Financial Modelling for Gambling Games

Ashe Raymond-Barker

September 17, 2025

Contents

Chapter 1

Introduction

Chance It is a skill-based, socially driven mobile gaming platform that merges the excitement of gambling with ethical play and charitable giving. Centred around three distinct daily experiences - Flip It, Crack It, and Play It - the app delivers accessible, low-stakes, skill-infused games designed to fit seamlessly into the lives of Gen Z and Millennial users. With a fixed-entry format, transparent prize structures, and optional charitable donations, Chance It is a new category of ethical, gamified micro-gambling.

In this report the company is referred to as "The House" and the collective of app users are referred to as "The Players". The collective of charities are referred to as "The Charities". I explore the nature of the first two games, "Flip It" and "Crack It":

In "Flip It" I explore the range of feasible odds to set the coin flip at to ensure that The House makes profit, whilst accounting for factors such as charitable giving and jackpots.

In "Crack It" I explore an effective method for distributing scratchcards and prizes that is highly adaptable to changes in variables such as entry fee, number of players and payout rate amongst others.

Chapter 2

Flip It

2.1 The Game

"Flip It" is our flagship experience, a once-per-day coin flipping opportunity that mimics the cultural rhythm of daily life, specifically timed around post-work downtime and pub hours. Much like BeReal, users are notified during a narrow window where they can place up to three bets. Users choose a bet amount (£2-10), flip the coins, and instantly find out if they've won or lost.

2.2 Making Profit

A standard coin flip is a zero sum game meaning that as long as the odds are in The House's favour, The House will make an almost guaranteed profit, so long as enough rounds of the game are played.

The amount of profit depends on the probabilities chosen by the company. This game essentially acts as an alternative to roulette and so we should consider those odds whilst choosing our probabilities.

In European roulette choosing a colour gives the player a $\frac{18}{37} \approx 48.65\%$ of winning, and in American roulette the odds sit at $\frac{18}{38} \approx 47.37\%$.

With this in mind, Monte Carlo modeling has been used with 100 iterations played to model the profit and loss potential of this game for both The House and the players, as the number of rounds of the game played increases. Here one coin flip is equal to one round. There are five different sets of graphs, visualising the results for five different sets of probabilities. They show both the mean profit/loss as well as the 95% confidence intervals, calculated by:

