Gambler Grandma?

Prakrat Agrawal

June 2025

Introduction

I was always taught by my grandmother that gambling is bad, but I would watch her buy tickets for the "Bumper Tambola" every Sunday and smile at the hypocrisy. She played in every league—from 20 Rs tickets to 500 Rs tickets, where she once won a refrigerator. I have fond memories of Tambola from my childhood—how my mom and grandmother would buy too many tickets and then make me strike the numbers for them, how hosts would make up funny names to call the numbers, and how games got heated when multiple people got a scheme at the same time.

This past semester, I took EECS 126 (Probability and Random Processes) at Berkeley. This paper is inspired by that class and tries to use some of the knowledge I learned to model this game mathematically and analyze its expected outcomes. Because there is no betting involved—players simply pay for a ticket and prizes are awarded based on random draws—the game should be fundamentally fair.

Game Rules and Ticket Constraints

Game Setup

- Number of players: 19
- Ticket price: 20
- Each player is given a ticket containing 15 distinct numbers from 1 to 90, arranged in a 3×9 grid.
- The host has a bag with all numbers 1–90 and draws one number at random *without replacement*. When a number is called, any player who has it on their ticket strikes it off.
- A player must be the *first* among all players to complete a particular pattern (scheme). They call out their win immediately after the draw that completes their pattern.
- Prizes are zero-sum: all money collected from ticket purchases is redistributed as prizes according to the various schemes.
- *Simplifying assumptions:*
  - In real Tambola, tickets come in pads of six per page, with no number repeated within those six. For simplicity, we assume tickets are generated independently and uniformly at random.
  - In practice, a player might buy multiple tickets. Here we assume each player holds exactly one ticket.
  - I might update this paper later to analyze how relaxing these assumptions changes the results.

Ticket Structure (3×9 grid)

- Each ticket is a 3×9 grid containing exactly 15 distinct numbers chosen from 1–90.
- **Column constraints:**
  - Column 1 holds numbers 1–9.
  - Column 2 holds numbers 10–19.
  - ...
  - Column 9 holds numbers 80–90.
- **Column counts:** Each column must contain 1, 2, or 3 numbers.
- **Row counts:** Each row must contain exactly 5 numbers (and 4 blank spaces).
- No number may appear more than once on a single ticket.

Table 1: Sample Tambola ticket

	8	17	21		54		88	
2		19		43		68		90
	7		30		53	75	81	

Winning Schemes and Prizes

Scheme	**Description**	**Prize**
Early 7	First to mark any 7 numbers on their ticket	20
Four Corners	First to mark the four corner numbers	20
Line 1	First to mark all five numbers in Row 1	40
Line 2	First to mark all five numbers in Row 2	40
Line 3	First to mark all five numbers in Row 3	40
First Half	First to mark all ticket numbers in the range 1–45	60
Second Half	First to mark all ticket numbers in the range 46–90	60
Full House	First to mark all 15 numbers on the ticket	100

Table 2: Winning schemes, descriptions, and prizes (total prizes = 380).

The prize scheme and number of players are arbitrary—chosen based on a typical Tambola game my grandmother described. Our goal is to compute, for each scheme, the expected draw index at which someone among 19 players will complete that scheme. The exact payout amounts do not affect those expectations but I have included them just to make the game structure a little more concrete. One can adjust the number of players in the formulas or code to see how the calculations change.

1 Setting Up the Problem

Whenever we draw without replacement from a finite population, the count of "successes" among the drawn items follows a *hypergeometric distribution.* Concretely:

- There are $N = 90$ total numbers, of which K are designated as "success states" (e.g., the numbers on a given ticket that we care about for scheme S).
- We draw n times *without replacement.* Let Y be the number of successes drawn in those n draws.
- To compute $P(Y = k)$, imagine all $\binom{N}{n}$ equally likely ways to choose n numbers from N. Exactly $\binom{K}{k}\binom{N-K}{n-k}$ of those subsets contain exactly k successes and $(n-k)$ failures. Therefore,
$$P(Y = k) = \frac{\binom{K}{k}\binom{N-K}{n-k}}{\binom{N}{n}},$$
- For scheme S, we require r successes. The probability the scheme remains incomplete after n draws is
$$P(Y \leq r - 1) = \sum_{k=0}^{r-1} \frac{\binom{K}{k}\binom{N-K}{n-k}}{\binom{N}{n}}$$
- Define
$$X_S \; = \; \text{draw-index at which a } \textit{single} \text{ ticket completes scheme } S.$$
Then
$$P(X_S > n) = P(Y \leq r - 1)$$
- When there are 19 independent tickets (each with its own X_S), the probability of the event "no one has completed scheme S by draw n" is
$$P(X_{S,1} > n,\ X_{S,2} > n,\ \ldots,\ X_{S,19} > n) = \left[P(X_S > n)\right]^{19}$$
Hence the expected draw index at which *someone* completes scheme S is
$$\mathbb{E}\left[\min(X_{S,1}, \ldots, X_{S,19})\right] \; = \; \sum_{n=0}^{90} \Big[P(X_S > n)\Big]^{19}$$
Below, we write down $P(X_S > n)$ for each scheme S.

