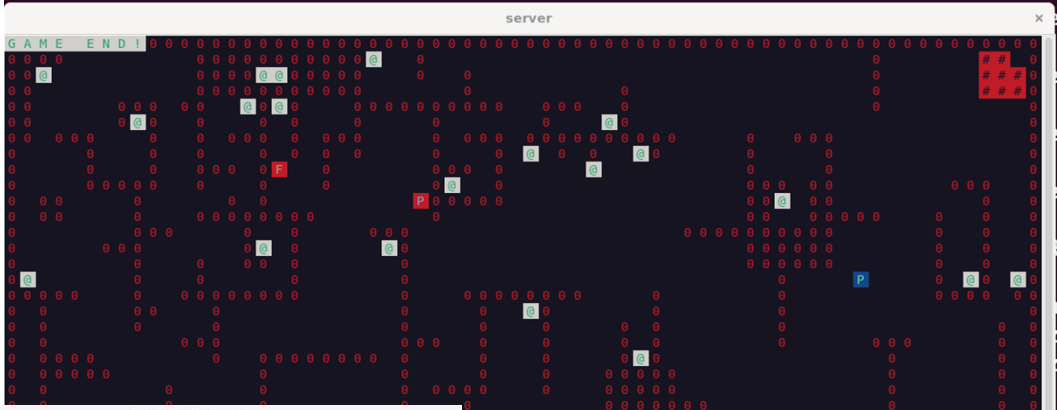
jps_maze

Philipp Grüber, Johan Bücker,
Simon Demuth,

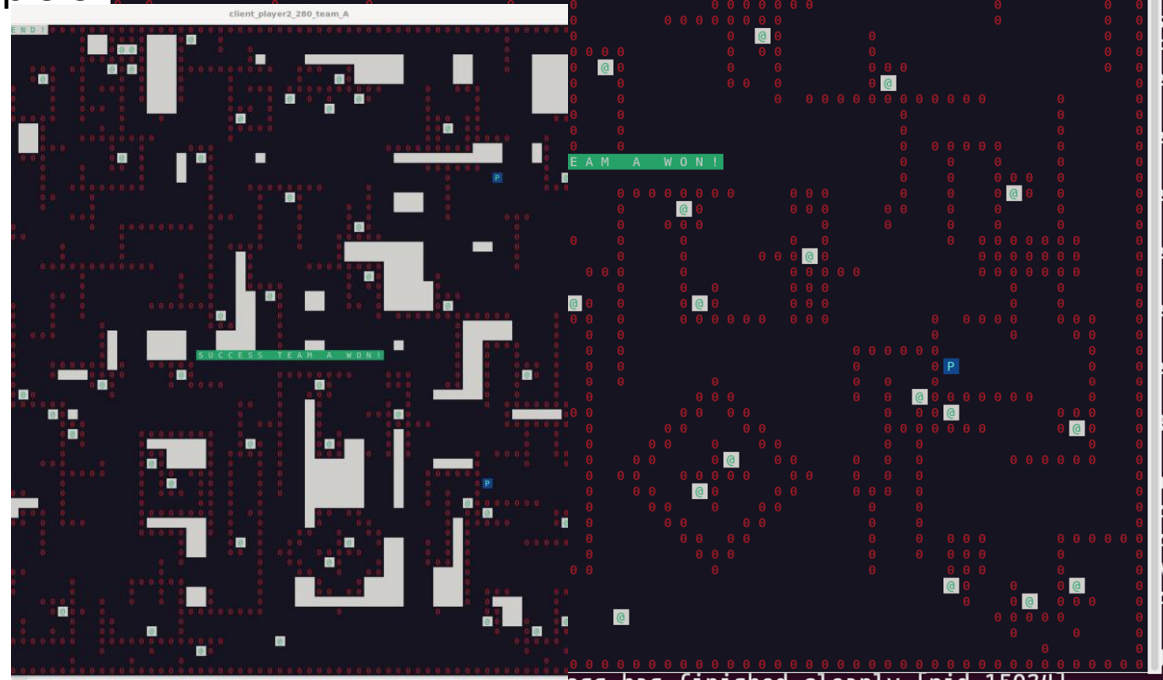
Konzept

- 1 Server, min. 2 Clients (=Spieler)
- Capture The Flag:
  Spieler müssen von ihrer Base aus die Flagge aus der anderen Base holen und wieder in ihre Base zurückbringen
- Hindernisse:
  - Wände
  - Portale (Spieler wird zu zufälligem Portal teleportiert)
- Dabei “mappen” Spieler ihre Umgebung
- Board wird zur Laufzeit aus .csv geladen
- Alles in Git Repo und komplett mit “ros2 launch” startbar
- Rollenverteilung:
  - Simon D.: Grafische Darstellung *(Node.js - nicht eingebunden)*
  - Johan B.: ROS & UDP Kommunikation
  - Philipp G.: Spielengine

Server



Spieler



Projektstruktur

- Zwei ROS Packete:
  - jps_maze_msgs: Enthält die .msg und .srv Definitionen für jps_maze
  - jps_maze: Enthält den Code für die eigentliche Applikation

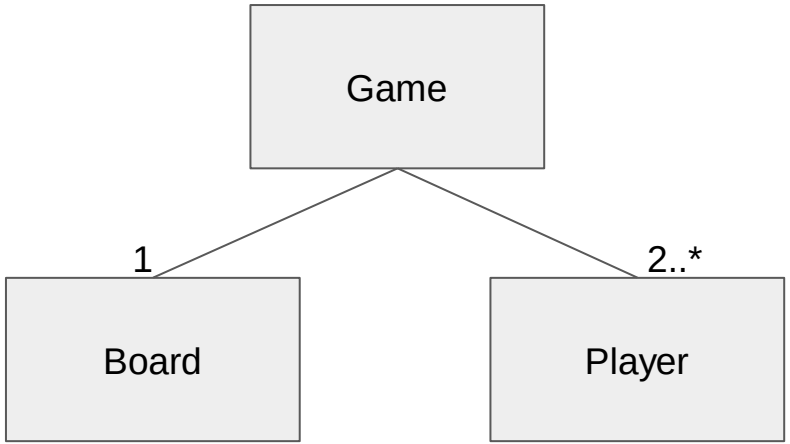
- jps_maze:
  - config: Enthält paramter.yaml zur definition der Parameter
  - jps_maze_client: Code für die Client Node
  - jps_maze_curses: Code für die Curses applikation
  - jps_maze_server: Code für die Server node
  - launch: Enthält die launch file zum starten des games
  - lib: Enthält die libraries game und visualizer, genutzt von beiden nodes
  - nodeenv: Nodejs code
  - res: Enthält die .csv files für die Boardlayouts

Launch-Prozess

- ros2 launch jps_maze jps_maze_launch.py
- Nutzt Pythons argparse um aus einer angeforderten Nutzerangabe die Launchkonfiguration zu bestimmen.
- Erstellt eine s.g. LaunchDescription, die zum einen die zu startende Node enthält, aber auch Prozessstart-eventhandler, die automatisch das gewünschte Visualisierungsfrontend starten.
- Über die parameters.yaml werden der node die relevanten Parameter zur Verfügung gestellt. Mit diesen bestimmt diese z. B. das zu verwendende Board.

