
- role "player"/"spectator"
- messagegroup "registration"

Cli ← Serv
new User registered
messagegroup "registration"

Cli ← RegisterResponseSuccess ← Serv
- success
- role
- username
- messagegroup "registration"

// in Case of multiple, differing requests, return which was chosen if success

Cli ← RegisterResponseFail ← Serv
- reason: "invalid username"
  "no more Players accepting"
  "not unique"
- messagegroup "registration"

Cli ← Timeout ← Serv
- Server not listening for new connections anymore, or is down.

Cli → leave / "UnregisterRequest" → Serv
- messagegroup "registration"

Cli ← UnregisterPlayer ← Serv
- messagegroup "registration"
- kicked /dc /...
- username
- role
- toastmessage
- list of players

HostCli → kick → Serv
- msggroup "registration"
- username

Cli -

HostCli → specifyRules → Serv
- messagegroup "rules"
- ruleset: JSON
- ~~turn order~~

// in between: HostCli -> Serv startgame

Cli ← ~~getRules~~ ← Serv
- messagegroup "rules"
- ruleset: JSON

// Client does not "know" rules → doesn't respond with Error if rule unknown

Cli → ackRules → Serv
- messagegroup "rules"

// for noticing Client timeout

HostCli → startGame → Serv
- messagegroup "startgame"

Cli ← initGame ← Serv
- turn-order
- messagegroup "startgame"
- all player names
- ruleset

Spielphase

Cli ← sendState — Serv { messagegroup "stateInfo", playerstate, globalstate } → States brauchen fromJSON(stateInfo) und toJSON

Cli — getState → Serv { messagegroup "stateInfo" }

Cli — playCard → Serv { messagegroup "playCard", ~~playerName~~ //possibly not needed?, Card }

Cli ← playCardResponse — Serv { messagegroup "playCard", success: "true"/"false" } // wenn false, dann aus Grund dass etwas anders als beim letzten broadcast-State. ↳ resend state.

Cli — drawCard → Serv { messagegroup "draw" }

Cli ← drawCardResponse — Serv { messagegroup "draw", ~~Anzahl~~ ~~Card-Draw: "2"~~, Cards-drawn: {Card1, Card2} } // Cards-drawn for UI animation // on failure num-draw == 0 and cardsdrawn empty

(Cli ← sendState — Serv again)

Cli ← playerFinished — Serv { messagegroup "endGame", username } → finished player knows as well ↳ can "spectate"

Cli ← endGame — Serv { messagegroup "endGame", Ranking } // abhängig von Regeln. Default: Ordered List ↳ pro Regel gibt Parser anderen MessageClass

(Cli — roleSwitchRequest → Serv { messagegroup "roleSwitch", intoRole only "spectator", not back }) not done

~~Cli~~ ~~Serv~~ ← roleSwitch — Serv { messagegroup "roleSwitch", intoRole "spectator" } // ~~store internally different in server to remember who was initial spectator.~~ // use this to let finished players watch.

Connectivity issues

keep passing keep-alive signal back-and-forth. Maybe allow timeout/maxPing to be set in Settings.

Cli — ~~sendToast~~ toast → Serv { messagegroup "toast", ~~target-user~~, message }

~~Cli~~ ← showToast — Serv { same again. + from + style }

State:

global:
- #Cards of each player
- Stack (no guarantees), maybe not all cards contained
- whose turn
- → evt hier implementieren

player:
- Handkarten
- min #karten ~~anzei~~ aufziehen (wenn er aufziehen würde)
- possible actions

internal:
- admin (EinzClientHandler, username)

Aktionen:
- spielbare Karten (based on attributes)
- (also admin) leave game
- Draw Cards (minimum possible)
- (admin) kick Player
- ((später evt zwischendurch) andere aktion)
- ((admin) transfer server (before leaving))

Action hat JSON Parameter

PossibleAction enthält alle möglichen Parameter?
e.g. alle spielbaren Karten

Represented as string
stored mappin somehow (see later)

```
{
  ~~"identifiers": ["...", "alpha", ...]~~,
  "actions": {
    "alpha": {        },
    "...": {          }
  }        ↑
           └ e.g. which cards are playable
}
```

Card: (only send Id of card)
- id
- image is local

Ruleset: only send Id of rules in Array. Reihenfolge irrelevant.
Regeln sollen voneinander unabhängig sein.

Logic initialize should be re-callable to restart game

Fail → Msg dass Karte nicht gelegt. Wenn playcard < newState < fail, dann fail immer noch anzeigen.
Meldung.
Nach fail schickt server neuen state. (Nutzt nicht, schadet nicht)