Aplicação de uma cadeia de Markov

criação de mapas de um RPG (Role Playing Games)

Stanislav Napravnik Neto - 12722124043
Geovane Souza Brasil do Couto – 1272218033
Igor Malaquias D'Amorim – 12722122087

Salvador 2025

Introdução

As cadeias de Markov são modelos matemáticos utilizados para representar sistemas que transitam entre diferentes estados de forma probabilística, onde a próxima etapa depende apenas do estado atual, e não da sequência de eventos anteriores. Esse tipo de processo é chamado de sem memória, e é amplamente aplicado em áreas como estatística, ciência de dados, inteligência artificial, biologia e, neste caso, em jogos.

Neste projeto, aplicamos o conceito de cadeia de Markov para gerar mapas de forma procedural em um jogo de RPG (Role-Playing Game). O objetivo é construir uma sequência de salas que o jogador deve atravessar, como "Início", "Inimigo", "Tesouro", "Armadilha", "Chefe (Boss)" e "Saída", respeitando regras estabelecidas pela lógica do jogo.

Cada tipo de sala representa um estado do sistema, e a matriz de transição define as probabilidades de ir de um estado para outro. O mapa é gerado aleatoriamente, mas guiado por essas probabilidades, garantindo variedade a cada execução, sem abrir mão de uma estrutura coerente.

Além disso, uma regra especial foi implementada: para alcançar a saída, o jogador deve obrigatoriamente passar por uma sala de chefe. Essa condição foi tratada logicamente dentro do código, controlando a transição para o estado "Saída" apenas após o estado "Boss" ter sido visitado.

Esse projeto exemplifica como conceitos matemáticos, como cadeias de Markov, podem ser aplicados de forma prática e criativa em contextos como o desenvolvimento de jogos e sistemas com comportamentos estocásticos.

Funcionamento do Algoritmo

Tendo em vista o funcionamento da cadeia de markov, os estados representam os tipos de sala que os jogadores encontram nos mapas de RPG gerados, no caso sendo:

- Start: ponto de início
- Enemy: sala com inimigos
- Treasure: sala com tesouro
- Trap: sala com armadilhas
- Boss: sala com um chefe
- Exit: sala de saída

Matriz de Transição

A lógica de transição entre os estados é definida por uma matriz de transição de probabilidades, onde cada linha representa o estado atual e cada coluna representa a chance de transição para outro estado. Exemplo:

```
const P = [
    # Enemy   Treasure   Trap   Boss   Exit
    0.5          0.3        0.2     0.0      0.0   # Start
    0.3          0.3        0.2     0.2      0.0   # Enemy
    0.3          0.2        0.3     0.2      0.0   # Treasure
    0.4          0.1        0.3     0.2      0.0   # Trap
    0.0          0.0        0.0     0.0      1.0   # Boss
]
```

Observe que a matriz não inclui transições para o estado "Start", pois ele só aparece no início e a transição para o estado "Exit" só pode ocorrer a partir do estado "Boss", garantindo que o jogador sempre enfrente o chefe antes de concluir o mapa.

