Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Tichu Bot

Semester Thesis

Peter Müller
pemuelle@student.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Zeta Avarikioti, Lukas Faber
Prof. Dr. Roger Wattenhofer

August 22, 2020

Acknowledgements

I would like to thank my supervisors, Lukas Faber and Zeta Avarikioti, for their valuable advice during our weekly meetings.

Abstract

Inspired by the recent success of reinforcement learning playing imperfect information games such as poker and Dota on a super-human level, we attempted to develop an agent to play the four-player card game Tichu. Tichu is a cooperative multi-agent game of imperfect information with a large state and action space. As such, it presents a challenging task for many current reinforcement learning methods. We applied a mixture of imitation learning, current reinforcement learning methods and Tichu-specific optimisation to the task. Our resulting agent is able to play the game on a decent human level.

Contents

Chapter 1

Introduction

Over recent years, artificial intelligence has become more proficient at beating humans in increasingly more difficult games. Following the success of reinforcement learning (RL) in the perfect information games of Go and chess [1], the challenge has shifted to imperfect information games such as poker [2] or Dota [3]. In Tichu, competing teams of two players each try to get rid of their cards as quickly as possible while securing points along the way. The cooperation required between the two team-players, while not knowing what cards the other players hold, and the vast number of playable combinations are just a few examples that make the game challenging from an RL point of view.

Most of the current approaches that play games on a super-human level rely heavily on tree-search methods. While these approaches generally perform quite well, they often require extensive amounts of computing power to be able to make their moves in a reasonable amount of time. With the goal of developing an agent which is playable without extensive computing resources, we primarily focus on purely RL-based methods in this thesis.

Chapter 2

Related Work

In perfect information games such as chess or Go the entire game state is visible to all the players. Go has long been recognised as the most challenging perfect information game for artificial intelligence (AI). In 2016, DeepMind was able to beat the Go world champion, Lee Sedol, with their RL agent AlphaGo. [4] AlphaGo learnt the game by playing thousands of matches with amateur and professional players and, therefore, acquired considerable human knowledge. In 2017, DeepMind published a new approach entitled AlphaZero, which learnt solely by playing against itself, starting from completely random play. [1] AlphaZero was not only able to outperform AlphaGo in the game of Go but also achieved state-of-the-art performance in the perfect information games of chess and Shogi.

In imperfect information games, only a part of the game state is observable by each player (e.g. in poker the players do not know the cards the other players are holding). While there has been some success in playing imperfect information games on a super-human level (i.e. poker [2] and Dota [3]), some unsolved challenges remain. One example of such a challenge concerns the reasoning involved in the hidden information based on the history of the game. The recently published Hanabi challenge [5], which titles as the new frontier for AI research, focuses on the challenge mentioned above, which is called theory of mind (ToM) reasoning. Hanabi is a multi-agent card game of imperfect information, which relies strongly on ToM reasoning to be played well. Recently, some progress has been made in this challenge. [6] They have presented a new deep multi-agent RL method, the Simplified Action Decoder. With this, they were able to achieve a new state-of-the-art performance in the four and five player games of Hanabi.

‘Big 2’ is a card game which can be viewed as a simpler version of Tichu. It is also a four-player game with imperfect information, and the cards and combinations are played similarly. The main differences between Big 2 and Tichu are that Big 2 is not played in teams of two and dispenses with several other Tichu rules (i.e. exchanging cards, calling ‘Tichu’ and the use of special cards). Researchers have applied RL to the task of playing Big 2. [7] They used the proximal policy optimisation algorithm and trained their model through self-play. The resulting agent was able to outperform amateur human players.

Chapter 3

Background

3.1 Tichu

Tichu[1] is a four-player card game, played in teams of two players with teammates sitting opposite each other at the table. Tichu is played with a deck of 56 cards, which consists of a standard 52 card set (2 to Ace in four suits) and four special cards (explained at the end of this section).

The game starts with each player being dealt eight cards. Each player can then decide whether they want to call a 'Grand Tichu' or not. When a player calls a 'Grand Tichu', they bet on being the first to get rid of all their cards, which, in return, will give them 200 points if they succeed and -200 points if they fail. The remaining cards are dealt, resulting in all the players holding 14 cards. From this point on until the player plays their first combination, they can call a 'Tichu'. A 'Tichu' is the same bet as a 'Grand Tichu', but it is only worth 100 or -100 points.

Then, each player selects three cards to simultaneously exchange with the other players by passing one card face down to each of the other players. After the exchange, the player with the Mah Jong card starts the game by playing the first combination. In what follows, only higher combinations of precisely the same type can be played, or a player can pass. If three players pass in a row, the trick is completed. The player who plays the last combination wins the trick and plays a new combination of his choice.

Combinations are based on poker hands with a few additions. Valid combinations are single, pair, trio, full house, straight (of at least five cards), pair steps (a straight of consecutive pairs) and bombs (four of a kind or a straight flush of at least five cards). Bombs can be played at any point in the game (even out of turn) and do not have to match the combination laying on the table.

