Department of Computer Science

Submitted in part fulfilment for the degree of BSc.

Exploring the balance of skill and luck in the card game Yaniv

Rory Anderson

19 May 2021

Supervisor: James Walker

Acknowledgements

Thanks to my patient supervisor, James Walker for providing useful feedback, and to the Ray open source community for providing excellent software and documentation.

Contents

List of Figures

List of Tables

Executive Summary

Yaniv is a complex, fast, and fun card game played throughout the world. Its design allows for rich strategy and the thrill of taking chances. Often while playing the game, conversations about the level of skill really required to play Yaniv well, or if it is indeed just luck.

This project addresses the question through a series of experiments which explore the effect chance events have on the game. First we look at the effect starting a round has, then the effects that starting a round with a specific hand or set of hands has.

To explore these questions required a simulation environment with several agents which could play the game and simulate different levels of skill. I implemented two agents based on human strategies – a novice and an intermediate player. I also created an environment in which to train a reinforcement learning agent – essentially an AI which learns to play the game through trail and error. This agent was used as a measure of continuously increasing skill.

I hypothesised that a player's level of skill would have an impact on the effects of the chance events. A strong player might be able to better exploit a good hand, conversely a good player should also be better able to defend against their opponent getting a good hand. This interplay was explored using the RL model's historical save-files – as the agent trained it saves a copy of itself at intervals. The idea behind this is to have a continuous measure of increasing skill: as an RL agent trains, it gets better at the game.

I found that starting a round does have a positive impact on a player's win rate, though it varies from player to player and their opponent. In self-play (when an agent plays a copy of themselves) the player going first was able to win about 6% more games.

The story was similar with starting hands. I broke them down into classes of hands, like 4 of a kind, or full house (similar to Poker). I found that starting with a bigger combination of cards in your hand lead to better chances of winning, up to a 90% win rate for a 5 card straight. But this was highly dependent on strategy. The intermediate agent was designed to put down the largest discard it could, and so took full advantage of the

5 card straight, whereas the RL model hadn't encountered enough 5 card straights in training to know what to do with it.

Finally, I used the RL models to conduct the same experiments but over the training life of the agent, to try and see the effect that skill had on these lucky events. The results, whilst interesting, were inconclusive. I discovered skill, win rate, and strategy were all part of the puzzle. Some strategies might win more games, but also score more points (which is bad), and others might win fewer but get less penalties. Likewise, some strategies are good at dealing with different parts of the game.

This project contributes a new environment and several baseline agents for developing and evaluating new and existing reinforcement learning algorithms. Further work would look at improving the RL agent's performance and introducing new heuristic agents.

This project does not use animal or human participants, so there are limited ethical implications. The method and implementation have been checked against the ACM code of ethics and the University of York code of practice on research integrity.

1 Introduction

Social card games (e.g. Bridge, Rummy, Poker) have been popular in Europe since the 14th century and have hundreds of variations [1]. One of the key reasons behind their popularity is the blend of intellectual stimulation and social entertainment they provide. The games are often a mix of strategy that tests one's skill and chance which adds the excitement element.

A common discussion whilst playing these games is how much skill a player used to win, or if they just "got lucky". The winner will almost certainly declare their skill lead them to victory whilst the losers are sure they just picked up the right card at the right time. This project aims to explore the influences that luck has on a game and how much skill is required to overcome a lucky opponent.

It's important here to differentiate between the two concepts of luck and chance. Chance is the random actor – when you draw a card off the top of a shuffled deck it's down to chance what you get, it should be completely random. Luck is the human concept that chance events can be impartial [2]. Someone might possess "good luck" which skews chance events towards favouring them, or the opposite with "bad luck". In this project we are mainly concerned with good luck or *lucky* events – those which are perceived to have a positive impact.

Yaniv is an unusual draw and discard game with some similarities to Rummy. The aim is to reduce the score of the cards in your hand to below a threshold at which point you can call the game and compare you hand to the other players, winning if you score the lowest or receiving a heavy penalty for getting it wrong. There's plenty of opportunity throughout a game for a player to strategize and plan, likewise each time a player draws a card chance comes into play. It offers the perfect environment to analyse skill and luck.

Reinforcement learning (RL) is a hot topic in the world of AI and has seen a recent explosion in development. Its aim is to solve sequential decision-making problems. Games are a very common test bed for reinforcement learning algorithms, as they've often been extensively studied and are rigorously defined [3]. As an RL agent learns to play a game its skill at the

game can be said to increase. I will use this along with several heuristic agents as variable skilled agents.

1.1 Aims and Objectives

This project is about exploring luck and skill, so it's important to be able to control both of these parameters in either the environment (luck) or players (skill). First we must be able to simulate the game mechanics, and allow for options which manipulate chance, simulating luck. Then we need to create models to play the game. It's possible to synthesize different levels of ability using both heuristics and reinforcement learning techniques.

With a simulation environment and set of agents we can move onto manipulating chance, forcing unlikely or lucky events to occur to visualize their impact on the game. Doing this throughout a range of skill levels will give us an indication of how skill impacts lucky events.

To achieve this I followed the following steps:

- Create an efficient configurable environment that can simulate Yaniv.
- Develop heuristic rule-based agents that mimic human behaviour as skill benchmarks.
- Use Deep Reinforcement Learning methods to train several agents
- Use the heuristic agents along with the RL agents in a simulation environment to run experiments described in Section 3.2.

1.2 Rules of Yaniv

Yaniv can be played by 2 or more people, though is best for 2 or 3. More players slow the game down and can remove some of the more interesting strategies [4].

Setup

Yaniv uses a standard 54 card deck with 2 jokers. Each card is assigned a score: ace is worth 1; number cards are worth their face value; picture cards are worth 10 and jokers score 0.

5 cards are dealt to each player one at a time. The remaining cards are then placed face down as the draw pile. The top card is turned over and placed next to the draw pile, this forms the discard pile.

Play

The starting player of the first round is predetermined, in subsequent rounds the player who won the previous round goes first.

On their turn a player can do one of two things:

1. Discard a combination of cards and pick one up
2. Call "Yaniv", ending the game. To call Yaniv the total score of the player's hand must be less than 8.

A player may discard the following:

- Any single card.
- A set of two or more cards of the same rank (eg two 5s, three queens)
- A straight - a sequence of consecutive cards of the same suit. (eg 2, 3, 4 of Clubs). Ace is always low, king is always high and jokers can be used as wildcards.

Once the player has discarded they must pickup one card, either from the top of the draw pile or from the top of the discard pile. If the previous player discarded multiple cards, only the top or bottom of the set may be picked up. Straights must be put down in sequential order, ascending or descending.

When a player calls Yaniv, they are stating that they have the lowest scoring hand. To call Yaniv a player's hand must be less than 8. Once Yaniv is called, the games ends and all the players reveal their cards. If the player who called Yaniv does have the lowest overall score then they win. If the caller doesn't have the lowest score they lose and the player with the lowest score wins. Miss-calling Yaniv is known as an Assaf.

Scoring

Scores are then assigned as follows: the winner scores 0; the losers score their hand total; if someone miss-called Yaniv they score 30 points plus the total of their hand.

Scores are then added to each player's total. Any player who has a total greater than 100 is knocked out of the game, and the next round continues

with the remaining players. The winner of the game is the last person left with less than 100 points.

There are two extra overall-game mechanics: if a player wins three rounds in a row they score -10; if a player lands on exactly 100 points their overall score is halved to 50.

1.3 Research Questions

The broad question this project aims to address is how much skill impacts lucky events and vice versa. It will give players the tools to know whether their opponent did just *get lucky*, or if they showed superior skill.

One of the most obvious areas to address is the effect of going first. In Yaniv the starting order is predetermined, either randomly at the start of an overall game or the winner of the previous round starts. Win streaks are fairly common in the overall game, where a player will continue to win many rounds in a row, which begs the question; how much better is it to start a round?

The first card that is turned over in the game is the top card of the deck after dealing the hands. That means it's a random card which is more likely to be desirable than what your opponent will discard, given a players propensity to hold on to low cards. So just how much advantage is gained by having the choice of the first card? Hypothesis 1.1 aims to test this.

Hypothesis 1.1. *Going first increases the likelihood of winning the game.*

As skill increases an agent's ability to exploit lucky events should increase, but also their ability to defend against a lucky opponent. Therefore, it is important to investigate the effects of lucky events over a set of skill levels which Hypotheses 1.2 and 2.3 aim to do.

Hypothesis 1.2. *As an agent's skill increases the effect of going first decreases.*

There are 2,598,960 possible 5 card hand configurations. Some of these are better to start with than others. As in poker, Yaniv hands can be broken down into classes, based on what they contain, they are detailed along with their frequency in Table 3.3.

Likewise, each starting hand can be scored using the Yaniv rules described in Section 1.2. Fig. 4.2a shows the probability of being dealt a hand of a certain score.