$$\mathrm{CI}_{95\%} = \bar{x} \pm 1.96 \cdot \frac{\sigma}{\sqrt{n}}$$

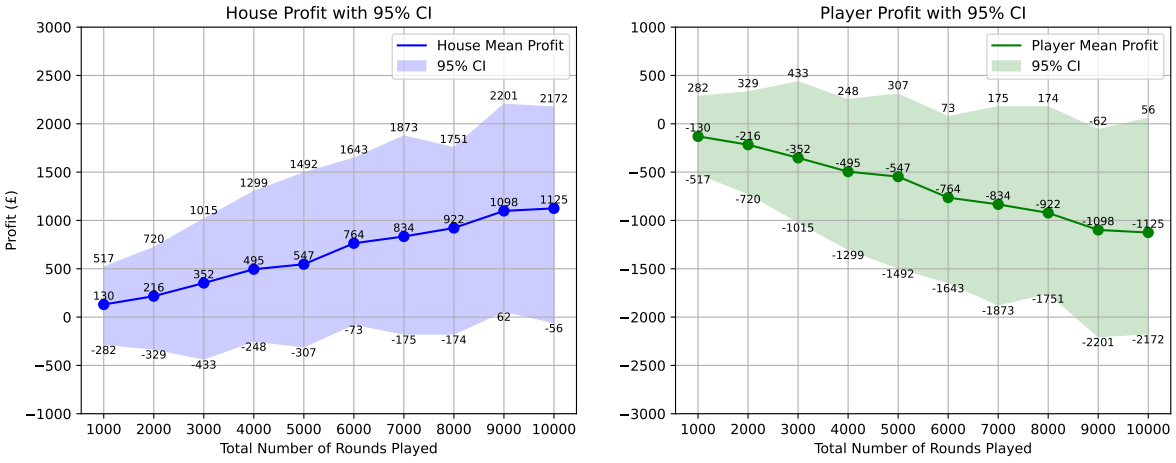

Figure 2.1: House/Player Odds: 51%/49%

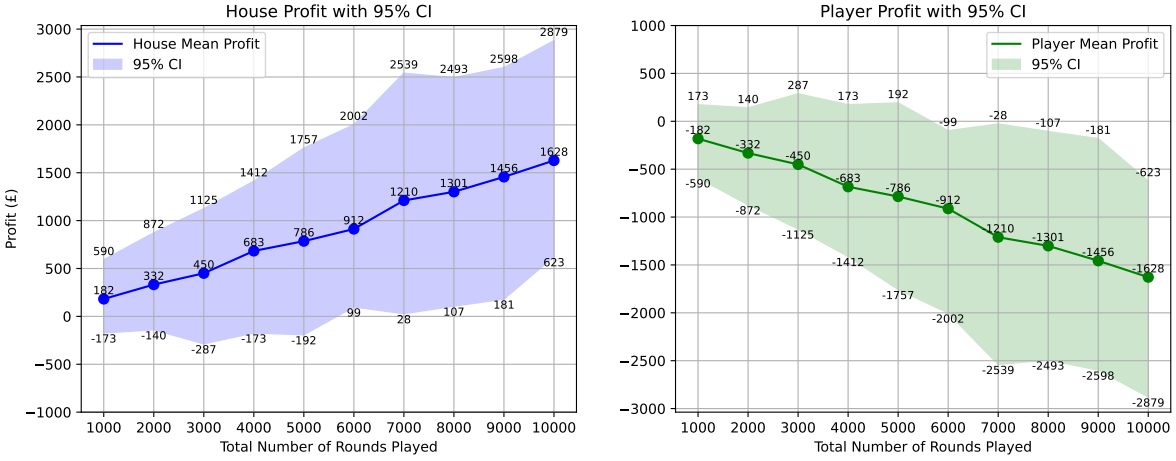

Figure 2.2: European Roulette House/Player Odds: 51.35%/48.65%

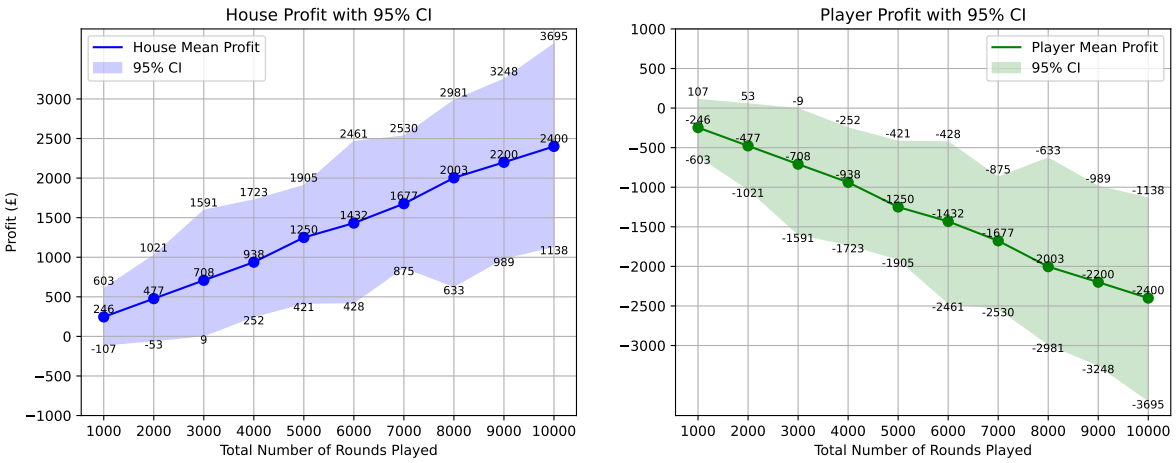

Figure 2.3: House/Player Odds: 52%/48%

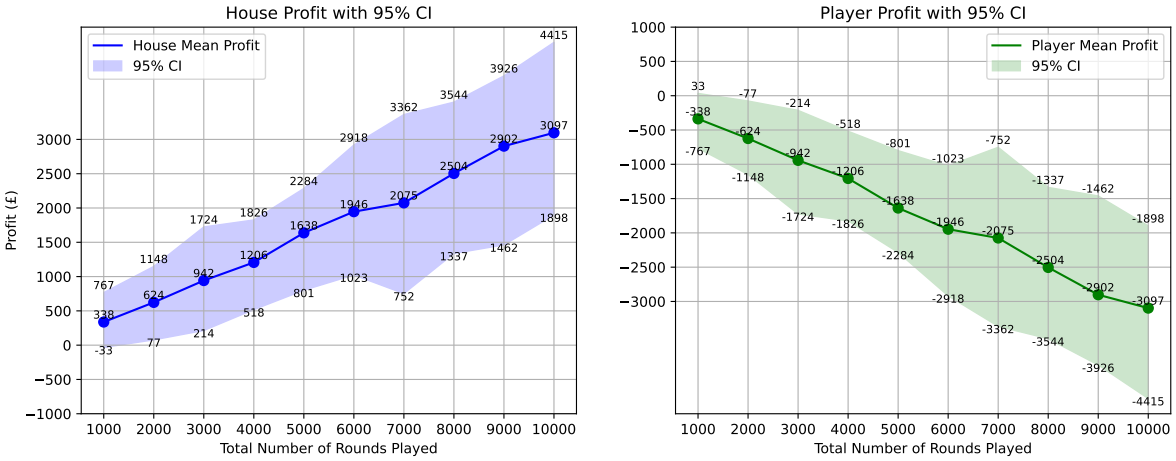

Figure 2.4: European Roulette House/Player Odds: 52.63%/47.37%

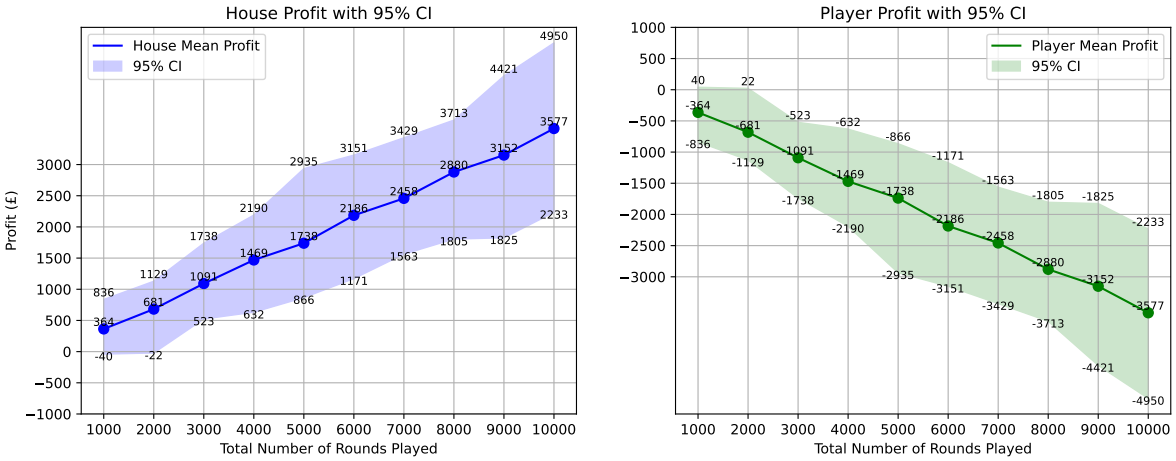

Figure 2.5: House/Player Odds: 53%/47%

2.3 Adding a charitable donation

With a coin flip, there is also the opportunity to have the coin land on its side, allowing for a third possible outcome. One such possibility for this outcome is that The Charities win the money in this case. We note that this is no longer zero-sum game, yet as long as The House's probability outweighs that of The Player's, profit will still be made with almost certainty as more rounds are played.

The graphs below show the potential profit and losses of The House, The Players and The Charities. In these graphs, 100 iterations of the game were played and each game modeled 10,000 players playing 3 rounds of the game each. The House's probability of a win was set between 51% and 53%, whilst the player's probability of a win was set between 45% and 49% and the charity's probability of a win was set between 0% and 4%.

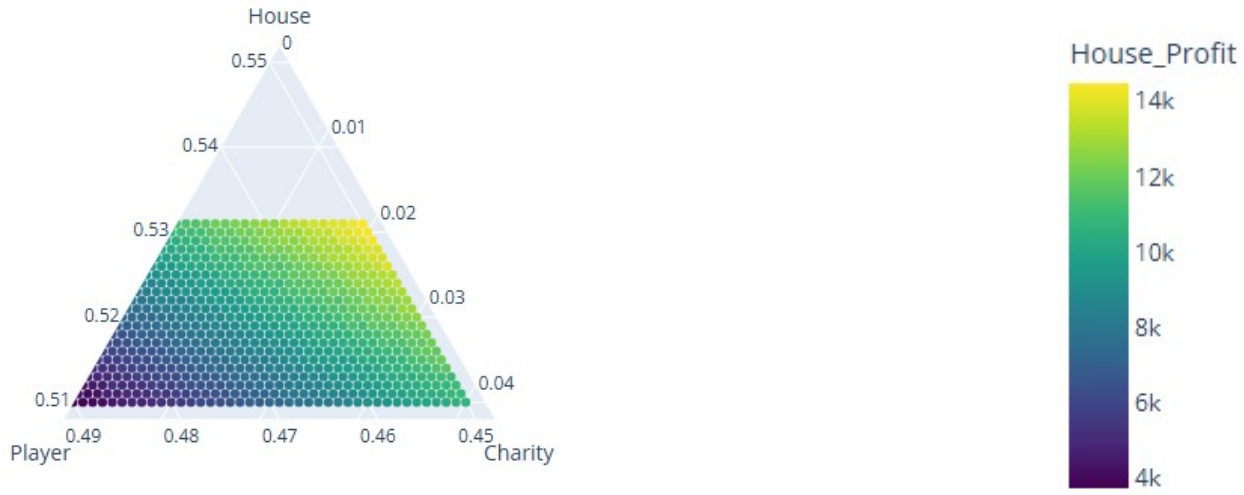

Figure 2.6: Mean House Profit (£)

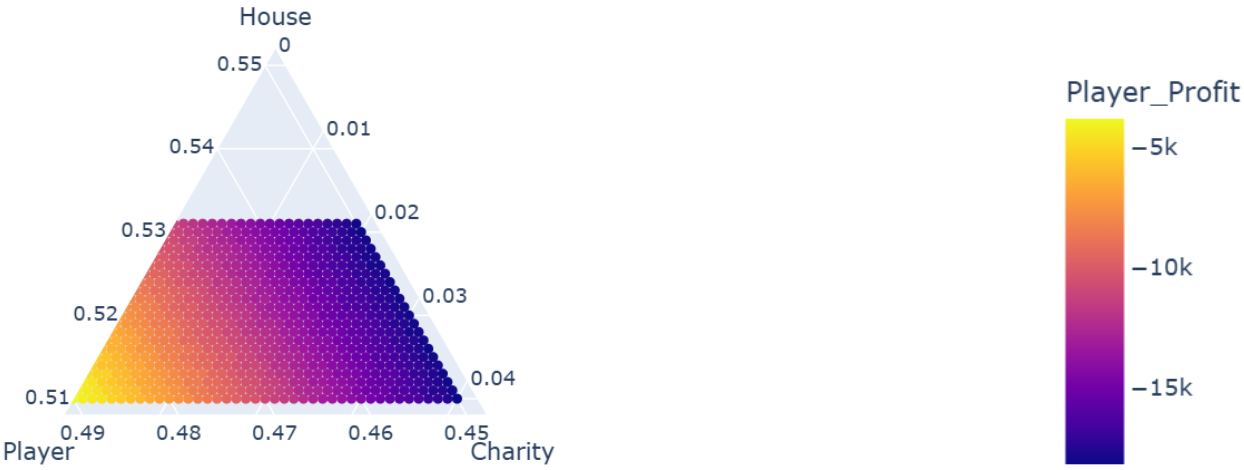

Figure 2.7: Mean Player Profit (£)

