

Simulação de Jogo de Tabuleiro

Gabriel Silva Delgado
g.delgado11@outlook.com

24 de julho de 2025

Jogo de Tabuleiro

Você e seu amigo estão jogando o jogo de tabuleiro descrito abaixo.

Regras do Jogo

Cada jogador começa na casa 1 e se reveza jogando um dado justo de 6 lados. O jogador se 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. Por outro lado, se você cair na cabeça de uma cobra, então você desce para a casa na ponta do rabo da cobra. O vencedor é a primeira pessoa a chegar ou passar pela última casa.

Objetivo do Teste

Estando interessado em análises, você decide fazer simulações de 10.000 jogos para entender suas chances de ganhar sob diferentes cenários. Considere cada cenário independente dos outros.

Por favor, responda às seguintes perguntas usando quaisquer meios disponíveis. Gostaríamos que você mantivesse um registro do seu trabalho e nos guiasse através da sua solução. Estamos principalmente interessados na sua abordagem e código, em vez das respostas finais.

Perguntas

Pergunta 1: Num jogo de duas pessoas, qual é a probabilidade do jogador que começa o jogo vencer?

Pergunta 2: Em média, em quantas cobras os jogadores caem a cada jogo?

Pergunta 3: Se cada vez que um jogador cair em uma escada houver apenas 50% de chance de ele poder subir, qual é o número médio de lançamentos de dados necessários para completar um jogo?

Pergunta 4: Você decide fazer experimentos para que os jogadores tenham aproximadamente chances iguais de vencerem o jogo. Você faz isso alterando a casa em que o Jogador 2 começa. Qual casa deveria ser a nova posição inicial do Jogador 2 para que as chances de vitória dos jogadores sejam aproximadamente iguais?



Figura 1: Tabuleiro do jogo

Pergunta 5: Em uma tentativa diferente de alterar as chances do jogo, em vez de mudar a posição inicial do Jogador 2, você decide dar-lhe imunidade para a primeira cobra em que ele cair. Neste caso, qual é a probabilidade aproximada de que o Jogador 1 vença?

1 Solução

Esse é um jogo bastante conhecido no contexto da educação matemática: Cobras e Escadas. No tabuleiro deste exemplo, há 36 casas, com 5 escadas e 5 cobras. As escadas permitem que o jogador avance mais rapidamente no percurso, enquanto as cobras dificultam o trajeto, fazendo com que ele retroceda.

As bases das escadas estão localizadas nas casas 3, 5, 15, 18 e 21, e levam, respectivamente, para as casas 16, 7, 25, 20 e 32. Já as cabeças das cobras estão nas casas 12, 14, 17, 31 e 35, levando o jogador de volta às casas 2, 11, 4, 19 e 22, respectivamente.

Precisamos modelar esse tabuleiro, para isso, usaremos a linguagem Julia e a ferramenta Visual Studio. Utilizando o ChatGPT podemos ter uma base para o código:

```

1      # --- Configuração do tabuleiro ---
2      const NUM_CASAS = 36
3
4      const escadas = Dict{
5          3 => 16,
6          5 => 7,
7          15 => 25,
8          18 => 20,
9          21 => 32
10     }
11
12     const cobras = Dict{
13         12 => 2,
14         14 => 11,
15         17 => 4,

```

```

16     31 => 19,
17     35 => 22
18 )
19
20 # --- Função para rolar um dado de 6 lados ---
21 function rolar_dado()
22     return rand(1:6)
23 end
24
25 # --- Função para mover jogador ---
26 function mover(posição_atual)
27     passo = rolar_dado()
28     nova_posição = posição_atual + passo
29
30     if nova_posição > NUM_CASAS
31         nova_posição = posição_atual # não anda se passar
32     end
33
34     # Verifica escadas
35     if haskey(escadas, nova_posição)
36         nova_posição = escadas[nova_posição]
37     # Verifica cobras
38     elseif haskey(cobras, nova_posição)
39         nova_posição = cobras[nova_posição]
40     end
41
42     return nova_posição, passo
43 end
44
45 # --- Função principal do jogo ---
46 function jogar_jogo()
47     jogador1 = 1
48     jogador2 = 1
49     turno = 0 # 0 = jogador1, 1 = jogador2
50
51     while jogador1 < NUM_CASAS && jogador2 < NUM_CASAS
52         if turno == 0
53             jogador1, dado = mover(jogador1)
54             println("Jogador 1 rolou $dado e foi para a casa $jogador1")
55         else
56             jogador2, dado = mover(jogador2)
57             println("Jogador 2 rolou $dado e foi para a casa $jogador2")
58         end
59         turno = 1 - turno # alterna o turno
60     end
61
62     if jogador1 >= NUM_CASAS
63         println("Jogador 1 venceu!")
64         return 1
65     else
66         println("Jogador 2 venceu!")