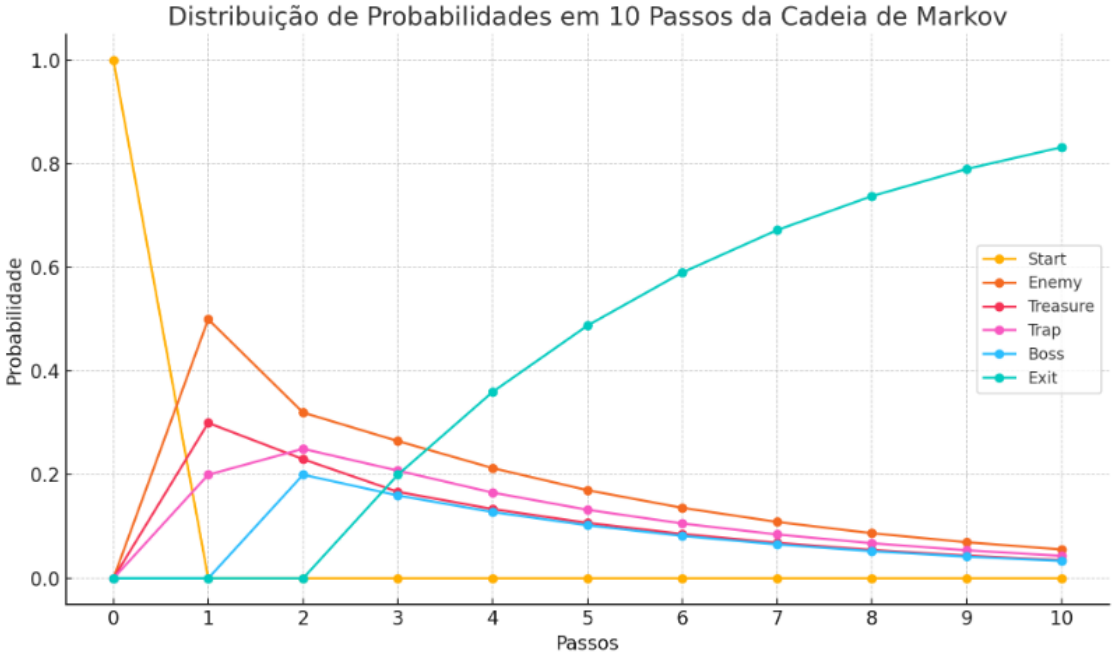
Lógica de Geração

O processo de geração funciona da seguinte forma:

1. Começamos sempre na sala "Start".
2. A cada iteração, usamos a linha da matriz de transição correspondente ao estado atual para escolher o próximo estado com base nas probabilidades.
3. Continuamos o processo até atingir o número máximo de salas ou chegar à sala "Exit", respeitando a regra de passar por um "Boss" antes.

Esse processo garante que cada execução do programa possa gerar um mapa diferente, mas sempre respeitando as regras impostas pela cadeia de Markov.

Evolução das probabilidades ao longo de 10 passos



O gráfico acima mostra a distribuição das probabilidades de estar em cada estado da cadeia ao longo de 10 passos, iniciando obrigatoriamente pelo estado "Start".

Essa analise permite observar que, por ser o ponto de partida, a probabilidade de estar no estado “Start” diminui rapidamente. Além disso os estados intermediários (Enemy, Treasure, Trap, etc), apresentam picos nos primeiros passos, refletindo seu papel como transições naturais antes de sua chegada ao objetivo final.

O estado “Boss” começa com probabilidade baixa, mas aumenta à medida que o caminho se aproxima do final, pois esse estado é necessário para chegar ao estado “Exit”.

O estado "Exit", sendo absorvente (não possui transições para outros estados), acumula probabilidade ao longo do tempo, tornando-se o destino mais provável após múltiplas iterações.

Esse comportamento valida a construção da cadeia de Markov para simular mapas de RPG, garantindo progressão lógica e estrutural no avanço do jogador.

Construção do Gerador de Mapas e explicação do Código-Fonte

Para desenvolver o gerador, foi utilizado a linguagem de programação Julia para a aplicação dos conceitos matemáticos abordados. Nesse caso, foram produzidas as seguintes linhas de código:

1. Definição dos Estados

```
# Estados possíveis
const estados = ["Start", "Enemy", "Treasure", "Trap", "Boss", "Exit"]
```

Os estados representam os tipos de salas possíveis no mapa. O jogo sempre começa em "Start" e termina em "Exit", sendo obrigatória a passagem por um "Boss" antes da saída.

2. Matriz de Transição

```
# Matriz de transição (probabilidades de ir para os próximos estados)
const P = [
    # Enemy  Treasure  Trap   Boss   Exit
    0.5    0.3       0.2    0.0    0.0;   # Start
    0.3    0.3       0.2    0.2    0.0;   # Enemy
    0.3    0.2       0.3    0.2    0.0;   # Treasure
    0.4    0.1       0.3    0.2    0.0;   # Trap
    0.0    0.0       0.0    0.0    1.0    # Boss (só vai para Exit)
]
```

