Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Development of a decentralised communication framework for an online Tichu game

Semester Thesis

Ian Boschung

boian@ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Lukas Faber
Prof. Dr. Roger Wattenhofer

January 7, 2021

Abstract

The problem of common online card game sites, including Tichu, is the need for a central server with full knowledge to relay all communication between the different players. This poses substantial challenges to a server when many games are played at the same time and increases the cost associated with running such a service. This thesis tries to develop a solution to this problem by taking a decentralised approach: The server connects four players who then play the game with only very little communication through the server. Following the development of a computer Tichu agent in a previous thesis, it should also be possible to substitute players with a computer agent.

Contents

Chapter 1

Introduction

During the Covid pandemic, many online card game sites have seen a substantial increase in activity. In some cases, this led to overloaded servers that could not deal with the increased traffic. Usually these websites are built around a central server which performs all computations and relays the playing information between the different players/clients. To make a more scalable system, this project aims towards an architecture, where the player clients communicate directly with each other without relaying through the central server. The central server solely serves as entry point to the game table and as card shuffling instance.

The game implemented in this project is Tichu, a four-player card game with incomplete information. We decided to use it in this project because a) a satisfying implementation is not already available and b) ETH Zürich recently developed a deep learning agent for this game that was missing a scalable user interface.

Chapter 2

Previous work

2.1 Tichu AI

The Tichu agent used in this project was developed at ETH in [1]. It uses reinforcement learning to train a neural network model on the game. As it doesn't calculate any future moves or doesn't take into account the probability of players still having certain cards, there are some limitations on the agents strategic abilities to play well. An improvement of the agents playing skills is not part of this work and will be investigated in other thesis. Moreover, the agent lacks an interface for human players. As an additional goal of this thesis, an intuitive user friendly interface shall be developed.

2.2 Jass Web UI

The User Interface of the Tichu Website is taken in big parts of the ETH Jass AI project ([2][3]), which is itself based on the result of a Zühlke competition [4]. The main changes between the user interface of Jass and Tichu are

- There can be up to 14 cards on the table.
- Players have more than nine cards on their hand.
- Players need to trade cards with the other players in the beginning of each round.
- Multiple cards can be played at the same time.
- New symbols for Tichu and pass actions.

Chapter 3

High-level architecture

3.1 Connection to the AI agent

One challenge in this project is the combination of human players and the Tichu AI. For the website, we are obliged to use the JavaScript language, as this is the only available one. On the other hand, the Tichu AI uses the tensorflow library and is written in python. For this reason, there needs to be some form of communication between these two entities. We facilitate the interaction of these components by separating the central server from the AI part and implementing a separate Tichu AI server. This program uses websockets to communicate with the other players and behaves mostly the same way as the Tichu frontend javascript application.

3.2 Connections between programs

During a game of Tichu, the following three different programs are in play: The main server where users willing to play find each other, the frontend browser application and the server hosting the computer Tichu agent. They all have different connections with each other as shown in Fig. 3.1

To start a game, a user initiates a connection with the central server. After choosing a name, the user can either join an existing table or create a new one. As soon as a player joins a table, all players at that table receive the name and team of the new player. Upon joining, the new player is responsible for opening a connection with all other peers that are already at the table. The clients connect to each other using the webRTC standard [5] that is available in all modern browsers. In cases where a direct connection is not possible because of strict firewall or NAT configurations, we host a TURN server that relays messages between peers. This is expected to happen for about 20% of all connections [6]. If a group wants to play with bot players, the server initiates a connection to the AI server for each bot needed and sends the necessary information to each player. One difference to a normal player is that the AI server never initiates a

