Snakes and Ladders

Código em: https://github.com/vininaka/snakes_ladders/blob/main/Ladders and Snakes.ipynb

1. Contexto e objetivo geral

Em um jogo de tabuleiro de *Snakes and Ladders*, cada jogador começa na casa 1 e se reveza jogando um dado de seis faces. O jogador move o número de casas indicado no dado. Se você cair na base de uma escada, você automaticamente se move para a casa no topo da escada. Da mesma forma, se você cair na cabeça de uma cobra, então você cai para a casa na cauda da cobra. O vencedor é a primeira pessoa a chegar na última casa ou passar por ela.

Assim como todo jogo de tabuleiro, as chances de cada jogador podem variar de acordo com o cenário e as situações apresentadas. Desse modo, o objetivo principal desse trabalho é entender o funcionamento do jogo, suas possíveis variações e com isso, realizar simulações para se realizar análises descritivas e estatísticas sobre os resultados apresentados.

2. Perguntas e questionamentos

Para guiar o desenvolvimento das análises, alguns questionamentos foram desenvolvidos:

- Em um jogo para duas pessoas, qual é a probabilidade de que o jogador que começa o jogo vença?
- Em média, quantas cobras são aterrissadas em cada jogo?
- Se cada vez que um jogador aterrissasse em uma escada e houvesse apenas 50% de chance de que eles pudessem usá-la, qual é o número médio de jogadas necessárias para completar um jogo?
- Começando com o jogo base, você decide que deseja que o jogo tenha probabilidades aproximadamente justas. Você faz isso mudando a casa em que o Jogador 2 começa. Qual casa para a posição inicial do Jogador 2 proporciona probabilidades mais próximas de igualdade para ambos os jogadores?
- Em uma tentativa diferente de mudar as probabilidades do jogo, em vez de começar o Jogador 2 em uma casa diferente, você decide dar imunidade ao Jogador 2 na primeira cobra em que ele aterrissa. Qual é a probabilidade aproximada de que o Jogador 1 vença agora?

3. Metodologia

Como estamos lidando com um cenário de **simulações**, essa abordagem objetiva consiste em **repetir um experimento muitas vezes e registrar a frequência relativa dos resultados**. Eventos mais prováveis ocorrem com mais frequência, eventos menos prováveis ocorrem com menos frequência. Proporcionalmente, os resultados indicam as probabilidades relativas. Quanto mais amostras ou experimentos você realizar, maior será a confiança que terá em seus resultados.

A ideia para resolver os problemas apresentados tem com uma solução, um processo semelhante as **simulações de Monte Carlo**, no qual um sistema é executado inúmeras vezes e os seus resultados são analisados.

As simulações de Monte Carlo têm como princípio básico, o desenvolvimento de um sistema. Desse modo, observando-se os questionamentos apresentados, esse trabalho será divido em 4 etapas:

- Sistema 1 – Desenvolvimento da base do jogo de tabuleiro de *Snakes and Ladders*

- Sistema 2 – **Adição de um ruído ao sistema 1** desenvolvido anteriormente, através de um condicional que possibilita ou não jogadores utilizarem escadas durante o jogo.
- Sistema 3 – **Inserção ao sistema 1 de uma posição variável** para início do jogador número 2, o que possibilita simulações para se obter chances iguais de vitória para cada um.
- Sistema 4 - **Inserção ao sistema 1 de uma variável *booleana*** de imunidade para o jogador número 2

Como pode ser observado, inicialmente será desenvolvido um código base, e partir dele, as modificações necessárias irão ser realizadas. Isso facilita para um melhor reaproveitamento de código e uma melhor otimização de tempo de desenvolvimento do trabalho como um todo.

4. Desenvolvimento

A base do desenvolvimento desse trabalho foi realizada utilizando-se a linguagem de programação **Python(https://www.python.org/)**, com o auxílio da ferramenta **Jupyter Notebook (https://jupyter.org/)**, desse modo foi possível realizar o desenvolvimento dos sistemas propostos acima e as análises necessárias, utilizando-se somente de uma única ferramenta.

