Inteligência Artificial para Jogos

Aula 06 - Máquina de Estados Finitos Alternativa

Apresentação da Aula

Na aula passada, você utilizou uma ferramenta muito útil para começar a construir a IA de um personagem de um jogo, a **Máquina de Estados Finitos**. Como você já viu, o conceito do mecanismo é simples de entender e espero que tenha sido fácil também de implementar. Caso ainda tenha alguma dúvida, sugiro que você revise o conteúdo da aula passada, pois nessa você irá se aprofundar um pouco mais sobre as **MEFs**.

Apesar de esse mecanismo ser um dos mais usados na indústria dos jogos, ele não é livre de problemas. Há inúmeras limitações ao seu uso e, em certos casos, ele não é o mais adequado.

Dessa forma, prepare-se para conhecer alternativas e variações de **Máquinas de Estados Finitos** para incrementar um pouco mais o jogo de futebol que você está desenvolvendo.

Então siga-me...

Objetivos

Conhecer as limitações, alternativas e variações da **Máquina de Estados Finitos**;

Compreender o que é e como representar uma **Máquina de Estados Finitos Não Determinística, Fuzzy** e **Hierárquica**;

Entender como e onde aplicar os diferentes tipos de **Máquinas de Estados Finitos**.

1. Máquina de Estados Finitos Não Determinísticas

Imagine, por exemplo, o jogo de futebol que você está desenvolvendo e que dois times controlados pelo computador vão disputar uma partida. Hum... imaginou? Ou seja, é uma simulação, e não um jogo. Se você usar a **Máquina de Estados Finitos** tal qual foi descrita na aula anterior, a posição inicial dos jogadores no campo irá determinar o resultado. Isso ocorrerá porque para um determinado evento apenas uma transição de estado é possível, fazendo com que essa "simulação" seja **determinística**. Assim, ao ser dado um conjunto de parâmetros de entrada para a simulação (posição inicial dos jogadores), uma única saída (resultado) será produzida.

Muitas vezes, não é possível perceber essa limitação, mesmo usando a **MEF Determinística**, porque a produção de aleatoriedade, que quebra o aspecto determinístico, fica a cargo do jogador. Ele movimenta seu personagem de um lado para o outro, atira, pula, faz combinações de golpes e *hadoukens (Caso não conheça, dá uma olhadinha no jogo Street Fighter!)* inesperados que criam a ilusão de que os personagens controlados pela IA não seguem padrões.

No filme Pixels (2015), os seres intergalácticos usam os jogos de arcade clássicos contra a Terra, até que Sam Brenner (Adam Sandler), campeão de competições de videogames dos anos 80, ajuda a salvá-la. A técnica dele era "identificar os padrões".

Figura 01 - Cena do filme Pixels

Fonte: MONTEIRO, Beatriz. **Pixels**. Disponível em: <http://www.magazine-hd.com/apps/wp/netflix-junho-filmes-recomendamos/4/>. Acesso em: 08 maio 2018

Os seres intergalácticos, tão inteligentes, criaram seus jogos com **Máquina de Estados Finitos Determinística!** Ainda bem que eles não viram essa aula que você está vendo! 😁

Uma alternativa às **MEFs** clássicas é a introdução de incertezas não através de aleatoriedade sobre as transições, mas através da dúvida sobre os eventos.

Você deve estar em "dúvida sobre essa dúvida". 🤔 Como um personagem controlado pela IA pode ter dúvida? Quando um evento ocorre em um cenário, por exemplo, e o personagem se encontra a menos de 50 m do inimigo, não há dúvida sobre isso. O computador pode, muito bem, calcular a distância do personagem ao seu inimigo e, se essa distância for menor que 50 m, é porque o evento ocorreu, caso contrário, não. É ou não é? Não há discussão sobre isso. É preto no branco!

Suponha que você tenha vários eventos concorrendo e um personagem que precisa defender uma base. Ele pode ir atrás do um inimigo que começou a atacá-la, mas se afastou para não morrer. O personagem, então, persegue o inimigo para não deixar que ele recupere sua saúde (*health*). Porém, outros inimigos estão também atacando a base. O que fazer, então? Voltar para protegê-la ou continuar e acabar com os últimos suspiros do inimigo inicial?

Essa dúvida é cruel... mesmo para nós quando estamos jogando. Imagine, então, para personagens controlados pelo computador. Bom, na verdade, não é tão cruel assim para eles. Eles vão apenas seguir um algoritmo. É cruel para o desenvolvedor

dos algoritmos: você! 😁

Mas, o que seria então uma **MEF Não Determinística**?