```

```

67         return 2
68     end
69 end
70
71 # --- Executar o jogo ---
72 jogar_jogo()

```

Embora o código esteja funcional, ele implementa uma regra que não se aplica neste caso específico. Algumas variações do jogo *Cobras e Escadas* exigem que o jogador caia exatamente na última casa para vencer. No entanto, como essa restrição não foi especificada no enunciado, as linhas 30 a 32 devem ser removidas para adequar o comportamento esperado.

Com essa modificação, obtemos um código simples, funcional e que respeita todas as regras descritas. Além disso, por utilizar apenas funções nativas da linguagem, o programa apresenta ótima eficiência de memória.

Utilizando a biblioteca `BenchmarkTools.jl`, foi constatado que cada chamada da rotina `jogar_jogo()` consome aproximadamente $232,600\ \mu s$ e aloca cerca de 1.76 KiB de memória. Ao desabilitar os comandos `println`, o tempo de execução cai para 269,456 ns (aproximadamente 1000 vezes mais rápido), com 0 bytes alocados, ou seja, sem consumo significativo de memória. Essa é uma das vantagens do uso da linguagem Julia, especialmente em aplicações de simulação em larga escala.

Com isso, foi simples escrever um laço que contabiliza a porcentagem de vitórias do Jogador 1:

```

1 function simular_n_jogos(n::Int; imprimir::Bool = true)
2     vitórias_j1 = 0
3
4     for _ in 1:n
5         vencedor = jogar_jogo(false)
6         vitórias_j1 += vencedor == 1 ? 1 : 0
7     end
8
9     porcentagem_j1 = 100 * vitórias_j1 / n
10
11     if imprimir
12         println("Jogador 1 venceu $(round(porcentagem_j1, digits=2))% das
13             ↪ $n partidas.")
14         println("Jogador 2 venceu $(round(100 - porcentagem_j1, digits=2))%
15             ↪ das $n partidas.")
16     end
17
18     return porcentagem_j1
19 end

```

Esse código executa em aproximadamente 2,969 ms, com alocação de cerca de 1.52 KiB. O uso de `println` nesse caso não impacta tanto quanto no exemplo anterior. Quando desabilitado, o tempo de execução reduz para cerca de 2,826 ms, com 0 bytes alocados. Isso representa uma redução inferior a 5% no tempo, embora essa diferença possa se tornar relativamente mais insignificativa conforme o valor de n aumenta.

Agora, já podemos responder as perguntas.

Pergunta 1: Num jogo de duas pessoas, qual é a probabilidade do jogador que começa o jogo vencer?

Com uma única simulação de 10.000 repetições, a porcentagem de vezes que o jogador 1 ganha é de 52.63%, porém, por isso ser um teste estatístico, é importante repetir o evento algumas vezes e pegar a média dos resultados, assim, após efetuar essa mesma simulação 1000 vezes, obtemos uma média de 52.55%. O código utilizado para isso foi:

```
1      using Statistics
2
3  function simular_repetidas(n_jogos::Int, n_repetições::Int)
4      resultados = [simular_n_jogos(n_jogos; imprimir=false) for _ in
5          ↪ 1:n_repetições]
6      média = mean(resultados)
7      println("Média de vitórias do Jogador 1 após $n_repetições
8          ↪ repetições de $n_jogos jogos: $(round(média, digits=2))%")
9      return média
10 end
```

Pergunta 2: Em média, em quantas cobras os jogadores caem a cada jogo?

Para responder a essa pergunta, foi necessário incluir um contador que é incrementado sempre que a condição `haskey(cobras, nova_posição)` for verdadeira. Esse contador foi adicionado ao retorno da função `jogar_jogo()`, e as funções `simular_n_jogos()` e `simular_repetidas()` foram adaptadas para processar essa nova informação.

Com essa modificação, foi possível estimar que os jogadores caem, em média, em aproximadamente **3,09 cobras por partida**, valor que se mostrou bastante consistente ao longo das simulações computacionais.