2 Calculating Expected Values

Numerically estimated these sums using Python.

2.1 Early 7

For Early 7, $K = 15$ and $r = 7$.

$$P(X_{\text{E7}} > n) = \sum_{k=0}^{6} \frac{\binom{15}{k}\binom{75}{n-k}}{\binom{90}{n}}, \quad \mathbb{E}[\text{first E7}] = \sum_{n=0}^{90} \left[P(X_{\text{E7}} > n)\right]^{19} \approx 22.4382$$

2.2 Four Corners

For Four Corners, $K = 4$ and $r = 4$.

$$P(X_{\mathrm{C}} > n) = \sum_{k=0}^{3} \frac{\binom{4}{k}\binom{86}{n-k}}{\binom{90}{n}}, \quad \mathbb{E}[\text{first Corners}] \approx 40.1262$$

2.3 Line Completion

For a given row (Line), $K = 5$ and $r = 5$.

$$P(X_{\mathrm{L}} > n) = \sum_{k=0}^{4} \frac{\binom{5}{k}\binom{85}{n-k}}{\binom{90}{n}}, \quad \mathbb{E}[\text{first Line}] \approx 47.0782$$

2.4 First Half and Second Half

Define:

$$K_1 = \big|\{\text{ticket} \cap \{1, \ldots, 45\}\}\big|, \quad K_2 = 15 - K_1$$

Since each column must hold at least one number, it is easy to show that $4 \le K_1, K_2 \le 11$.

Due to the intricate combinatorial nature of generating a valid Tambola ticket (satisfying both column and row constraints), a direct closed form derivation of $P(K_1 = k)$ is complex. The probability distribution $P(K_1 = k)$ can be calculated analytically by enumerating all 1,554 valid column configurations and, for each configuration, computing the probability of achieving k first-half numbers. Since columns 1-4 contain only first-half numbers and column 5 contains a mix (6 first-half, 4 second-half), we use:

$$P(K_1 = k) \;=\; \frac{1}{1554} \sum_{\text{configs}} P\big(K_1 = k \mid \text{config}\big),$$

where $P(K_1 = k \mid \text{config})$ follows from the hypergeometric distribution of column 5's number selection.

k	$P(K_1 = k)$	$P(K_2 = k)$
4	0.004354	0.009331
5	0.037387	0.062055
6	0.137473	0.186594
7	0.267503	0.295302
8	0.295302	0.267503
9	0.186594	0.137473
10	0.062055	0.037387
11	0.009331	0.004354

Table 3: Distribution of K_1 and K_2

Note: Our calculation assumes each valid column configuration is equally likely, which matches our ticket generation algorithm that uniformly selects configurations before attempting row placement.

Let X_{1stHalf} be the draw index at which a ticket completes its first half. Condition on $K_1 = k$:

$$P\big(X_{\text{1stHalf}} > n \mid K_1 = k\big) = \sum_{m=0}^{k-1} \frac{\binom{k}{m}\binom{90-k}{n-m}}{\binom{90}{n}}$$

Then

$$P\big(X_{\text{1stHalf}} > n\big) = \sum_{k=4}^{11} P(K_1 = k)\, P\big(X_{\text{1stHalf}} > n \mid K_1 = k\big)$$

Define $K_2 = 15 - K_1$ and X_{2ndHalf} analogously.

$$P\big(X_{\text{2ndHalf}} > n\big) = \sum_{k=4}^{11} P(K_2 = k) \sum_{m=0}^{k-1} \frac{\binom{k}{m}\binom{90-k}{n-m}}{\binom{90}{n}}$$

In both cases the time to the first completion is the minimum of 19 independent copies:

$$\mathbb{E}\big[\min_i X_{*\text{Half},i}\big] = \sum_{n=0}^{90} \big[P(X_{*\text{Half}} > n)\big]^{19}$$

Numerical evaluation gives

$$\mathbb{E}[\text{first 1st Half}] \approx 58.56, \quad \mathbb{E}[\text{first 2nd Half }] \approx 57.35$$

The column-5 contribution is 6 first-half values vs. 4 second-half values. Hence the Second Half scheme completes slightly earlier on average.