The round ends when only one player with cards remains. The last player has to give their remaining cards to the opposing team and the tricks they have won

[1] https://www.fatamorgana.ch/tichu/tichu_english.asp

to the player who was the first to get rid of their cards in this round. Points are counted based on the cards from the tricks won by each team. Kings and tens are worth 10 points, fives are worth five points, the Dragon is worth 25 points, and the Phoenix is worth -25 points. All the other cards are worth 0 points. In the case that the players of one team finish first and second, the round ends, the team earns 200 points, and the counting of points from the tricks won by each team is omitted. Additionally ± 100 and ± 200 points are added for won and lost 'Grand Tichus' and 'Tichus'.
The rounds continue until one team reaches a total score of 1000 points or more.

Special Cards:

- **The Mah Jong** card is the starting card, the player with the Mah Jong card starts the game by playing the first trick. The Mah Jong card has a value of 1 and can either be played as a single card or in a straight from 1 to 5 or further. Additionally, it allows the player to wish for a card rank when playing the card. Every player who has a card in the desired rank and can play a valid combination must do so. The wish remains active until one player fulfils it.
- **The Dog** card can only be played as a single card after a player wins a trick (or as the first card). It passes the lead to the player's partner.
- **The Phoenix** card can be used as a wildcard in any combination (except for bombs). When played as a single card, it assumes a rank 0.5 higher than the previous card played.
- **The Dragon** card is the highest card in the game and can only be played as a single card. If the trick is won with the Dragon, the player has to give the trick to a chosen opponent.

3.1.1 Challenges

We noted the following challenges that make Tichu difficult for an RL agent to learn.

- **Imperfect Information:** The game-state of Tichu is only partially observable, which means that the cards of all the other players are hidden. An RL agent needs a way to extract additional information from the history of the observable states and actions of the other players.
- **Cooperative Multi-Agent:** The multi-agent setting itself already poses a challenge for RL. Additionally, Tichu requires a mix of cooperative and competitive play, which is an additional difficulty.

- **Large State and Action Space:** There are over 10^{30} possible initial hand distributions in Tichu, meaning the number of possible game states is even larger. Additionally, one can form approximately 10^9 different valid combinations from 14 cards.
- **Multiple Game Stages:** The different stages of Tichu (announcing Tichus, trading cards and playing combinations) result in multiple learning tasks for the RL agent.
- **Sparse Rewards** A single round of Tichu lasts on average 120 turns. The reward of the round is only known at the end. This results in the credit assignment problem, meaning the agent has to discern which of the actions he took during the round contributed to the reward at the end. Additionally, there are conflicting long- and short-term rewards; for instance, when winning a trick worth many points, it is not guaranteed that the player will keep these points. In the case of finishing last, these points go to the winning team.

3.2 Reinforcement Learning

Reinforcement learning is a subset of machine learning which focuses on agents learning via trial and error within a simulated environment. The general RL

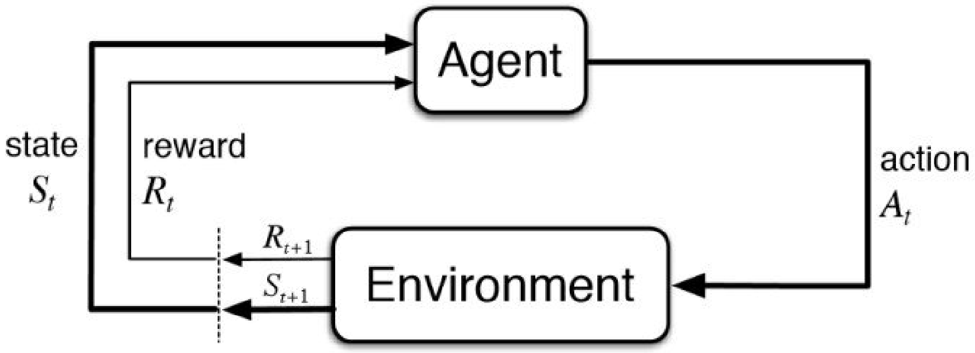

Figure 3.1: RL cycle [2]

cycle can be seen in Figure 3.1 and works as follow. At every time-step t, the agent receives the current state S_t from the environment. Based on this state, the agent then chooses an action A_t to take. The environment executes the action and returns a reward R_{t+1} and a new state S_{t+1} to the agent. This cycle repeats until a terminal state is reached within the environment. The goal of RL is to find a policy which maximises the cumulative reward when the agent acts according to it. A policy in RL defines a mapping from state to action.

3.2.1 Self-Play and Multi-Agent Reinforcement Learning

The general single-agent setting in RL focuses on a single agent solving a task within a mostly static environment. In contrast, learning in multi-agent settings is fundamentally more difficult. An example of this is the non-stationarity of the environment, as multiple agents interact simultaneously with the environment and potentially with each other. The optimal policy for an agent, therefore, also depends on the policies of the other agents.

3.3 Imitation Learning

Imitation learning (IL), also referred to as learning from demonstration, is the method of learning by observing and imitating the behaviour of a human expert.