Antes que eu responda essa pergunta, pense um pouco e recapitule o que foi visto na aula 04. Nela, você viu o conceito de probabilidade sendo usado para "modelar incertezas". Foi demonstrado que um chute ao gol pode gerar vários resultados. Pode ser um *chute a la Roberto Baggio* *(manda a bola para os torcedores atrás do gol)* ou um chute *a la* Zico (tira a teia de aranha do canto da trave). Esse mesmo conceito pode ser usado para criar **Máquina de Estados Finitos Não Determinística.** Assim como uma ação pode gerar diferentes resultados, um evento pode também gerar diferentes transições de estados. Simples, não é?

Os mesmos elementos e mecanismos apresentados na aula 4 (**Probabilidade**) podem ser usados sobre os elementos e mecanismos apresentados na aula 05 (**Máquina de Estados Finitos**). A diferença das **MEFs** anteriores é que pode-se ter, para o mesmo evento, mais de uma transição de um estado para o outro.

Por exemplo, a Figura 02 representa uma **MEF** similar à da aula 04, porém introduz uma transição de um estado para ele mesmo. Isso significa dizer que, caso o personagem tenha o inimigo à vista, ele pode continuar no mesmo estado, ou seja, patrulhando.

Figura 02 - Diagrama de uma Máquina de Estados Finitos de um personagem com duas transições para o mesmo evento (Inimigo à vista)

Como você deve ter percebido, há uma duplicidade de transição quando o inimigo estiver à vista durante o estado de patrulha. Como, então, resolver qual transição executar? Uma solução é definir uma probabilidade para cada transição de cada uma das possíveis opções. Lembra da fórmula que define a probabilidade de um evento específico ocorrer?

$$p = \frac{evento\ específico}{todas\ opções\ de\ eventos\ possíveis}$$

Com essa fórmula, é possível estabelecer uma regra que determine a frequência possível de cada transição. Suponha que um estado possua 10 possíveis transições a partir do mesmo evento. Nesse caso, se todas as transições tiverem a mesma chance de serem a escolhida para serem executadas, a probabilidade de cada uma seria dada por:

$$p = \frac{evento\ específico}{todas\ opções\ de\ eventos\ possíveis} = \frac{1}{10} = \frac{1}{100} = 10$$

Ou seja, cada uma teria uma chance de 10% de ser executada.

Mas esse é o caso de todas as transições terem chances iguais. E se você fosse tendencioso? Poderia dar uma chance maior àquelas transições nas quais queira que aconteça com mais frequência. Por exemplo, no comportamento modelado na Figura 02, podem ser introduzidas probabilidades associadas às transições, como representado na Figura 03.

Figura 03 - Diagrama de uma Máquina de Estados Finitos com probabilidades associadas às transições duplicadas

Observe que quando o personagem perceber o inimigo ele continuará patrulhando em 20% das vezes, mas passará a perseguir o inimigo em 80% das vezes.

Talvez pareça estranho para você modelar um comportamento de, por exemplo, um personagem soldado que às vezes não persegue o inimigo, mesmo o tendo visto. De fato, não faz muito sentido se você pensar em um jogo com um único soldado para perseguir o jogador. E se for um cenário com vários desses soldados e que eles sejam controlados por uma **MEF Determinística** como a da Figura 02? Quando o jogador (inimigo, para os soldados) entrar nesse cenário, todos os personagens vão sair correndo atrás dele, de uma vez só. Nesse caso, vai parecer estranho também, pois mostra que eles não possuem coordenação alguma entre eles. A princípio, espera-se que uma tropa de soldados possua um mínimo de coordenação. O uso de probabilidade nas transições, como a **MEF** da Figura 03, fará com que você possa reduzir o ritmo em que os soldados irão em direção ao jogador, simulando um pouco de coordenação sem precisar elaborar algoritmos complexos para isso.

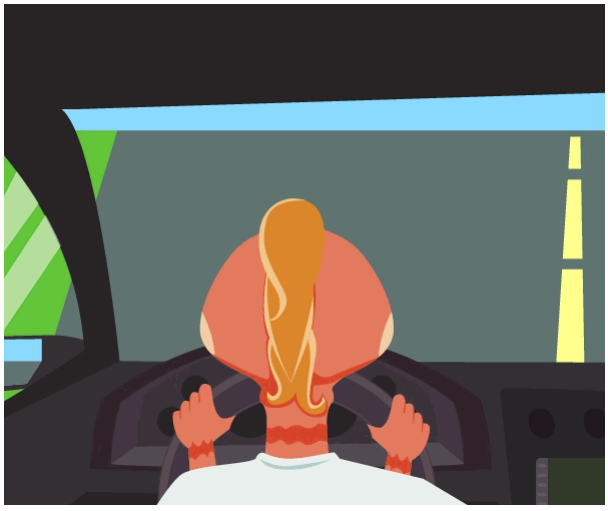
2. Máquina de Estados Finitos Fuzzy

Na seção anterior, foi apresentada uma forma de fazer com que as **MEFs** não sejam tão determinísticas, ou melhor, que os comportamentos gerados por elas não criem padrões que os jogadores possam identificar. Isso torna os jogos desenvolvidos mais imprevisíveis, às vezes mais realistas, mas principalmente mais

divertidos. Só que há outras formas de criar a sensação de imprevisibilidade e realismo usando **MEFs**. Uma delas é **introduzindo a noção de incerteza sobre os eventos**.