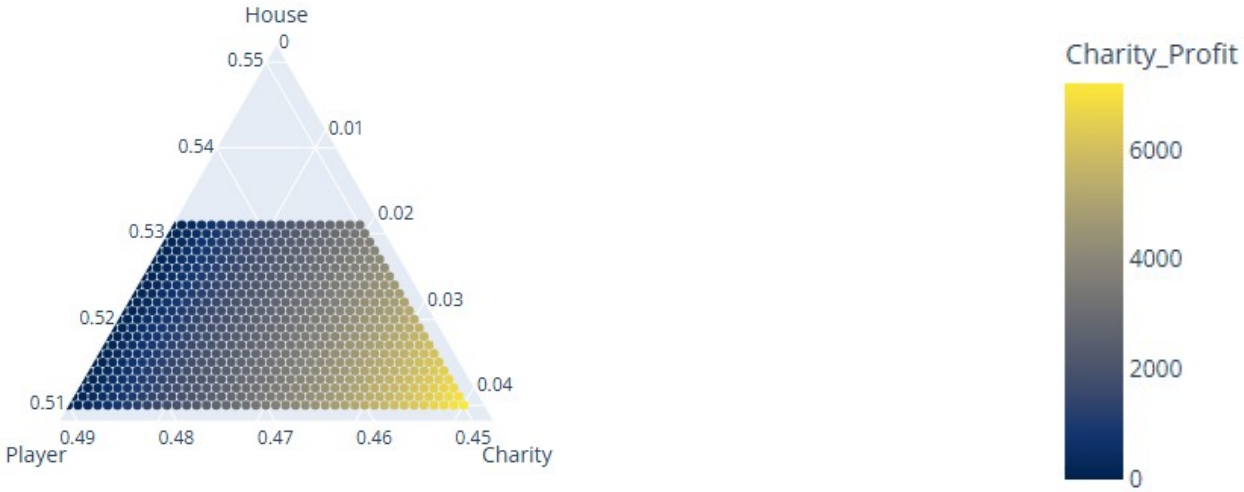

Figure 2.8: Mean Charity Profit (£)