There is a clear advantage to starting with a hand score below 8; it means you can call Yaniv immediately. There are 32 5-card hand combinations which score below 8 which means if you're dealt one there's a 99.9988% chance that your opponent has a score >8 which means you can call the game and will likely win.

Since the goal in Yaniv is to reduce the number of points in your hand to less than 8 it follows that starting with fewer points in your hand will increase your chances of winning, which is what Hypothesis 2.1 aims to test.

Hypothesis 2.1. *Starting with a lower hand score will increase your chances of winning.*

Similarly, starting a game with a combination of cards that you can put down immediately should have a positive impact on your chances of winning. One of the key strategies in Yaniv is to collect combinations of cards, so that you can discard multiple cards at once. Each time you discard you only pick up one card, so discarding multiple cards is an efficient way to reduce your hand score. Starting with a combination in your hand is a bonus as it means you're immediately able to reduce the number of cards in your hand. Hypothesis 2.2 aims to test the effect this has on a player's success.

Hypothesis 2.2. *Starting with a larger legal discard combination in your hand will increase your chances of winning.*

As with Hypothesis 1.2, Hypothesis 2.3 aims to test how the effect of these events changes with an agent's skill level. A more skilled opponent might be able to take better advantage of starting with a pair, eventually building it into a bigger combination. Likewise, a skilled opponent should be able to neutralise some of the effects that starting with a better combination gives.

Hypothesis 2.3. *As an agent's skill increases, its ability to exploit better starting hands increases.*

2 Related Work

Artificial intelligence (AI) in games has seen significant advances in recent years. With the advent of deep reinforcement learning, AI researchers have been able to achieve superhuman performance in many game spaces. In particular games with perfect information have been subject to success – those where the entire game state is visible to each player. In 2016 DeepMind [5] mastered the game of Go, previously held as one of the greatest challenges in AI due to the large intractable search space. They used neural networks to approximate the value of the board state and search techniques select the next move. With this technique, AlphaGo was able to defeat the Go world champion Lee Sedol.

AlphaGo learned the game by playing games with both professional and amateur players, acquiring human domain knowledge. In 2017 DeepMind published a new algorithm, AlphaGo Zero, which required zero domain knowledge [6]. It learned only by playing games with itself (self-play.) The new approach was able to outperform AlphaGo. Later DeepMind [7] was able to generalise the approach to Chess and Shogi achieving similarly groundbreaking results.

Imperfect information games, those where only part of the state is visible to each player, present a different challenge. Any player that wants to do well in these games will have to be able to reason about the hidden state. In games likes Poker and other card games this explodes the combinatorial complexity. Recent advances have made good progress on this though. Facebook recently released ReBeL a generalised self-play RL+search framework that converges to a Nash equilibrium in two-player zero-sum games [8].

2.1 Challenges

Before looking closely at the related RL literature, it is worth noting the challenges that creating a successful RL agent in Yaniv poses.

- **Imperfect Information**: A player has access to a limited subset of the current game sate, i.e., the cards in their hand and the cards on

the discard pile. A successful agent must be able to reason about unknown states.
- **Multi-agent**: Yaniv is a multi-player game. Using an MDP would require the other players to be part of the environment rather than separate entities interacting with the environment [9]. This means that any opponent is stationary.
- **Large State and Action Space**: Due to the combinatorial nature of card games the action and state spaces are much larger than the classic environments that RL have had successes in [10].
- **Sparse Rewards**: In Yaniv, your actions do not have immediate consequences – only at the end of the round do you find out whether you win. So the learner has to figure out what series of actions lead to the reward at the end of the round, whether picking up that card 6 turns ago was a good choice or not.
- **Multiple Turn Phases**: Needs to learn 3 tasks: whether to Yaniv, what to discard, and what to pick up.

2.2 Reinforcement Learning

Reinforcement learning (RL) is learning through interaction. The learning agent interacts with its environment, observes the results of its actions, and then can learn to change its behaviour based on the reward given to it. This is essentially trial-and-error learning and is similar on the way children learn to interact with the world [11].

Reinforcement learning is a framework for solving optimisation problems. The typical RL problem can be described as a control loop where the agent learns from interacting with an environment, receiving state descriptions and rewards in order to make decisions which lead to the maximum expected reward [12]. The agent observes a state s_t at time step t, then interacts with the environment by taking an action a_t. This leads to a new state s_{t+1} and the environment generates a reward r_{t+1}.

2.2.1 Markov Decision Process

The environment that an RL agent interacts with is usually formed as a Markov decision process (MDP) [9]. MDPs are a mathematical model for sequential decision-making and control problems. A finite MDP can be defined by the 5-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ where:

- $\mathcal{S}$ is the set of all possible states, where $s_t \in \mathcal{S}$ represents the state

of the environment at time step t.
- $\mathcal{A}$ is the set of all possible actions in the environment. $\mathcal{A}_t \subset \mathcal{A}$ is the subset of actions which are legal at time t.
- $\mathcal{P}$ is the state transition probability function. $\mathcal{P}(s_{t+1}|s_t, a_t) \in [0, 1]$, where $s_t, s_{t+1} \in \mathcal{S}$ and $a_t \in \mathcal{A}$. It defines the probability of going to state s_{t+1} form state s_t after doing an action a_t.
- $\mathcal{R}$ is the reward function which maps the immediate reward an agent will receive after doing an action a_t in state s_t and transitioning to state s_{t+1}. $\mathcal{R}(s_t, a_t, s_{t+1}) \in \mathbb{R}$
- $\gamma \in [0, 1]$ is the reward discount factor. Which is the relationship between future and immediate rewards. If $\gamma = 1$ all rewards future are considered equally to the present reward, if $\gamma = 0$ then the agent only considers the present reward [13].

Solving an MDP yields a policy $\pi : \mathcal{S} \to \mathcal{A}$ which maps states to actions. An optimal policy π^* is a policy which maximises the expected return (2.1) for every state [14]. A common technique to solve MDPs is the value iteration algorithm [15], which requires a complete representation of the states, actions, rewards, and transitions of the environment. Often this information is difficult or impossible to obtain. To solve this RL algorithms learn by interacting with the environment, gathering experience as they go [11].

$$G_t = r_{t+1} + \gamma r_{t+2} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{2.1}$$

In order for a problem to be described by an MDP it must satisfy the Markov property, i.e., the future depends only on the current state and action, not the past. Many problems don't satisfy this constraint, including Yaniv. There are extensions and modifications to the MDP which are able to model many different environments.

To fully model Yaniv an extensive-form game can be used. It is able to model the challenges mentioned in Section 2.1 [10], [16]. The model is grounded in game theory and allows for multi-agent, partially observable games.

2.2.2 RL methods

The aim of any RL algorithm is to learn an optimal policy for an environment. That is a policy that maximises the value of every state. There are many different approaches to this using RL. Many factors should be considered when choosing the right RL algorithm, such as the size and shape of the

state and action space (continuous vs. discrete actions), computational resources available, and even the expert human data available.

RL algorithms learn on trajectories, that is the sequences of state, action and reward through an episode. The difference between on and off-policy algorithms is how they generate the trajectories learned on:

- On policy algorithms use the policy that is learning to sample actions which means that the RL algorithm uses the current iteration of the policy to generate the trajectories that are fed into the algorithm to learn.
- Off policy algorithms use trajectories generated by a different policy to the one being trained. This might be another policy specifically designed to explore the useful parts of the state space or trajectories pre-generated from some other source, like human expert play.

The trade-off between on and off policy is mostly sample efficiency. On policy learning has to always generate new samples and so has a low sample efficiency. This can be a positive as it means more stability since the algorithm directly tries to optimise the policy's performance. With off-policy learning, you can reuse old samples which means algorithms can achieve a much higher sample efficiency [11].

Popular on-policy algorithms include: Asynchronous Advantage Actor Critic (A3C) [17], Trust Region Policy Optimisation (TRPO) [18], Proximal Policy Optimisation (PPO) [19].

Popular off-policy algorithms include: Deep Deterministic Policy Gradient (DDPG) [20], Deep Q-Network (DQN) [5], Soft Actor Critic [21].

Most RL algorithms either use a value or policy-based approach, or a mix of the two. Value-based algorithms learn a value function that approximates the value of the current state and use that to derive their policy. Q-learning is similar in that the function being optimised is the value of a given state-action pair, $Q(s, a)$ [22]. The actions to be taken is calculated by:

$$a(s) = \underset{a}{\operatorname{argmax}}\, Q(s, a)$$

Policy-only and value-only based algorithms both have serious downsides that make them difficult to scale to complex environments. In 2000 Konda and Tsitsiklis [23] summarised their drawbacks:

- Policy-only algorithms estimate the gradient of the policy's performance and update in the direction of improvement. Gradient estimators

have may have a large variance which causes instability. Policy updates are calculated independently of previous updates which means there is no learning done on previously acquired information.

- Value-only algorithms use all resources to calculate a value function which is then used to generate an optimal policy which is an indirect way of getting a policy.