Se você parar para pensar, faz todo sentido! Veja bem, coloque-se no lugar de um personagem controlado por regras. Quando você vai atravessar a rua, você não segue a regra “se não houver carro até 200 m de distância, então posso atravessar a rua”. No máximo, você segue a regra “se não houver carro próximo, então posso passar atravessar a rua”. Mas, o que é “próximo”? Talvez, para você, “próximo” signifique menos de 200 m, mas para alguns significa menos de 100 m (estes atravessam a rua correndo, e não é interessante fazer isso 😉), enquanto para outros menos de 500 m (estes são idosos e precisam atravessar a rua lentamente). O termo “próximo” pode, então, assumir diferentes valores em função de inúmeros dados, como idade do personagem, velocidade dos carros que estão vindo, entre outros.

Além da multitude de variáveis, não dá para fazer um cálculo preciso porque há chances de o carro aumentar a velocidade, desviar de um buraco e mudar um pouco de direção etc., ou seja, o resultado do cálculo pode mudar e não dar mais tempo de você sair do meio da rua! Então, é preciso ter uma margem de segurança, mas que margem de segurança é essa? Enfim, são muitas informações imprecisas.



O termo “próximo” não deveria ser modelado através de um valor ou intervalo fixo. Há um certo grau de importância em diferentes valores. Por exemplo, atravessar a rua tendo um carro vindo a 50 m de distância tem a mesma importância de atravessar com um carro a 500 m? Se você considerar um carro

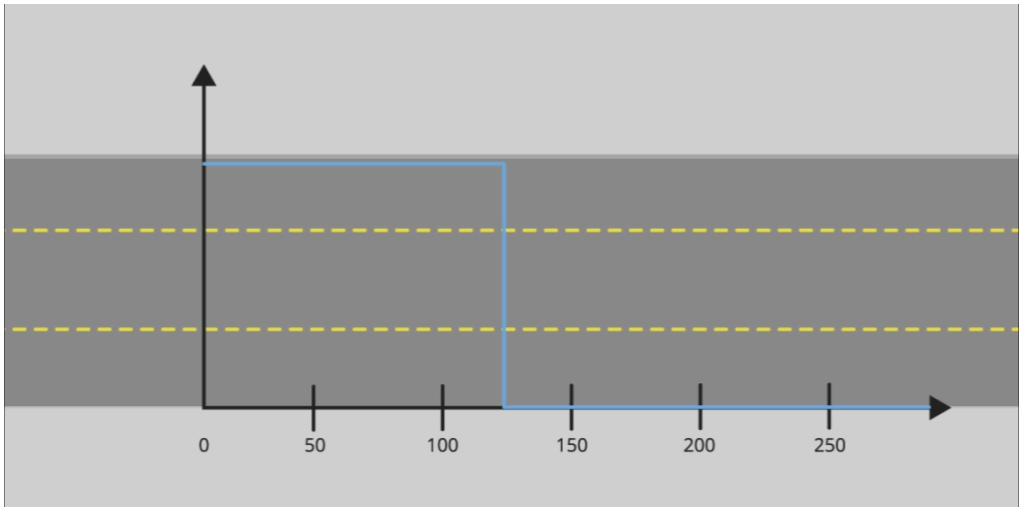
vindo em alta velocidade, não seria prudente atravessar a rua com esse carro vindo a uma distância de 50 m, mas com ele a 500 m... bom, nesse caso, talvez seja possível. Atravessar a rua no primeiro caso é bem mais grave que no segundo.