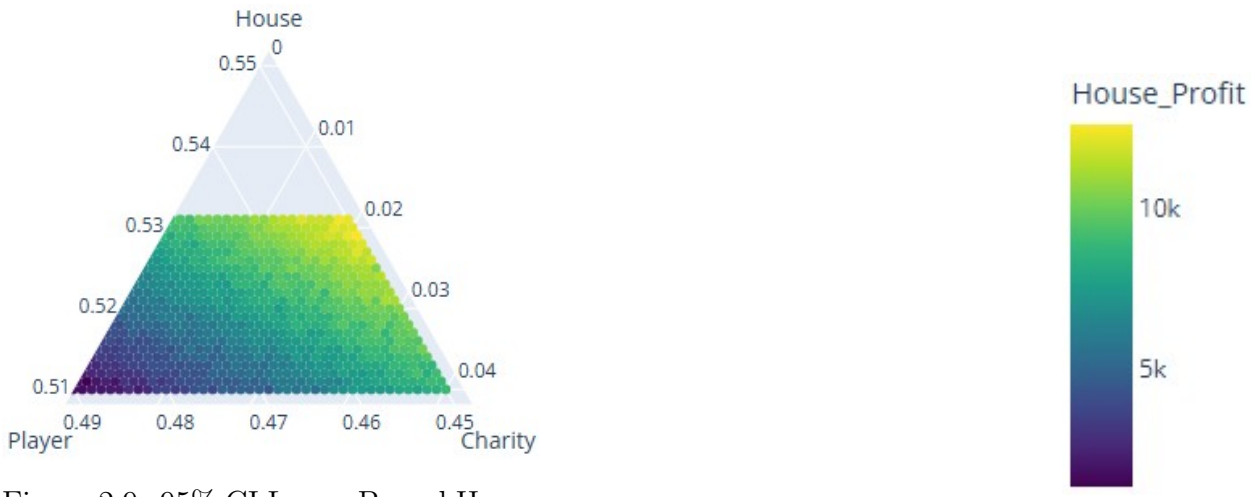

Figure 2.9: 95% CI Lower Bound House Profit

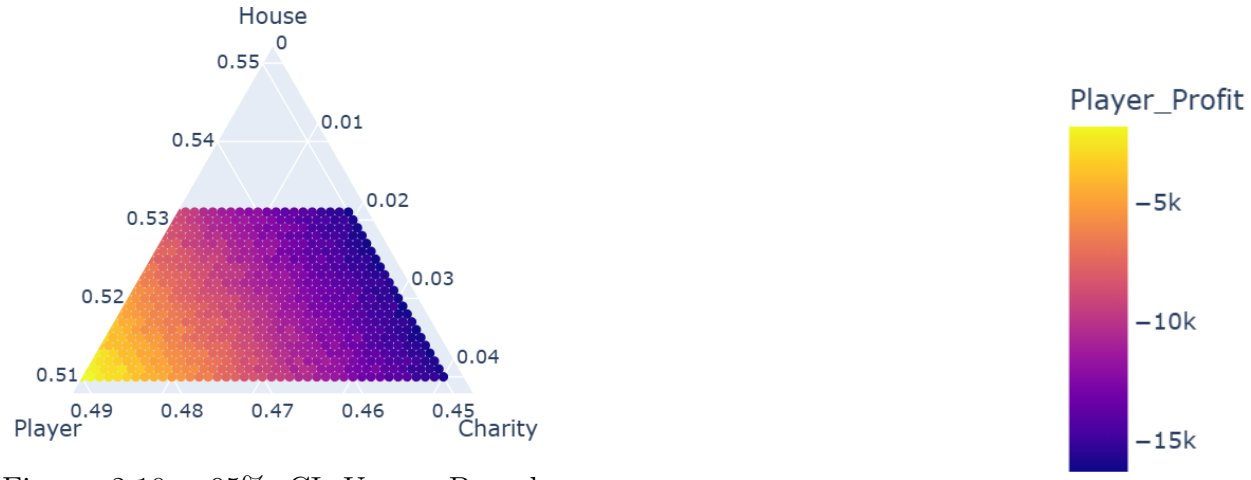

Figure 2.10: 95% CI Upper Bound Player Profit

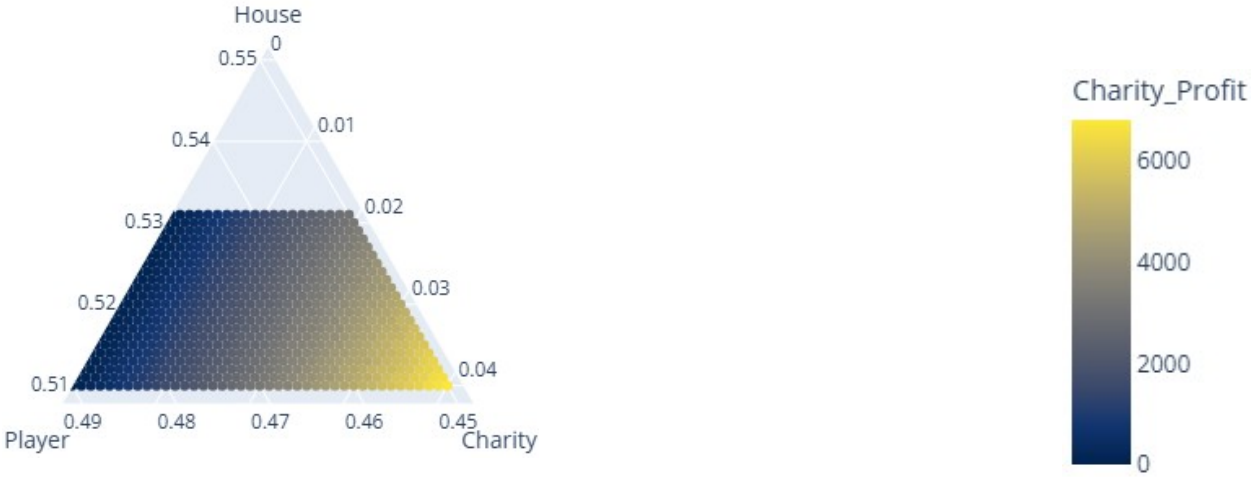

Figure 2.11: 95% CI Lower Bound Charity Profit

Evidently, this is an effective way of introducing an interactive charity element to the game, with good profits being made by The House even at the lower bounds of the confidence interval in the given probability space. However, with this option players aren't excited for the event that the coin lands on its side. Rather than being a rare desirable occurrence, it is just a rare and less bad occurrence. Instead, The House could simply give a percentage of its profits from the game to charity. This would cause players to feel less bad when they lose to the house as they know a percentage of that money is going to charity. It would also leave space for another option that could solve the problem.

2.4 Adding a Jackpot

Another option would be, should the coin land on its side, to instead reward the player with a jackpot value. This would give the player a great rush of excitement and makes the event both a rare and desirable outcome for the player. This again is no longer a zero-sum game and because of the multiplier effect of the jackpot, it is no longer enough for The House's odds to be greater than that of The Player's.

Below we have explored The House's profit when setting the jackpot rate between 0% and 5% (again simulating 100 iterations of 10,000 players playing 3 rounds each). The jackpot is set at 10x the value of the bet placed by the player. It is evident that the jackpot rate must be significantly lower than the charity rate could be. In other words, the coin landing on its side would have to be a significantly less common event.

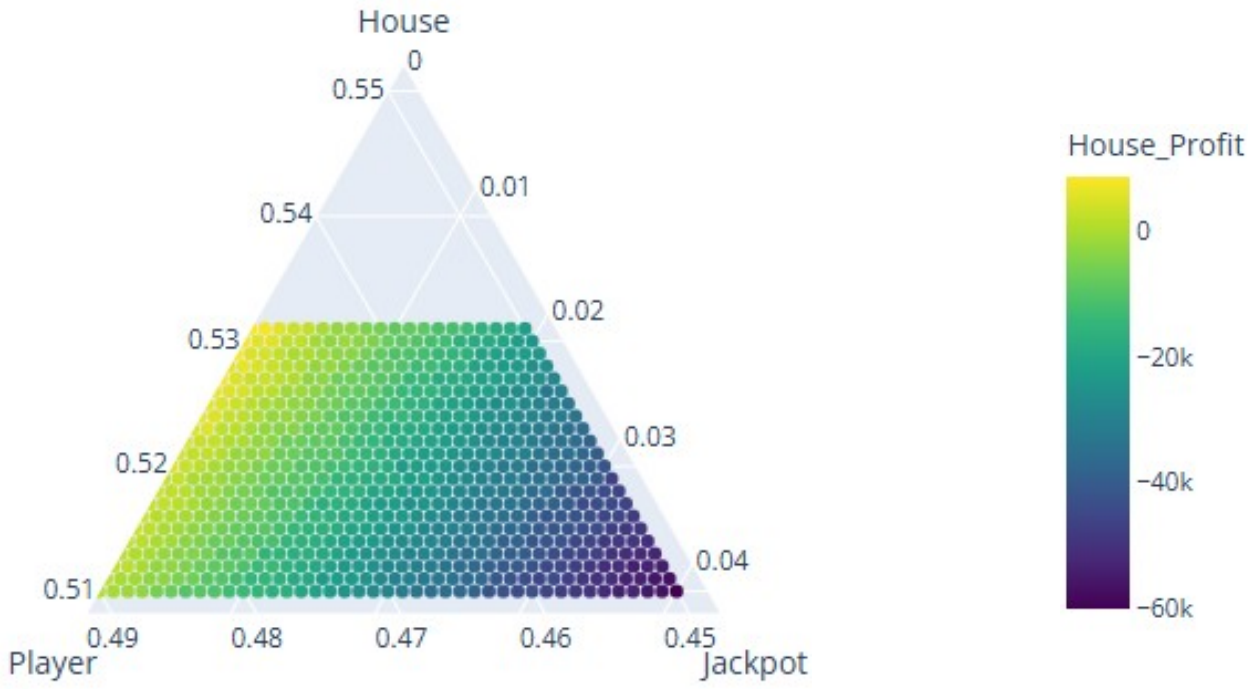

Figure 2.12: House Mean Profits

By setting the jackpot rate between 0 and 0.002 (ie. 2 in 1000 flips are jackpots), and using a step size of 0.0002, we can find combinations of probabilities with profit being made

by The House within the 95% confidence interval. However, one should note that the profits are significantly less than those made with similar odds when the coin landing on its side leads to a charity donation.