Behaviour Cloning (BC) focuses on learning a policy using supervised learning over state-action pairs of expert trajectories. It is the simplest form of IL because it treats the problem as a supervised learning task. However, the main issue with this approach is the compounding errors when using the trained policy in a test environment. [8] The reason for this is that the trained policy will make mistakes which compound over time. These compounding errors result in states that did not appear in the expert data set, leading to the BC agent failing.

3.4 Theory of Mind

One challenge which arises in the cooperative multi-agent setting with imperfect information is the beliefs concerning the hidden information. Humans can interpret the actions of others and act in a certain way such that others who observe their actions can reason over their intents and beliefs. This capability is commonly referred to as the theory of Mind (ToM) [9]. In our Tichu setting, an example of this would be making assumptions about the hands of the other players based on their actions.

Chapter 4

Reinforcement Learning

4.1 Tichu Environment

As the foundation for our Tichu environment, we used an existing implementation from Github.[1] The code originates from a previous attempt at applying Deep Q-Learning on Tichu, but no written work referencing the code and presenting the results could be found online.

4.1.1 Simplified Tichu Version

To simplify our problem and increase our training performance, we made the following changes to the original rules of Tichu.

1. **No straight bombs.** The discrete action space of Tichu is huge with approximately 10^9 different valid combinations that can be formed from 14 cards. We can reduce the action space to approximately 10^3 by ignoring the colours of the cards. Since the colour of a card is only considered when playing a straight bomb combination, removing it does not change the game very much but rather makes learning easier.
2. **No asynchronous bombs.** We wanted to work with a turn-based environment to simplify the RL process. To incorporate the asynchronous bombs into a turn-based environment, we would have had to ask all four players whether they wanted to play a bomb or not after every move. In the worst-case scenario, this would increase the episode length by a factor of 4.

Additionally, we defined a single round of Tichu as one episode instead of a whole game to 1000 points, as otherwise the episode length explodes.

[1] `https://github.com/lukaspestalozzi/Tichu`

4.1.2 State, Actions and Reward

In order to train an RL agent on Tichu, the environment has to encode the current state of the game into a vector of input features. A few examples of our input features which only consider the currently observable state of the game are the cards the player is holding, the number of remaining cards for all other players and information about the current trick on the table. Ideally, the input features would also contain a complete history of observable state-action pairs from all the players. However, with rounds lasting over 100 steps and the large state and action space of Tichu, this would increase the input vector excessively. For this reason, we added a set of human-engineered features about the history of the game. Some examples of these features were counts of the already played cards and information on which combinations the other players had passed. This resulted in a total of 85 features for our state vector. A detailed description of all the input features can be found in the appendix (Table A.2).

The action space of our environment consisted of all the legal actions possible according to our simplified Tichu setting. This included all the valid combinations that could be formed with 14 cards but also all other actions such as calling Tichu and trading cards. This resulted in a total of 1,415 different actions. With every state update, the environment also provides a legal-actions-mask to the agent. This mask states what actions from the action space are allowed in the current state of the game. A detailed description of all the actions can be found in the appendix (Table A.1).

As the episode reward, we counted the points according to the Tichu rules at the end of the round. We then subtracted the points of the opponent's team from the points of the agent's team. We did this in order to maximise the point difference for the opponent's team instead of just the points the agent's team scored. Through this, our agent will be rewarded for denying the enemy team a Tichu, which otherwise would only result in negative points for the enemy team but would not affect the score of the agent's team.

4.2 Policy Optimisation

We used proximal policy optimisation (PPO) [10] as our RL algorithm, based on its previous success in imperfect information games. [3, 7, 11] We made use of the more recent PPO2 implementation based on the OpenAI baselines[2]. In contrast to PPO, PPO2 is optimised for distributed training. It uses one process to train the policy network and multiple processes running the environment to generate training data in parallel.

Due to the long training time, hyperparameter tuning tends to be difficult.

[2]https://github.com/openai/baselines

For this reason, most of the hyperparameters for the policy network were left at the default value. Only the hidden layer configuration was changed to five layers with 512; 1024; 2048; 1024 and 512 neurons, which is considerably larger than the default. This configuration was found by performing a grid search on the supervised learning task for behavioural cloning.

4.3 Training

The agent is trained using self-play utilising a shared policy network for both players on the agent's team. We used opponent sampling as proposed by previous work [12, 3], meaning that our agents play against the latest policy for 80% of the games and against random older policies for 20% of the games. We saved the policies every 200,000 time-steps and added them to the opponent pool. Playing occasionally against past policies is performed in order to achieve a more robust policy and avoid strategy collapse. Strategy collapse refers to the phenomenon in which the agent forgets how to play against a wide range of opponents because it only requires a small set of strategies to defeat its latest past version. [3]

Due to the time complexity within the Tichu environment, which mainly resulted from calculating the legal actions at every step in the game, making use of a GPU would have required many environments running in parallel, and this far exceeded our available computing resources. For this reason, we trained our policy network solely on CPU machines. Running the environment and training the network on 16 CPUs resulted in a training speed of around 10 million time-steps per 24 hours, which corresponds to approximately 200,000 rounds of Tichu.