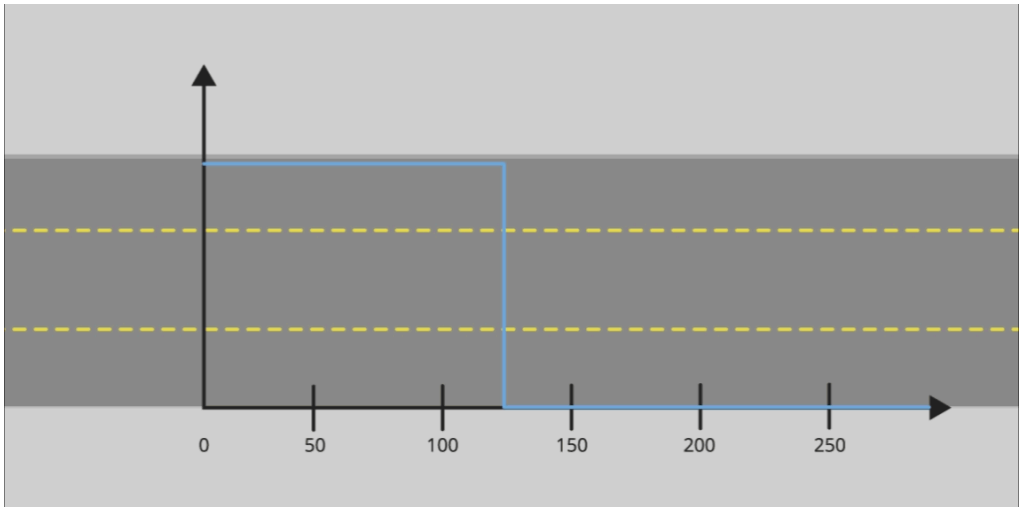
O que eu quero reforçar é que, muitas vezes, você raciocina, não sobre valores exatos, mas sobre graus de importância que você dá aos valores. Para inserir esse mesmo conceito no comportamento dos personagens virtuais, você precisa utilizar algumas ferramentas matemáticas: **conceito de conjunto** e o de **pertinência de conjunto**.

Pode-se definir o conceito de "próximo", denominado aqui de P, por exemplo, através do conjunto de números positivos reais entre 0 e 125 (inclusos) representando as distâncias em metros. **Testar se uma distância está "próxima" significa verificar se seu valor pertence ou não ao conjunto P**. Então, como você deve saber, valores como 50, 100 e 125 pertencem ao conjunto P, enquanto valores como 150 não pertencem.

Na Figura 04 observe a situação de um pedestre tentando atravessar a rua detalhada por meio do gráfico de uma função para verificar se um valor pertence (se o carro está numa distância próxima ao pedestre) ou não ao conjunto P (se o carro está numa distância mais afastada do pedestre).