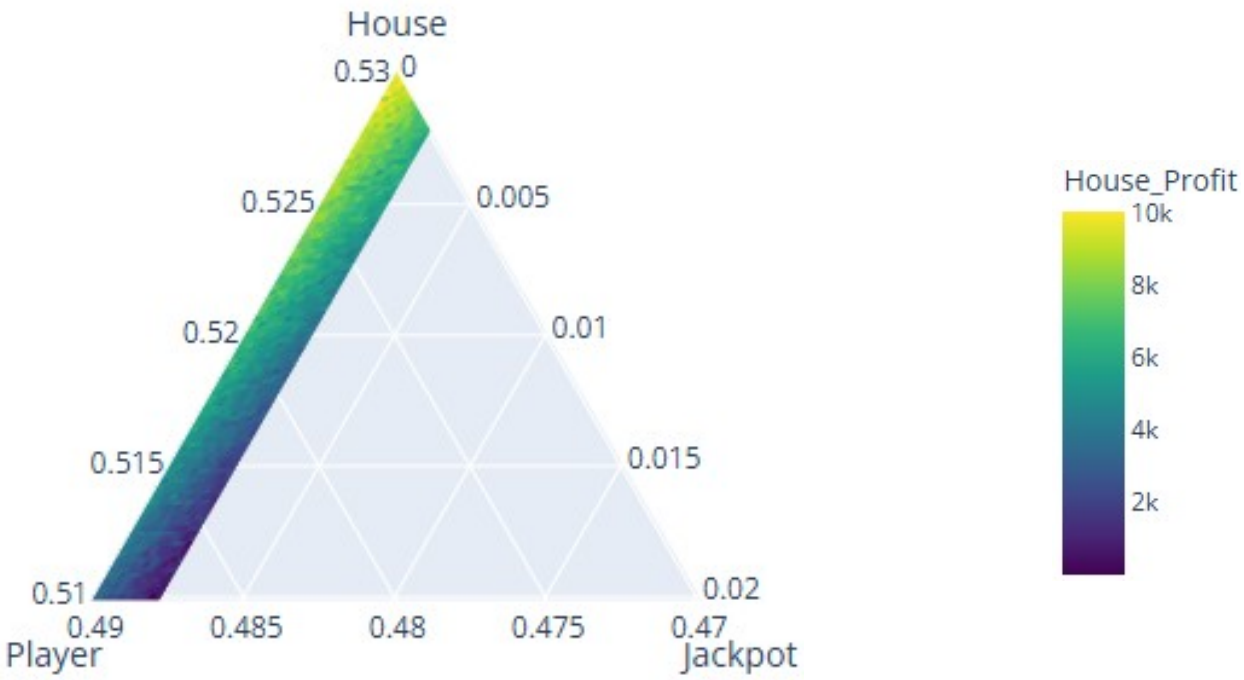

Figure 2.13: House Mean Profits

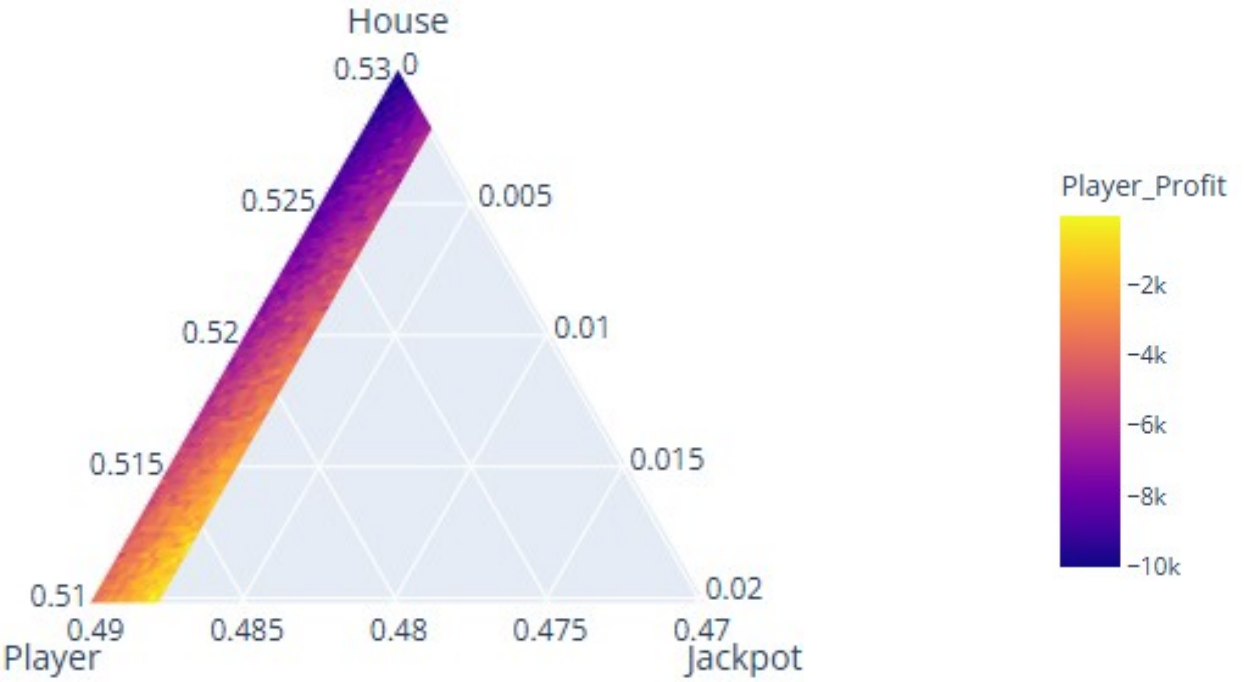

Figure 2.14: Player Mean Profits

Figure 2.15: 95% CI Lower Bound Player Profit

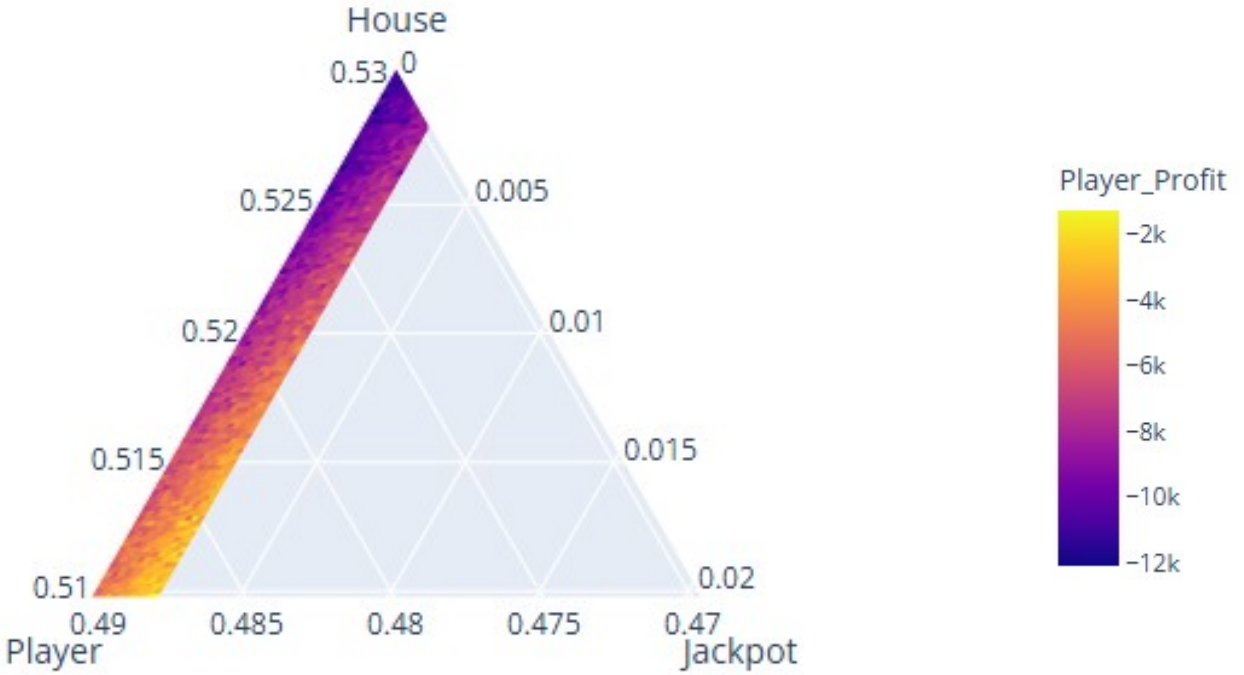

Figure 2.16: 95% CI Lower Bound Charity Profit

2.5 Recommendations

This game is similar in nature to roulette and so the odds should reflect this. A charitable element to the game is important. It can feasibly be included as an interactive part of the game (The coin landing on its side), without significant damage to The House's profits. However, this can just as effectively be included outside of the game as a percentage of The House's profits, which also has the bonus effect of players not feeling as bad when they lose money to The House. This then leaves a space for a different option, such as a jackpot. Whilst it is possible to include a jackpot and make profit, it does cause The House's profit to fall considerably, and thus should only be included if there is confidence that it will be a key factor that influences usership of the game.

2.6 Modelling Assumptions

The main assumption made here is that the value of all bets are randomly uniformly distributed between £2 and £10. In reality, i suspect we would likely see clumping at integer numbers and particularly around numbers such as £2, £5, and £10.

Additionally, when discussing The House's profit, the modelling does not account for additional costs such as the costs of running the servers or transaction fees.

Chapter 3

Crack It

3.1 The Game

This is our daily headline game and USP, a skill-based charitable lottery that merges the thrill of winning with the satisfaction of earning your place. Designed for players that crave more than blind luck, "Crack It" appeals to those who enjoy gameplay, challenges and feeling in control of their odds. Unlike traditional scratch cards or number draws, this daily event introduces a layer of competition and mastery, keeping users coming back to test their knowledge, speed, or accuracy in rotating game modes.

Players will play a minigame. Their performance in this minigame determines their rank amongst other players. The higher you rank, the more scratchcards you win. Each scratchcard then has a select chance of winning you money, ranging from earning back your entry fee, to earning 20x your entry fee.

3.2 Making Profit

Being a self-contained game, there is no chance of The House losing money in this game. The House hosts the game and thus takes a cut of the entrance fee, which is its source of profit. The question then becomes, how much of a cut should The House take and how do we both distribute scratchcards and choose the odds of those scratchcards so that the player pot is exactly distributed to the players and players keep coming back to play?