- **Sistema 1 – Jogo Tradicional**

Inicialmente, como apresentado pela Figura 1, foi realizado o desenvolvimento do sistema base, com apenas o jogo de tabuleiro em sua versão tradicional com **36 espaços possíveis**.

```
def play_game():
    player1_position = 1
    player2_position = 1

    turns = 0
    count_ladders_p1 = 0
    count_ladders_p2 = 0
    count_snakes_p1 = 0
    count_snakes_p2 = 0

    while player1_position < 36 and player2_position < 36:
        # Player 1's turn
        roll = random.randint(1, 6)
        player1_position += roll
        if str(player1_position) in ladder_heads:
            count_ladders_p1+=1
            player1_position = int(ladder_heads[str(player1_position)])

        elif str(player1_position) in snake_heads:
            count_snakes_p1+=1
            player1_position = int(snake_heads[str(player1_position)])

        if player1_position < 36:
            # Player 2's turn
            roll = random.randint(1, 6)
            player2_position += roll
            if str(player2_position) in ladder_heads:
                count_ladders_p2+=1
                player2_position = int(ladder_heads[str(player2_position)])

            elif str(player2_position) in snake_heads:
                count_snakes_p2+=1
                player2_position = int(snake_heads[str(player2_position)])

        turns+=1

    info_dict = {
        'turns':turns,
        'ladders_p1': count_ladders_p1,
        'ladders_p2': count_ladders_p2,
        'snakes_p1': count_snakes_p1,
        'snakes_p2': count_snakes_p2,
        'final_p1': player1_position,
        'final_p2': player2_position
    }
    return info_dict
```

Figura 1 - Desenvolvimento do jogo base de *Snakes and Ladders*

De maneira geral, na implementação acima, cada jogador começa na **posição inicial 1** e os turnos vão acontecendo até que um jogador chegue em uma **posição maior ou igual que 36**. Caso um jogador caia em uma escada, a sua posição será atualizada para a ponta dessa escada e caso um jogador caia na cabeça de uma cobra, a sua posição vai ser atualizada para a cauda dessa cobra.

Como pode ser observado abaixo, a Figura 2 apresenta as configurações utilizadas pelas simulações e mapeamento de objetos do cenário (cobras e escadas). A cada iteração da simulação, são salvas as seguintes informações em um **Pandas DataFrame (https://pandas.pydata.org)** :

- Número de turnos da simulação
- Número de escadas utilizadas por cada jogador
- Número de cobras utilizadas por cada jogador
- Posição final de cada jogador

Inital simulations configs

```
# Mapping Ladders and Snakes
ladder_heads = {'3': '16', '5': '7', '15': '25', '18': '20', '21': '32'}
snake_heads = {'12': '2', '14': '11', '17': '4', '31': '19', '35': '22'}
```

```
# Define number of simulations
num_games = 10000
```

Simulation - Case 1

```
# Generate 10000 simulations for the traditional implementation of snakes and ladders and get the following information:

* Number of turns in simulation
* Number of ladders taken by each player
* Number of snakes landed by each player
* Final position of each player
```

```
case1 = []
for _ in range(num_games):
    case1.append(play_game())

case1_stats = pd.DataFrame(case1)
case1_stats['round'] = np.arange(1,10001)
case1_stats['win_p1'] = case1_stats['final_p1'] >= 36

case1_stats['final_ladders'] = case1_stats['ladders_p1'] + case1_stats['ladders_p2']
case1_stats['final_snakes'] = case1_stats['snakes_p1'] + case1_stats['snakes_p2']

case1_stats = case1_stats[['round','turns', 'ladders_p1', 'ladders_p2', 'snakes_p1',
                           'snakes_p2','final_ladders','final_snakes','final_p1', 'final_p2','win_p1']]
```

Figura 2 – Configuração de simulação para o sistema 1

- **Sistema 2 – Jogo Tradicional com adição de um pequeno ruído**

Para o desenvolvimento do sistema 2, realizou-se o reaproveitamento do código base apresentado anteriormente pela Figura 1, com adição de um condicional de aleatoriedade que vai de 0 a 1 (biblioteca **random do Python**). Desse modo, valores acima de 0.5 permitem que usuários utilizem a escada em cada iteração. Como pode ser observado na Figura 3, **as setinhas em vermelho,** indicam as modificações feitas no código base.

Não houve modificações nos códigos de simulação, por isso não foi necessário a inserção de uma representação dessa etapa para simulações nesse caso.

```python
## Insert a little randomness on our code for the ladders
def play_game():
    player1_position = 1
    player2_position = 1

    turns = 0
    count_ladders_p1 = 0
    count_ladders_p2 = 0
    count_snakes_p1 = 0
    count_snakes_p2 = 0

    while player1_position < 36 and player2_position < 36:
        # Player 1's turn
        roll = random.randint(1, 6)
        player1_position += roll
        if str(player1_position) in ladder_heads:
            if random.random() > 0.5:
                count_ladders_p1+=1
                player1_position = int(ladder_heads[str(player1_position)])

        elif str(player1_position) in snake_heads:
            count_snakes_p1+=1
            player1_position = int(snake_heads[str(player1_position)])

        if player1_position < 36:
            # Player 2's turn
            roll = random.randint(1, 6)
            player2_position += roll
            if str(player2_position) in ladder_heads:
                if random.random() > 0.5:
                    count_ladders_p2+=1
                    player2_position = int(ladder_heads[str(player2_position)])

            elif str(player2_position) in snake_heads:
                count_snakes_p2+=1
                player2_position = int(snake_heads[str(player2_position)])

        turns+=1

    info_dict = {
        'turns':turns,
        'ladders_p1': count_ladders_p1,
        'ladders_p2': count_ladders_p2,
        'snakes_p1': count_snakes_p1,
        'snakes_p2': count_snakes_p2,
        'final_p1': player1_position,
        'final_p2': player2_position
    }
    return info_dict
```