Chapter 5

Optimisations

In order to further improve the performance of our agents, we tried to address a few specific shortcomings, which are described in the following.

5.1 Imitation Learning

One problem which arises when applying RL algorithms to games such as Tichu with a large state and action space is their sample efficiency. An agent might need to play millions of games before finding a good policy. In past work, IL has been used to initialise a policy from human expert plays in order to accelerate learning. [4] For this reason, we initialised our policy network via BC using game logs from an online Tichu platform.

5.1.1 Behavioural Cloning Pre-Train

We gathered our expert trajectories by scraping the Tichu game logs[1] of the online board-game platform *Brettspielwelt*. We only used the state-action pairs of the winning team for each round. Additionally, all the games containing straight bombs and asynchronous bombs were filtered out since they do not work in our environment.

We pre-trained our policy network for ten epochs on the entire expert dataset. After this, we continued training via PPO, as explained in the previous chapter.

5.2 Predict Hidden State and Monte Carlo Tree Search

A crucial aspect of Tichu is the reasoning that occurs with regard to the hands of the other players based on their past actions. Since our previous approach did not consider the actions taken by the other players during the game, in our

[1] `http://tichulog.brettspielwelt.de/202001`

opinion crucial information was missing that was needed to play the game on a human level. With games lasting over 100 time-steps and the history containing much unnecessary information, including the entire history of partially observable states and actions in the input to our RL agent was not feasible.

As previously stated, the reasoning over the hidden state based on the actions of others is part of the ToM. Solving this problem for multi-agent imperfect information games is still an ongoing research topic, that is, the currently unsolved Hanabi challenge. [5] We tried to apply an approach from previous work conducted on that topic to improve our end-game performance. [13] Accordingly, we saved the entire state-action history of the game. The saved state only includes the information observable by all the players. In our case, this means the information about the players' hands is masked. With the knowledge of their own hand, the already played cards and the number of remaining cards of each player, the agent can generate all the possible hand distributions for the other players at the current state of the game. The already played cards for every state in the history can be reconstructed from the observed actions. For each possible distribution, the agent can, therefore, fill in the masked information in the state-action history. This means we can now check for every state-action pair in the history if the action taken corresponds to the action selected by the policy for our reconstructed state. Undertaking this for all possible hand distributions allows us to reconstruct the hidden information.

This method has some limitations. First, the number of possible hand distributions for the remaining cards is enormous early in the game, making this method only possible for the end-game. Second, for this method to work, we have to know the policy the other agents are following or at least be able to approximate it. In a setting where the agents play against humans, this is almost impossible. However, we can assume that a policy network trained via BC on human game-play approximates a human policy.

With this approximation of the hidden information, we can convert the imperfect information game into a perfect information one. Perfect information games (i.e. Go and chess) have already been solved successfully via Monte Carlo Tree Search (MCTS). [4] For this reason, we tried replacing the RL agent with an MCTS agent during the end-game when approximating the hidden information was possible.

5.3 Separate Tichu Calling Model

During the training of our agent, we observed that it never called a 'Grand Tichu' or 'Tichu'. With the policy network initialised via BC, the initial 'Tichu' calling rate was around 0.15 per game, meaning the agent called a Tichu approximately every seventh game. However, during further training via RL, this rate always dropped to 0. We were not able to find a reason for this behaviour. However, one could hypothesise that by calling 'Tichu', a player and their teammate would mainly focus on finishing first, possibly neglecting the other 100 points from won tricks. When training from scratch, the behaviour can be explained in the following way. In a random setting, the probability that the agent wins a Tichu is 0.25, which results in an expected reward of -50 when calling 'Tichu' and -100 when calling 'Grand Tichu'. Not calling any 'Tichus' results in an expected reward of 0. For this reason, it makes sense that the policy collapses to refraining from calling Tichus early during training.

In an attempt to force the agent to play more Tichus and possibly to improve its overall performance, we trained two separate supervised neural networks on the task of calling 'Tichu' and 'Grand Tichu'. As an input for the networks, we took the eight cards for 'Grand Tichu' and 14 for 'Tichu'. The network has one single output stating whether this hand will finish first or not. As our training data, we again used the game logs and extracted all the initial hands and whether they finished first for every game. The datasets were balanced using random under-sampling, and two simple fully connected neural networks were trained on the task. Both networks had an input size of 17 (a count between 0 and 4 for each of the 17 different cards) and two hidden layers with 48 and 96 neurons. The trained network was then used in conjunction with a trained RL agent for the calling 'Tichu' decisions. Since the output of the network is a probability between 0 and 1, we were able to tune the cutoff and, therefore, the level of aggressiveness at which the agent calls 'Tichu'. We ran a simple grid search for the optimal cutoff value by evaluating different agents for 1,000 games and optimising for the highest reward. The search resulted in an optimal cutoff value of 0.75 for both 'Tichu' and 'Grand Tichu'.