2.5 Full House

For Full House, $K = 15$ and $r = 15$.

$$P(X_{\text{FH}} > n) = \sum_{k=0}^{14} \frac{\binom{15}{k}\binom{75}{n-k}}{\binom{90}{n}}, \quad \mathbb{E}[\text{first Full House}] \approx 73.2970$$

3 Summary of Theoretical Expectations

Scheme	**Expected Draws (19 players)**
Early 7	22.4382
Four Corners	40.1262
Line Completion	47.0782
First Half	58.5600
Second Half	57.3500
Full House	73.2970

Table 4: Theoretical expected draw-indices for each scheme (19 players).

4 Python Simulation Code

Below is the Python code used to generate valid Tambola tickets, simulate players drawing numbers until each scheme is first completed, and repeat the trial many times to compute empirical averages.

```
import random

def generate_ticket():
    """
    Generate one valid Tambola ticket (3×9 grid, 15 numbers total).
    - Each column j has between 1 and 3 numbers from its range:
        Column 1 → {1..9}, Column 2 → {10..19}, ..., Column 9 → {80..90}.
    - Each row has exactly 5 numbers.
    Returns a 3×9 list of lists: grid[r][c] = number or 0 if empty.
    """
    while True:
        # 1) Decide how many numbers in each of 9 columns. Sum must be 15.
        col_counts = [1] * 9
        rem = 15 - 9  # six extra slots
        for _ in range(rem):
            choices = [i for i in range(9) if col_counts[i] < 3]
            idx = random.choice(choices)
            col_counts[idx] += 1

        # 2) For each column j, pick col_counts[j] numbers from that column's range.
        columns = []
        for j, cnt in enumerate(col_counts):
            if j == 0:
                pool = list(range(1, 10))
            elif j == 8:
                pool = list(range(80, 91))
            else:
                pool = list(range(j * 10, j * 10 + 10))
            nums = random.sample(pool, cnt)
            columns.append(nums)

        # 3) Attempt to place each column's chosen numbers into 3 rows, so each row has 5 nu
        for _ in range(100):
            grid = [[0]*9 for _ in range(3)]
            row_count = [0,0,0]
            valid = True
            for c_idx, nums in enumerate(columns):
                c = len(nums)
                chosen_rows = random.sample([0,1,2], c)
                for i, r in enumerate(chosen_rows):
                    grid[r][c_idx] = nums[i]
                    row_count[r] += 1
                    if row_count[r] > 5:
                        valid = False
                        break
                if not valid:
                    break

            if valid and row_count == [5,5,5]:
                return grid
        # If assignment failed 100 times, restart.

def run_trial(num_players):
    """
    Run one Tambola trial with 'num_players' players.
    Return a dict with draw-indices when each scheme is first completed.
    """
    tickets = [generate_ticket() for _ in range(num_players)]
    scheme_info = []

    for grid in tickets:
        ticket_nums = {
            grid[r][c]
            for r in range(3)
            for c in range(9)
            if grid[r][c] != 0
        }

        # Four Corners
        def get_corners(row_i):
            cols_with = [c for c in range(9) if grid[row_i][c] != 0]
            if not cols_with:
                return []
            lc = min(cols_with)
            rc = max(cols_with)
            return [grid[row_i][lc], grid[row_i][rc]]

        tl, tr = get_corners(0)
        bl, br = get_corners(2)
        corners_set = {tl, tr, bl, br}

        # Three rows
        line_sets = [
            {grid[r][c] for c in range(9) if grid[r][c] != 0}
            for r in range(3)
        ]

        # First Half & Second Half
        first_half_set = {x for x in ticket_nums if x <= 45}
        second_half_set = {x for x in ticket_nums if x >= 46}

        scheme_info.append({
            'ticket_set': ticket_nums,
            'corners': corners_set,
            'lines': line_sets,
            'first_half': first_half_set,
            'second_half': second_half_set
        })

    draw_order = list(range(1,91))
    random.shuffle(draw_order)

    marked = [set() for _ in range(num_players)]
    scheme_times = {
        'Early7': None,
        'Corners': None,
        'Lines': [None, None, None],
        'FirstHalf': None,
        'SecondHalf': None,
        'FullHouse': None
    }
    claimed = {
        'Early7': False,
        'Corners': False,
        'Lines': [False, False, False],
        'FirstHalf': False,
        'SecondHalf': False,
        'FullHouse': False
    }

    for draw_idx, number in enumerate(draw_order, start=1):
        for i in range(num_players):
            if number in scheme_info[i]['ticket_set']:
                marked[i].add(number)

        # Early 7
        if not claimed['Early7']:
            winners = [i for i in range(num_players) if len(marked[i]) >= 7]
            if winners:
                scheme_times['Early7'] = draw_idx
                claimed['Early7'] = True

        # Four Corners
        if not claimed['Corners']:
            winners = [
                i for i in range(num_players)
                if scheme_info[i]['corners'].issubset(marked[i])
            ]
            if winners:
                scheme_times['Corners'] = draw_idx
                claimed['Corners'] = True

        # Lines
        for r in range(3):
            if not claimed['Lines'][r]:
                winners = [
                    i for i in range(num_players)
                    if scheme_info[i]['lines'][r].issubset(marked[i])
                ]
                if winners:
                    scheme_times['Lines'][r] = draw_idx
                    claimed['Lines'][r] = True

        # First Half
        if not claimed['FirstHalf']:
            winners = [
                i for i in range(num_players)
                if scheme_info[i]['first_half'].issubset(marked[i])
            ]
            if winners:
                scheme_times['FirstHalf'] = draw_idx
                claimed['FirstHalf'] = True

        # Second Half
        if not claimed['SecondHalf']:
            winners = [
                i for i in range(num_players)
                if scheme_info[i]['second_half'].issubset(marked[i])
            ]
            if winners:
                scheme_times['SecondHalf'] = draw_idx
                claimed['SecondHalf'] = True

        # Full House
        if not claimed['FullHouse']:
            winners = [
                i for i in range(num_players)
                if scheme_info[i]['ticket_set'].issubset(marked[i])
            ]
            if winners:
                scheme_times['FullHouse'] = draw_idx
                claimed['FullHouse'] = True
                break

    return scheme_times

def run_simulation(n_trials, num_players):
    """
    Run 'n_trials' Tambola trials with 'num_players' players.
    Return average draw indices for each scheme.
    """
    totals = {
        'Early7': 0,
        'Corners': 0,
        'Lines': [0, 0, 0],
        'FirstHalf': 0,
        'SecondHalf': 0,
        'FullHouse': 0
    }
    for _ in range(n_trials):
        res = run_trial(num_players=num_players)
        totals['Early7'] += res['Early7']
        totals['Corners'] += res['Corners']
        for r in range(3):
            totals['Lines'][r] += res['Lines'][r]
        totals['FirstHalf'] += res['FirstHalf']
        totals['SecondHalf'] += res['SecondHalf']
        totals['FullHouse'] += res['FullHouse']

    averages = {
        'Early7': totals['Early7'] / n_trials,
        'Corners': totals['Corners'] / n_trials,
        'Lines': [totals['Lines'][r] / n_trials for r in range(3)],
        'FirstHalf': totals['FirstHalf'] / n_trials,
        'SecondHalf': totals['SecondHalf'] / n_trials,
        'FullHouse': totals['FullHouse'] / n_trials
    }
    return averages

if __name__ == "__main__":
    n_trials = 20000
    num_players = int(input("Enter number of players: "))
    print(f"Running {n_trials} trials with {num_players} players ...")
    avg = run_simulation(n_trials, num_players)
    print("Empirical average draw indices:")
    print(f"  Early 7:      {avg['Early7']:.2f}")
    print(f"  Four Corners: {avg['Corners']:.2f}")
    print(f"  Line 1:       {avg['Lines'][0]:.2f}")
    print(f"  Line 2:       {avg['Lines'][1]:.2f}")
    print(f"  Line 3:       {avg['Lines'][2]:.2f}")
    print(f"  1st Half:     {avg['FirstHalf']:.2f}")
    print(f"  2nd Half:     {avg['SecondHalf']:.2f}")
    print(f"  Full House:   {avg['FullHouse']:.2f}")
```