Figura 3 – Desenvolvimento do sistema 2

- **Sistema 3 – Jogo Tradicional com inserção ao sistema 1 de uma posição variável**

O objetivo principal do sistema 3 apresentado pela Figura 4 foi de encontrar qual seria a posição inicial do jogador número 2 para que as chances de vitórias entre ambos os jogadores fossem iguais.

Desse modo, de acordo com a Figura 4, foi criado um método que tem como entrada uma posição variável a posição de início do jogador 2 e para cada iteração do jogo, esse método tem como retorno qual jogador foi o vencedor de cada partida.

Para encontrarmos a melhor posição de início, como apresentado pela Figura 5, foram realizadas **350000** simulações (35 posições variáveis de início e 10000 simulações cada) e em cada uma, foi calculada a diferença de vitórias entre os jogadores. A posição de melhor início para o jogador 2, foi definida com base nas simulações em que essa **diferença foi a menor possível**.

```python
def play_game(player2_start):
    player1_position = 1
    player2_position = player2_start

    while player1_position < 36 and player2_position < 36:
        # Player 1's turn
        roll = random.randint(1, 6)
        player1_position += roll
        if str(player1_position) in ladder_heads:
            player1_position = int(ladder_heads[str(player1_position)])

        elif str(player1_position) in snake_heads:
            player1_position = int(snake_heads[str(player1_position)])

        if player1_position < 36:
            # Player 2's turn
            roll = random.randint(1, 6)
            player2_position += roll
            if str(player2_position) in ladder_heads:
                player2_position = int(ladder_heads[str(player2_position)])

            elif str(player2_position) in snake_heads:
                player2_position = int(snake_heads[str(player2_position)])

    return player1_position >= 36
```

Figura 4 – Implementação do sistema 3

```python
# Simulate games with different starting positions for Player 2
num_games = 10000
best_start_position = None
min_difference = float('inf')

df_ranges = pd.DataFrame(columns = ['range','diff'])
for start_position in range(2, 36):
    player1_wins = sum(play_game(start_position) for _ in range(num_games))
    player2_wins = num_games - player1_wins
    difference = abs(player1_wins - player2_wins)
    df_ranges = df_ranges.append({
        'range':start_position,
        'diff':difference
    },ignore_index=True)
    if difference < min_difference:
        min_difference = difference
        best_start_position = start_position

print(f"Best starting position for Player 2: {best_start_position}")
```

Figura 5 – Simulação do sistema 3 para se obter a melhor posição de início do jogador 2

- **Sistema 4 – Jogo Tradicional com inserção ao sistema 1 de uma variável *booleana*** de imunidade para o jogador número 2

Como pode ser observado pela Figura 6, foi adicionado uma **variável *booleana*** (verdadeiro/falso) ao jogo tradicional base criado anteriormente que permite ao jogador 2 evite a primeira vez que encontrar uma cobra na simulação.

```python
## Insert immunity one time to snakes for player 2
def play_game():
    player_immunity = True
    player1_position = 1
    player2_position = 1

    turns = 0
    count_ladders_p1 = 0
    count_ladders_p2 = 0
    count_snakes_p1 = 0
    count_snakes_p2 = 0

    while player1_position < 36 and player2_position < 36:
        # Player 1's turn
        roll = random.randint(1, 6)
        player1_position += roll
        if str(player1_position) in ladder_heads:
            count_ladders_p1+=1
            player1_position = int(ladder_heads[str(player1_position)])

        elif str(player1_position) in snake_heads:
            count_snakes_p1+=1
            player1_position = int(snake_heads[str(player1_position)])

        if player1_position < 36:
            # Player 2's turn
            roll = random.randint(1, 6)
            player2_position += roll
            if str(player2_position) in ladder_heads:
                count_ladders_p2+=1
                player2_position = int(ladder_heads[str(player2_position)])
            elif str(player2_position) in snake_heads:
                if player_immunity:
                    count_snakes_p2+=1
                    player2_position = int(snake_heads[str(player2_position)])
                    player_immunity = False

        turns+=1

    info_dict = {
        'turns':turns,
        'ladders_p1': count_ladders_p1,
        'ladders_p2': count_ladders_p2,
        'snakes_p1': count_snakes_p1,
        'snakes_p2': count_snakes_p2,
        'final_p1': player1_position,
        'final_p2': player2_position
    }
    return info_dict
```

Figura 6 – Desenvolvimento sistema 4 com imunidade para o jogador 2

Assim como anteriormente, não houve modificações nos códigos de simulação, por isso não foi necessário a inserção de uma representação dessa etapa para simulações nesse caso.