In addition to using these Tichu calling networks on top of already trained agents, we trained an agent, utilising it during training. In theory, this should allow an agent to learn to play with this more aggressive way of calling 'Tichu' and, therefore, further improve its overall performance.

Chapter 6

UI

In order to test the trained RL agents against humans, we developed a web application consisting of a React front-end and a python back-end. The app is deployed on a Heroku Dyno and accessible via this URL for the reader to test: `https://tichu-ai.herokuapp.com`

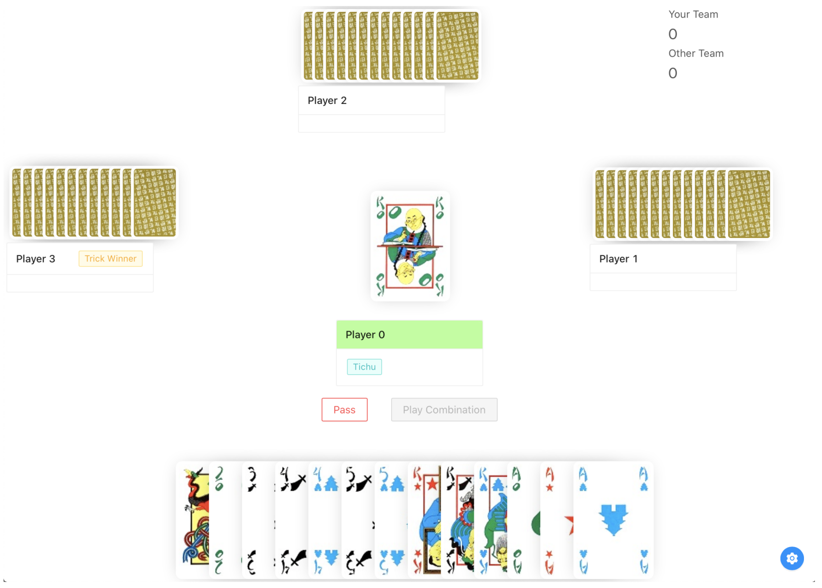

Figure 6.1: Screenshot of our Tichu UI

The front-end is written in React and allows one or multiple humans to play against the trained RL agents.

The back-end handles the game state by running the same environment used for the RL part. It holds the trained agents and performs moves for the AI. To keep the game state synchronised on all the open web clients, we broadcast updates via a WebSocket. Additionally, some REST endpoints are exposed to handle different commands from the clients (i.e. game setup, applying human actions and checking the validity of a build combination).

Chapter 7

Results

7.1 Evaluation

We evaluated our different approaches against two baselines and in an All vs. All tournament. As a performance measure, we took the reward of the agent averaged over 10,000 rounds of Tichu. It should be noted that we defined our reward as the point difference with the opposing team, which means the expected reward against an opponent of the same strength would be 0.

To keep the influence of chance as small as possible during the evaluation, we used the following methods. We generated a fixed set of initial hand distributions used for all the evaluation rounds. Additionally, each round was played a second time with the same initial hand but shifted one position to the right. This shift meant that the agent played the same round twice, but the hands were switched between teams, which cancels out wins based solely on good hands.

7.1.1 Baselines

In order to compare our different approaches, we used the following two baseline agents:

- **Random Agent**: At every turn, the random agent selects an action at random from the set of legal actions for the current state of the game.
- **Rule Agent**: The rule agent selects its actions based on the following rules: It never calls 'Grand Tichu' or 'Tichu'. It always trades the two lowest cards to its opponents and the highest card to its partner. When possible, it never passes and always plays the lowest possible combination. When leading a new trick, it selects the combination with the highest number of cards.

7.2 Experiments

The following agents were evaluated against our two baselines and in an All vs. All tournament style.

1. **Behavioural Cloning** This agent's policy was trained via BC only.
2. **PPO** We trained different agents using PPO with one or multiple of the following variations. All the PPO agents were trained for 40 million steps (approx. 800,000 rounds of Tichu), which took five days on average.
   (a) **Scratch** Policy network randomly initialised. Trained from scratch without any human expert demonstration.
   (b) **BC** Policy network initialised via behavioural cloning rather than random.
   (c) **Tichu Model** The separate Tichu calling model from Section 5.3 was used on top of the trained agent during evaluation.
   (d) **Tichu Model (Training)** The separate Tichu calling model was used during training and evaluation.
   (e) **MCTS** During the end-game, the MCTS approach from section 5.2 was used.

The summarised evaluation results can be seen in Table 7.1. The entire evaluation matrix from the All vs. All tournament can be found in Appendix B.1.