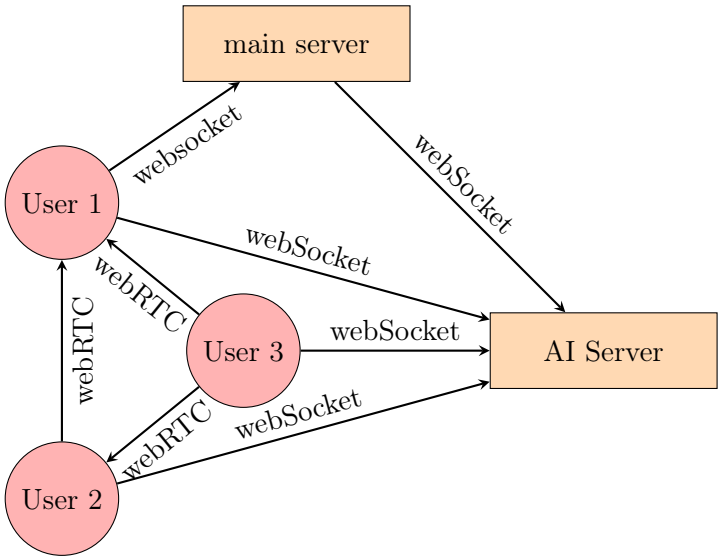

Figure 3.1: All communication channels between the different programs, in this example a Tichu game with three users and one computer player. Arrows indicate which instance initiates the connection.

connection.

3.3 Communication during game

During the game, the four players or bots exchange actions. All actions are sent to every player except the Trade actions, which are only communicated between the two affected players. Every player also keeps his connection with the server open. At the beginning of each round, the server shuffles the 56 cards and distributes them among the players and at the end of each round, every player sends the new point total to the server.

The problem of distributing cards between multiple players was first discussed by Shamir, Rivesta and Adleman in a chapter about Mental Poker[7] and since then multiple approaches to this problem have been developed. Such a solution could be implemented, but would go beyond the scope of this work. Here, we still rely on a central entity to generate a random permutation of all cards and distribute these cards to the players.

Chapter 4

Implementation

4.1 Tichu node library

4.1.1 Motivation

In order to make code reusable, we took the decision to write a javascript library that fully encapsulates the Tichu logic. Javascript is the dominant web scripting language, so this library can directly be used by the browser frontend. To reuse this code in the python application, we use the js2py library that is able to translate it to native python code.

The library also sends the actions to the other players. To keep this part flexible, the actual send function can be passed to the library on creation of a new game.

The advantage of such a library is that it could be replaced by a library with the same interface but for another card game, e.g. Jass. The overall website structure could remain the same with only some adaptions to the user interface.

4.1.2 TichuState class

The TichuState class saves all information associated with one round of Tichu. A Tichu game can be in different states. Depending on the state, players have different actions available to them that modify the state. The possible state and action combinations can be seen in Fig. 4.1. The game always starts at the CARDS8 state where every player only sees 8 cards and needs to decide whether to announce a grand Tichu or not. After every player has decided, the game advances to the CARDS_FULL state. After trading cards, the player who posesses the Mahjong transitions to the MY_TURN state and all other players to the WAITING state. During play, three players are always in the WAITING state and one player is either in the MY_TURN or in one of the two special states.