Spielengine

Übersicht der (wichtigsten) Klassen:



7 Dateien mit **874** Zeilen C++ Source Code

Übersicht der Structs & Enums:

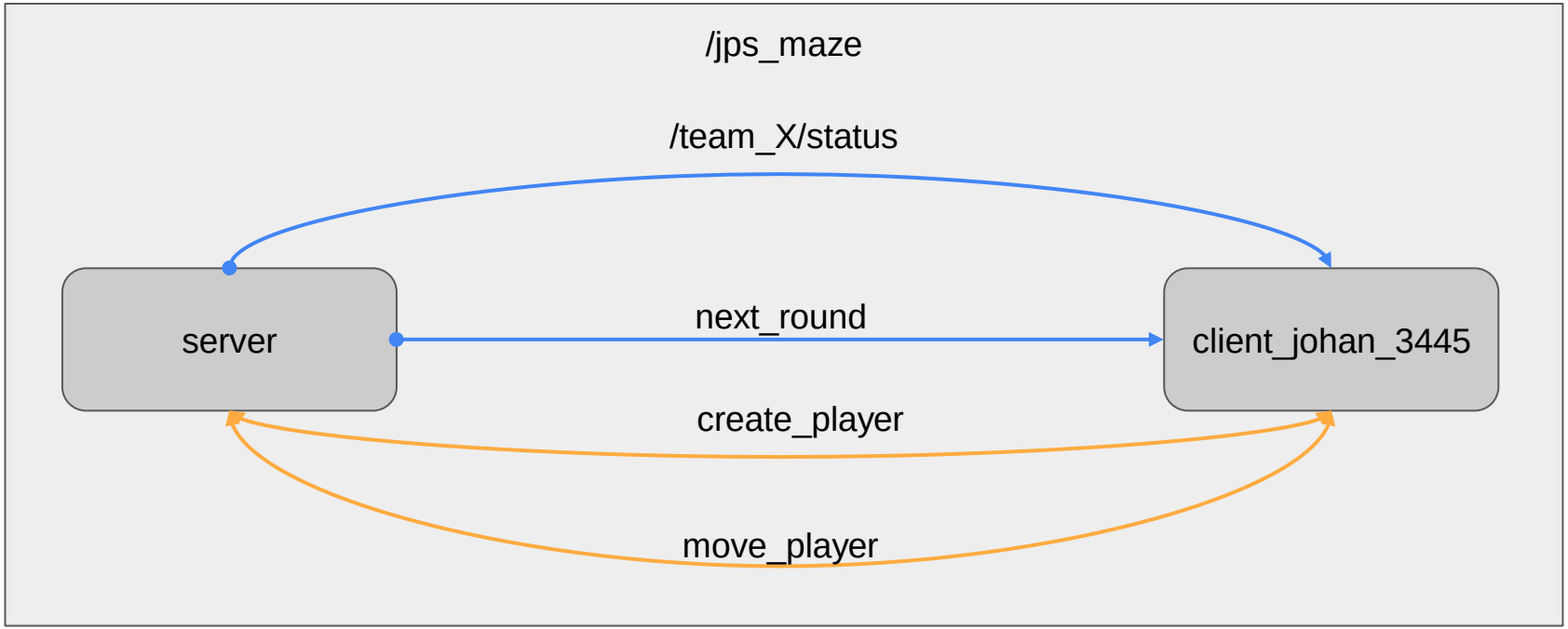
Game	game_flag_state_t	*enum*
	game_state_t	*enum*
Board	game_block_t	*struct*
	game_block_type_t	*enum*
Player	team_t	*enum*
	direction_t	*enum*

Spieler Logik

1. Geht die bisherige Richtung weiter
2. Wenn nicht möglich:
    a. Wenn vorherige Richtung UP / DOWN, dann zufällig LEFT oder RIGHT
    b. Wenn vorherige Richtung LEFT / RIGHT, dann zufällig UP oder DOWN
3. Wenn das nicht geht: Entgegengesetzte Richtung zu 2.
4. Wenn das nicht geht:
    a. Wenn Team A: Rechts drehen und durchprobieren
    b. Wenn Team B: Links drehen und durchprobieren
5. Sonst Exception