5. Resultados

Nessa etapa, serão apresentados os resultados obtidos nas simulações em cada sistema desenvolvido, assim como as respostas para os questionamentos da **Seção 2**.

Com o objetivo de entender as chances de cada jogador em um jogo tradicional de *Snakes and Ladders,* após as 10000 simulações realizadas, inicialmente, foi verificado a probabilidade de vitória de cada jogador de acordo com os resultados obtidos. Como pode ser observado pela Figura 7, o jogador 1 obteve um total de **5288** vitórias dentro os **10000 jogos disputados**. Desse modo a probabilidade de vitória de cada jogador é dada pela seguinte fórmula:

Probabilidade jogador 1 ganhar: 5288/10000 = **52,88%**
Probabilidade jogador 2 ganhar: 1 – P(Jogador 1 ganhar) = **47,12%**

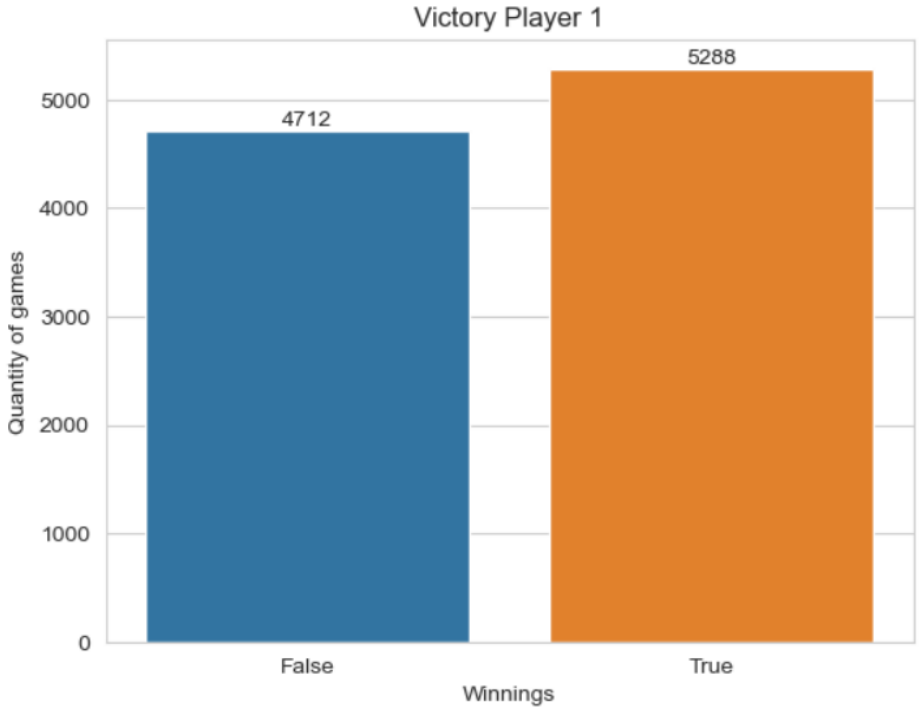
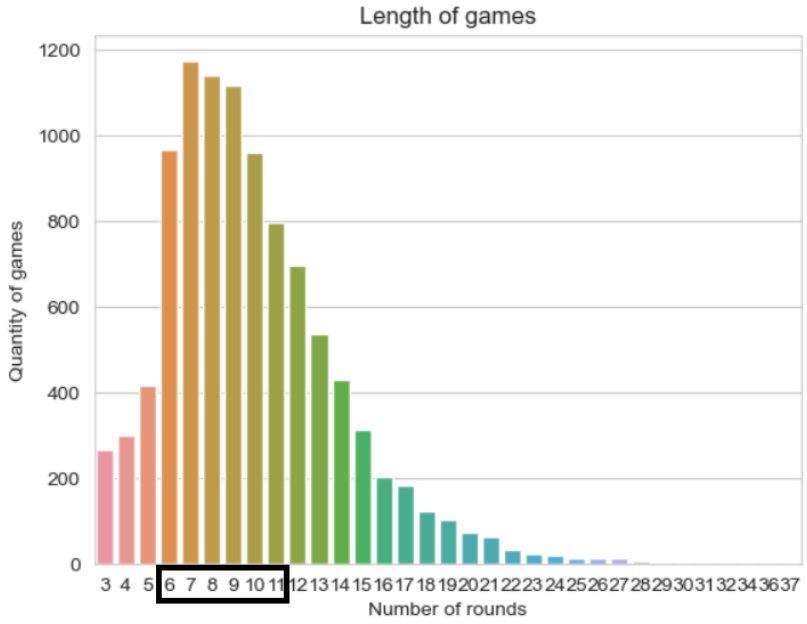


Figura 7 – Número de vitórias e derrotas para o jogador 1 dentro as 10000 simulações

Além da probabilidade de cada jogador ganhar uma partida, uma análise interessante de se realizar é referente a quantidade média de jogos necessários para se concluir uma partida.