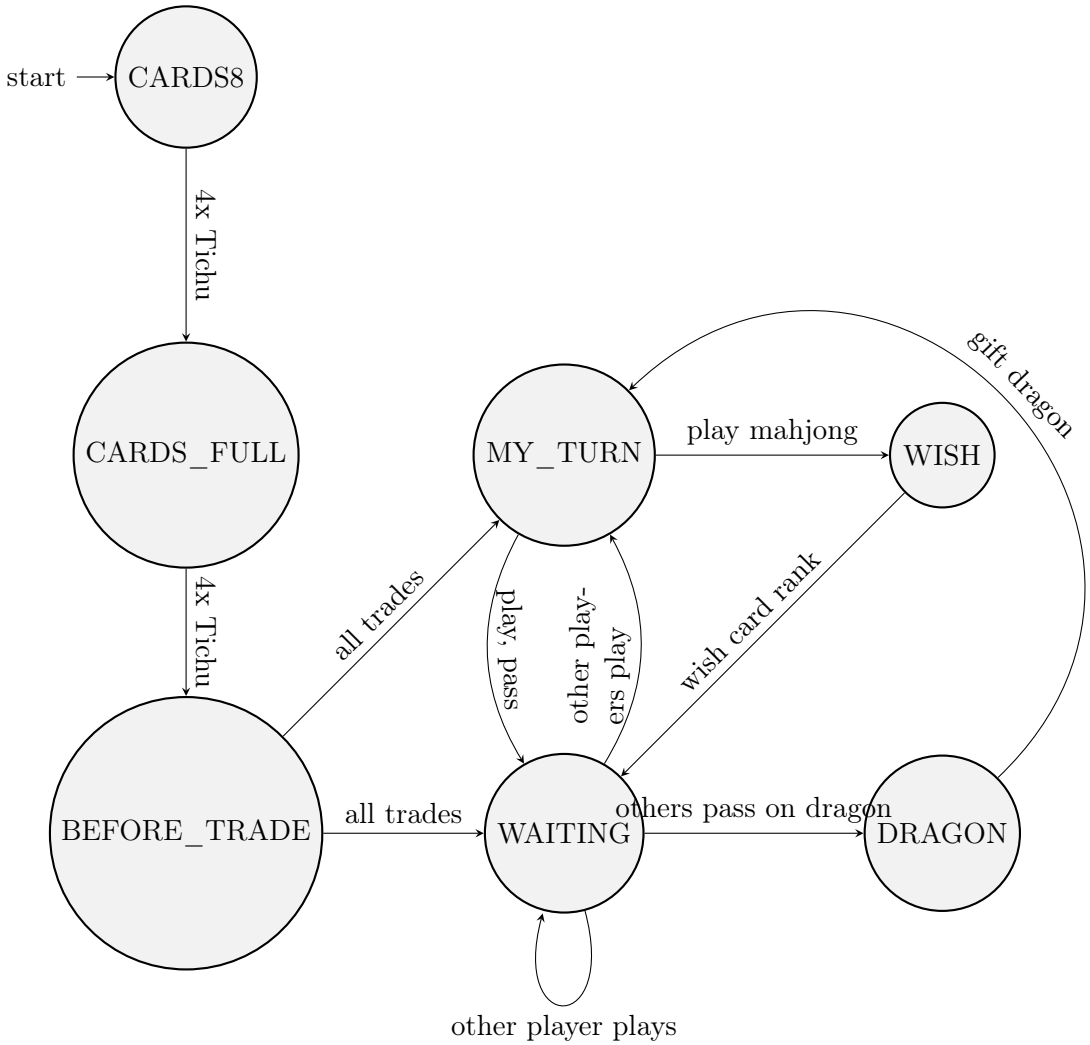

Figure 4.1: State diagram of one round of Tichu.

4.1.3 Action classes

There exists a corresponding python class for every action available to a player. Actions can be serialized to the JSON format to send them between players and can be applied to a state. If an action is applied to a state, it performs additional checks to see if the action is valid and changes the Tichu-state according to the games rules.

4.1.4 Combination class

Card combinations are the most important part of Tichu and similar to the ones found in Poker: Players can play single cards, pair, triple, straight, full house and consecutive pairs. In addition there are two especially powerful combination called bombs.

The Combination class encapsules all information needed to store and compare these card combinations. It also includes a function that calculates a combination from a set of cards. To do this efficiently, we use the framework of a finite automaton that processes the sequence of cards and has accepting states for all valid combinations. To simplify the automaton, we preprocess the input two times. In a first step the cards are sorted in ascending order of their rank. Because order doesn't matter in a Tichu combination, this doesn't change the result of our automaton. Because the combinations are all also valid if shifted by a certain number, e.g. a street from 2 to 6 and a street from 3 to 7, we input not the rank of the actual cards, but the difference between every two adjacent cards to the automaton. Our input alphabet then consists of the following symbols:

- 0: Two cards with the same rank.
- 1: Next cards rank is one higher than the previous card.
- 2: Next cards rank is two higher than the previous card (only used in the automaton with the phoenix).
- x: Next card is anything except the 0 or 1 higher.

It is very easy to construct a nondeterministic finite automaton by introducing one path for each combination as seen in Fig. 4.2. Using the subset construction algorithm this NFA can be simplified to less states and a DFA Fig. 4.3.