Actor-critic (mixed policy and value) methods aim to overcome some weaknesses of using a policy only or value only method. The actor refers to the policy and the critic refers to the value function approximation. It's often the case that the actor and critic will share the same neural network [17].

2.2.3 RL in Imperfect Information Games

Many card games have imperfect information, including Yaniv. It's a challenge in AI research as any successful agent must be able to reason about information it doesn't have access to. The hidden states mean that game tree traversal is difficult as there needs to be a separate branch for each possible state [24].

RLCard and OpenSpeil are both reinforcement learning toolkits which are designed to aid in game AI research [25], [26]. Neither of these toolkits implement Yaniv.

Successful agents in card games require some domain knowledge and often include some sort of search, either at the training or evaluation state [8], [27], [28].

Model and search free approaches have also seen some success. Charlesworth 2018 [29] trains an agent using PPO on the trick taking game Big 2.

2.2.4 Self-play

Multiagent online reinforcement learning requires trajectories to be gathered whilst training the agent. In a multiagent setting the opponents to the learning agent are important. In order to learn successful the agent needs to face correct challenge for its current skill level [16], [30]. Imagine if you just learned the rules of a game, and your only opponent was an expert, or if you are an intermediate player and can only face a novice – the experience you can gather is less valuable than playing someone of a similar skill level.

It's important to match the level of skill between opponents during training. One easy way to achieve this is with self play: the opponents are played by the learning agent. That is, the same policy is used to generate actions for each player – this is known as naive self play [30].

There are different approaches to self play, which centre around how the opponent's policies are sampled. In naive self play the opponent is just the most recent version of the learning agent. A more sophisticated approach is to sample historical policies. This is ensures that the agent doesn't *forget* strategies [31].

2.3 Synthesis

Reinforcement learning continues to grow and tackles a constantly evolving pool of challenges. This project aims to add a new environment and baselines to test new RL algorithms against. Yaniv is particularly interesting due to its large state and action space, and the possible depth of strategy. An intermediate player can make decisions based on an instant observation, but to really master the game you need to use the game history to reason about your opponent's hand and what might still be in the deck. Yaniv offers the scope that a good RL environment should.

The main goal of this project is to further understand Yaniv, and the balance of luck and skill in the game. Using RL it's possible to model an improving player and run simulations using agents of different skill levels. There is no research on the game of Yaniv.

3 Methodology

In this chapter I describe in detail the steps outlined in Section 1.1. In Section 3.1 I discuss the method used and decisions made to create the environment and agents required. In Section 3.2 I discuss the experiments designed to test the hypotheses outlined in Section 1.3.

3.1 Implementation

In order to simplify the process of training RL agents a few changes have been made to the rules mentioned in Section 1.2. First, jokers are removed – keeping them in increases the combinatorial complexity of the game, massively increasing the number of actions an RL agent has to learn which is difficult to deal with using my limited compute resources.

Second, we're only concerned with rounds, rather than the overall game. The meat of playing Yaniv comes from the rounds. There's a bit of meta-game you can play, most of it comes down to strategies to hit exactly 100, but again this would increase the state needed to represent the game and add complexity without much benefit to our research.

3.1.1 Simulation Environment

Any game is interacted with through a series of actions and observations. A player will make an observation, then take an action that results in a new observation. This cycle is implemented for Yaniv in the `yaniv_rl/game` module in a series of Python classes.

As with most games, Yaniv only allows certain actions to be taken at any point. For instance, you cannot discard a card that you do not hold in your hand. This narrows down the list of possible actions you can take for a given state to the *legal actions*.

Yaniv has two phases per turn, first, the discard phase, the player can discard a legal combination of cards or call Yaniv if their total hand score is

less than 8. In the second phase, the draw phase, they can either pick up from the discard pile or draw a card from the deck. A human will usually consider the two phases together and make one action, draw and discard.

The order of the cards you discard are important in Yaniv, as it dictates what the next player can pick up – they can pick up either the top or the bottom card. With this in mind, there are 1072 possible legal card combinations that can be discarded. Some of these actions are redundant, however. For example, when discarding a pair the order of the pair is irrelevant since the next player can always pick up either card. When discarding three or four of a kind, which cards are in the middle is important, as they will not be available to the next player. This can be used to deny cards you know the player is trying to collect, by hiding them in the middle of a discard.

Using this information it is possible to reduce the total number of discard combinations to 484 by ignoring actions with duplicate effects on the game. The algorithm to generate the actions is located in `yaniv_rl/game/jsondata/gen_discard_actions.py`. Discard actions are represented as a string containing the cards to be discarded. A card is encoded as a 2 character string `SUIT + RANK`, where suit is one of `["C", "D", "H", "S"]` and rank is `["A", "2", "3", "4", "5", "6", "7", "8", "9", "T", "J", "Q", "K"]`. So the ace of spades would be `"SA"` and the 5 of Hearts `"H5"`. The possible ways of discarding a three of a kind are: `["D2C2H2", "C2D2H2", "C2H2D2"]`.

In total there are 488 possible actions a player can take, the 484 discard actions, a `yaniv` action, and 3 pick up actions; `pickup_top_card`, `pickup_bottom_card`, and `draw_card`. When the previous player has only discarded single card the two pickup actions are synonymous.

3.1.2 Heuristic Agents

One of the important questions in Section 1.3 is how skill effects luck, and how different events change with more or less able players. In order to fully test the differences in skill it's important to be able to simulate different levels of ability. One way to do that is to craft heuristic agents based on human knowledge.

To do this I have created three simple agents. The first is a random action agent which takes random legal actions. This simulates a player with no skill, they only know the rules of the game (i.e. what actions they're allowed to take) but cannot use strategy.

Next is the novice agent, which aims to simulate a new player who has just

	Random	Novice	Intermediate
Random	0.0674	0.0151	0.0111
Novice	0.9417	0.4797	0.3231
Intermediate	0.9842	0.6746	0.4953

Table 3.1: Rule-based agents' win rates. Represents a tournament Row vs. Column with 10000 games played. A win rate of 0.5 means that the agent on the row won 5000 games against the column agent.

learned the rules and the basic objective. Often when a new player picks up the game they just put down the highest scoring card in their hand and don't think about making card combinations. While it's possible to win like this, its inefficient.

Finally, the intermediate agent, which implements the basic strategy in Yaniv, which is to build combinations of cards – even if that means your hand score will temporarily increase. This simulates a player with a few games of experience, they understand the importance of building multiple card combinations but lack further strategy.

Another important reason for the heuristic agents is to benchmark the RL agent as it trains. Yaniv is a relatively new game with a small player base and so there are no available baselines to test our reinforcement learning agents' performance against.

The strategies for the three agents are outlined below:

1. **Random Action**: Uniformly samples legal actions.

2. **Novice Strategy**: Always call Yaniv if the action is available. Discard the highest scoring combination of cards. If the available discard is less than 2, pick it up otherwise draw from the deck.

3. **Intermediate Strategy**: Checks to see if picking up from the discard pile will increase the number of cards that can be discarded next turn. If so pick the highest scoring discard combination that doesn't include the cards used for next turns combinations. Otherwise, act like the novice strategy.

    - This strategy aims to reduce the number of cards in its hand as quickly as possible, usually by making pairs.

Table 3.1 details the win rates of each agent. When playing itself the random action agent will often end in a draw.

Feature	Size	Description
Current Hand	52	Players current hand
Dead cards	52	Discarded cards which are no longer in play
Top Card	17	Available cards to pick up from the previous play
Bottom Card	17	
Known Cards	(n-1) * 52	Known cards in the opponent hands
Opponent Hand Sizes	(n-1) * 6	One-hot encoding of opponent hand size
	58n + 80	

Table 3.2: The Yaniv State Encoding.

3.1.3 Gym Environment

In order to train an RL agent, the gym environment needs to encode the state into a vector of inputs. The gym environment provides a wrapper around the human-friendly game environment. This project uses Ray's RLlib framework and algorithm implementations [32]. RLlib provides a base interface `MultiAgentEnv`, an extension of OpenAI's gym environment for use in a multi-agent setting [33], [34].

Observation Encoding

One of the more important steps when designing an RL environment is to come up with an efficient and sensible way to encode the current state into a set of input features for a neural network. The actual learning performance of an RL algorithm can be limited by the design [11]. The full state of the game would include the current player's hand and all the actions taken before the current time step. As there is technically no limit to how long a game of Yaniv can last, encoding all this information would lead to a non-stationary and potentially huge state space. Both of these problems can be mitigated by designing a state encoding using human game knowledge.

Table 3.2 shows the initial state encoding chosen for this project. There are 4 main features. They aim to approximate human knowledge at any given step in time. A player knows what is in their hand, what cards have been played and are out of the game, the two possible cards that can be picked up, and how many cards each opponent has left in their hand. A human with perfect memory should also know what cards any opponent has picked up from the discard pile. This is encoded in the known cards feature for each opponent. This can be vital information as it should help avoid making wrong calls.