ROS Aufbau

Services | Publisher



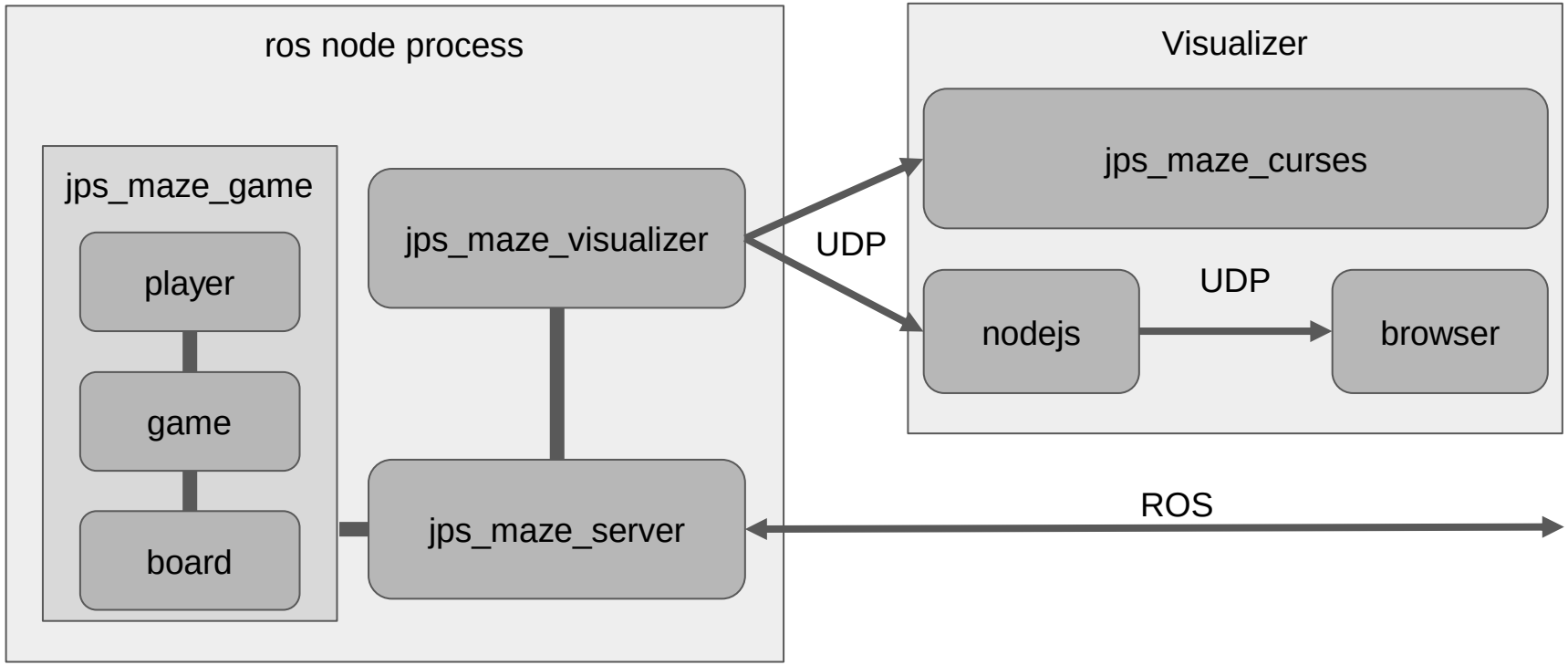
2 Dateien mit 294 Zeilen C++ Source Code