In Tichu there is also one special card called the Phoenix that acts as a joker: It can replace any other card in a combination. If played as a single card, it takes the value one half higher than the card played before. The Problem with this automaton is that it doesn't take into account the possibility of the phoenix replacing one card. To solve this problem, we built a second automaton that

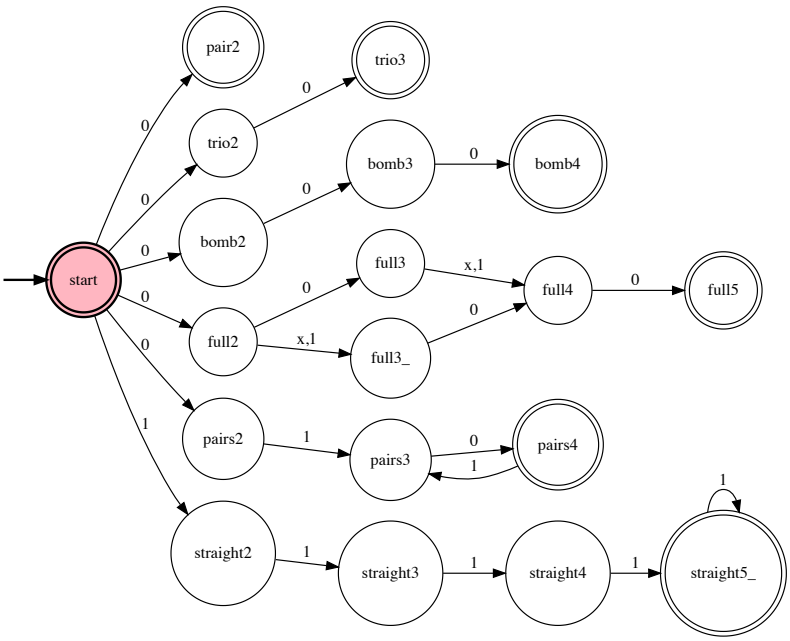

Figure 4.2: Nondeterministic finite automaton to decide if a sequence of cards is a valid combination

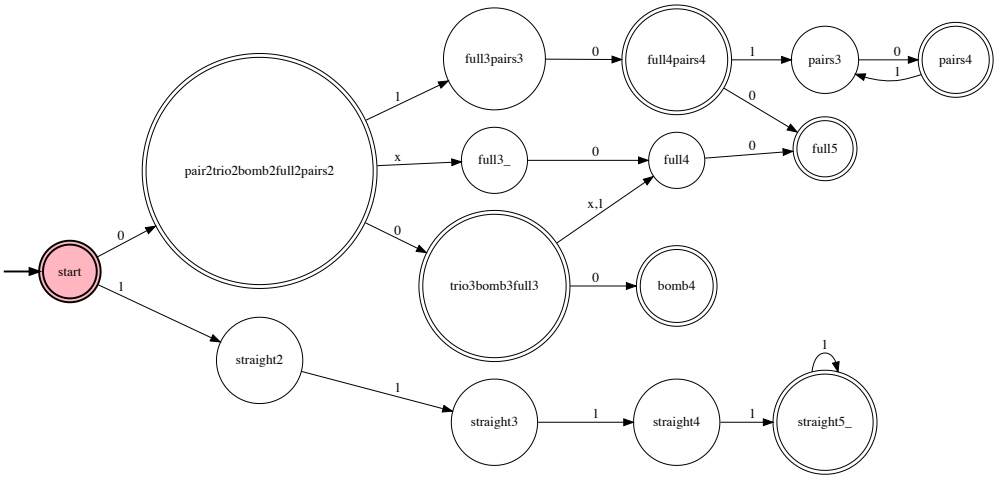

Figure 4.3: Simplified Deterministic automaton with less states

assumes it can repair a combination with a phoenix. This automaton uses a new symbol in the input alphabet: 2 for a difference in rank of two. This transition is needed when the phoenix replaces a card in a street. When combined with the previous DFA this leads to the complete automaton in Fig. 4.4 that can recognize all possible combinations. If the phoenix is available, the start state is the phoenix state, otherwise the nophoenix state. This can be thought of as an additional phoenix/nophoenix transition after the real start state.

To know which value the phoenix finally takes we maintain a list with all possible phoenix values. Some transitions add one or two additional values to this list (marked in red in Fig. 4.4). Also if the automaton ends in a state marked in red a phoenix value needs to added to the list. Which value to add is a function of the last rank processed and the current rank processed.

4.2 Peer2Peer communication

4.2.1 Signaling

To establish a webRTC connection with a peer, a browser first submits a stun request to a special stun server. The stun server responds with a list of possible ip and port pairs that may be used for a connection. From this, the browser creates an ICE candidate that needs to be sent to the other peer via a existing connection, a process called signaling. In this project, the central Tichu server is also used to transmit the signaling data. The peer receiving the ICE offer can use this information to try and open a connection to the other peer.

