Server-Protokoll (v0.9)
Send + Receive

Patrick Schäfer

patrick.schaefer@hu-berlin.de

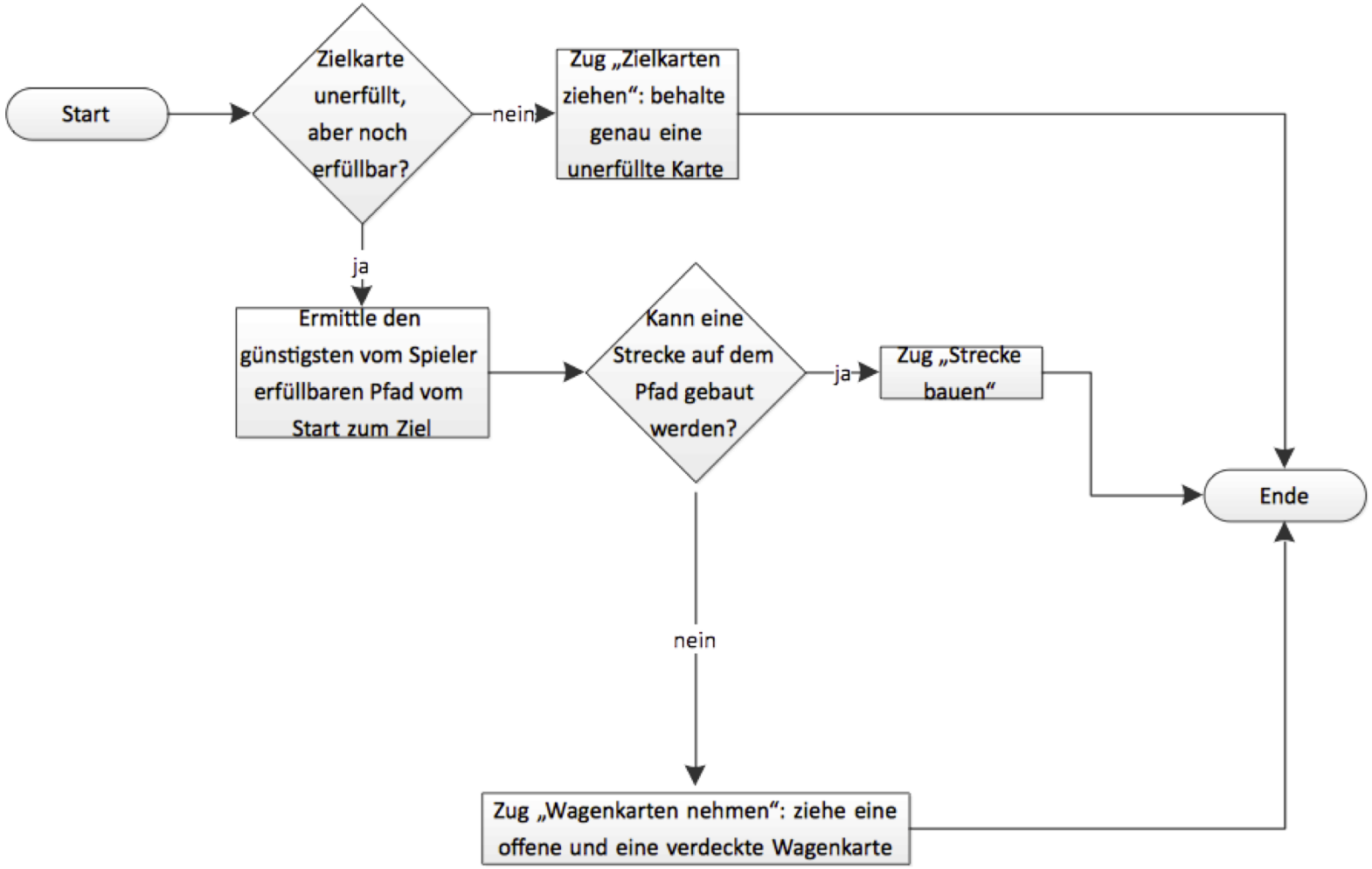
Agenda

- Today
  - New User-Story „Einfacher Computerspieler“
  - Talk: Server-Protokoll: Send+Receive