Pergunta 3: Se cada vez que um jogador cair em uma escada houver apenas 50% de chance de ele poder subir, qual é o número médio de lançamentos de dados necessários para completar um jogo?

Utilizando uma estratégia semelhante à da questão anterior, modificamos a verificação `haskey(escadas, nova_posição)` para que, ao cair na base de uma escada, o jogador tenha apenas 50% de chance de efetivamente subir. Para isso, introduzimos um sorteio binário com `rand() < 0.5`, e controlamos essa regra adicional por meio de uma variável booleana, permitindo ativá-la ou desativá-la conforme o cenário da simulação.

Também reformulamos o controle da variável `turno` na função principal `jogar_jogo()`. Anteriormente, ela alternava entre os valores 0 e 1; agora, ela é incrementada a cada jogada, e o jogador da vez é determinado pela expressão `turno % 2 == 0`. Além disso, o número total de turnos passou a ser registrado e retornado pela função, possibilitando a análise da quantidade média de lançamentos por partida.

É razoável supor que exista um certo equilíbrio estatístico entre a chance de cair na base de uma escada e na cabeça de uma cobra. Ao introduzir essa nova restrição, esperávamos observar um impacto nesse equilíbrio, o que de fato ocorreu. A média de cobras por partida aumentou de aproximadamente 3,09 para 3,61, o que pode ser interpretado como um "incentivo" ao aumento de dificuldade.

Com relação à quantidade de turnos, sem a nova restrição, tínhamos uma média de 19,05 lançamentos por jogo. Com a regra dos 50% nas escadas ativada, esse valor sobe para 22,49, indicando que a progressão no tabuleiro torna-se visivelmente mais lenta.

Pergunta 4: Você decide fazer experimentos para que os jogadores tenham aproximadamente chances iguais de vencerem o jogo. Você faz isso alterando a casa em que o Jogador 2 começa. Qual casa deveria ser a nova posição inicial do Jogador 2 para que as chances de vitória dos jogadores sejam aproximadamente iguais?

Existem diferentes abordagens para resolver esse problema. Em contextos de maior escala, seria apropriado utilizar ferramentas como o *irace* (Iterated Racing for Automatic Algorithm Configuration), disponível em <https://mlopez-ibanez.github.io/irace/>. No entanto, como temos apenas 34 opções viáveis (considerando que a casa 1 favorece o Jogador 1, e a casa 36 garante vitória imediata ao Jogador 2), é viável realizar uma busca exaustiva.

Para isso, foi adicionado um novo parâmetro à função principal, permitindo especificar a posição inicial do Jogador 2. Em seguida, realizamos 34 simulações, uma para cada posição inicial de $p = 2$ até $p = 35$, mantendo o Jogador 1 sempre na casa 1. Cada simulação consiste em 10.000 jogos repetidos 1.000 vezes. O tempo médio para cada repetição é de aproximadamente 3,629 segundos, o que totaliza cerca de dois minutos para todas as simulações, um custo computacional bastante razoável, que dispensa estratégias de otimização mais sofisticadas.

A função abaixo foi utilizada para conduzir esse experimento:

```
1  function encontrar_p_equilibrado()
2      resultados = Float64[]
3
4      for p in 2:35
5          média_vitórias, _, _ = simular_repetidas(10_000, 1000;
6              ↪ jogador2=p, imprimir=false)
7          push!(resultados, média_vitórias)
8          println("p = $p → Jogador 1 venceu $(round(média_vitórias,
9              ↪ digits=2)))% das partidas.")
10     end
11
12     # Calcula o índice (posição p) cuja vitória do J1 mais se
13     ↪ aproxima de 50%
14     diferenças = abs.(resultados .- 50.0)
15     melhor_p = argmin(diferenças)
16     melhor_taxa = resultados[melhor_p]
17
18     println("\nMelhor posição inicial para o Jogador 2:
19     ↪ $melhor_p")
20     println("Ela resulta em aproximadamente $(round(melhor_taxa,
21     ↪ digits=2)))% de vitórias para o Jogador 1.")
22
23     return melhor_p, melhor_taxa, resultados
24 end
```

Como resultado, identificamos que a casa **6** é a que produz maior equilíbrio entre os jogadores, resultando em aproximadamente **49,66%** de vitórias para o Jogador 1 quando a regra nova da escada está ativa e **p=7**, com aproximadamente **49,69%** quando essa nova regra não está ativa.