5 Simulation Results vs. Theory

We ran 20,000 trials with 19 players. Table 5 shows the empirical average draw indices from simulation alongside the theoretical expectations from Table 4.

Scheme	**Empirical (20,000 trials)**	**Theoretical**
Early 7	22.42	22.4382
Four Corners	40.97	40.1262
Line 1	47.08	47.0782
Line 2	47.26	47.0782
Line 3	47.13	47.0782
1st Half	58.86	58.5600
2nd Half	57.55	57.3500
Full House	73.27	73.2970

Table 5: Comparison of empirical (simulation) vs. theoretical expected draws.

We can see using this table that our theoretical and empirical calculations are largely consistent with each other.

6 Future Work

These are some other related questions that I think can be modeled using simple probability. I may update this paper in the future to include one or more of these.

- **Relaxing simplifying assumptions:**
  - Model tickets in pads of six with no repeats across the pad.
  - Allow multiple tickets per player
- **Conditional scheme probabilities:** Compute, for example,

  $$P\big(\text{Full House by } n \mid \text{First Half by } m\big) \quad \text{or} \quad P\big(\text{Line 1} \mid \text{Early 7}\big),$$

  to understand dependencies among patterns.
- **Generalization to other bingo-style games:** Extend the framework to variants with different grid sizes, number ranges, or pattern definitions (e.g. “X” or “T” patterns).