Média	9,83
Desvio Padrão	4,08
Mínimo	3,0
25%	7,0
50%	9,0
75%	12,0
Máximo	37,0

Figura 8 – Número de rodadas em cada jogo dentro as 10000 simulações e suas distribuições estatísticas

Observando-se a Figura 8 e a sua distribuição estatística, pode-se concluir que em **média**, os jogos tradicionais têm uma duração de **9,83** rodadas e uma **mediana de 9** rodadas.

Um fator que tem grande impacto na quantidade de rodadas em cada jogo é a quantidade de escadas e cobras utilizada por jogador. Observando-se essas informações nas 10000 simulações realizadas, obtemos os seguintes resultados:

- Distribuições de **escadas** utilizadas por cada jogador:

Jogador	1
Média	1.27
Desvio Padrão	0.82
Mínimo	0.0
25%	1.0
50%	1.0
75%	2.0
Máximo	8.0

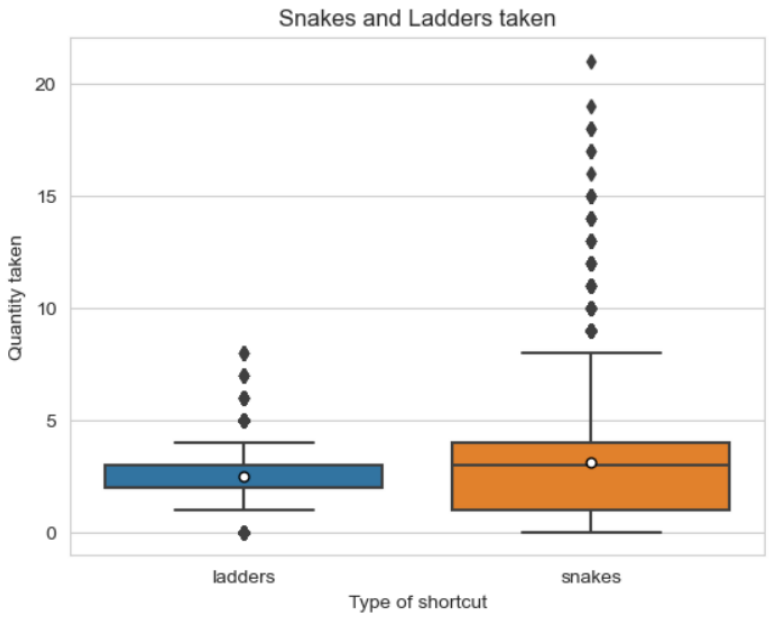
Jogador	2
Média	1.23
Desvio Padrão	0.83
Mínimo	0.0
25%	1.0
50%	1.0
75%	2.0
Máximo	5.0

- Distribuições de **cobras** utilizadas por cada jogador:

Jogador	1
Média	1.59
Desvio Padrão	1.49
Mínimo	0.0
25%	0.0
50%	1.0
75%	2.0
Máximo	12.0

Jogador	2
Média	1.51
Desvio Padrão	1.45
Mínimo	0.0
25%	0.0
50%	1.0
75%	2.0
Máximo	10.0

- Distribuições gerais de cobras e escadas:



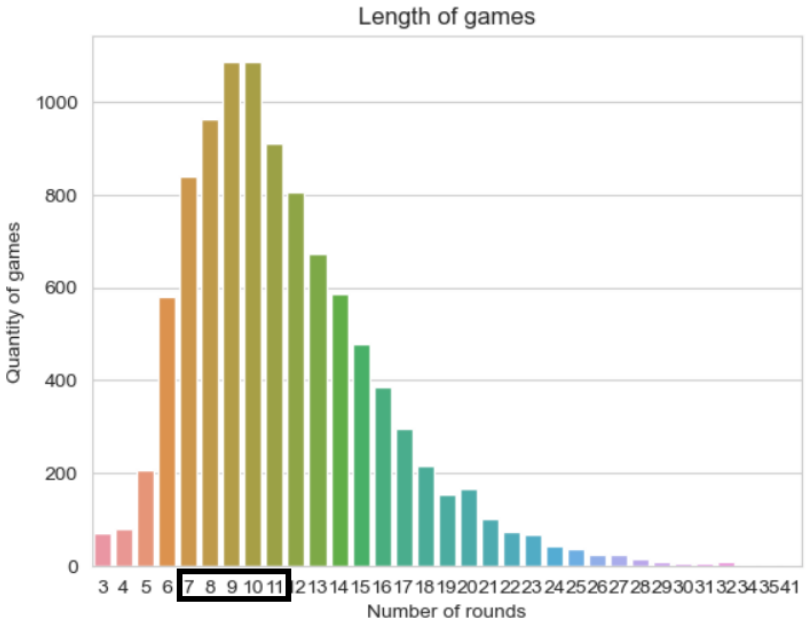
Atalho	Escada
Média	2.50
Desvio Padrão	1.12
Mínimo	0.0
25%	2.0
50%	2.0
75%	3.0
Máximo	8.0