3.3 Set Up

We denote the following:
Entry fee (y): The fee paid by each player to take part in the game
Game participation (g): The number of players
Payout rate (x): The percentage of the total pot that is distributed between the players
Platform margin (h) = 1 - Payout rate = $1 - x$
Charitable rate (c) = $\frac{1}{10}$ * Platform margin = $\frac{1}{10} * h$

Then we have:
Total pot = Game participation * Entry fee = $g * y$
Player pot $\approx$ Total pot * Payout Rate = $g * y * x$
Charity pot $\approx$ Total pot * Charitable Rate = $g * y * c$
House Profit $\approx$ (Total pot * Platform margin) - Charity pot = $(g * y * h) - (g * y * c) = (g * y) * (0.9 * h)$

We use approximation symbols as these values are altered slightly to ensure the smooth distribution of prizes to players.

3.4 Scratchcards

This game is essentially a scratchcard lottery where your rank in the game decides how many scratchcards you get. All players who enter should have a chance to win something and therefore all players who enter should be given at least one scratchcard. Working from this assumption I have split the players into the following percentiles and awarded each group a different number of scratchcards.

Percentile	Scratchcards per player
$0 - 5\%$	10
$5 - 20\%$	6
$20 - 50\%$	3
$50 - 100\%$	1

Note that these percentiles and number of scratchcards per player should be able to be adjusted as wanted without affecting the integrity of the distribution method of prizes.

From this we can calculate the total number of scratchcards required.

```
# number of scratchcards

# G1 = 0 - 5 %
# G2 = 5 - 20 %
# G3 = 20 - 50 %
# G4 = 50 - 100 %

G1 = round(Fraction(5, 100) * game_participation) * 10 #10 cards per person
G2 = round(Fraction(15, 100) * game_participation) * 6 #6 cards per person
G3 = round(Fraction(30, 100) * game_participation) * 3 #3 cards per person
G4 = round(Fraction(50, 100) * game_participation) #everyone gets 1

total_scratchcards = G1 + G2 + G3 + G4
```

Figure 3.1: Calculating Total Number of Scratchcards

3.5 Prizes

I have modeled the prize values to be multiples of the entry fee. This is important both for to ensure consistency with with changing entry fees throughout the week and also to help with smooth distribution of the Player pot to The Players. There are five categories of prizes, listed as follows:

Prize Level	Prize Value
Prize 1 (P_1)	y
Prize 2 (P_2)	$2 * y$
Prize 3 (P_3)	$4 * y$
Prize 4 (P_4)	$10 * y$
Prize 5 (P_5)	$20 * y$

3.6 Odds

Having defined the prize values, we need to then calculate the number of each prize to give out in order to fairly and fully distribute the prize pot between the players.

We now define the following:

$$\begin{aligned}\textbf{Player pot} &= \text{round}(\text{Game participation} * \text{Payout rate}) * \text{Entry fee} \\ &\approx \text{Total pot} * \text{Payout Rate} \\ &= g * y * x\end{aligned}$$

Scratchcard win % (z): The percentage chance of a scratchcard being a winner
Total winning scratchcard (Z) = round(z * Total scratchcards)
Entry fee multiplier (Q) = round(Game participation * payout rate) = round($g * x$)

With these values we now want to make sure that the total number of prizes we give out is equal to **Z** and the sum of each prize value multiplied by the number of them given out is equal to the Player pot.

If we denote the numbers of each prize awarded as N_1, N_2, N_3, N_4, N_5 respectively we can write the first condition as:

$$N_1 + N_2 + N_3 + N_4 + N_5 = Z \tag{3.1}$$

The second condition can be written as:

$$\begin{aligned}P_1N_1 + P_2N_2 + P_3N_3 + P_4N_4 + P_5N_5 &= \text{Player pot} \\ \Rightarrow\ yN_1 + 2yN_2 + 4yN_3 + 10yN_4 + 20yN_5 &= \text{round}(g \cdot x)\, y \\ \Rightarrow\ N_1 + 2N_2 + 4N_3 + 10N_4 + 20N_5 &= Q \end{aligned} \tag{3.2}$$

Solving these simultaneous equations will ensure that prizes are distributed in such a way that each flash card has a chance of winning approximately equal to that which has

been chosen by The House and that the Player pot, which is a fraction of the Total pot approximately equal to the Payout rate, will be fully distributed to The Players.

Below is an example where the app has 72,315 users, with a game of "Crack It" being played by 20% of these players. The entry fee is set at £3.25 and the Payout rate is set at 68%.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import random
import statistics
import math
from fractions import Fraction

# Set the seed
np.random.seed(42)
random.seed(42)sn
```

```
# Set the variables
players = 72315
participation_rate = Fraction(1, 5)
game_participation = players * participation_rate
entry_fee = Fraction(13, 4) # 3.25
payout_rate = Fraction(68, 100)
platform_margin = 1 - payout_rate
charitable_rate = Fraction(1, 10)

print(f"Game Participation: {game_participation}")
```

```
Game Participation: 14463
```

```
# Pot values
total_pot = game_participation * entry_fee

player_pot = round(game_participation * payout_rate) * entry_fee
leftover_pot = total_pot - player_pot
charity_pot = leftover_pot * charitable_rate
platform_profit = leftover_pot - charity_pot

print(f"Total pot: £{total_pot:.2f}")
print(f"Player pot: £{player_pot:.2f}")
print(f"Charity pot: £{charity_pot:.2f}")
print(f"Platform profit: £{platform_profit:.2f}")
```

```
Total pot: £47004.75
Player pot: £31963.75
Charity pot: £1504.10
Platform profit: £13536.90
```

```
# number of scratchcards

# G1 = 0 - 5 %
# G2 = 5 - 20 %
# G3 = 20 - 50 %
# G4 = 50 - 100 %

G1 = round(Fraction(5, 100) * game_participation) * 10 #10 cards per person
G2 = round(Fraction(15, 100) * game_participation) * 6 #6 cards per person
G3 = round(Fraction(30, 100) * game_participation) * 3 #3 cards per person
G4 = round(Fraction(50, 100) * game_participation) #everyone gets 1

total_scratchcards = G1 + G2 + G3 + G4

print(f"G1: {G1}")
print(f"G2: {G2}")
print(f"G3: {G3}")
print(f"G4: {G4}")
print(f"Total Scratchcards: {total_scratchcards}")
```

```
G1: 7230
G2: 13014
G3: 13017
G4: 7232
Total Scratchcards: 40493
```

```
# Prize Values
P1 = entry_fee * 1
P2 = entry_fee * 2
P3 = entry_fee * 4
P4 = entry_fee * 10
P5 = entry_fee * 20

print(f"P1: £{P1:.2f}")
print(f"P2: £{P2:.2f}")
print(f"P3: £{P3:.2f}")
print(f"P4: £{P4:.2f}")
print(f"P5: £{P5:.2f}")
```

```
P1: £3.25
P2: £6.50
P3: £13.00
P4: £32.50
P5: £65.00
```

```
z = Fraction(1, 10)
total_winning_scratchcards = round(z * total_scratchcards)
Val=round(game_participation * payout_rate)

print(f"Total Winning Scratchcards: {total_winning_scratchcards}")
print(f"Entry Fee Multiplier: {Val}")
```

```
Total Winning Scratchcards: 4049
Entry Fee Multiplier: 9835
```

From here we can form our simultaneous equations:

$$N_1 + N_2 + N_3 + N_4 + N_5 = 4049$$

$$N_1 + 2N_2 + 4N_3 + 10N_4 + 20N_5 = 9835$$

Solving these gives us the number of each type of prize to distribute amongst the flash-cards. It should be noted that in this example

One way of solving these simultaneous equations is by choosing 3 variables manually and working out the other two. There are certainly more effective ways which can be completely automised in the coding, but I will need to do more research into understanding and implementing these methods.

Appendix A

Code Appendix

The following is the code used to plot the House and Player mean profit graphs with 95% confidence intervals in Chapter 2:

```
outcomes = ["house", "player", "charity", "jackpot"]
probabilities = [0.51, 0.49, 0, 0]  # Outcome probabilities
num_rounds = list(range(1, 11))  # How number of rounds affects p/l
num_players = 1000  # Number of players per round
num_simulations = 100  # Number of Monte Carlo runs
jackpot_val = 100

