

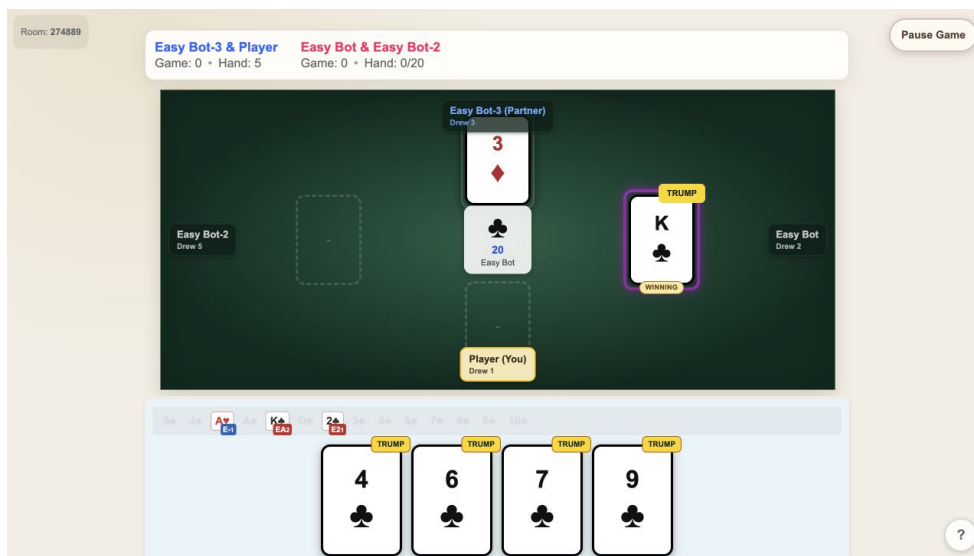
30for60

Feb 05, 2026 • spencerburleigh.com/30for60

My family loves to play an obscure card game called “**forty-fives**” ^[a] (45) popular in northern New England and Atlantic Canada.

We play an even more specific set of rules (the probabilistically most interesting and exciting variant!) where you sit across from a teammate and play another team of two. Each person “bids” for the number of points they expect their team can get in exchange for receiving four hidden cards in the “kitty” and choosing the “trump” suit that defines the hierarchy of cards. Many a game has been turned by our most distinct bid: “30for60” where a player who bids that they can get all 30 points in a hand is rewarded with 60 points if they do so.

Living thousands of miles away from anyone I’m related to is hard sometimes. The long tail of the internet had not produced a way to play online so I built 30for60.com ^[b]: a fully functional version of the game complete with self-play trained bots that can join a game if we don’t have enough human players.



I started this project just wanting a website to play this stupid game with my family in a way that didn’t involve manually dealing cards with a mouse. And it’s been pretty great! I’ve spent quite a few hours playing with my immediate family in New Hampshire and grandparents in Georgia, but 45 is best played with four people and grandparents don’t seem to come from the store in threes. Life seemed like it could be so much easier if a friendly and extremely competent bot could join us...

How to play

Play at 3ofor60.com^[b]. There is a how-to, and the table will nudge you in the right directions.

Part for Nerds

Webapp Stack

The site is a React frontend on Cloudflare Pages talking to a single Node/TypeScript Socket.IO process on Fly.io. That backend owns the game: rooms, cards, bidding, scoring. Redis holds snapshots so a restart or a dropped connection can pick the hand back up instead of wiping it. Card rankings and trick rules live in a shared TypeScript package, so the UI and the server stay aligned.

Why this is an interesting learning problem

45 has a few distinct phases of gameplay: bidding, trump selection by the bid winner (based on their hand and the four additional kitty cards they pick up), discard of unwanted cards and replenishment from the deck after trump is selected, and playing the five tricks that make up a hand.

While trump selection and which cards to discard are ~deterministic once you've seen the kitty, bidding and playing the five tricks in a hand are not. The optimal action varies drastically on: position (leading trump forces everyone including your partner to play trump too), overall score (past 100, points only count if your team enters a bid), and a belief about the others' hands (inferred from the draw phase and how much trump they're playing).

Training the best (and only?) 45 bot in the world

Self-play

Self-play was the only approach that fit. 45 is obscure; there is no expert dataset, only the family games this app is starting to log. The decisions that matter are sequential and imperfect-information: you never see the other hands, and the right bid at 40–40 is the wrong bid at 110–105. A giant heuristic policy of how I know how to play would have encoded my biases and taken a ton of time.

Reward function

I started as sparse as possible: at the end of each hand, the change in score differential, [1] plus a terminal bonus for winning the game about the size of the reward it would pile up in a typical game.

I started adding extra shaping penalties from here since infrequent rewards (and the especially high reward of setting your opponent back) discouraged bidding. A 20 is often an easy bid once you include the four unknown kitty cards, but a single sample does not show that, so the policy learned to

pass. I added a pass penalty and a penalty for pulling trump out of partner, then gradually annealed the pass penalty so it would not become the main objective.

Splitting stages

Sparse PPO methods to keep the bot learning from the full game only reached an elementary proficiency, so I factored the problem. Trump and discard became a deterministic heuristic and I turned bidding into a Monte Carlo EV table — simulated expected scores, looked up at bid time. Trick-taking stayed a neural policy, conditioned on role: bidding team vs defense, and bidder vs supporter, because the bid winner holds most of the trump and controls the hand.

EV table for bidding

45's auction decision space is tiny — pass / 20 / 25 / 30for60, plus dealer hold. The explosion is the hands: $C(52,5) = 2.6$ million, but most of those variations do not matter when choosing what bid to make. Once you name trump, off-suit cards don't matter, and a 5 + J in diamonds is mathematically identical to a 5 + J in spades.