Figura 04 - Teste pra ver se precisa de ambulância ou não 😁



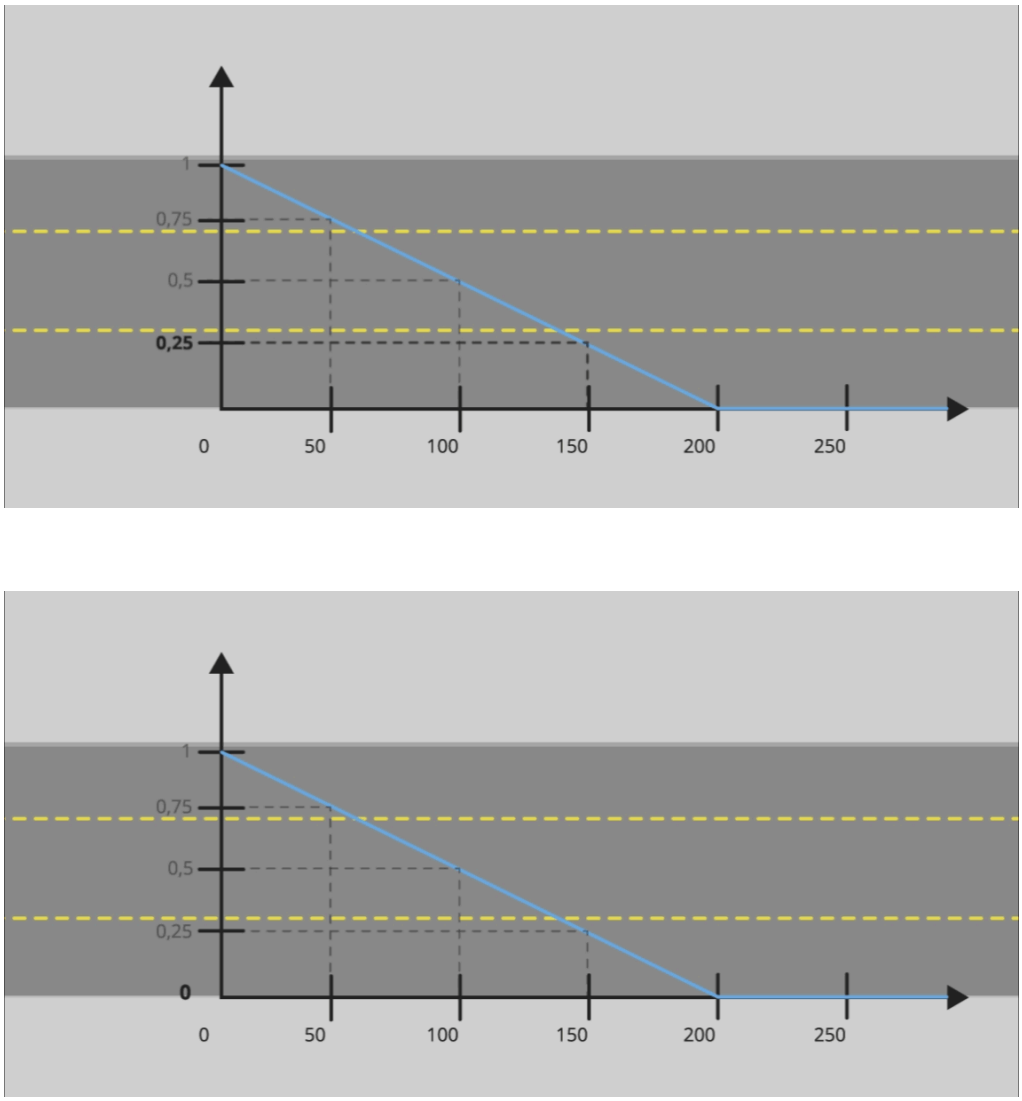


Na coordenada Y, há apenas dois possíveis valores: “Sim”, indicando quando um valor da coordenada X pertence ao conjunto P, ou “Não”, quando um valor da coordenada X não pertence ao conjunto P. Veja na figura que os valores até 125 não são distâncias confiáveis para o pedestre atravessar a rua, já os valores acima são.

Espero que as informações presentes na Figura 04 tenham ajudado o seu entendimento, pois irei complicar agora um pouco mais... Para introduzir o grau de importância dos valores de um conjunto, você irá, em vez de ter apenas duas possibilidades no eixo Y, ter diferentes valores. Esses valores indicam o “grau de pertinência” de um valor no conjunto P, ou seja, um valor pode pertencer a P, mas com um grau de pertinência menor do que outro valor.

A Figura 05 mostra uma possível forma de representar os graus de pertinência de diferentes valores para o conjunto P. No caso, o valor 0 tem valor de pertinência 1, o que significa que o carro está realmente próximo (na verdade, já está amassando a lataria)! O valor 50 pertence ao conjunto “próximo” com grau 0,75, enquanto o valor 100 pertence com grau 0,5, o valor 150 pertence com grau 0,25, e os valores 200 e 250 com grau 0, ou seja, não pertencem.

Figura 05 - Função de pertinência de valores no conjunto de números positivos reais entre 0 e 200



Com esse tipo de representação, você consegue modelar incertezas através dos **graus de pertinência de valores nos conjuntos** ou **"graus de verdade"**. Por exemplo, se alguém me perguntasse agora se um carro a 100m de distância está próximo de mim, eu diria que "isso é meia verdade", pois o **grau de verdade** dessa afirmação é de 0,5. 😁

Essa forma de modelar as afirmações vai ser bastante útil nas **MEFs Fuzzy**, pois dois eventos podem ser verdadeiros simultaneamente. Lembra do exemplo apresentado na primeira seção desta aula? Em que foi citado o caso do personagem que precisa defender uma base. Então, ele pode ir atrás do inimigo que começou a atacar a base, pois se encontra próximo a ele, e pode matá-lo; ou voltar para

proteger a base, pois ela está sendo atacada e pode ser destruída. As duas condições que geram as transições para os seus respectivos estados podem ser verdadeiras... porém, com graus de pertinência ou de verdade diferentes.

A Figura 06 ilustra o caso descrito. O estado de Patrulha é o estado no qual o personagem volta a sua base para protegê-la. Nesse exemplo, o personagem pode estar próximo o suficiente de um inimigo a ponto de iniciar uma perseguição, mas ao mesmo tempo sua base pode estar em perigo porque está sendo atacada por outros inimigos. O que o personagem deve, então, fazer? Deve testar qual o **grau de verdade** dessas condições, verificando o grau de pertinência da distância que ele se encontra ao inimigo no conjunto definido por “Próximo”, e também verificar o grau de pertinência do nível de destruição da base em um conjunto, chamado “Perigo”. Com posse desses dois valores de pertinência, ele poderá tomar uma decisão.