Atalho	Cobra
Média	3.10
Desvio Padrão	2.55
Mínimo	0.0
25%	1.0
50%	3.0
75%	4.0
Máximo	21.0

Figura 9 – Distribuição de cobras e escadas utilizadas dentro as 10000 simulações

Como pode ser observado nas distribuições acima, em média, cada jogador utiliza as escadas em torno de **1,2 vezes** e as cobras em torno de **1,5 vezes** por jogo. Totalizando-se assim, **uma média geral de 3 cobras utilizadas e 2,5 escadas utilizadas por jogo,** resultado apresentado pela Figura 9. Vale observar, que o número médio de vezes de cada atalho utilizado por cada jogador tem sido bastante parecido de maneira geral, o que demonstra que os resultados apresentados pela simulação estão bem equilibrados entre si.

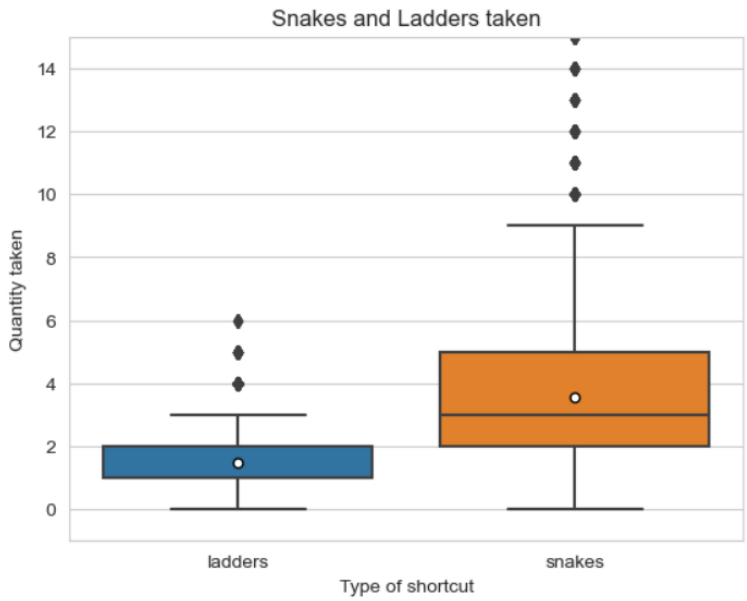
Com o objetivo de medir o impacto da utilização de escadas na duração dos jogos, foi introduzido um pequeno ruido probabilístico para se decidir se escadas serão utilizadas em cada jogo ou não. Com base nas simulações, foram obtidos os seguintes resultados sobre o sistema 2:



Média	11.42
Desvio Padrão	4.45
Mínimo	3.00
25%	8.00
50%	11.0
75%	14.0
Máximo	41.0

Figura 10 – Distribuição de duração média de jogos para simulações do sistema 2

- **Distribuições gerais de cobras e escadas:**



Atalho	Escada
Média	1.46
Desvio Padrão	0.97
Mínimo	0.0
25%	1.0
50%	1.0
75%	2.0
Máximo	6.0

Atalho	Cobra
Média	3.55
Desvio Padrão	2.78
Mínimo	0.0
25%	2.0
50%	3.0
75%	5.0
Máximo	22.0

Figura 11 – Distribuição de cobras e escadas utilizadas para simulações do sistema 2

Comparando-se os resultados das simulações obtidas para o sistema 2 conforme apresentado pelas Figura 10 e 11, constatou-se que o número médio de escadas utilizadas **diminui significativamente em torno de 1 escada por partida**, o que levou a um aumento no número de rodadas necessárias para se concluir uma partida para **11,42**. Um aumento de aproximadamente 16% quando comparado ao número anterior obtido (9,83). Com relação as cobras utilizadas durante a partida, não houve uma mudança estatisticamente relevante.

