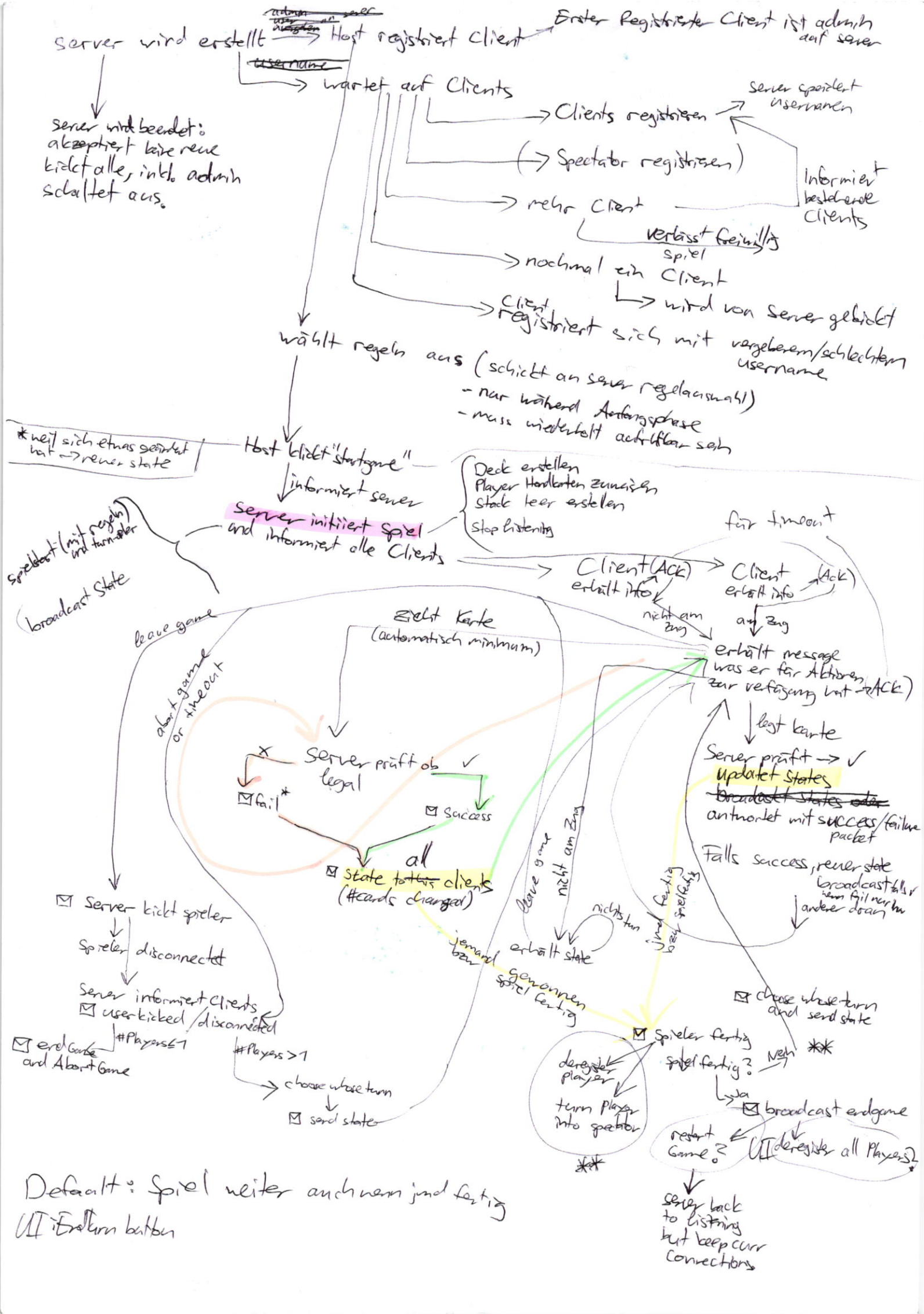

Server wird erstellt
Host registriert Client
Erster Registrierter Client ist admin auf server
wartet auf Clients
Clients registrieren
Server speichert Usernamen
(Spectator registrieren)
mehr Client
informiert bestehende Clients
verlässt freiwillig Spiel
nochmal ein Client
wird von Server gekickt
Client registriert sich mit vorgegebenem/schlechtem username
Server wird beendet: akzeptiert keine neue kickt alle, inkl. admin schaltet aus.
wählt regeln aus (schickt an server regelauswahl)
- nur während Anfangsphase
- muss wiederholt abrufbar sein
Host klickt "startgame"
informiert server
Server initiiert Spiel und informiert alle Clients
Deck erstellen
Player Handkarten zumischen
Stack leer erstellen
Stop listening
*weil sich etwas geändert hat → neuer state
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)
zieht Karte (automatisch minimum)
leave game
abort game or timeout
legt karte
Server prüft ob legal
fail*
success
Server prüft → ✓
updatet states
broadcastet states oder
antwortet mit success/failure packet
all
state [illegible] clients (#cards changed)
Falls success, neuer state broadcasten, wenn fail nur an anderen denken
leave game
nicht am Zug
nichts tun
jemand fertig
jemand gewonnen spiel fertig
erhält state
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
choose whose turn and send state
Spieler fertig
spiel fertig?
Nein
**
deregister player
turn player into spectator
***
Ja
broadcast endgame
restart Game?
UI deregister all Players?
send back to listening but keep curr connections
Default: Spiel weiter auch wenn jmd fertig
UI: Endturn button

Messages

Anmeldephase

Cli → Serv (register): { username, role "player"/"spectator", messagegroup "registration" }

Cli ← Serv (RegisterResponseSuccess): { success, role, username, messagegroup "registration" } // in Case of multiple, differing requests, return which was chosen if success

Cli ← Serv (RegisterResponseFail): { 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 ← Serv: new User registered, messagegroup "registration"

Cli → Serv ("UnregisterRequest"): { messagegroup "registration" }

Cli ← Serv (UnregisterPlayer): { messagegroup "registration", kicked name /dc/..., username, role, toast message, [list of players] }