# Initialise arrays to track profit/loss
house_value = np.zeros((num_simulations, len(num_rounds)))
player_value = np.zeros((num_simulations, len(num_rounds)))
charity_value = np.zeros((num_simulations, len(num_rounds)))

for sim in range(num_simulations):
    for r_idx, rounds in enumerate(num_rounds):
        # Generate all bets for all players across all rounds at
            once
        bet_vals = np.random.uniform(2, 10, (rounds, num_players)).
            round(2)
        # Define jackpot amount
        jackpot_vals = bet_vals * 10
        # Generate all outcomes in one go
        outcomes_idx = np.random.choice(len(outcomes), size=(rounds,
            num_players), p=probabilities)

        # Vectorized profit calculations
        house_profit = np.round(bet_vals[outcomes_idx == 0].sum() -
            bet_vals[outcomes_idx == 1].sum(), 2)
        player_profit = np.round(bet_vals[outcomes_idx == 1].sum() -
            bet_vals[outcomes_idx == 0].sum(), 2)
        charity_profit = np.round(bet_vals[outcomes_idx == 2].sum(),
            2)
```

```
26
27          # Jackpot adjustment
28          jackpot_mask = (outcomes_idx == 3)
29          jackpot_payout = jackpot_vals[jackpot_mask].sum()
30
31          player_profit += np.round(jackpot_payout, 2)
32          house_profit  -= np.round(jackpot_payout, 2)
33
34          # Store results
35          house_value[sim, r_idx] = house_profit
36          player_value[sim, r_idx] = player_profit
37          charity_value[sim, r_idx] = charity_profit
38
39  # Statistics for house_value
40  house_mean = np.mean(house_value, axis=0).tolist()
41  house_sd   = np.std(house_value, axis=0).tolist()
42  house_var  = np.var(house_value, axis=0).tolist()
43  house_ci_lower = np.percentile(house_value, 2.5, axis=0)  # 2.5th
       percentile
44  house_ci_upper = np.percentile(house_value, 97.5, axis=0) # 97.5th
       percentile
45
46  # Statistics for player_value
47  player_mean = np.mean(player_value, axis=0).tolist()
48  player_sd   = np.std(player_value, axis=0).tolist()
49  player_var  = np.var(player_value, axis=0).tolist()
50  player_ci_lower = np.percentile(player_value, 2.5, axis=0)
51  player_ci_upper = np.percentile(player_value, 97.5, axis=0)
52
53  # Define total trials (rounds    players)
54  total_rounds = [r * num_players for r in num_rounds]
55
56  fig, axes = plt.subplots(1, 2, figsize=(14, 5), sharex=True)
57
58  # Plot house mean profit
59  axes[0].plot(total_rounds, house_mean, label="House␣Mean␣Profit",
       color="blue")
60  axes[0].fill_between(total_rounds, house_ci_lower, house_ci_upper,
       color="blue", alpha=0.2, label="95%␣CI")
61  axes[0].scatter(total_rounds, house_mean, color="blue", s=50, zorder
       =3)
62
63  for xi, yi, lower, upper in zip(total_rounds, house_mean,
       house_ci_lower, house_ci_upper):
64      axes[0].text(xi, yi + 50, f"{yi:.0f}", ha="center", fontsize=8,
           color="black")
65      axes[0].text(xi, upper + 50, f"{upper:.0f}", ha="center",
```

```
        fontsize=8, color="black")
    axes[0].text(xi, lower - 100, f"{lower:.0f}", ha="center",
        fontsize=8, color="black")

axes[0].set_xticks(total_rounds)
axes[0].set_yticks(np.arange(-1000, 3500, 500))
axes[0].set_title("House Profit with 95% CI")
axes[0].set_xlabel("Total Number of Rounds Played")
axes[0].set_ylabel("Profit (  )")
axes[0].legend()
axes[0].grid(True)

# Plot player mean profit
axes[1].plot(total_rounds, player_mean, label="Player Mean Profit",
    color="green")
axes[1].fill_between(total_rounds, player_ci_lower, player_ci_upper,
    color="green", alpha=0.2, label="95% CI")
axes[1].scatter(total_rounds, player_mean, color="green", s=50,
    zorder=3)

for xi, yi, lower, upper in zip(total_rounds, player_mean,
    player_ci_lower, player_ci_upper):
    axes[1].text(xi, yi + 50, f"{yi:.0f}", ha="center", fontsize=8,
        color="black")
    axes[1].text(xi, upper + 50, f"{upper:.0f}", ha="center",
        fontsize=8, color="black")
    axes[1].text(xi, lower - 100, f"{lower:.0f}", ha="center",
        fontsize=8, color="black")

axes[1].set_xticks(total_rounds)
axes[1].set_yticks(np.arange(-3000, 1500, 500))
axes[1].set_title("Player Profit with 95% CI")
axes[1].set_xlabel("Total Number of Rounds Played")
axes[1].legend()
axes[1].grid(True)

plt.savefig("PnL1.pdf", format="pdf", bbox_inches="tight")
plt.tight_layout()
plt.show()
```

The following is the code used to plot the House, Player and Charity mean profit Ternary graphs in Chapter 2. The code for the 95% confidence interval lower bound graphs is nearly identical:

```
prob_list = [] # Initialise list of probabilities
step = 0.001 # Set step value
p1_values = np.arange(0.51, 0.53 + step, step)

```

```
for p1 in p1_values:
    # p2 must be     p1 and     1-p1
    max_p2 = min(0.55, p1, 1 - p1)
    min_p2 = 0.45
    p2_values = np.arange(min_p2, max_p2+step, step)
    for p2 in p2_values:
        p3 = round(1 - (p1 + p2), 3)
        if p3 >= 0:
            prob_list.append([round(p1,3), round(p2,3), p3, 0.0])

outcomes = ["house", "player", "charity", "jackpot"]
probabilities = prob_list  # Outcome probabilities
num_rounds = 3  # How number of rounds affects p/l
num_players = 10000  # Number of players per round
num_simulations = 100  # Number of Monte Carlo runs

# Initialise arrays to track profit/loss
house_value = np.zeros((num_simulations, len(prob_list)))
player_value = np.zeros((num_simulations, len(prob_list)))
charity_value = np.zeros((num_simulations, len(prob_list)))

for sim in range(num_simulations):
    for r_idx, probs in enumerate(prob_list):
        # Fixed number of rounds
        bet_vals = np.random.uniform(2, 10, (num_rounds, num_players
           )).round(2)
        # Define jackpot amount
        jackpot_vals = bet_vals * 10
        # Generate outcomes
        outcomes_idx = np.random.choice(len(outcomes), size=(
           num_rounds, num_players), p=probs)

        # Profit calculations
        house_profit = np.round(bet_vals[outcomes_idx == 0].sum() -
           bet_vals[outcomes_idx == 1].sum(), 2)
        player_profit = np.round(bet_vals[outcomes_idx == 1].sum() -
            bet_vals[outcomes_idx == 0].sum() - bet_vals[
           outcomes_idx == 2].sum(), 2)
        charity_profit = np.round(bet_vals[outcomes_idx == 2].sum(),
            2)

        # Jackpot adjustment
        jackpot_mask = (outcomes_idx == 3)
        jackpot_payout = jackpot_vals[jackpot_mask].sum()

        player_profit += np.round(jackpot_payout, 2)
        house_profit  -= np.round(jackpot_payout, 2)
```