		Average Reward vs.		
#	Agent	All	Rule	Random
1	Behavioural Cloning	19.8	190.2	481.7
2	PPO (Scratch)	−110.0	86.3	432.1
3	PPO (BC)	17.2	166.3	458.0
4	PPO (BC) + Tichu Model	33.7	**208.4**	**510.3**
5	PPO (BC) + Tichu Model (Training)	**39.3**	199.9	508.5
6	PPO (BC) + Tichu Model (Training) + MCTS	(37.2)[1]	193.4	507.7

Table 7.1: Evaluation Results

7.2.1 Training from Scratch

To monitor our policy's training performance, we evaluated our policy every 500,000 steps by playing 1,000 rounds against random previous versions of itself

[1]Avg. Reward vs. Agent #5

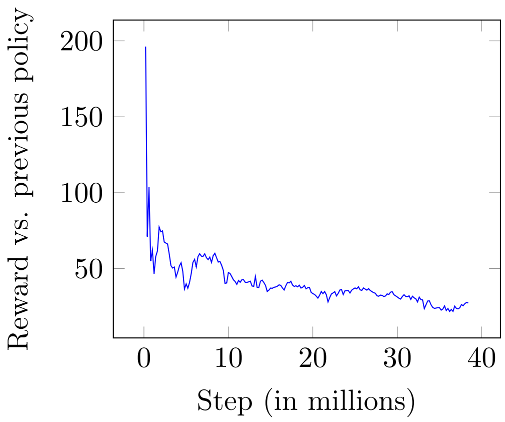

Figure 7.1: Reward PPO (Scratch)

(Figure 7.1). Additionally, we tracked the frequency of different combinations during these 1,000 evaluation games (Figure 7.2).

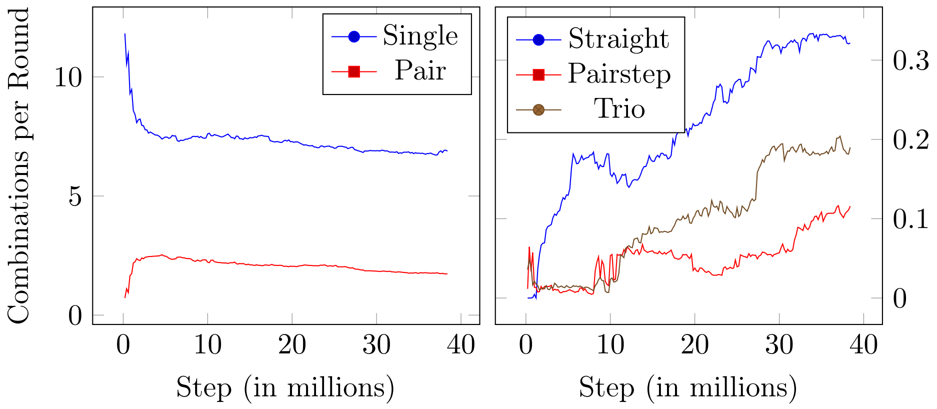

Figure 7.2: Combination Frequency PPO (Scratch)

We observe that the learning progress seems to continue steadily throughout the training, with the current policy being able to outperform its previous versions consistently. In Figure 7.2 we can see that the frequency of the ‘simple’ combinations decreases and the frequency of the ‘larger’ combinations increases during training, which is also a sign of the agent improving in our opinion. It appears that if we train the policy for longer, the performance will continue to improve.

Training from scratch without human knowledge was, however, somewhat inefficient. Even after five days of training, the agent made many mistakes and mainly played single cards when evaluated via the UI.

7.2.2 Behavioural Cloning

The agent trained via BC is already able to beat the agent trained from scratch by, on average, more than 100 points per round.

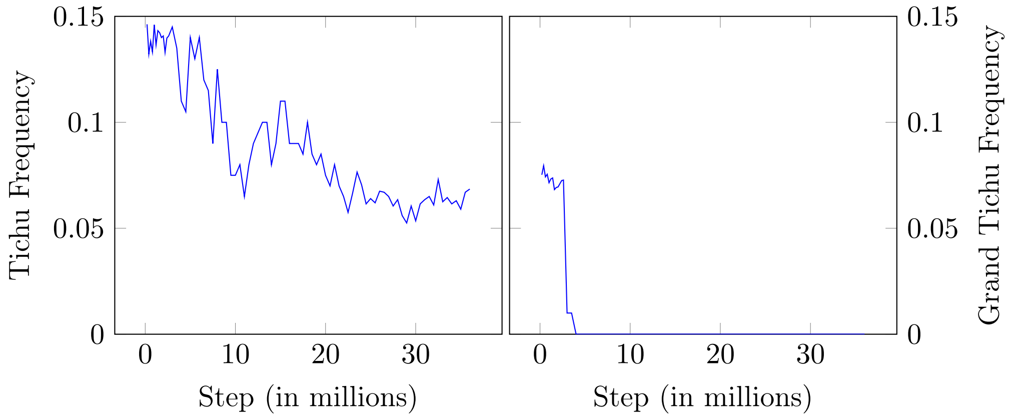

Figure 7.3: Tichu Frequency PPO (BC)

When continuing to train the agent with the BC initialised policy, we observe a slow but constant training improvement. As mentioned earlier, the Tichu calling rate during training decreased considerably (Figure 7.3). Nevertheless, the trained PPO (BC) agent is able to beat the initial BC agent.