HostCli → Serv (kick): { msggroup "registration", username }

Cli .

HostCli → Serv (specifyRules): { messagegroup "rules", ruleset: JSON-Obj, ~~turn order~~ }

// in between: HostCli → Serv startgame

Cli ← Serv (sendRules): { messagegroup "rules", ruleset: JSON-Obj }

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

Cli → Serv (ackRule): { messagegroup "rules" } // for noticing Client timeout

HostCli → Serv (startGame): { messagegroup "startgame" }

Cli ← Serv (initGame): { 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", ~~dev-rem~~ // 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", ~~num-draw : "2"~~, cards-drawn: {Card?, Card?} } // 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", into role only "spectator", not back }) not done

~~Cli~~ ← roleswitch — Serv { messagegroup "roleswitch", into-role "spectator" } // ~~store internally different in server to remember~~ role 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 → Serv { messagegroup "toast", ~~target-user~~, message })

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

State:

global: #Cards of each player & names of players (don't send spectators)
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
~~#Karten de anderen Player~~

internal:

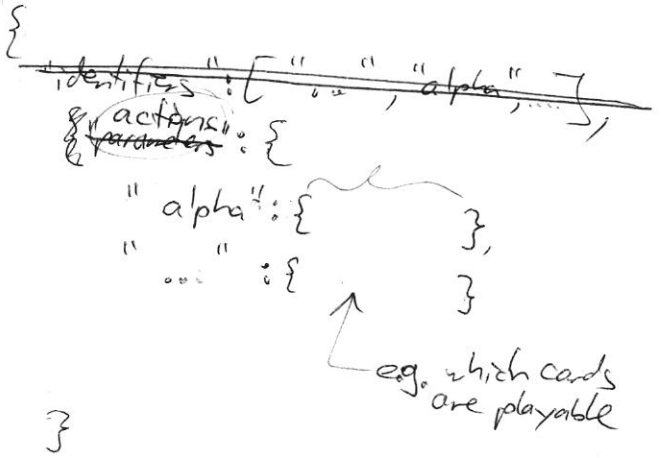
admin (EinzClientHandler, username)

Aktionen:

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

Action hat SSONParameter
PossibleAction enthält alle möglichen Parameter?
e.g. alle spielbaren Karten

Represented as string
stored mapping somehow (see later)



}

}

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)

ready: HashSet<String>

startgame(Players, Spectators)

Idee: wenn 4 oder so

Multiple Servers

Anonym

Eric:
server stopp unexpected by client
kickPlayer -> removePlayer
ServerName: "server"
restartgame -> create new serverFunctions
playerLeft/kicked: handle connections and call ~~playerLeft()~~ removePlayer()

Fabian:
- Messages
- Locks Clarific: which functions read/write
- Class1: GlobalState
  Class2: PlayerState
- List of actions
- Card ID ?
- Send message: game has started (-> rename startgame)
- Karte Zielen etc. kontrollieren
- Play until 1 player left
- move finished players to spectators (remember he was a player)
- after player gets removed: check #players!