```
46
47          # Store results
48          house_value[sim, r_idx] = house_profit
49          player_value[sim, r_idx] = player_profit
50          charity_value[sim, r_idx] = charity_profit
51
52  # Statistics for house_value
53  house_mean = np.mean(house_value, axis=0).tolist()
54  house_sd   = np.std(house_value, axis=0).tolist()
55  house_var  = np.var(house_value, axis=0).tolist()
56  house_ci_lower = np.percentile(house_value, 2.5, axis=0)  # 2.5th
        percentile
57  house_ci_upper = np.percentile(house_value, 97.5, axis=0) # 97.5th
        percentile
58
59  # Statistics for player_value
60  player_mean = np.mean(player_value, axis=0).tolist()
61  player_sd   = np.std(player_value, axis=0).tolist()
62  player_var  = np.var(player_value, axis=0).tolist()
63  player_ci_lower = np.percentile(player_value, 2.5, axis=0)
64  player_ci_upper = np.percentile(player_value, 97.5, axis=0)
65
66  # Statistics for charity_value
67  charity_mean = np.mean(charity_value, axis=0).tolist()
68  charity_sd = np.std(charity_value, axis=0).tolist()
69  charity_var = np.var(charity_value, axis=0).tolist()
70  charity_ci_lower = np.percentile(charity_value, 2.5, axis=0)
71  charity_ci_upper = np.percentile(charity_value, 97.5, axis=0)
72
73  p1_vals = [p[0] for p in prob_list]  # house probability
74  p2_vals = [p[1] for p in prob_list]  # player probability
75
76  # Create DataFrame
77  df = pd.DataFrame(prob_list, columns=["House", "Player", "Charity",
        "Jackpot"])
78
79  # Add profit columns
80  df["House_Profit"] = house_mean
81  df["Player_Profit"] = player_mean
82  df["Charity_Profit"] = charity_mean
83
84  # Ternary Plot: House Profit
85  fig_house = px.scatter_ternary(
86      df,
87      a="House", b="Player", c="Charity",
88      #size=df["House_Profit"].abs(),
89      color="House_Profit",
```

```
    hover_data=["House_Profit"],
    color_continuous_scale="Viridis",
)
fig_house.update_layout(title="House Profit across Probability Space")
fig_house.show()
fig_house.write_html("tri_HP1.html")

# Ternary Plot: Player Profit
fig_player = px.scatter_ternary(
    df,
    a="House", b="Player", c="Charity",
    #size=df["Player_Profit"].abs(),
    color="Player_Profit",
    hover_data=["Player_Profit"],
    color_continuous_scale="Plasma",
)
fig_player.update_layout(title="Player Profit across Probability Space")
fig_player.show()
fig_player.write_html("tri_PP1.html")

# Ternary Plot: Charity Profit
fig_charity = px.scatter_ternary(
    df,
    a="House", b="Player", c="Charity",
    #size=df["Charity_Profit"].abs(),
    color="Charity_Profit",
    hover_data=["Charity_Profit"],
    color_continuous_scale="Cividis",
)
fig_charity.update_layout(title="Charity Profit across Probability Space")
fig_charity.show()

fig_charity.write_html("tri_CP1.html")
```

The following is the code used to plot the House and Player mean profit Ternary graphs for the addition of jackpots in Chapter 2. The code for the 95% confidence interval lower bound graphs is nearly identical:

```
prob_list = []
step = 0.0002
p1_values = np.arange(0.51, 0.53 + step, step)

for p1 in p1_values:
    # p2 must be between 0.45 and min(p1, 1-p1, 0.55)
    max_p2 = min(0.55, p1, 1 - p1)
```

```
    min_p2 = 0.47
    p2_values = np.arange(min_p2, max_p2 + step, step)

    for p2 in p2_values:
        p3 = 0.0
        p4 = round(1 - (p1 + p2 + p3), 5)

        # Apply constraints
        if p4 >= 0 and p4 <= 0.002:
            if p1 > (p2 + p4) + 0.02:
                prob_list.append([round(p1, 4), round(p2, 4), p3, p4
                    ])

outcomes = ["house", "player", "charity", "jackpot"]
probabilities = prob_list  # Outcome probabilities
num_rounds = 3  # How number of rounds affects p/l
num_players = 10000  # Number of players per round
num_simulations = 10  # Number of Monte Carlo runs

# Initialise arrays to track profit/loss
house_value = np.zeros((num_simulations, len(prob_list)))
player_value = np.zeros((num_simulations, len(prob_list)))
charity_value = np.zeros((num_simulations, len(prob_list)))

for sim in range(num_simulations):
    for r_idx, probs in enumerate(prob_list):
        # Fixed number of rounds
        bet_vals = np.random.uniform(1, 10, (num_rounds, num_players
            )).round(2)
        # Define jackpot value
        jackpot_vals = bet_vals * 10
        # Generate outcomes
        outcomes_idx = np.random.choice(len(outcomes), size=(
            num_rounds, num_players), p=probs)

        # Profit calculations
        house_profit = np.round(bet_vals[outcomes_idx == 0].sum() -
            bet_vals[outcomes_idx == 1].sum(), 2)
        player_profit = np.round(bet_vals[outcomes_idx == 1].sum() -
            bet_vals[outcomes_idx == 0].sum() - bet_vals[
            outcomes_idx == 2].sum(), 2)
        charity_profit = np.round(bet_vals[outcomes_idx == 2].sum(),
            2)

        # Jackpot adjustment
        jackpot_mask = (outcomes_idx == 3)
        jackpot_payout = jackpot_vals[jackpot_mask].sum()
```

```
48
49          player_profit += np.round(jackpot_payout, 2)
50          house_profit  -= np.round(jackpot_payout, 2)
51
52          # Store results
53          house_value[sim, r_idx] = house_profit
54          player_value[sim, r_idx] = player_profit
55          charity_value[sim, r_idx] = charity_profit
56
57  # Statistics for house_value
58  house_mean = np.mean(house_value, axis=0).tolist()
59  house_sd   = np.std(house_value, axis=0).tolist()
60  house_var  = np.var(house_value, axis=0).tolist()
61  house_ci_lower = np.percentile(house_value, 2.5, axis=0)  # 2.5th
       percentile
62  house_ci_upper = np.percentile(house_value, 97.5, axis=0) # 97.5th
       percentile
63
64  # Statistics for player_value
65  player_mean = np.mean(player_value, axis=0).tolist()
66  player_sd   = np.std(player_value, axis=0).tolist()
67  player_var  = np.var(player_value, axis=0).tolist()
68  player_ci_lower = np.percentile(player_value, 2.5, axis=0)
69  player_ci_upper = np.percentile(player_value, 97.5, axis=0)
70
71  # Statistics for charity_value
72  charity_mean = np.mean(charity_value, axis=0).tolist()
73  charity_sd = np.std(charity_value, axis=0).tolist()
74  charity_var = np.var(charity_value, axis=0).tolist()
75  charity_ci_lower = np.percentile(charity_value, 2.5, axis=0)
76  charity_ci_upper = np.percentile(charity_value, 97.5, axis=0)
77
78  p1_vals = [p[0] for p in prob_list]  # house probability
79  p2_vals = [p[1] for p in prob_list]  # player probability
80
81  # Create DataFrame
82  df = pd.DataFrame(prob_list, columns=["House", "Player", "Charity",
       "Jackpot"])
83
84  # Add profit columns
85  df["House_Profit"] = house_mean
86  df["Player_Profit"] = player_mean
87  df["Charity_Profit"] = charity_mean
88
89  # Ternary plot house profit (House, Player, Jackpot)
90  fig_house = px.scatter_ternary(
91      df,
```

```
    a="House", b="Player", c="Jackpot",   # <-- swapped Charity Jackpot
    #size=df["House_Profit"].abs(),
    color="House_Profit",
    hover_data=["House_Profit", "Charity"],  # keep Charity in hover if you still want to inspect
    color_continuous_scale="Viridis",
)
fig_house.update_layout(title="House Profit across Probability Space (Jackpot axis)")
fig_house.show()

# Ternary plot player profit (House, Player, Jackpot)
fig_player = px.scatter_ternary(
    df,
    a="House", b="Player", c="Jackpot",   # <-- swapped
    #size=df["Player_Profit"].abs(),
    color="Player_Profit",
    hover_data=["Player_Profit", "Charity"],
    color_continuous_scale="Plasma",
)
fig_player.update_layout(title="Player Profit across Probability Space (Jackpot axis)")
fig_player.show()
```