In two-player the state vector is of length 196 and in three-player it is 254. It is also possible for an agent trained in three-player to play in two-player

games as the inputs for the second opponent are ignored and set to zero. Likewise for a two-player trained agent.

The current hand, dead cards and known cards are encoded in a vector of length 52, where all inputs are zero unless the card ID is present in the feature. The discards are encoded in a two-plane vector, one for the rank and one for the suit and then concatenated. This state representation was chosen as it includes most of the information that goes into making a decision in Yaniv in a compact way.

Action Representation

As Yaniv is a multi-phase game it is important to decide how the agent will interact with the model. It can either do its two actions sequentially, i.e. calling step with the discard action, then receiving a new observation and calling step again with the pickup action. Or it can take two actions at once, telling the environment to step with both its discard and pick up action.

The output of the policy network is the same length as the number of actions. It's normalised by a `softmax` function, so the output is a list of probabilities each one corresponding to a distinct action.

To implement the second action representation you have to create a separate action for each combination of discard and pickup, i.e. the cross product of the discard action and the pickup actions. The Yaniv action is still a separate action. This means that the total number of actions is now almost tripled to 1,462. This causes the algorithm to train far slower as shown in Fig. 3.1.

Reward Function

Reward shaping is another important part of the gym interface. The reward function is what determines the reward given to an agent when they do an action. It is possible to use dense rewards, which give a reward signal to the agent at every step, or more sparse rewards where an agent takes many steps before receiving a reward. The reward signal is extremely important [35], as it's what the agent uses to learn, it's what the agent is trying to maximise.

Yaniv is a relatively simple game where the reward comes at the end. Either the agent wins, loses, or the game runs out of cards and a draw is called. The simplest reward scheme would assign a reward of +1 for a win, -1 for a loss, and zero for a draw [36].

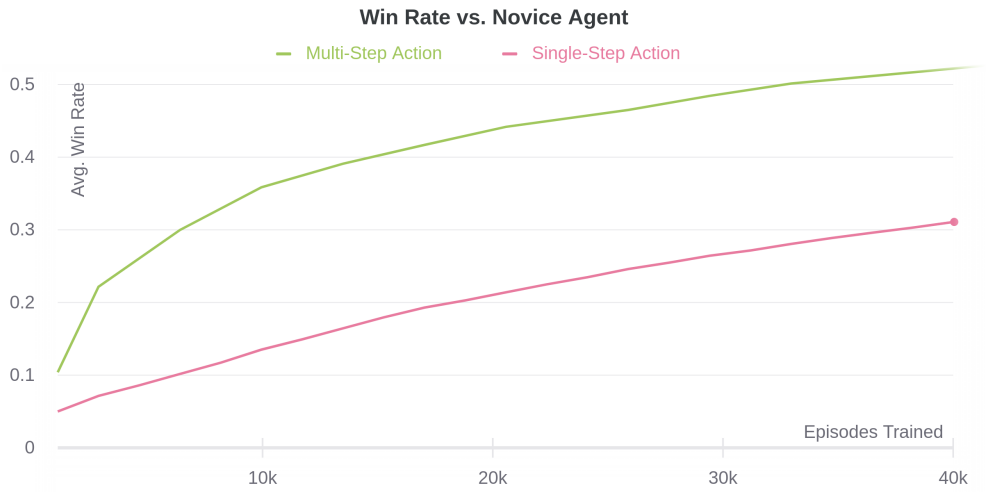


Figure 3.1: Win rate vs Episodes Trained. This uses the novice rule agent to evaluate its performance for 500 episodes. This compares the two different action designs, single-step (where both discard and draw actions are taken together) and the opposite, multi-step.

At the end of a Yaniv hand, the losers score points based on what is left in their hand. This number contributes to an overall game, where the aim is to score as few points as possible. To integrate this into the learning scheme a different reward function is proposed.

$$r = \begin{cases} max(losingScores)/scoreCutoff & \text{if win} \\ 0 & \text{if draw} \\ losingScore/scoreCutoff & \text{if loss} \end{cases} \tag{3.1}$$

The reward described in (3.1) is proportional to the scores at the end of the hand. The losers of the game score their hand divided by the cut-off variable and the winner scores the absolute value of the minimum reward. The score cut-off is set at 30. The aim of an agent is to maximise the score of the opponent, whilst minimising its own score. Losing by -2 has far less impact on the overall game than losing by -20. This reward scheme tries to incentivize agents to cause their opponent to lose by as large a score as possible, whilst minimizing its own losses, either by winning or scoring low.

Fig. 3.2 shows the impact good reward shaping has on the training outcome. The green plot uses the simple reward scheme and the red plot uses (3.1). The reward scaling idea means that an agent isn't penalised for getting close to winning, which happens with the simple reward scheme. Equation (3.1) more accurately describes the goal of the game - to reduce the overall score of your hand.

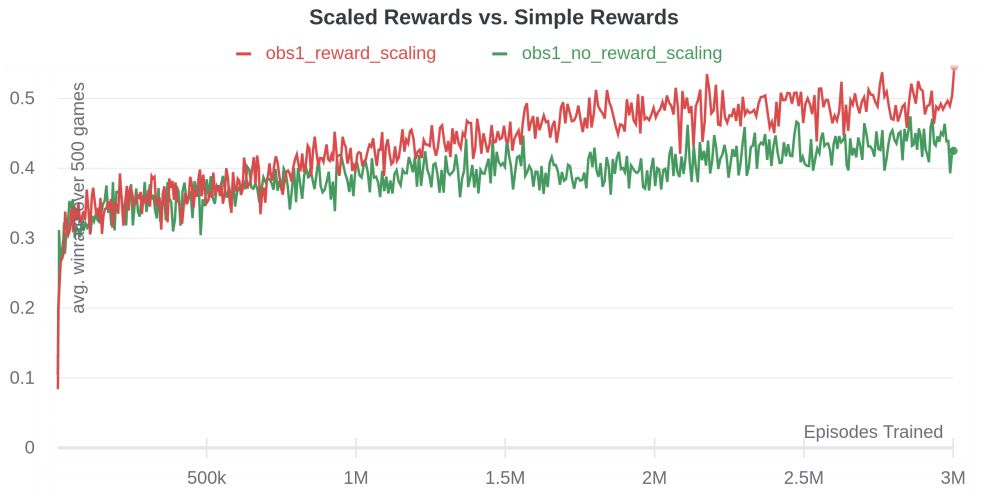


Figure 3.2: Win rate vs Training Iteration. The win rate is calculated over 500 games against an evaluation agent policy trained for 10M episodes. Both plots use A3C. Represents 20 hours of training time.

3.1.4 Training RL Agents

To streamline the development process this project uses RLlib and Ray as the RL and distributed computing framework [32]. RLlib has implemented and benchmarked many popular RL algorithms.

For this project I decided to use two actor-critic methods PPO [19] and A3C [17]. They're both online policy gradient algorithms.

Self-play

I implemented δ-Uniform Self-Play with a small menagerie size of 4 due to low compute resources [30]. The scheme trains policy 0 and uses it as player 0 when collecting samples. The opponents are sampled uniformly from the menagerie, which is populated with historical versions of policy 0 [31]. The gating function is as follows: if, after a training iteration, the trained policy has an average win rate of >0.55 then insert it into the menagerie.

As described in Section 2.2.4, this means that during training the agent will play against historical versions of itself. This is so that as the agent grows in skill, so does the challenge. The reason to sample historical policies is to ensure that the agent does not *forgot* older strategies.

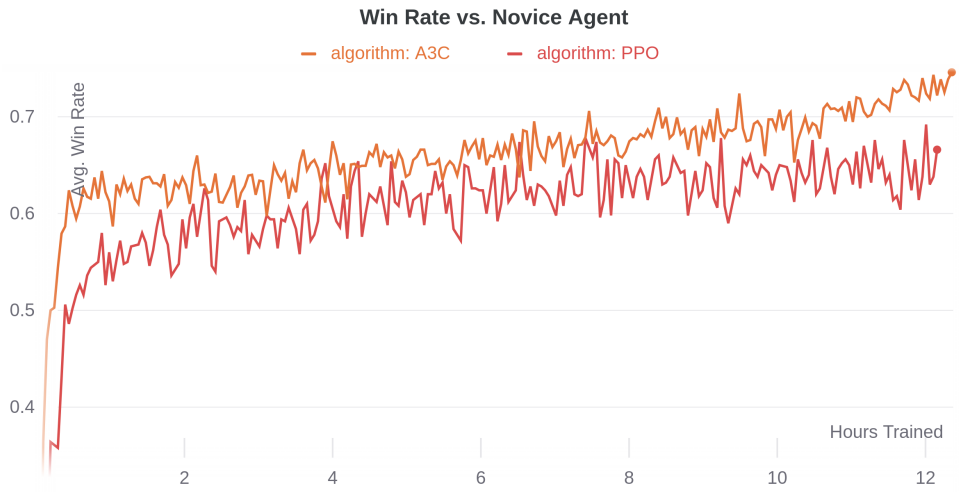


Figure 3.3: Comparison of the A3C and PPO algorithm trained in 2 player. The win rate is calculated over 500 games against an evaluation agent policy trained over 12 hours.