7.2.3 Separate Tichu Calling Model

Using the separate Tichu calling model on top of our trained PPO (BC) agent, further improved its performance. The agent with the Tichu calling model is able to beat the same agent without it by 10 points on average. Using the Tichu calling model during the training of the agent also further improved the performance by a few points.
When evaluating the agent via the UI against humans, the performance increase is quite noticeable in our opinion. Even though it calls Tichus quite aggressively, most of the time it is able to win them.

7.2.4 Monte Carlo Tree Search

Due to the time complexity of our MCTS agent, we only evaluated it against our best PPO agent from the previous section. The cutoff value to begin the approximation of the hidden information was set to 1000, meaning the MCTS begins once the number of possible hand distributions is below 1000. Additionally, we set a time limit of 10 s per move to perform the tree search.

From the results, we can see that the best PPO agent with MCTS in the endgame was able to beat the same agent without MCTS by 37 points on average. However, in this scenario, we know the exact policy the opponent is following and are, therefore, able to reconstruct the hidden information with high accuracy. In a real-world scenario when playing against humans, the opponent's policy is unknown, and it is, therefore, unlikely we would be able to reconstruct the hidden information with high accuracy.

7.3 Evaluation against Humans

With the help of the developed UI, we were able to test our trained agent against humans. The evaluation rounds were played by two human players against two trained agents. We used the *PPO (BC) + Tichu Model (Training)* agent as the opponent for this evaluation. In total, 43 rounds were played, resulting in an average reward of 21.2 for our agent.

We must note that 43 rounds is a relatively small sample size. In addition, we could not apply the switching of hands to reduce the influence of chance as human players are able to remember the hands of previous rounds.

Even though our agent was able to beat humans by 21.2 points on average per round, we still think it is not performing on a super-human level. When observing the agent playing, we can sometimes still see some obvious mistakes, which should not occur in a perfect play. Nevertheless, the results are quite promising, and we would argue that the agent is playing on a decent human level.

Chapter 8

Conclusion

During this thesis, we were able to create an RL agent that plays Tichu on a decent human level. This was mainly achieved through a mixture of IL, RL and optimisation specific to the setting of Tichu. The combination of the different challenges presented by Tichu made it difficult to find a general solution for achieving super-human performance. In particular, an efficient belief state construction in a large game of imperfect information is still an ongoing research topic, which was not the primary focus of this thesis.

8.1 Future Work

More work has to be performed in order to apply RL to the full version of Tichu with more than 10^9 possible actions. Some methods have been proposed to deal with large discrete action spaces by grouping similar actions together. [14] The authors demonstrated the capability of their method for tasks of up to 10^6 actions but suggest it could also be applied to larger action spaces.

Further, it would be interesting to apply the methods used by the recently published paper on the Hanabi challenge to the case of Tichu. [6] In particular, because an efficient belief state construction is, in our opinion, crucial to playing Tichu on a super-human level.

Bibliography

[1] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. v. d. Driessche, T. Graepel, and D. Hassabis, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, Oct. 2017, number: 7676 Publisher: Nature Publishing Group. [Online]. Available: https://www.nature.com/articles/nature24270

[2] M. Moravčík, M. Schmid, N. Burch, V. Lisý, D. Morrill, N. Bard, T. Davis, K. Waugh, M. Johanson, and M. Bowling, "DeepStack: Expert-level artificial intelligence in heads-up no-limit poker," *Science*, vol. 356, no. 6337, pp. 508–513, May 2017. [Online]. Available: https://www.sciencemag.org/lookup/doi/10.1126/science.aam6960

[3] C. Berner, G. Brockman, B. Chan, V. Cheung, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, and S. Zhang, "Dota 2 with Large Scale Deep Reinforcement Learning," p. 66.

[4] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. v. d. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–503, 2016. [Online]. Available: http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html

[5] N. Bard, J. N. Foerster, S. Chandar, N. Burch, M. Lanctot, H. F. Song, E. Parisotto, V. Dumoulin, S. Moitra, E. Hughes, I. Dunning, S. Mourad, H. Larochelle, M. G. Bellemare, and M. Bowling, "The Hanabi challenge: A new frontier for AI research," *Artificial Intelligence*, vol. 280, p. 103216, Mar. 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0004370219300116

[6] H. Hu and J. N. Foerster, "Simplified Action Decoder for Deep Multi-Agent Reinforcement Learning," *arXiv:1912.02288 [cs]*, Dec. 2019, arXiv: 1912.02288. [Online]. Available: http://arxiv.org/abs/1912.02288

[7] H. Charlesworth, "Application of Self-Play Reinforcement Learning to a Four-Player Game of Imperfect Information," *arXiv:1808.10442*

[cs, stat], Aug. 2018, arXiv: 1808.10442. [Online]. Available: http://arxiv.org/abs/1808.10442

[8] S. Ross and J. A. Bagnell, “Efcient Reductions for Imitation Learning,” p. 8.