Pergunta 5: Em uma tentativa diferente de alterar as chances do jogo, em vez de mudar a posição inicial do Jogador 2, você decide dar-lhe imunidade para a primeira cobra em que ele cair. Neste caso, qual é a probabilidade aproximada de que o Jogador 1 vença?

Para implementar essa nova regra, foi necessário inserir uma condição específica na função `mover()`, de forma que o Jogador 2 ignore o efeito da primeira cobra que encontrar no tabuleiro. Essa imunidade é válida apenas uma vez por partida.

Com essa alteração, o jogo voltou a apresentar um desequilíbrio significativo. As simulações indicaram que a taxa de vitória do Jogador 1 caiu para aproximadamente **38,61%** quando a regra da escada com chance de subida (`escadas_metade`) está desativada, e para cerca de **37,19%** quando essa regra está ativada. Esses resultados mostram que a imunidade concedida ao Jogador 2 representa uma vantagem considerável em relação ao equilíbrio original do jogo.

Por fim, com todas perguntas respondidas, o código final:

```
1      using Statistics
2
3      # --- Configuração do tabuleiro ---
4      const NUM_CASAS = 36
5
6      const escadas = Dict(
7          3 => 16,
8          5 => 7,
9          15 => 25,
10         18 => 20,
11         21 => 32
12     )
13
14     const cobras = Dict(
15         12 => 2,
16         14 => 11,
17         17 => 4,
18         31 => 19,
19         35 => 22
20     )
21
22     # --- Função para rolar um dado de 6 lados ---
23     function rolar_dado()
24         return rand(1:6)
25     end
26
27     # --- Função para mover jogador ---
28     function mover(posição_atual, jogador::Int ; cont_cobras::Bool
29         ↪ = false, escadas_metade::Bool = false,
30         ↪ imunidade_primeira_cobra::Bool, imprimir::Bool = false)
31         passo = rolar_dado()
```

```

30     nova_posição = posição_atual + passo
31
32     # Verifica escadas
33     if haskey(escadas, nova_posição)
34         if escadas_metade
35             if rand() < 0.5 # 50% de chance de subir
36                 nova_posição = escadas[nova_posição]
37             end
38         else
39             nova_posição = escadas[nova_posição]
40         end
41     # Verifica cobras
42     elseif haskey(cobras, nova_posição)
43         if jogador == 2 && imunidade_primeira_cobra
44             imunidade_primeira_cobra = false
45             cont_cobras = false
46             imprimir && println("Jogador 2 passou imune pela
↳ primeira cobra")
47         else
48             nova_posição = cobras[nova_posição]
49             cont_cobras = true
50         end
51     end
52
53     return nova_posição, passo , cont_cobras ,
↳ imunidade_primeira_cobra
54 end
55
56 # --- Função principal do jogo ---
57 function jogar_jogo(imprimir::Bool, escadas_metade::Bool;
↳ jogador1::Int = 1, jogador2::Int = 1,
↳ imunidade_primeira_cobra_j2::Bool)
58     turno = 0 # 0 = jogador1, 1 = jogador2
59     contador_cobras = 0
60     cont_cobras = false
61
62     while jogador1 < NUM_CASAS && jogador2 < NUM_CASAS
63         if turno%2 == 0
64             jogador1, dado, cont_cobras =
↳ mover(jogador1,1;escadas_metade=escadas_metade,
↳ imprimir=imprimir,
↳ imunidade_primeira_cobra=false)
65             imprimir && println("Jogador 1 rolou $dado e foi
↳ para a casa $jogador1")
66         else
67             jogador2, dado, cont_cobras,
↳ imunidade_primeira_cobra_j2 =
↳ mover(jogador2,2;escadas_metade=escadas_metade,