Training

Agents were trained over a series of environment configurations and hyperparameters. Overall the A3C algorithm performed best with the limited hyperparameter search I was able to conduct. It is simple and robust enough to provide suitable defaults out of the box, only needing to tune batch size parameters to achieve desirable results. PPO also offered a good option, but it is more sensitive to hyperparameters and with my limited compute resources tuning was difficult. A3C is slightly better suited as it scales better with CPUs versus PPO. A comparison of the two is shown in Fig. 3.3.

A3C is able to converge to a good level of play quicker than PPO, and then reaches a higher plateau. For this reason the rest of the experiments are done using A3C, as it trained quicker on the resources I had available with minimal hyperparameter tuning.

In order to run experiments I had to chose one model which performed best and had historical policies saved. The configuration is listed in Appendix A. The model's policy is saved every 10 training steps (approximately 3,000 episodes.) Fig. 3.4 shows the results of training over 10k steps.

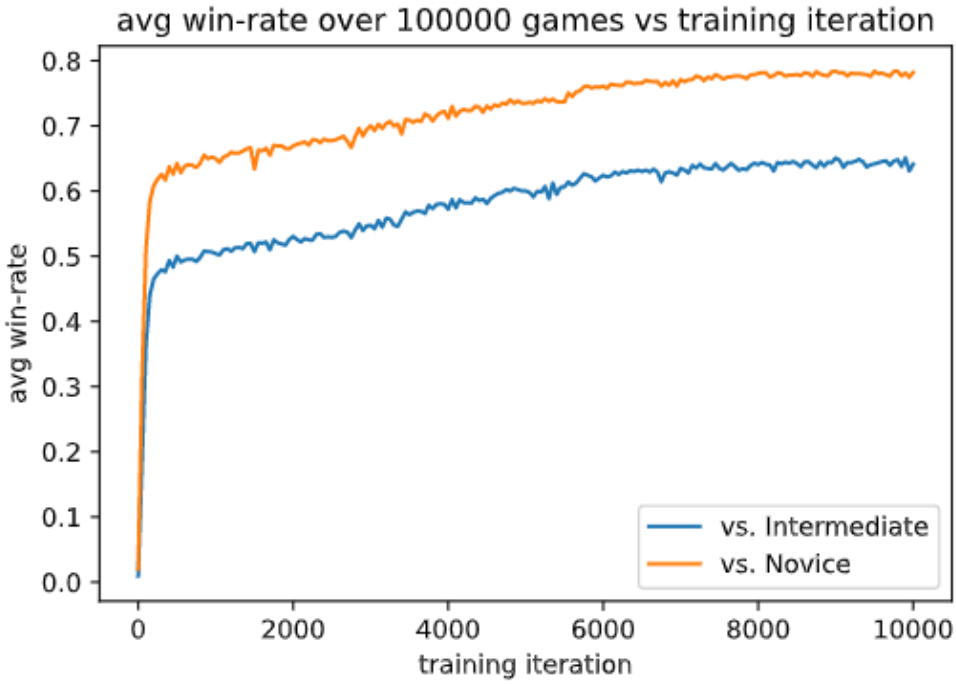


Figure 3.4: Performance of the model over training iteration. Evaluated at every 100th iteration against the novice and intermediate heuristic agent for 100,000 games. Trained with the configuration listed in Appendix A.

3.2 Experiment Design

The ideas of luck and skill are difficult to quantify together. One way to try and eliminate the effects of chance is to run simulations over many thousands of games [37]. Allowing us to gain a truer measure of skill.

In order to see the effects luck has on the game it is possible to synthesise events which have a positive impact on the player but may be unlikely to occur. To investigate this idea I propose two sets of experiments, the first looks at the effect of starting a game, and the second at the effect of a player's starting hand.

All of these experiments involve simulating games with the environment and agents developed in Section 3.1. In each experiment one of the events mentioned in Section 1.3 is fixed to simulate luck, while all other parameters of the environment remain standard.

3.2.1 The effect of going first

In order to test Hypothesis 1.1 – the effect going first has on a player's win rate – I used the rule based agents developed in Section 3.1.2 and the same environment used to generate Table 3.1 but with the starting player fixed over the tournament.

For each combination of the heuristic agents, set up a tournament to simulate games. Fix the starting player to the first agent, simulate 10,000 games, then fix the starting player to the second agent and simulate another 10,000 games. For the case when the agent plays itself, it is only necessary to simulate one set of games with the starting player fixed. The win-rate of the starting player is then recorded. The results are shown in Table 4.1, along with the increases in performance over Table 3.1 shown in brackets.

In order to test Hypothesis 1.2 I use the RL model's saved policies as a measure of increasing skill. Every 10th training iteration the RL model saves the current version of its policy. Using these policy snapshots its able to run experiments over the RL agents training life, which simulates the idea of getting better. Fig. 3.4 clearly shows the increase in skill over training iteration against the rule based agents.

At each saved model 10,000 games are simulated with player 0 going first each time. The win rate of player 0 at each model is plotted on Fig. 4.1a.

Class	Description	Example	Frequency
5S	5 card straight	C2 C3 C4 C5 C6	36
4K	4 of a kind	C2 H2 S2 D2 C7	624
4S	4 card straight	C2 C3 C4 C5 D4	1848
3S2K	3 card straight and a pair	C2 C3 C4 D6 S6	2796
3K2K	3 of a kind and a pair	C2 D2 H2 S4 H4	3744
3S	3 card straight	C2 C3 C4 H5 S3	45144
3K	3 of a kind	C2 D2 H2 S6 H5	54516
2K2K	Two pair	C2 D2 H4 S4 H7	2376
2K	Pair	D7 C7 H2 D6 H3	1202604
X	High Card	D7 S9 H2 C5 D3	1287648

Table 3.3: Yaniv hand classes.

3.2.2 The effect of starting hands

In order to test Hypothesis 2.1 – the effect of starting a round with a hand of a specific score – I have simulated games where the starting hand of one player is forced to be of a certain score. I used one RL agent to make decisions for both players. 20,000 games were simulated per starting hand score. Results in Fig. 4.2b.

In order to Hypothesis 2.2 I used both the intermediate agent and RL Model. The first thing to do is determine the influence starting hand has during self-play. This eliminates the difference in skill between two agents. For each hand class simulate a number of games where the starting hand of player 0 is sampled from the current hand class, while player 1's hand is sampled randomly from the remaining deck. The results of this can be seen in Fig. 4.3a.

Next it is useful to do the same experiment but with different opponents. Fig. 4.3b shows the results of the intermediate model vs the 10,000th iteration of the RL model. The experiment is conducted in the same way but for each hand class both the agents get a turn at sampling from the class. The aim of this is to show whether an agent can become more skilled at defending against certain unfair advantages.

To test Hypothesis 2.3 I use a similar experiment to Hypothesis 2.2 and Hypothesis 1.2. For each model and starting hand class run a series of game simulations where the model is used as the policy for both players. The starting player is chosen randomly per game, and the opponents hand is sampled from the deck after a hand is sampled from the hand class. Results in Fig. 4.4.

4 Results and Discussion

In this chapter I present the results of the experiments described in Section 3.2, and their implications with respect to the research questions from Section 1.3.

4.1 The effect of going first

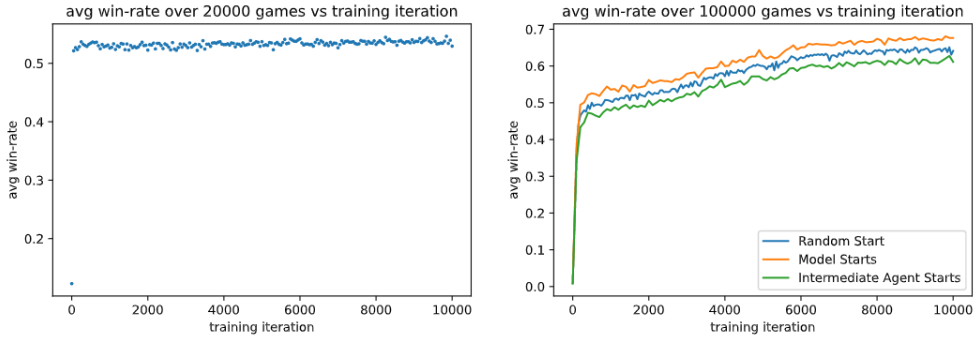
Table 4.1 shows the results from the first experiment. Games against the random agent do not see significant change as any strategy employed against a random agent will yield high win rates. The increase over the original comparison (Table 3.1) is shown in brackets.