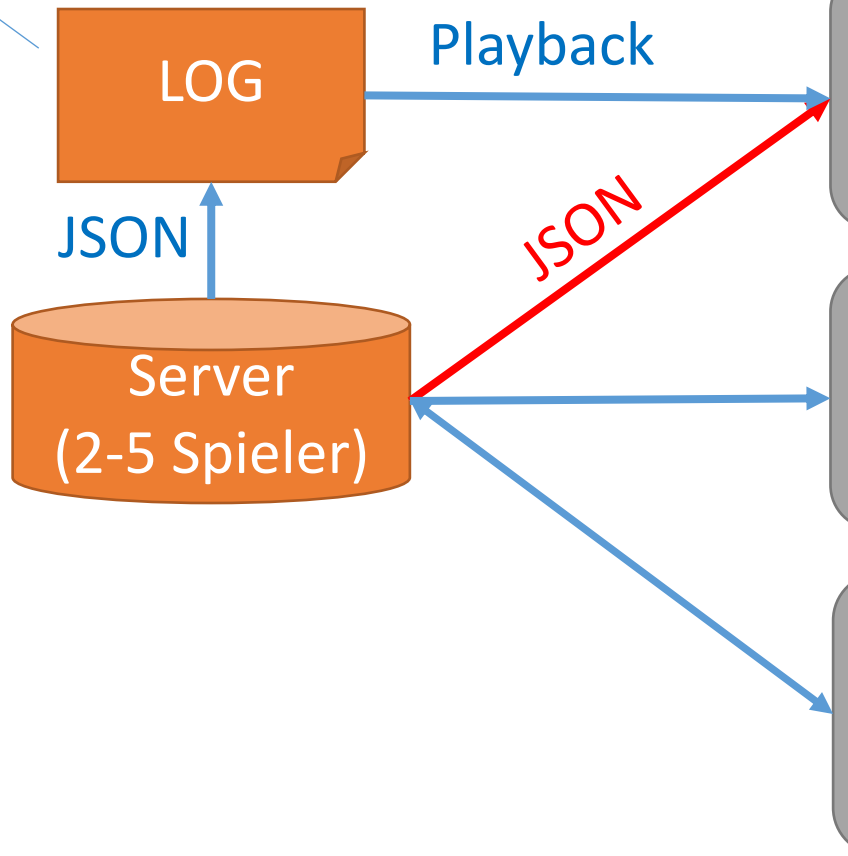
- This week:
  - technical refinement for the new user story
  - finalize tasks in your sprint backlog (incl. tests, code review)

- Next Monday, 13:30
  - Sprint #1 Review Meeting; bring a laptop & presentable prototype
  - Sprint #2 Sprint planning; kickoff; present your Sprint Backlog

User-Story: Einfacher KI-Spieler



Anforderungskatalog



Start des Servers

- Benötigt Java JVM
- Starten des Servers:
  JAVA -JAR SERVER.JAR <NR_OF_PLAYERS>
- Der Server öffnet einen Socket auf Port 8080

Client-Server Kommunikation

- Die Kommunikation der Clients mit dem Spielserver erfolgt über den Socket

- Ein Client erstellt eine Verbindung zum Spieleserver über eine Socket mit gegebener Kombination IP-Adresse/Ports
  - 127.0.0.1, 8080 ist die Default-Kombination, falls Server und Client auf dem gleichen Rechner laufen

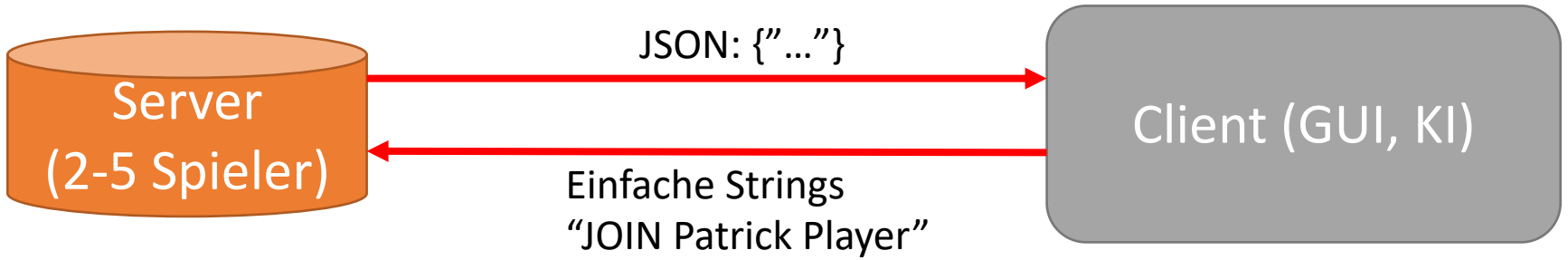
Server-Nachrichten: JSON-Protokoll

- Server Nachrichten werden ...
  - mittels JSON serialisiert
  - über einen Socket verschickt und in eine Log-Datei geschrieben
  - müssen von euch deserialisert werden