```



```

69             imprimir && println("Jogador 2 rolou $dado e foi
        ↪ para a casa $jogador2")
70         end
71         turno +=1 # alterna o turno
72         if cont_cobras
73             contador_cobras += 1
74         end
75     end
76
77     if jogador1 >= NUM_CASAS
78         imprimir && println("Jogador 1 venceu!")
79         return 1, contador_cobras, turno
80     else
81         imprimir && println("Jogador 2 venceu!")
82         return 2 , contador_cobras, turno
83     end
84 end
85
86
87 function simular_n_jogos(n::Int; imprimir::Bool = true,
    ↪ escadas_metade::Bool = true,
88                                     jogador1::Int = 1, jogador2::Int = 1,
    ↪ imunidade_primeira_cobra_j2::Bool
    ↪ = true)
89
90     vitórias_j1 = 0
91     cobras_total = 0
92     turnos_total = 0
93
94     for _ in 1:n
95         vencedor, cobras, turnos =
96             ↪ jogar_jogo(false,escadas_metade;
97             ↪ jogador1=jogador1, jogador2=jogador2,
98             ↪ imunidade_primeira_cobra_j2)
99         vitórias_j1 += vencedor == 1 ? 1 : 0
100         cobras_total += cobras
101         turnos_total += turnos
102     end
103
104     porcentagem_j1 = 100 * vitórias_j1 / n
105     media_cobras = cobras_total / n
106     media_turnos = turnos_total / n
107     imprimir && println("Número médio de turnos:
    ↪ $media_turnos")
108
109     if imprimir
110         println("Jogador 1 venceu $(round(porcentagem_j1,
    ↪ digits=2))% das $n partidas.")
111         println("Jogador 2 venceu $(round(100 -
    ↪ porcentagem_j1, digits=2))% das $n partidas.")
112         println("Número médio de cobras encontradas por jogo:
    ↪ $(round(media_cobras, digits=2))")

```

```

109         end
110
111         return porcentagem_j1, media_cobras, media_turnos
112     end
113
114
115     function simular_repetidas(n_jogos::Int, n_repetições::Int;
116     ↪ imprimir::Bool = false, escadas_metade::Bool = false,
117     ↪ jogador1::Int = 1, jogador2::Int = 1, imunidade::Bool =
118     ↪ true)
119         vitórias = Float64[]
120         cobras = Float64[]
121         turnos = Float64[]
122
123         for _ in 1:n_repetições
124             porc_j1, media_cobras, media_turnos =
125             ↪ simular_n_jogos(n_jogos; imprimir=imprimir,
126             ↪ escadas_metade=escadas_metade, jogador1=jogador1,
127             ↪ jogador2=jogador2,
128             ↪ imunidade_primeira_cobra_j2=imunidade)
129             push!(vitórias, porc_j1)
130             push!(cobras, media_cobras)
131             push!(turnos, media_turnos)
132         end
133
134         média_vitórias = mean(vitórias)
135         média_cobras = mean(cobras)
136         média_turnos = mean(turnos)
137
138         println("Média de vitórias do Jogador 1 após $n_repetições
139         ↪ repetições de $n_jogos jogos: $(round(média_vitórias,
140         ↪ digits=2))%")
141         println("Média de cobras encontradas por jogo:
142         ↪ $(round(média_cobras, digits=2))")
143         println("Média de turnos por partida:
144         ↪ $(round(média_turnos, digits=2))")
145
146         return média_vitórias, média_cobras, média_turnos
147     end
148
149     function encontrar_p_equilibrado(escadas_metade::Bool,
150     ↪ imunidade::Bool)
151         resultados = Float64[]
152         pmin=2
153         pmax=35
154         for p in pmin:pmax
155             média_vitórias, _, _ = simular_repetidas(10_000, 1000;
156             ↪ jogador2=p, imprimir=false,
157             ↪ escadas_metade=escadas_metade,
158             ↪ imunidade_primeira_cobra_j2=imunidade)
159             push!(resultados, média_vitórias)
160         end
161     end

```

```

145         println("p = $p → Jogador 1 venceu
           → $(round(média_vitórias, digits=2))% das
           → partidas.")
146     end
147
148     # Calcula o índice (posição p) cuja vitória do J1 mais se
           → aproxima de 50%
149     diferenças = abs.(resultados .- 50.0)
150     melhor_p = argmin(diferenças)
151     println("Melhor p encontrado: $melhor_p")
152     melhor_taxa = resultados[melhor_p]
153
154     println("\nMelhor posição inicial para o Jogador 2:
           → $(melhor_p+pmin -1)")
155     println("Ela resulta em aproximadamente
           → $(round(melhor_taxa, digits=2))% de vitórias para o
           → Jogador 1.")
156
157     return melhor_p, melhor_taxa, resultados
158 end

```

Uma versão um pouco diferente foi hospedada no github, disponível em: <https://github.com/Delg11/Cobras-e-Escadas/blob/main/main.jl>