In the match ups between the novice and intermediate agents, each agent when going first saw an increase in their win rate, between 1.75-3.52%. This shows that going first does increase the win rate of a player as hypothesised in Hypothesis 1.1. This holds for players that use a strategy. For the random agent it is unsurprising that there is little to no effect.

The notable change is in self-play mode where the agent going first increases their win percentage by over 2%. Since the agents are of the same skill level it shows the impact that going first has. As hypothesised in Hypothesis 1.2 the effect of going first is lessened. The increase of the intermediate agent over the novice agent is smaller than the other way around. I imagine this is because the stronger strategy of the intermediate agent is more significant.

	Random	Novice	Intermediate
Random	0.0655 (−0.0019)	0.0186 (+0.0035)	0.0111 (+0.0000)
Novice	0.9414 (−0.0003)	0.5149 (+0.0352)	0.3421 (+0.0190)
Intermediate	0.9865 (+0.0023)	0.6921 (+0.0175)	0.5206 (+0.0253)

Table 4.1: Win rates table (Row vs. Column) rule based agents where the row is fixed as the starting player. Averaged over 10000 games per pairing. Difference from Table 3.1 shown in brackets.



(a) Win rate vs Training Iteration in self play when the agent going first is fixed. Every 100th training iteration using 20000.

(b) Win-rate for the model versus the intermediate agent. 20,000 games simulated at every 100th training iteration.

Figure 4.1: Going first over training iteration.

Fig. 4.1 shows the results of the RL agent going first in self play as it is trained. There is a small increase of 3% over the usual 50% win rate. This stays constant over the course of training. Likewise, Fig. 4.1b plots effects of going first using the intermediate agent and the RL agent as it's trained.

Both of these plots do not show a significant change of the effect that starting a round has on win-rate as the agent's skill increases.

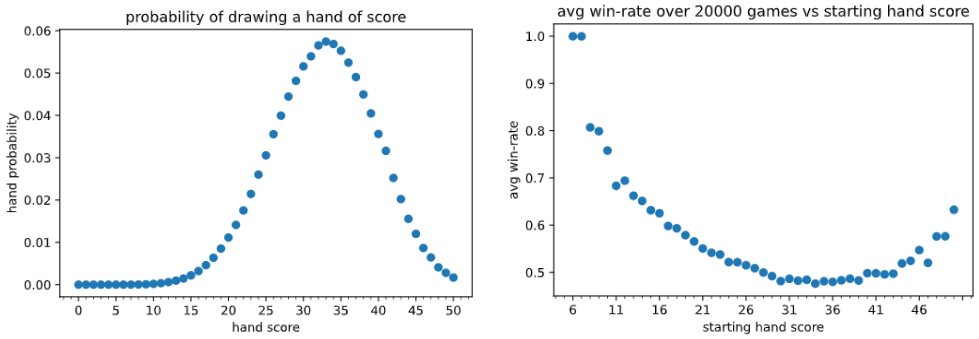
4.2 The effect of starting hands

Starting with a specific score

The first experiments looked at the effect a starting a game with a hand that had a specific total score. Fig. 4.2b shows the results. There is a clear increase in win rate as initial hand score reduces, to 1 – as mentioned in Section 1.3 starting with a hand less than 8 should mean a 99.99% win chance if the player calls Yaniv straight away. This seems to support Hypothesis 2.1.

However, as a player's hand score increases past 40 points their chances of winning increases too. This can be attributed to the fact that there are fewer possible ways of making large scores. This means that a higher value hand is more likely to have a better hand combination in it. There is a clear relationship between the score of a starting hand and its win rate, but it is not linear, and so we must reject Hypothesis 2.1.

It is worth noting, that the vast majority of the hands are distributed around



(a) Frequency of 5 card hand combinations which score. Note that the lowest possible hand score is 6 (4 aces and a two).

(b) Win-rate for a given starting hand score.

Figure 4.2: Hand score results.

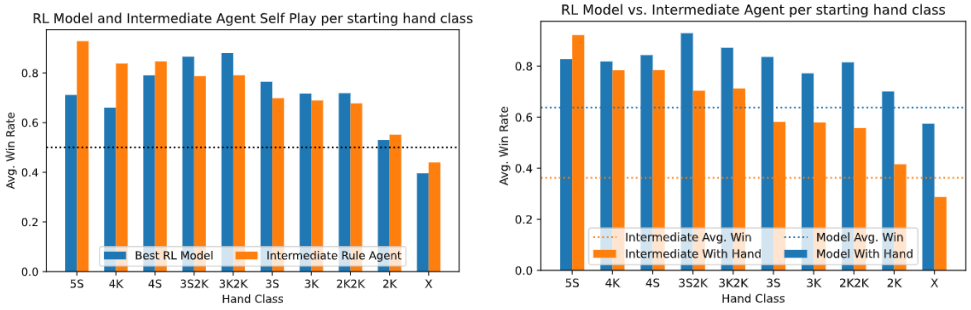
the 50% win rate area as shown in Fig. 4.2a.

Starting with a specific hand class

Fig. 4.3a shows the results of the experiment to test Hypothesis 2.2. Using the intermediate agent as an example, in self play mode there is clear evidence to accept Hypothesis 2.2. The hand classes can be broken down in to 6 categories: 2X, 2X2X, 3X, 3X2X, 4X, 5X (3X2X = 3 card combination + 2 card combination). 2X2X and 3X have similar win rates, then there is a marked increase to 3X2X, again to 4X and again to 5X.

However, the RL model behaves differently to the intermediate agent. It's better able to take advantage of 3K2K but at the higher order combinations it does worse. This is understandable considering the frequency of 5S and 4K (shown in Table 3.3). Since it is unlikely that the agent has encountered situations where it can put down a 5 card straight or 4 of a kind, the agent doesn't know how to deal with these rare occurrences. Unlike the heuristic agent which has the rules that cover these events. Under the limitations of the experiment it is not possible to accept Hypothesis 2.2. It is true for the intermediate rule agent, and perhaps any *good* player.

Fig. 4.3b shows the results when pairing up the intermediate rule agent and RL model. Orange shows when the intermediate agent starts the game with the class, and the RL model has a random hand, the opposite for blue. The baseline win rates are also shown as the dotted line for each agent when starting with random hands, so any increase above that line is positive.



(a) Win rates for each hand class. Agents play against themselves for 100,000 games per hand class. The 50% line is marked, anything above this in an increase.

(b) Hand class win rates for Model vs Intermediate. Along with the baseline win-rate for each (random hands).

Figure 4.3: Win rates for different hand classes. Uses the intermediate rule agent and the 10,000th iteration of the RL model. Averaged over 100,000 games. The opponent's starting hand is sampled from the deck after generating the starting hand of the given class.

The shape of the increases over the baseline win rates in Fig. 4.3b is similar to self play mode (a). The advantage gained is amplified for the intermediate agent, since its base win rate was so low against the RL model – it's still able to win almost 90% of games when starting with a 5 card straight. Likewise, the effects are dampened somewhat for the RL model, since it already wins over 60% of the time against the intermediate agent. This goes some way to showing that the effect a starting hand has on your win rate will depend on the baseline win rate – i.e. the difference in skill between players.

Fig. 4.4 displays the results of the final experiment. Using samples of the RL model as it's trained gives an estimation of increasing skill. It quickly picks up on basic strategy, then around iteration 4000 the model starts to get better at 3K, which has the effect of also improving 3K2K. I believe this is where the agent learns to exploit three of a kind, instead of just putting down two cards.

This shows that in order to get the most out of certain hand classes a player must know how to exploit them. This is the essence of Hypothesis 2.3. As the RL model gathers more in game experience its able to increase its win rate when starting with specific hand classes.

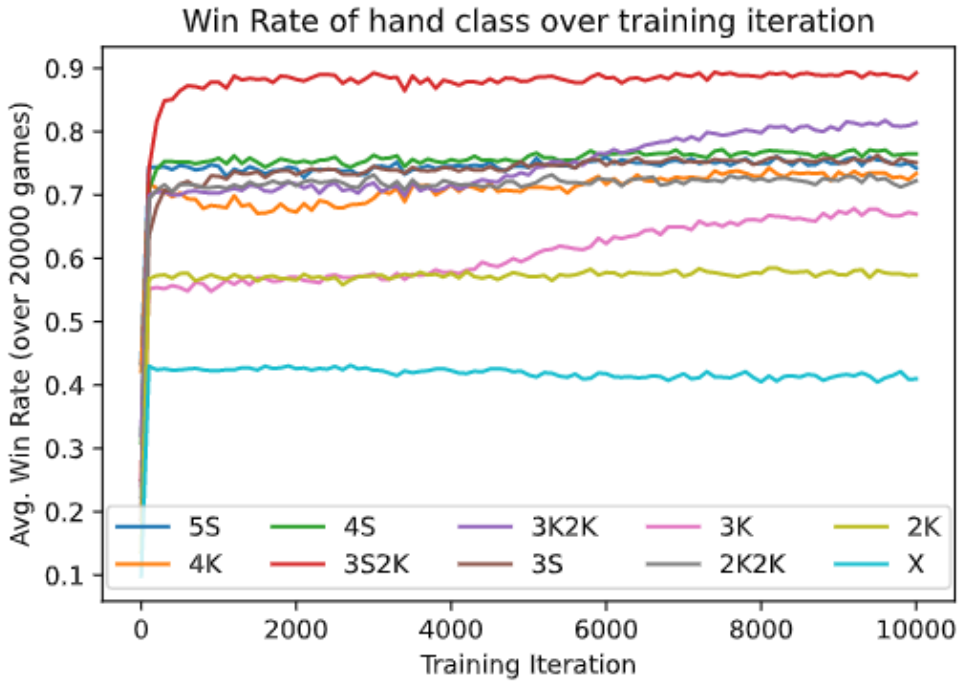


Figure 4.4: Win-rate for given class over training iterations.