- Der Server sendet zwei Arten von Nachrichten:
  - **Info**: Züge anderer Spieler
  - **Request**: Benachrichtig Spieler-Client, welcher Zug von ihm erwartet wird

Client-Nachrichten: einfache Strings

- Zu jeder Zeit könnt ihr die Befehle der API verwenden, um den Server anzuweisen, eine Aktionen für euch durchzuführen
- Die Befehle werden dabei als einfache Strings übertragen: „JOIN PATRICK PLAYER“
- Jede Nachricht vom Client an den Server **muss mit einem Zeilenumbruch ’\n’ enden**



Übersicht der Client-Befehle

	Befehl (<Parameter>)
Verbindungsaufbau	**Join** <Name> <ClientType>
Wagenkarten „offen“ ziehen	**DrawPassengerCars** <false> < PassengerCarColor >
Wagenkarten „verdeckt“ ziehen	**DrawPassengerCars** <true>
Zielkarten ziehen	**DrawDestinationTickets**
Zielkarten behalten	**ClaimDestinationTickets** <boolean> <boolean> <boolean>
Eine Strecke nutzen	**ClaimRoute** < Destination > < Destination > < PassengerCarColor >
Aktuelles Spielbrett	**BoardState**
Alle Strecken des Spielbretts ausgeben	**ListAllRoutes**

Beispiele:

- ClaimDestinationTickets True false false
- DrawPassengerCars false Red
- DrawPassengerCars true
- ClaimRoute SaintLouis LittleRock Yellow

DEMO

Erinnerung: JSON-Format

- Beispiele für Server-Nachrichten:
  - Request
    {"type":"Request","player":"Blue","turnType":"**Join**"}
  - Info
    {"type":"Info","player":"Blue",
    "turnType":"**Join**","success":true, "playerName":"**Patrick**"}

- Über Info-Nachrichten werden Änderungen am Spielbrett kommuniziert
- Request benötigen wir für die KI und den Single-Player-Modus

Erinnerung: JSON-Format

- Eine JSON-Nachricht vom Server enthält die folgenden Felder:

Type	: Info oder Request
turnType	: Antwort auf euren Zug
success	: Die Anfrage war erfolgreich (nur wenn Type = Info)
player	: Der Spieler von dem der Zug gemacht wurde
errorCode	: Ein Fehler-code (nur wenn success = false)

Verbindungsaufbau

- Es gibt zwei Arten von Clients: OBSERVER oder PLAYER
- Die Art des Clients wird beim Join-Aufruf angegeben
- Ein Observer bekommt niemals Requests, aber alle Nachrichten die den Spielfortschritt protokollieren

SERVER>
{"TYPE":"REQUEST","PLAYER":"BLUE",
"TURNTYPE":"**JOIN**"}

CLIENT>
JOIN PATRICK PLAYER

SERVER>
{"TYPE":"INFO","PLAYER":"BLUE","SUCCESS":TRUE,
"TURNTYPE":"**JOIN**","PLAYERNAME":"**PATRICK**"}

// Der Server wartet auf den Join-Befehl
Server>{"type":"Request", "turnType":"**Join**"}

// Ein Spieler verbindet sich als Patrick
Client> **Join Patrick Player**

Server>
{"type":"Info","player":"Red","success":true,"turnType":"Join","clientType":"Player","playerName":"**Patrick**"}
Server>
{"type":"Request","player":"Red","turnType":
"ClaimDestinationTickets","activeDestinationTickets": [{"city1":"Chicago","city2":"NewOrleans","points":7},
{"city1":"Montreal","city2":"NewOrleans","points":13},{"city1":"NewYork","city2":"Atlanta","points":6}]}

// Der Spieler behält 2 Karten
Client> **ClaimDestinationTickets False True True**

Server>
{"type":"Info","player":" Red ","success":true,"turnType":"ClaimDestinationTickets",
"drawnCards":[{"city1":"Montreal","city2":"NewOrleans","points":13},
{"city1":"NewYork","city2":"Atlanta","points":6}]}
{"type":"Request","player":" Red ","turnType":"Turn"}

Server

- Der Server v0.9 kann von der Veranstaltungswebseite heruntergeladen werden
- Wir werden eine C#-Bibliothek bereitstellen, die die JSON-Nachrichten mittels JSON.NET deserialisiert
- Die Daten müssen von dort in euer Modell übertragen werden
- Zum Format und den vorhanden Feldern siehe auch aktuelles Word-Dokument auf der Veranstaltungswebseite

Offene Fragen

- Was passiert im Fehlerfall?
  - Unendlich viele Versuche?
  - Default-Zug?