4.2.2 Card Trades

In Tichu, each player has to trade one card with each other player before playing cards on the table. During the real game, all players must choose their cards and put them face down on the table. Only when all players have decided what card to trade can they look at the cards received from the others. This is more difficult in a decentralized game, because there is no place one can "deposit" a card face down. The solution to this problem is a so called commitment scheme. The simple protocol we used in this project uses the properties of hash functions that it is difficult to find another input with that results in the same hash and the hash function is not reversible. Instead of putting a card face-down on the table, the player generates a random number and concatenates it with the card he wants to trade away. He applies a hash function to this string and sends the result to the other player. After all players have decided on the cards to trade, the verifying phase begins: Each player sends the value of the card he traded and the random string he used to generate the hash. The player receiving the card

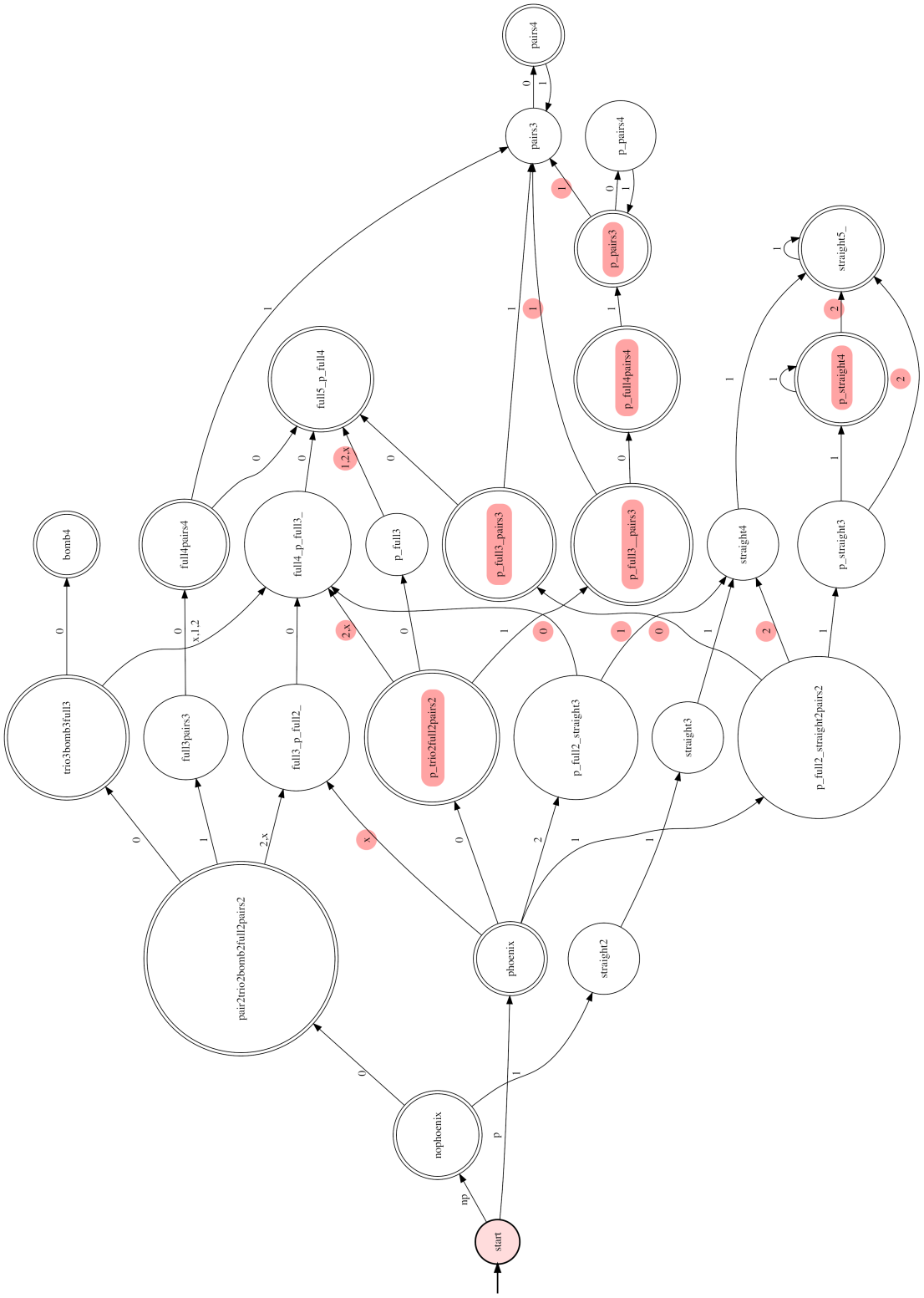

Figure 4.4: Complete DFA including the special phoenix card. Transitions marked in red add a possible phoenix value, states in red if the automaton ends in it.