4.3 Limitations and assumptions

Throughout this project we have used a player's win-rate versus the baseline heuristic agents to determine their skill level. There is a clear reason for this; if an agent wins 50% of the time against their opponent then they must have the same level of skill. Likewise, if they win 60% of the time then you can assume they are better. However, there is more to Yaniv, as the score a player loses on is important. If two players have an equally matched win rate (50%), but one scores less on average when they lose, then they must have more skill. A better measure of performance might be the average round score, that is the loss proportion multiplied by the average losing score.

5 Conclusion

This project aimed to explore the relationship between luck and skill in the card game Yaniv. Through a series of research questions we looked at the effects that certain luck based events had on the game – going first, and starting with a specific hand.

5.1 Implications of Results

The results detailed in Chapter 4 give a clearer picture on the effects of some luck based events in Yaniv.

We were able to show that starting a round does increase the chances of winning it (Hypothesis 1.1). However, we weren't able to conclusively show that skill increased the effects (Hypothesis 1.2). That said it was clear that the opponent the increase was not constant. It changed with each match-up of agents. The largest changes were seen in self-play when the two agents had the same level of skill (on average winning about a 6% more games). I suspect this is because one's win rate is harder to increase the higher it is.

Next we looked at the effects of starting with a specific set of cards in your hand. We showed that starting with a smaller total hand score did increase your chances of winning (up to 100% wins.) However, we also discovered that starting with a hand score of over 40 gave increasing odds of winning. We hypothesised that this is because to make the higher score hands you need more card combos which can be put down together. It is clear that there is a relationship between starting hand score and win rate, though it is not strictly linear. So, as long as your starting hand is not within the range 30-40 then you stand better than average odds of winning.

Finally, we investigated the effect of starting with a specific combination of cards in your hand, like a four of a kind. Using the intermediate agent it was clear that the more cards included in a combination the better the win rate. However, the results from the RL model in self play showed that it had not learned to deal with the less probable cases and wasn't able to exploit the five card straight as well as the intermediate agent. So it is difficult to

accept Hypothesis 2.2 as it's dependent on strategy. For the intermediate agent it is true, but not for the RL model.

Likewise with it is difficult to show that this effect increases with an increase in skill (Hypothesis 2.3). Using the win rate as the only measure of skill, then the RL model was more skilled than the intermediate model, but could not take full advantage of some of the higher order hand classes. This shows that skill level is more nuanced than just win rate.

In any case, it's clear that starting with hand that has some combination increases your odds of winning. Starting with a hand that has nothing in it gives only a 40% chance of winning.

All the conclusions drawn have the base assumption that the agent plays rationally with the most basic aim of reducing one's hand score to the Yaniv threshold. The random action agent – takes a random legal action at each step – doesn't see significant change when going first.

5.2 Success of the Project and Contributions

The main goal to explore luck and skill in Yaniv was achieved, with multiple experiments being run.

In order to meet the main aim I set several smaller objectives listed in Section 1.1. I successfully created a highly configurable full Yaniv simulator, that can be used in further AI research. Along with this I created 2 rule based agents which mimic human play. These can be used as benchmarks for any further AI research. Finally, I designed a configurable environment to train RL agents in, which achieved a good win rate over the heuristic agents.

Overall the implementation of the project was successful, though the experiments designed had limitations and were based on the assumption that an RL agent could be used as a continuous measure of skill. Further work needs to be done to quantify what skill means in Yaniv and how it effects luck.

Though the objectives were successfully met, it was difficult to fully realise the aim of exploring luck and skill together. It is clear that the two concepts are more nuanced winning performance. We observed the effects of luck and skill independently, which gave a picture as to how they might interact but were not able to draw concrete conclusions.

5.3 Further Work

One major shortcoming of the RL agent was the lack of forward planning. The agent is based only on an MLP network, which has no time element. Incorporating an LTSM network or some other temporal mechanism could greatly improve performance. Another way to improve the agent would be to add some form of search for action selection both during training and evaluation. This would increase the forward planning ability of the agent.

The RL models were all trained with limited resources and could do with more training time and proper hyperparameter tuning. With more compute resources it would be possible to train agents on the entire game, rather than the slightly simplified version described in Section 3.1. Adding Jokers in adds some complexity to the action and state space.

Evaluating and training agents to play the overall game also offers opportunity for further study. Training an agent to reliably hit certain scores, in order to hit 100 points for a bonus, would be a challenging task.

In order to play Yaniv well, an agent needs to reason about the mental state of their opponents and their possible strategy. Training an agent with a theory of mind to make assumptions about a players' evolving [38]. It is possible to foil an opponent's strategy by holding on to cards you suspect they need. Likewise, it is important to make estimations about an opponent's hand [39].

More rigorous statistical analysis needs to be done in order to gain a fuller understanding of luck and skill in the Game [37].

A Training configuration for RL model


```
    "model": {
        "custom_model": "yaniv_mask",
        "fcnet_hiddens": [
            512,
            512
        ]
    },
    "algorithm": "A3C",
    "batch_mode": "complete_episodes",
    "env_config": {
        "n_players": 2,
        "single_step": false,
        "step_reward": 0,
        "state_n_players": 2,
        "early_end_reward": 0,
        "end_after_n_steps": 130,
        "observation_scheme": 1,
        "max_negative_reward": -1,
        "negative_score_cutoff": 30,
        "use_dead_cards_in_state": true,
        "use_unkown_cards_in_state": false,
        "use_scaled_negative_reward": true,
        "use_scaled_positive_reward": true,
        "end_after_n_deck_replacements": 0
    },
    "rollout_fragment_length": 100,
    "update_self_play_param_win_rate": 0.55
}
```

Bibliography

[1] David Sidney Parlett, *The Oxford guide to card games*, eng. Oxford University Press, 1990, ISBN: 978-0-19-214165-1. [Online]. Available: http://archive.org/details/oxfordguidetocar00parl (visited on 13/05/2021).

[2] H. C. Levinson, *Chance, luck, and statistics*. Courier Corporation, 2001.

[3] D. Ye, G. Chen, W. Zhang, S. Chen, B. Yuan, B. Liu, J. Chen, Z. Liu, F. Qiu, H. Yu, Y. Yin, B. Shi, L. Wang, T. Shi, Q. Fu, W. Yang, L. Huang and W. Liu, 'Towards Playing Full MOBA Games with Deep Reinforcement Learning,' *arXiv:2011.12692 [cs]*, Dec. 2020, arXiv: 2011.12692. [Online]. Available: http://arxiv.org/abs/2011.12692 (visited on 18/05/2021).

[4] *Rules of Card Games: Yaniv*. [Online]. Available: https://www.pagat.com/draw/yaniv.html (visited on 29/12/2020).

[5] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den 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,' en, *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016, Number: 7587 Publisher: Nature Publishing Group, ISSN: 1476-4687. DOI: 10.1038/nature16961. [Online]. Available: https://www.nature.com/articles/nature16961 (visited on 15/04/2021).

[6] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan and D. Hassabis, 'Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm,' *arXiv:1712.01815 [cs]*, Dec. 2017, arXiv: 1712.01815. [Online]. Available: http://arxiv.org/abs/1712.01815 (visited on 15/04/2021).

[7] 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. van den Driessche, T. Graepel and D. Hassabis, 'Mastering the game of Go without human knowledge,' en, *Nature*, vol. 550, no. 7676, pp. 354–359, Oct. 2017, Number: 7676 Publisher: Nature Publishing Group, ISSN: 1476-4687. DOI:

10.1038/nature24270. [Online]. Available: https://www.nature.com/articles/nature24270 (visited on 15/04/2021).

[8] N. Brown, A. Bakhtin, A. Lerer and Q. Gong, 'Combining Deep Reinforcement Learning and Search for Imperfect-Information Games,' *arXiv:2007.13544 [cs]*, Nov. 2020, arXiv: 2007.13544. [Online]. Available: http://arxiv.org/abs/2007.13544 (visited on 26/02/2021).