2 Dateien mit 366 Zeilen C++ Source Code

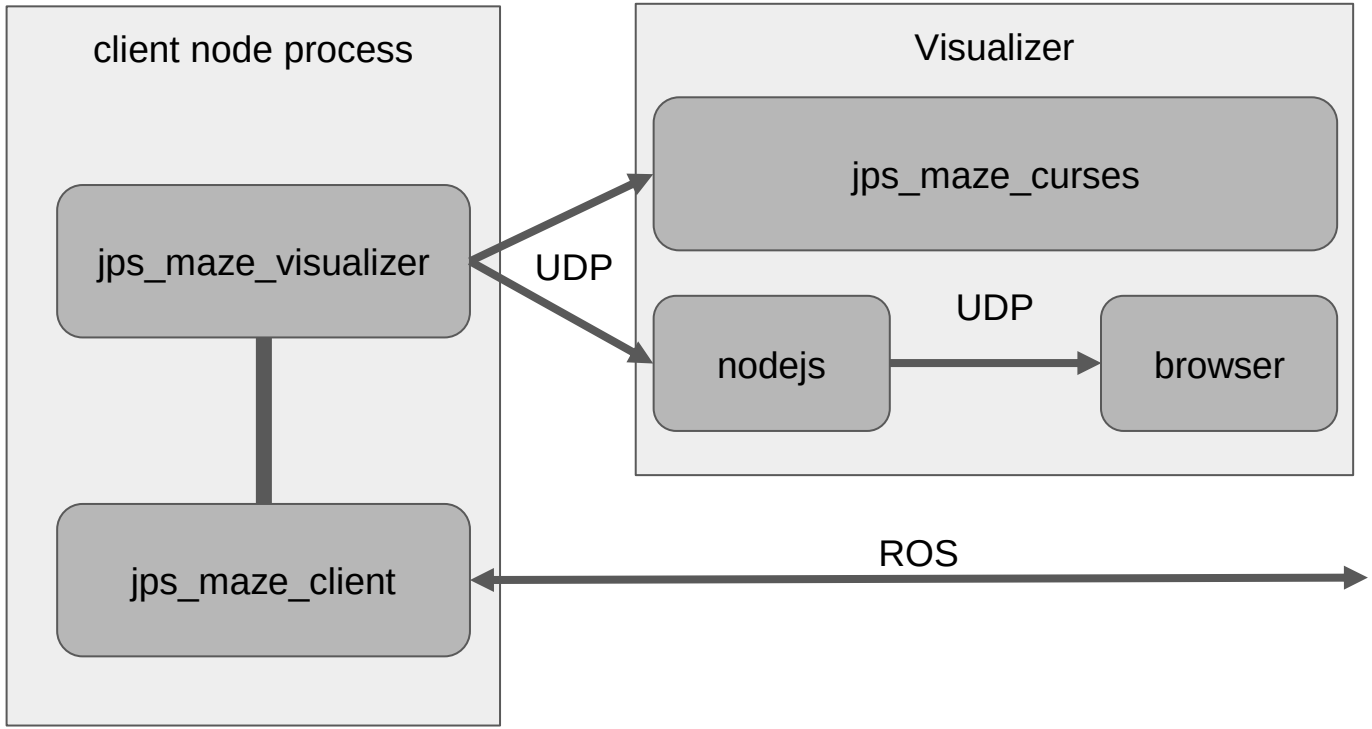
ROS Packet Flow

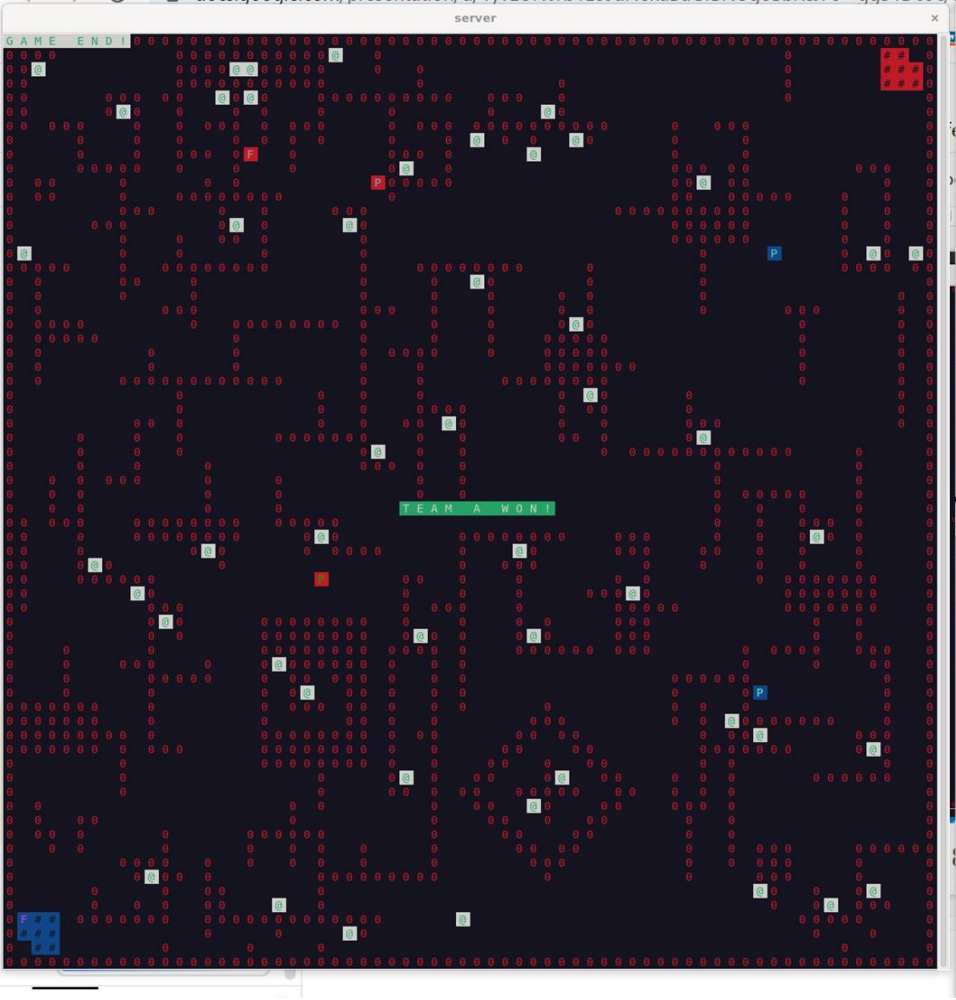
Services | Publisher

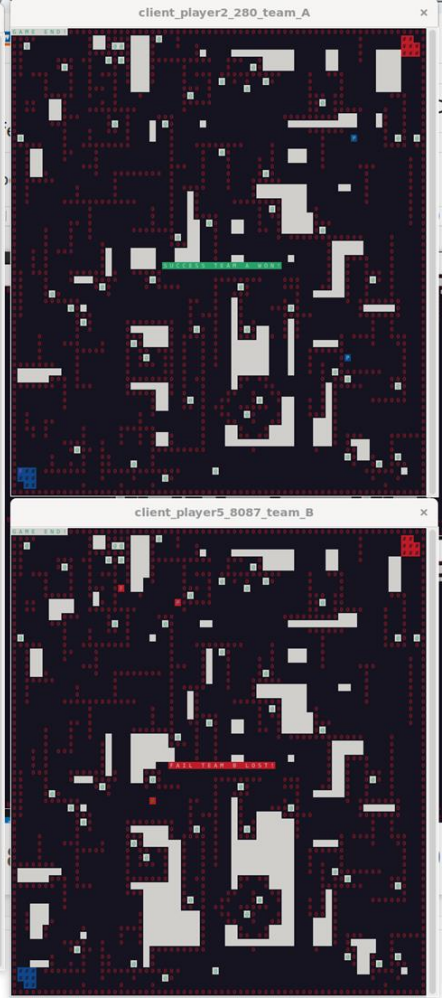
Server structure



Client structure







Ergebnis **16** Dateien mit **2118** Zeilen C++ Source Code | **300+** Commits

Link zum Repository

https://github.com/THI-FF-I/INTP-ros