Uma observação interessante de se analisar, é que o ruído inserido no sistema não alterou as probabilidades de chances de vitória de cada jogador, permanecendo em linha com os valores obtidos para as simulações realizadas para o sistema base.

Probabilidade jogador 1 ganhar: 52.36%
Probabilidade jogador 2 ganhar: 47.64%

Com isso, temos a constatação de que de maneira geral, o jogador que começa a partida, sempre tem uma pequena vantagem na probabilidade de vitória de uma partida. Entre as maneiras de amenizar esse problema, podemos colocar o jogador 2 em algumas posições posteriores a inicial, ou até mesmo conceder uma imunidade a atalhos como cobras sobre um número determinado de rodadas.

Realizando-se experimentos sobre o sistema 3, no qual foi possível definir uma posição inicial mais justa para o jogador 2, foram observados os seguintes resultados.

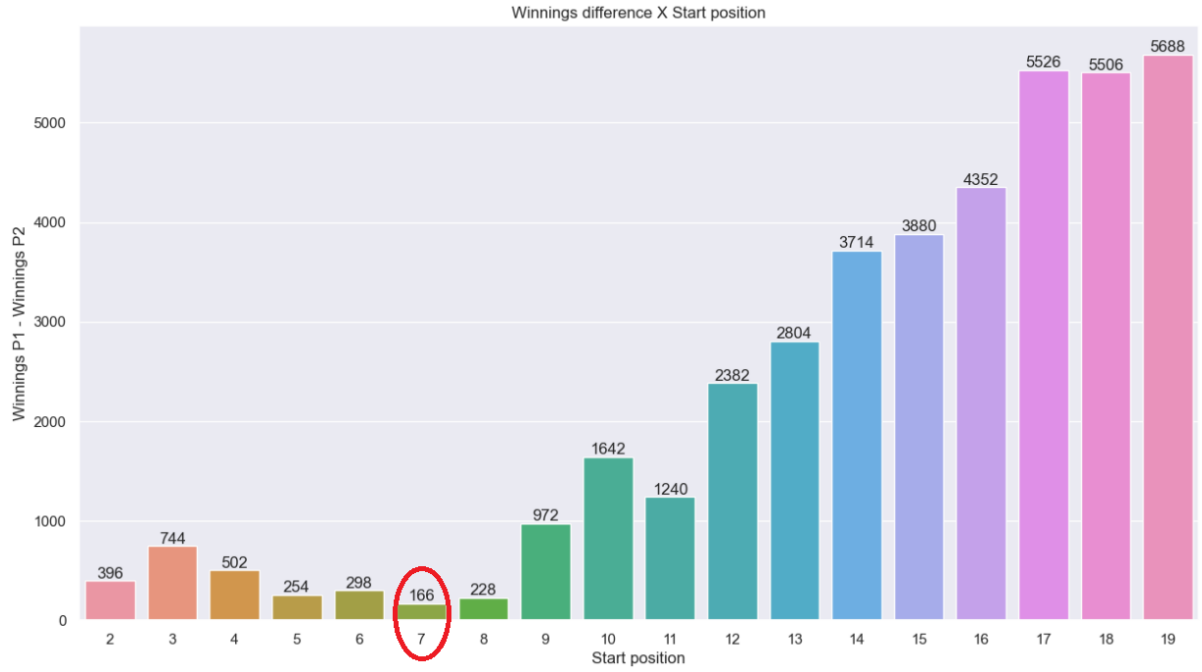


Figura 12 – Gráfico que representa a diferença de vitórias entre jogadores por posição inicial do jogador 2

Para cada posição inicial diferente, foram realizadas 10000 simulações e em seguida, foi calculada a diferença no número de vitórias entre os jogadores. Com base nos valores apresentado pela Figura 12, a posição inicial mais justa para o jogador número 2 seria a **posição 7, com uma pequena diferença de apenas 166 vitórias de partidas entre os jogadores.**

Analisando-se a outra abordagem sugerida para minimizar a vantagem do jogador 1, adicionando-se uma pequena imunidade a cobras para o jogador 2 por uma rodada, obtemos os seguintes resultados para as simulações utilizando-se o sistema 4:

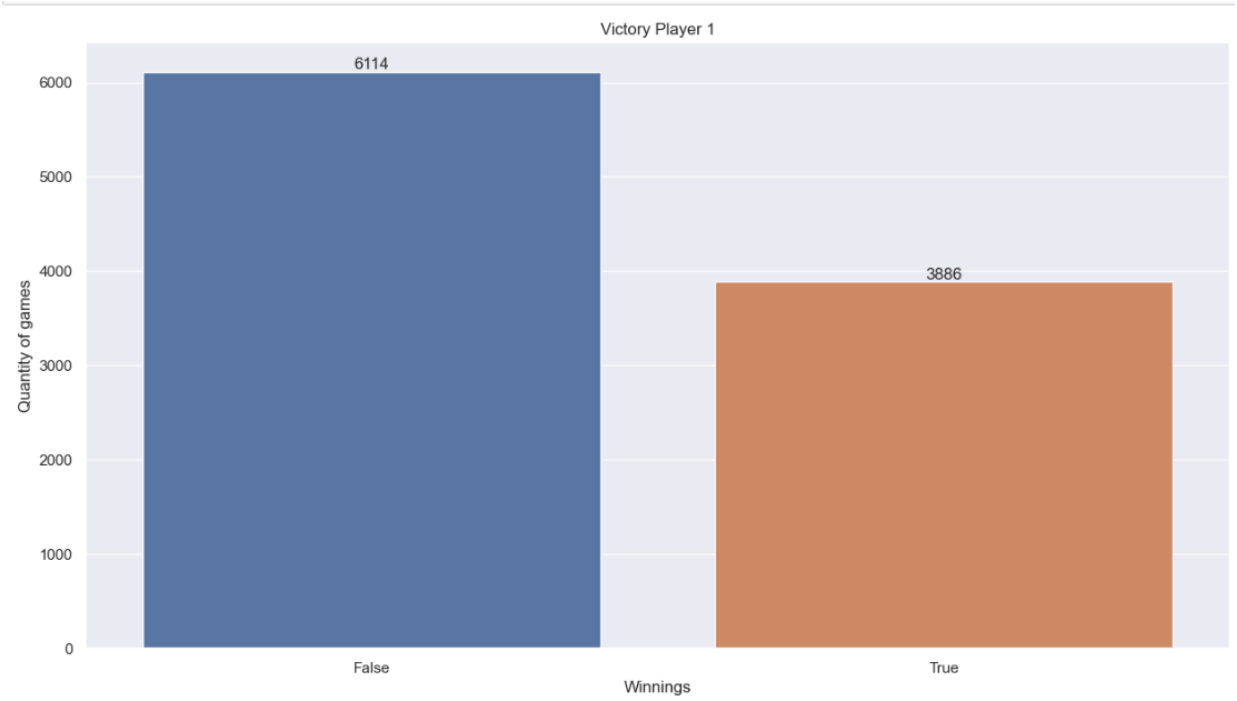


Figura 13 – Número de vitórias e derrotas para o jogador 1 para o sistema 4

Probabilidade jogador 1 ganhar: 3886/10000 = **38,86%**
Probabilidade jogador 2 ganhar: 1 – P(Jogador 1 ganhar) = **61,14%**

- Distribuições de **cobras** utilizadas por cada jogador:

Jogador	1
Média	1.42
Desvio Padrão	1.27
Mínimo	0.0
25%	0.0
50%	1.0
75%	2.0
Máximo	9.0

Jogador	2
Média	0.71
Desvio Padrão	0.46
Mínimo	0.0
25%	0.0
50%	1.0
75%	1.0
Máximo	1.00

Com a imunidade concedida, foi constatado que essa maneira de solucionar o problema da desvantagem entre as probabilidades de vitória do jogador 1 sobre o jogador 2 **não foi satisfatória**. Com base nos valores obtidos da simulação apresentado pela Figura 13 e nas distribuições de cobras por jogador por jogo, temos que o número médio de cobras utilizadas pelo jogador 2 **caiu pela metade (0,71)** quando comparados as **1,5 vezes** nas simulações anteriores, o que representa um grande impacto na probabilidade vitória desse jogador que agora apresenta um valor de **61,14%.**

Para mais detalhes sobre implementações ou abordagens utilizadas, sugiro a visualização do Jupyter Notebook que vai ser entregue juntamente com este relatório e está disponível no link do github inserido na primeira página.

6. Conclusão

De maneira geral, com base nas simulações realizadas no jogo de tabuleiro tradicional, podemos concluir que o jogador que começa a partida geralmente tem uma leve vantagem em termos de probabilidade de vitória, como evidenciado nas simulações para o sistema 1. Além disso, observamos que a introdução de incertezas relacionadas à capacidade de usar uma escada não afeta diretamente essa probabilidade de vitória, mas desempenha um papel significativo na determinação do número médio de rodadas necessárias para concluir uma partida.

Entre as diferentes maneiras de ajustar essa probabilidade de vitória para tornar o jogo mais equilibrado, nossos resultados indicam que a abordagem mais eficaz é conceder ao jogador 2 uma vantagem de algumas casas, conforme demonstrado nas simulações com 7 casas de vantagem. Por outro lado, a tentativa de fornecer imunidade a cobras sob circunstâncias específicas não produziu os resultados desejados.