[9] M. L. Littman, 'Markov games as a framework for multi-agent reinforcement learning,' en, in *Machine Learning Proceedings 1994*, W. W. Cohen and H. Hirsh, Eds., San Francisco (CA): Morgan Kaufmann, Jan. 1994, pp. 157–163, ISBN: 978-1-55860-335-6. DOI: 10.1016/B978-1-55860-335-6.50027-1. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9781558603356500271 (visited on 12/05/2021).

[10] Q. Jiang, K. Li, B. Du, H. Chen and H. Fang, 'DeltaDou: Expert-level Doudizhu AI through Self-play,' en, in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, Macao, China: International Joint Conferences on Artificial Intelligence Organization, Aug. 2019, pp. 1265–1271, ISBN: 978-0-9992411-4-1. DOI: 10.24963/ijcai.2019/176. [Online]. Available: https://www.ijcai.org/proceedings/2019/176 (visited on 29/12/2020).

[11] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg and D. Hassabis, 'Human-level control through deep reinforcement learning,' en, *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015, ISSN: 0028-0836, 1476-4687. DOI: 10.1038/nature14236. [Online]. Available: http://www.nature.com/articles/nature14236 (visited on 22/02/2021).

[13] R. S. Sutton, D. McAllester, S. Singh and Y. Mansour, 'Policy Gradient Methods for Reinforcement Learning with Function Approximation,' in *In Advances in Neural Information Processing Systems 12*, MIT Press, 2000, pp. 1057–1063.

[14] D. Silver, *Lectures on Reinforcement Learning*. 2015, Published: URL: https://www.davidsilver.uk/teaching/.

[15] R. BELLMAN, 'A Markovian Decision Process,' *Journal of Mathematics and Mechanics*, vol. 6, no. 5, pp. 679–684, 1957, Publisher: Indiana University Mathematics Department, ISSN: 00959057, 19435274. [Online]. Available: http://www.jstor.org/stable/24900506 (visited on 11/05/2021).

[16] J. Heinrich and D. Silver, 'Deep Reinforcement Learning from Self-Play in Imperfect-Information Games,' *arXiv:1603.01121 [cs]*, Jun. 2016, arXiv: 1603.01121. [Online]. Available: http://arxiv.org/abs/1603.01121 (visited on 02/01/2021).

[17] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver and K. Kavukcuoglu, 'Asynchronous Methods for Deep Reinforcement Learning,' *arXiv:1602.01783 [cs]*, Jun. 2016, arXiv: 1602.01783. [Online]. Available: http://arxiv.org/abs/1602.01783 (visited on 09/01/2021).

[18] J. Schulman, S. Levine, P. Moritz, M. I. Jordan and P. Abbeel, 'Trust Region Policy Optimization,' *arXiv:1502.05477 [cs]*, Apr. 2017, arXiv: 1502.05477. [Online]. Available: http://arxiv.org/abs/1502.05477 (visited on 16/04/2021).

[19] 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 (visited on 09/01/2021).

[20] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver and D. Wierstra, 'Continuous control with deep reinforcement learning,' *arXiv:1509.02971 [cs, stat]*, Jul. 2019, arXiv: 1509.02971. [Online]. Available: http://arxiv.org/abs/1509.02971 (visited on 16/04/2021).

[21] T. Haarnoja, A. Zhou, P. Abbeel and S. Levine, 'Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor,' *arXiv:1801.01290 [cs, stat]*, Aug. 2018, arXiv: 1801.01290. [Online]. Available: http://arxiv.org/abs/1801.01290 (visited on 16/04/2021).

[22] C. J. Watkins and P. Dayan, 'Q-learning,' *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992, Publisher: Springer.

[23] V. R. Konda and J. N. Tsitsiklis, 'Actor-Critic Algorithms,' en, *Advances in neural information processing systems*, p. 7, 2000.

[24] D. Seitz, N. Milyukov and V. Lisy, 'Learning to guess opponents information in large partially observable games,' en, p. 8,

[25] D. Zha, K.-H. Lai, Y. Cao, S. Huang, R. Wei, J. Guo and X. Hu, 'RLCard: A Toolkit for Reinforcement Learning in Card Games,' *arXiv:1910.04376 [cs]*, Feb. 2020, arXiv: 1910.04376. [Online]. Available: http://arxiv.org/abs/1910.04376 (visited on 29/12/2020).

[26] M. Lanctot, E. Lockhart, J.-B. Lespiau, V. Zambaldi, S. Upadhyay, J. Pérolat, S. Srinivasan, F. Timbers, K. Tuyls, S. Omidshafiei, D. Hennes, D. Morrill, P. Muller, T. Ewalds, R. Faulkner, J. Kramár, B. De Vylder, B. Saeta, J. Bradbury, D. Ding, S. Borgeaud, M. Lai, J. Schrittwieser, T. Anthony, E. Hughes, I. Danihelka and J. Ryan-Davis, 'OpenSpiel: A Framework for Reinforcement Learning in Games,' *arXiv:1908.09453 [cs]*, Sep. 2020, arXiv: 1908.09453. [Online]. Available: http://arxiv.org/abs/1908.09453 (visited on 08/01/2021).

[27] N. Shi, R. Li and S. Youran, 'ScrofaZero: Mastering Trick-taking Poker Game Gongzhu by Deep Reinforcement Learning,' *arXiv:2102.07495 [cs]*, Feb. 2021, arXiv: 2102.07495. [Online]. Available: http://arxiv.org/abs/2102.07495 (visited on 02/03/2021).

[28] N. Brown and T. Sandholm, 'Superhuman AI for heads-up no-limit poker: Libratus beats top professionals,' en, *Science*, vol. 359, no. 6374, pp. 418–424, Jan. 2018, Publisher: American Association for the Advancement of Science Section: Research Article, ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.aao1733. [Online]. Available: https://science.sciencemag.org/content/359/6374/418 (visited on 05/01/2021).

[29] 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 (visited on 15/04/2021).

[30] D. Hernandez, K. Denamganai, S. Devlin, S. Samothrakis and J. A. Walker, 'A Comparison of Self-Play Algorithms Under a Generalized Framework,' *arXiv:2006.04471 [cs]*, Jun. 2020, arXiv: 2006.04471. [Online]. Available: http://arxiv.org/abs/2006.04471 (visited on 27/03/2021).

[31] 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 (visited on 29/03/2021).

[32] E. Liang, R. Liaw, P. Moritz, R. Nishihara, R. Fox, K. Goldberg, J. E. Gonzalez, M. I. Jordan and I. Stoica, 'RLlib: Abstractions for Distributed Reinforcement Learning,' *arXiv:1712.09381 [cs]*, Jun. 2018, arXiv: 1712.09381. [Online]. Available: http://arxiv.org/abs/1712.09381 (visited on 27/04/2021).

[33] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang and W. Zaremba, 'OpenAI Gym,' *arXiv:1606.01540 [cs]*, Jun. 2016, arXiv: 1606.01540. [Online]. Available: http://arxiv.org/abs/1606.01540 (visited on 27/04/2021).

[34] D. Seita, *Scaling Multi-Agent Reinforcement Learning*. [Online]. Available: http://bair.berkeley.edu/blog/2018/12/12/rllib/ (visited on 25/03/2021).

[35] Y. Li, 'Deep Reinforcement Learning,' *arXiv:1810.06339 [cs, stat]*, Oct. 2018, arXiv: 1810.06339. [Online]. Available: http://arxiv.org/abs/1810.06339 (visited on 29/04/2021).

[36] M. Warchalski, D. Radojevic and M. Milosevic, 'Deep RL Agent for a Real-Time Action Strategy Game,' *arXiv:2002.06290 [cs]*, Feb. 2020, arXiv: 2002.06290 version: 1. [Online]. Available: http://arxiv.org/abs/2002.06290 (visited on 07/04/2021).

[37] Z. Guo, I. Khuu, K. Zhu, J. S. Rosenthal and F. P. Schoenberg, 'Distinguishing luck from skill through statistical simulation: A case study,' en, *Communications in Statistics - Simulation and Computation*, pp. 1–23, Dec. 2019, ISSN: 0361-0918, 1532-4141. DOI: 10.1080/03610918.2019.1698746. [Online]. Available: https://www.tandfonline.com/doi/full/10.1080/03610918.2019.1698746 (visited on 01/05/2021).

[38] A. Fuchs, M. Walton, T. Chadwick and D. Lange, 'Theory of Mind for Deep Reinforcement Learning in Hanabi,' *arXiv:2101.09328 [cs]*, Jan. 2021, arXiv: 2101.09328. [Online]. Available: http://arxiv.org/abs/2101.09328 (visited on 28/04/2021).

[39] B. Mishra and A. Aggarwal, 'Opponent Hand Estimation in Gin Rummy Using Deep Neural Networks and Heuristic Strategies,' en, p. 7,