[9] C. L. Baker, J. Jara-Ettinger, R. Saxe, and J. B. Tenenbaum, “Rational quantitative attribution of beliefs, desires and percepts in human mentalizing,” *Nature Human Behaviour*, vol. 1, no. 4, pp. 1–10, Mar. 2017, number: 4 Publisher: Nature Publishing Group. [Online]. Available: https://www.nature.com/articles/s41562-017-0064

[10] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal Policy Optimization Algorithms,” *arXiv:1707.06347 [cs]*, Aug. 2017, arXiv: 1707.06347. [Online]. Available: http://arxiv.org/abs/1707.06347

[11] B. Baker, I. Kanitscheider, T. Markov, Y. Wu, G. Powell, B. McGrew, and I. Mordatch, “Emergent Tool Use From Multi-Agent Autocurricula,” *arXiv:1909.07528 [cs, stat]*, Feb. 2020, arXiv: 1909.07528. [Online]. Available: http://arxiv.org/abs/1909.07528

[12] T. Bansal, J. Pachocki, S. Sidor, I. Sutskever, and I. Mordatch, “Emergent Complexity via Multi-Agent Competition,” *arXiv:1710.03748 [cs]*, Mar. 2018, arXiv: 1710.03748. [Online]. Available: http://arxiv.org/abs/1710.03748

[13] A. Lerer, H. Hu, J. Foerster, and N. Brown, “Improving Policies via Search in Cooperative Partially Observable Games,” *arXiv:1912.02318 [cs]*, Dec. 2019, arXiv: 1912.02318. [Online]. Available: http://arxiv.org/abs/1912.02318

[14] G. Dulac-Arnold, R. Evans, H. van Hasselt, P. Sunehag, T. Lillicrap, J. Hunt, T. Mann, T. Weber, T. Degris, and B. Coppin, “Deep Reinforcement Learning in Large Discrete Action Spaces,” *arXiv:1512.07679 [cs, stat]*, Apr. 2016, arXiv: 1512.07679. [Online]. Available: http://arxiv.org/abs/1512.07679

Appendix A

Tichu Actions & State Features

Type	Action	Count
Basic	Pass	1
	Win Trick	1
	Grand Tichu	2
	Tichu	2
Trade	Left	17
	Right	17
	Team Mate	17
Play Combination	Single	17
	Pair	26
	Trio	26
	Full House	468
	Straight	495
	Pair Steps	296
	Bomb	13
Special	Play Dog	1
	Give Dragon Trick Right/Left	2
	Wish After 1	14
Total Possible Actions		1415

Table A.1: Tichu Action Space

Type	Feature	Count	Range
Private Information	Hand [1]	17	0 to 4
	Cards Traded (Given) [2]	3	0 to 17
	Cards Traded (Received) [2]	3	0 to 17
Global Information	Announced Tichus	4	0 or 1
	Announced Grand Tichus	4	0 or 1
	Player Cards Counts	4	0 to 14
Trick on Table	Last Combination Type [3]	1	0 to 7
	Last Combination Height [4]	1	0 to 14
	Last Combination Length [5]	1	0 to 14
	Trick Points [6]	1	−25 to 125
	Trick Leader [7]	1	0 to 3
Card Counting	Played Cards Count [1]	17	0 to 4
	Played Combination Count [8]	7	0 to 55
	Combination Pass Height [9]	12	0 to 14
	Number of Won Tricks [10]	4	0 to 14
	Points in Won Tricks [11]	4	−25 to 125
Total Features		85	

Table A.2: Tichu State Features

[1] Count for each of the 17 different card values
[2] Value of the cards given/received
[3] i.E. single (=0), pair (=1)
[4] i.E. single 5 (=5), straight 4-8 (=8)
[5] i.E. single 5 (=1), straight 4-8 (=5)
[6] points the trick on the table is worth
[7] i.E. you are the leader (=0), player to the right is leader (=1)
[8] Count how many times the following combinations have been played during the current round: Single, Pair, Trio, Full House, Pair Steps, Straights, Bombs
[9] For the following combinations: Single, Pair, Trio, Full House, Pairsteps (of exactly 4 cards) and Straights (of exactly 5 cards) and the Opponent players: Left and Right; the lowest height of that combination the player passed on. i.E. Trick on the table is a pair of 4s and the player to the left passed (Combination: Pair, Player: Left (=4))
[10] Count of won tricks for each player
[11] Points in the won tricks of each player

Appendix B

Evaluation Matrix

	Opponent				
Agent	Behavioural Cloning	PPO (Scratch)	PPO (BC)	PPO (BC) + Tichu Model	PPO (BC) + Tichu Model (Training)
Behavioural Cloning		111.5	−3.9	−14.0	−14.3
PPO (Scratch)	−111.5		−90.1	−115.5	−122.9
PPO (BC)	3.9	90.1		−12.1	−13.1
PPO (BC) + Tichu Model	14.0	115.5	12.1		−6.8
PPO (BC) + Tichu Model (Training)	14.3	122.9	13.11	6.8	

Table B.1: Evaluation Matrix