Figura 06 - Diagrama de uma Máquina de Estados Finitos com transições **Fuzzy**

Apesar de as **MEFs fuzzy** representarem uma lógica mais “natural” nos personagens virtuais, elas possuem uma certa desvantagem em relação ao desempenho computacional. Veja que em uma **MEF** clássica, os testes das condições dos eventos de transição podem ocorrer pontualmente. Por exemplo, no momento que o personagem passar da distância de 199m para 200m, ele muda de estado e não precisa mais ficar fazendo verificações daquela condição (bom, ele terá outras condições para testar, mas não mais aquela). Porém, em uma **MEF Fuzzy**, em geral, as condições precisam ser testadas continuamente porque as informações são parciais, ou melhor, as verdades parciais, como no caso da proximidade de 100 m que era uma “meia verdade”. Isso faz com que as **MEFs** requeiram, em geral, mais processamento do computador do que as **MEFs** clássicas. Se os personagens possuem comportamentos simples, então não há problema. Porém, quando o

número de estados e transições começa a aumentar, assim como o número de personagens controlados por elas, então você pode começar a repensar as estratégias.

Uma das **MEFs** utilizadas em casos de IAs com muitos estados e transições será apresentada na seção seguinte.

3. Máquina de Estados Finitos Hierárquica

O exemplo da Figura 02 ilustra uma **MEF** com apenas 3 estados. Com esse número de estados, dá para ver e entender as transições e os eventos. Ou seja, compreender todo o funcionamento do comportamento do personagem apenas olhando rapidamente o diagrama.

Infelizmente, nem todos os casos de IA são assim. Basta imaginar no cenário do jogo de futebol. Em quantos estados diferentes um jogador pode se encontrar? Nos exemplos da aula passada, você criou algo pontual: fez com que eles fossem capazes de cobrar pênaltis! Só isso já gerou um certo número de estados. E quantos estados seriam necessários para criar o comportamento de um jogo completo?

Para exemplificar melhor, suponha que alguém peça para você descrever o seu comportamento durante um dia qualquer, um dia comum. Se essa descrição for necessária para programar um robô que se comporte de forma similar a você, seria necessário, então, descrever todos os detalhes, desde a ação de se levantar da cama (dar aquela “espreguiçada”! 😁), percorrer o caminho até o banheiro e assim por diante. Em cada um desses estados, inúmeros eventos podem ocorrer, desde sua mãe falar para você se apressar ou até mesmo você receber um aviso de que não haverá mais aula e que, por isso, não precisa mais se arrumar. Caramba! Coisa de louco, não é mesmo? Simplesmente, não dá para você gerenciar tantos estados e tantas transições assim. Você ficará perdido quando for necessário dar uma manutenção no comportamento do robô se uma **MEF** clássica for utilizada.

Resumindo, a **MEF** tal qual descrita na aula passada é excelente e muito usada. Entretanto, basta adicionar um grande número de possíveis estados que o gerenciamento começa a ser intratável. É desejável, portanto, ter um mecanismo

mais adequado quando o número de possíveis estados começa a extrapolar a sua capacidade de gerenciá-los.

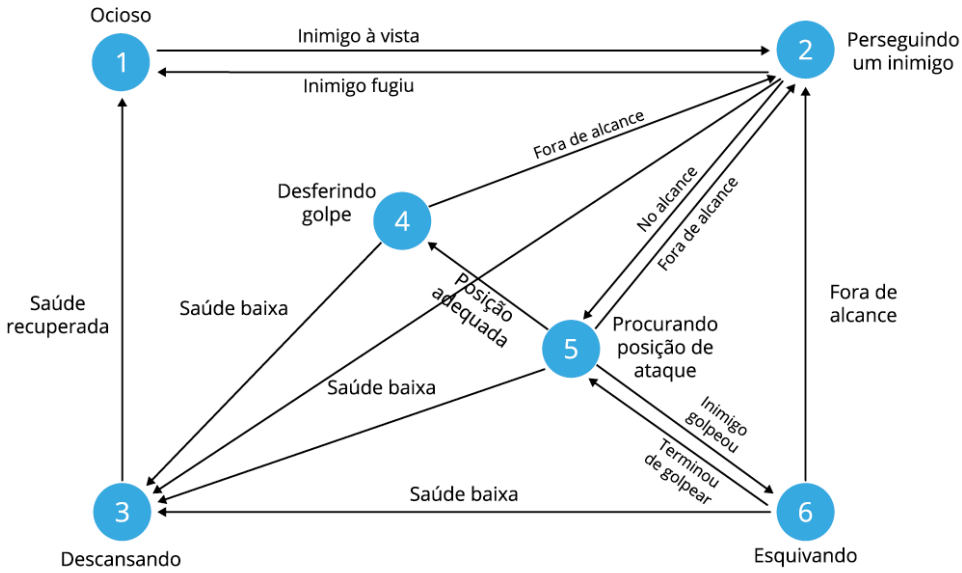
Pare para pensar um pouco...