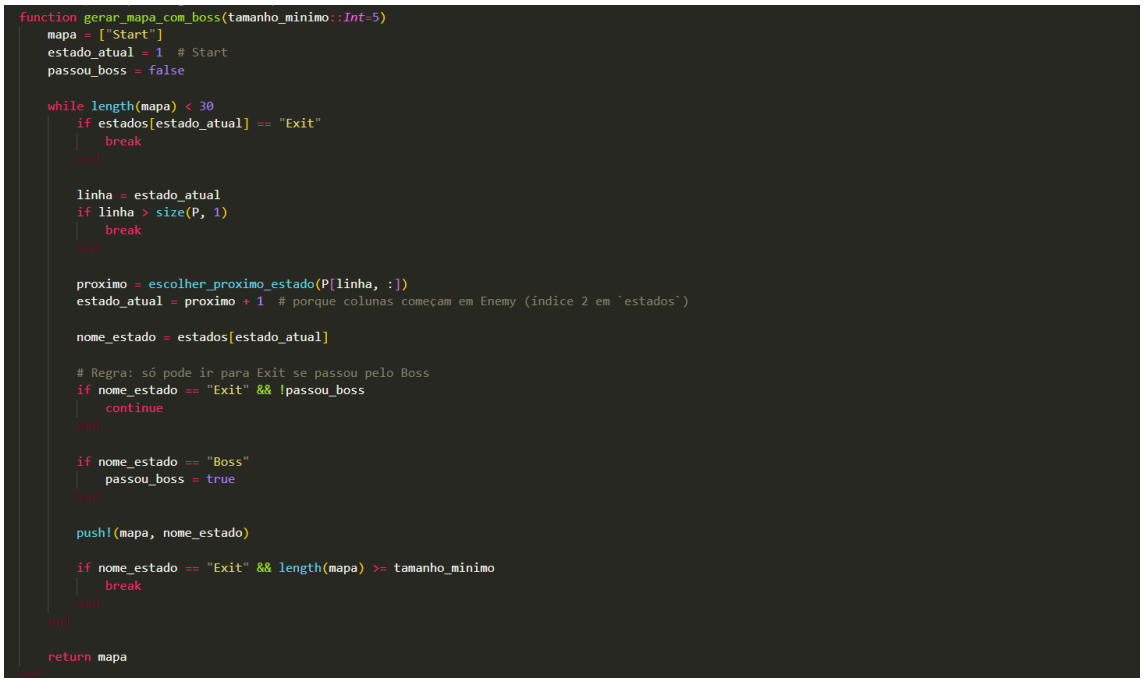
Cada linha da matriz representa o estado atual e cada coluna representa a probabilidade de transição para o próximo estado. Por exemplo, partindo de "Enemy", há 30% de chance de continuar em "Enemy", 30% de ir para "Treasure", 20% para "Trap" e 20% para "Boss".

3. Escolha do Próximo Estado

```
# Função para escolher o próximo estado com base nas probabilidades
function escolher_proximo_estado(probs)
    r = rand()
    acumulado = 0.0
    for i in 1:length(probs)
        acumulado += probs[i]
        if r < acumulado
            return i
        end
    end
    return length(probs)  # fallback (último)
end
```

Essa função simula o sorteio do próximo estado com base nas probabilidades da matriz de transição. É essencial para garantir a aleatoriedade do mapa, respeitando os limites impostos.

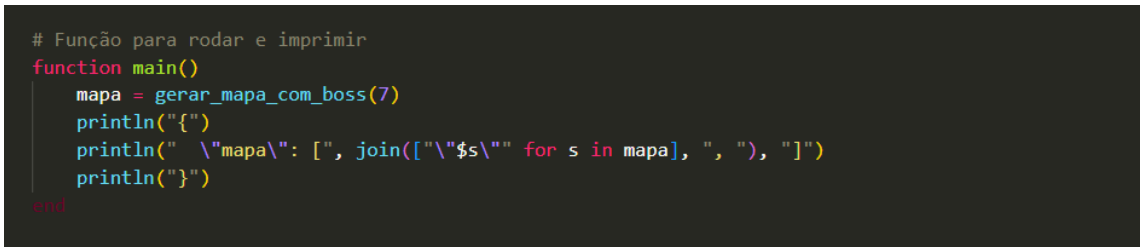
4. Geração do Mapa



```
function gerar_mapa_com_boss(tamanho_minimo::Int=5)
    mapa = ["Start"]
    estado_atual = 1  # Start
    passou_boss = false

    while length(mapa) < 30
        if estados[estado_atual] == "Exit"
            break
        end

        linha = estado_atual
        if linha > size(P, 1)
            break
        end

        proximo = escolher_proximo_estado(P[linha, :])
        estado_atual = proximo + 1  # porque colunas começam em Enemy (índice 2 em `estados`)

        nome_estado = estados[estado_atual]

        # Regra: só pode ir para Exit se passou pelo Boss
        if nome_estado == "Exit" && !passou_boss
            continue
        end

        if nome_estado == "Boss"
            passou_boss = true
        end

        push!(mapa, nome_estado)

        if nome_estado == "Exit" && length(mapa) >= tamanho_minimo
            break
        end
    end

    return mapa
end
```

A função gerar_mapa simula o trajeto do jogador, escolhendo aleatoriamente os próximos estados até atingir a condição de parada: alcançar a sala "Exit" **após** passar por um "Boss". O parâmetro max salas define o número máximo de salas possíveis.

5. Impressão do Resultado



```
# Função para rodar e imprimir
function main()
    mapa = gerar_mapa_com_boss(7)
    println("{")
    println("  \"mapa\": [", join(["\"$s\"" for s in mapa], ", "), "]")
    println("}")
end
```

Essa parte final imprime o resultado gerado no formato JSON.