Almost none of the variation between hands matters for the auction and I mapped each dealt hand to the set of trump ranks it would hold under its best suit: 14 slots in 45's order (5, J, A♥, A, K, Q, then the eight remaining ranks, red high / black low), and combinations of from 0 through 5 of trump. This narrowed us to 3,473 keys, about a $750\times$ cut from the original 2.6 million. I also implemented dominance pruning: if key A is stronger than key B and key A cannot make the minimum bid, you don't need to calculate B to know that B also cannot make the minimum bid [2].

For each remaining key, the bidder is dealt that hand, names trump with a deterministic heuristic, discards, and plays the five tricks with a previously trained card playing model against a random rest of the deck. Each sim records the bidder team's raw score (tricks $\times$ 5 plus 5-points for best-trump). A Welford running mean and variance stop a key when we're within 5 points (the smallest unit of points in 45), 95% of the time, or at 200 sims. The table stores mean raw score and empirical make-rates at 20 / 25 / and 30for60 bids.

With the table, bidding is a lookup: use the hand's expected values to decide your max bid and adjust for the scoreboard (over-100, 30for60).

Reducing variance

The return is high-variance because the kitty and the other three hands' draws are hidden. For closeout hands (score over 100) and for risky 25/30for60 bids I ran a paired rollout: same pre-draw decision, resample the random cards, and averaged the two returns. PPO's default is one playthrough of the hand, and these pivotal hands are likely to create disproportionate variance in

rewards, so I forked them after the bid phase and averaged the results of two hands' end score to better estimate the expected value of the hidden cards instead of the luck of one draw.

Overemphasizing the most consequential types of hands

Many hands in 45 have a quite deterministic play pattern but there are a few types of hands (over 100, very close bid where you have good cards and can keep someone from getting the bid, opportunity to renege (hold back) a powerful card) that players' skill shines.

Since these states are rare in the distribution of normal hands, they barely update the policy. To train faster and ensure these moments were learned, I put 30% of later training runs on a curriculum. This included closeout scores (100–120), situations where you the bidder are exceptionally weak but your partner is exceptionally strong, and other marginal situations. The remaining 70% stayed on fair shuffle to prevent catastrophically forgetting the typical gameplay.

Distilling the game to speed up training

I trained this on my M1 Pro MacBook Pro over a few nights, so a sparse and not suit-symmetric 52-card one-hot encoding was doing me no favors. The policy kept collapsing onto spades just because that suit won first. But suits don't really matter: once trump is named, the only relevant coordinates are is-trump and rank-within-trump, so I encoded that way. Next, I quantized the low trump and off-suit ranks into buckets to shrink the state further [3].

Hitting the limits of self-play

Self-play is non-stationary: the opponent is your last saved version of the bot. It stops exploring: entropy dies, everyone passes, and you have a locally consistent policy that is wrong. I watched entropy, hand length, and return, and killed runs that collapsed. Later training used a league: the learner faced a rotating mix of older saved versions, and only played copies of itself after it could beat that mix. Partner assignment rotated too. If the same two networks are always teammates they learn a private convention, which my grandma won't understand when she partners with it.

Checkpoint 26.02.05

I've now trained a policy that as partners can often beat most people in my family. They still underbid, and they still do silly things at the table: failing to ruff, cross-ruffing low cards [4]. I do not think a few more tweaks get you a bot that always beats humans. I think a longer run [5], on this engine, with a bit more shaping or examples from logged family games, would get something that could quickly beat almost any human player.

September 2026 Update: Astra ~one shot UI upgrade

I am a ml / backend engineer by training and while I have a lot of opinions about design I haven't thought I had many skills to actually make websites look super nice. When I started this project in ~January 2026 the models were not very good at design either and the UI took disproportionate time to get something I was 6/10 happy with. Now in September 2026 I pointed Astra at the UI and with 10 minutes of reminders to keep things consistent, I think we are now an 8/10!

Notes

[1] So not bidding and setting your opponent back 20 points while you go forward 15 points is +35 whereas making a bid of 25 points where your opponent earns 5 points is only +20.

[2] I think I oversimplify this a bit (which underestimates the EV) since the four unknown kitty cards also come with optionality. Eg reducing the trump you think you have to your 5S and QS when you have J♣ and Q♣ in another suit is undervalued. What if you get 5♣ in the kitty? The best hand with the hidden kitty is the thing you ignored...

[3] I resolved ties where there were two cards from the same bucket as highest with a coin flip.

[4] The first trick in a hand of 45 starts with the player (opponent) to the bid winner's left. Opponents are generally trying to save their trump as much as possible but the bid winner presumably has a lot of trump and can compel others to play trump, so it's typical for the opponent right before the bidder to play their ~largest card to force the bidder to use a big trump card (that can't be used against their partner) or lose the flow of how the cards are moving. Cross roughing is a much more controversial strategy when the bidder's partner plays their biggest (typically held "bare" with no other trump cards) on the first turn. This can be good because it signals to the bidder that they can't rely on help from their partner but if the cross rough is small it is also just wasting the trump card since you should expect the next player to rough regardless. I am moderately surprised this hasn't been emergent?

[5] I looked into moving this into the cloud but it seemed extremely expensive for something I could just run overnight when I wasn't using my computer anyway. Lots of thinking about local AI/compute from this project... ;)

Links

- a. en.wikipedia.org/wiki/Forty-fives
- b. 30for60.com