can do the same hash calculation and compare the results: If they are equal, the other player didn't change his card value.

4.3 Changes to AI interface

A big part of the processing done in the previous Tichu Bot implementation was rewritten in javascript and moved to the Tichu library. This allowed us to avoid code duplication and have all Tichu-related logic in one place, but it also introduces some AI-specific parts to the library. The decision process of an AI action is as follows: The Tichu library calculates the state vector for the AI. This vector is used as an input to the neural network that gives a probability distribution for all Tichu actions. The AI server then chooses the highest rated possible action and uses the Tichu library to convert the index of this action to an instance of a action class.

Chapter 5

User interface

The overall goal while designing the user interface was to have a slim and tidy screen, yet being user-friendly and fully functional. For this reason we decided to use icons instead of text on the most important buttons. For first-players this increases the learning curve to use the site, but users should quickly get accustomed to the most important functions. A mock up of the main playing is shown in Fig. 5.1. To select cards, the user can click on each card he wants to play before clicking on the play button.

Figure 5.1: The main Tichu user interface: (1) players handcards, (2) top combination on table, (3) play and pass button, (4) play bombs or announce tichu, display help window, (5) the player who is highest at the moment is highlighted in green, (6) cards remaining on a players hand, (7) overall score.

Chapter 6

Conclusion

This project shows a possible approach to implement a multi-user card game by choosing a decentralised architecture using the webRTC standard. This architecture is easily scalable yet economical, decreasing significantly the server load in most situations. To guarantee connections in all cases, a TURN server is used to forward messages when a direct connection is not possible. The bot players are also scalable by hosting multiple instances of the AI server program on different servers and assigning them to different games. As an example, this thesis implemented the well-known Tichu game using these concepts.

6.1 Future work

The implemented system lacks in resiliency: It is not possible to recover from a connection failure or a lost message. Another drawback is that no cheat-prevention mechanism is implemented except the simple commitment scheme used in the card trading phase. A series of protocols have been proposed in [8] to prevent different kinds of cheating attempts, e.g. the lockstep protocol to avoid lookahead cheats. In case of a conflict, i.e. not all players having the same state, a resolution can be achieved by assuming the majority is right. In a four player game this signifies that if only one player wants to cheat, he can be identified by the other three players working together. This approach does not work if an entire team tries to cheat, because there would be a two against two situation. In some situations, the central server could be included as an arbiter to further reduce the possibilities for a team to cheat.

In terms of user-friendliness a user-management and ranking system would improve long-term user experience and motivation.

Bibliography

[1] P. Müller, "Tichu bot," Aug. 2020.

[2] J. L. Roman Flepp, "Developing a jass ai platform [confidential]," 2019.

[3] N. Rimensberger, "Developing a jass ai [confidential]," 2020.

[4] webplatformz, "Zühlke jass challenge," https://github.com/webplatformz/challenge, 2016.

[5] B. Sredojev, D. Samardzija, and D. Posarac, "Webrtc technology overview and signaling solution design and implementation," in *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 2015, pp. 1006–1009.

[6] P. Hancke, "What kind of turn server is being used?" https://medium.com/the-making-of-whereby/what-kind-of-turn-server-is-being-used-d67dbfc2ff5d, 2017.

[7] A. Shamir, R. L. Rivest, and L. M. Adleman, "Mental poker," in *The mathematical gardner*. Springer, 1981, pp. 37–43.

[8] N. E. Baughman, M. Liberatore, and B. N. Levine, "Cheat-proof playout for centralized and peer-to-peer gaming," *IEEE/ACM Transactions On Networking*, vol. 15, no. 1, pp. 1–13, 2007.