E se nas situações complexas, como seu comportamento diário, a descrição inicial para o robô for mais abstrata? Por exemplo, você poderia descrever os estados do seu comportamento em “arrumar-se para ir à escola”, “ir para a escola”, “assistir as aulas na escola”, “voltar para casa” etc. Esses são estados mais abstratos do que os descritos anteriormente. Enquanto o robô que te imita estiver em um determinado estado, pode haver uma descrição mais detalhada dele através de outra **MEF**. Por exemplo, o estado “arrumar-se para ir para a escola” pode ser detalhado com o estado de se levantar da cama, ir ao banheiro etc. Ou seja, a **MEF** do robô é definida por estados, que por sua vez podem ser definidos como outras **MEFs**.

Essa é uma forma bastante utilizada nos jogos para organizar a complexidade de alguns comportamentos. Ela permite pensar inicialmente em estados mais gerais e depois ir especializando cada um desses estados gerais através de outras **MEFs** até chegar a ações concretas dos personagens.

Dando outro exemplo, mas desta vez sem invadir sua privacidade 😁. Suponha que você esteja construindo o modelo de comportamento de um personagem guerreiro e, à medida que vai se lembrando de casos e situações, vai inserindo estados e transições, chegando a um ponto em que sua **MEF** fique como a apresentada na Figura 07.

Figura 07 - Diagrama de uma Máquina de Estados Finitos de um personagem guerreiro

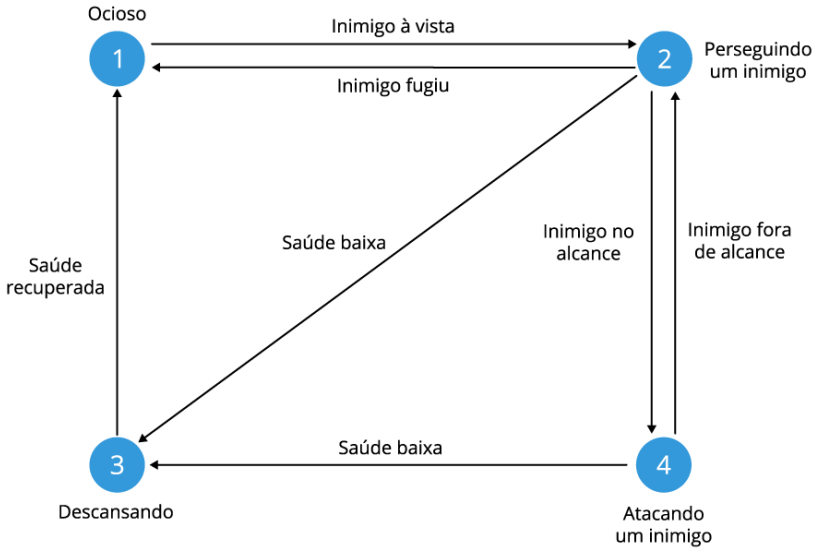


Nessa **MEF**, o personagem possui seis possíveis estados: ou ele encontra-se sem fazer nada (ocioso); ou está descansando (procurou um refúgio para recuperar sua saúde); ou está perseguindo um inimigo (alguém está à vista); ou está procurando uma posição de ataque (quando o inimigo estiver ao alcance do ataque); ou está desferindo golpe (quando a posição estiver adequada); ou está se esquivando (quando o inimigo estiver atacando).

As transições de estados são muitas e fica difícil de entender toda a dinâmica, então preste atenção no diagrama para entender quais os eventos que geram as transições. Veja que em algumas situações, o mesmo evento gera uma transição de diferentes estados para um único estado. Quando a saúde está baixa, por exemplo, o personagem procurará abrigo para descansar de qualquer estado em que ele se encontra (exceto quando ele está ocioso). Algo similar ocorre quando o inimigo se encontra fora do alcance, e o personagem precisa persegui-lo. Quando um mesmo evento precisa ser tratado em diferentes estados, está claro que a Máquina começa a ficar complexa e uma saída é criar uma hierarquia.

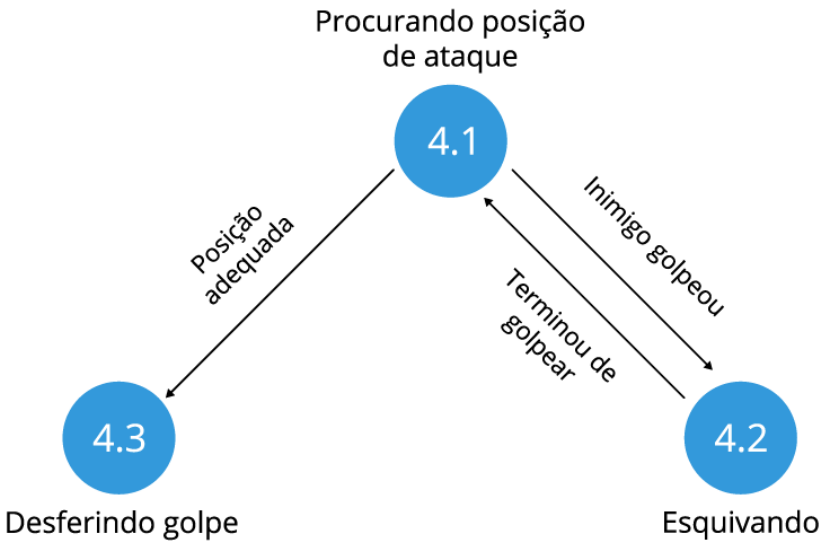
A saída é generalizar, definindo estados mais abstratos, e criar **MEFs** para cada um desses estados. Assim, a **MEF** da Figura 07 poderia ser simplificada para a **MEF** da Figura 08.

Figura 08 - Diagrama simplificado de uma Máquina de Estados Finitos de um personagem guerreiro



Com ela, você consegue entender melhor o comportamento. Porém, o comportamento de atacar o inimigo precisa ser detalhado melhor através de outra **MEF**, como especificado no diagrama da Figura 09.

Figura 09 - Diagrama de uma Máquina de Estados Finitos do estado 'Atacando um inimigo'



É interessante notar que há transições do estado abstrato “Atacando um inimigo” do diagrama da Figura 08, mesmo que haja transições internas “dentro” desse estado. O que acontece então no caso de o personagem desferir um golpe (estado 4.1 - Figura 09) e o inimigo passar a estar fora de alcance (transição do estado 4 para o 2 - Figura 08)?

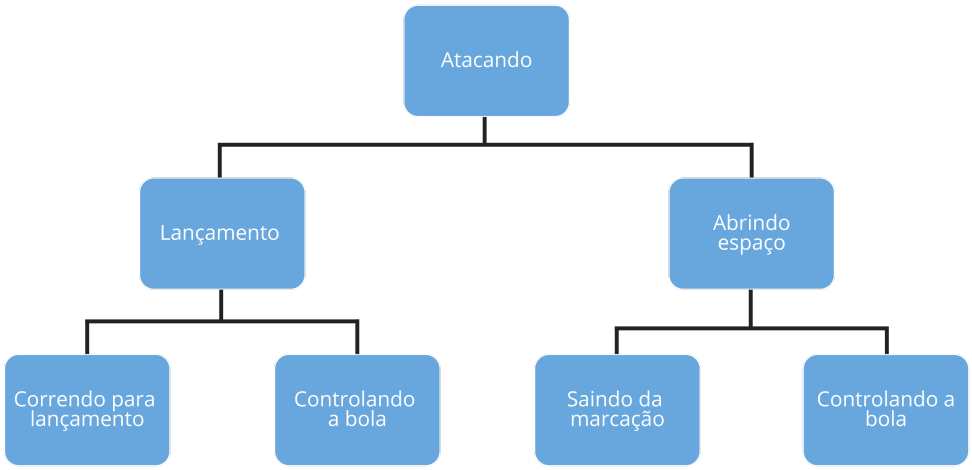
As transições de um estado mais abstrato são válidas para todo e qualquer sub-estado interno. Então, seja qual for o estado em que o personagem se encontra (4.1, 4.2 ou 4.3 - Figura 09), se a saúde dele passar a estar baixa ou se o inimigo ficar fora de seu alcance, então o personagem irá para o estado 3 ou o 2 (Figura 08), respectivamente.

Nessa variação, um estado é composto de outros estados internos, cada um podendo ser definido também como um conjunto de sub-estados, e assim por diante.

Por que isso é útil?

Retorne ao jogo de futebol que você está desenvolvendo. Praticamente todos jogadores estarão, por vezes atacando, por vezes defendendo. São dois tipos de comportamentos bem diferentes, o que significa que acaba refletindo em dois estados distintos. Porém, não há apenas uma forma de “atacar” ou de “defender”. O jogador pode estar atacando e, de forma mais específica, correndo para que seu colega faça um lançamento em profundidade, ou então saindo para abrir espaço mais para direita etc. Ou seja, mesmo que o estado dele seja “atacando”, é necessário detalhar melhor esse estado através de estados internos a ele. Essa organização de estados dentro de outros estados é chamada de **Máquina de Estados Finitos Hierárquica**, pois reflete as estruturas hierárquicas nas organizações humanas (chefe – subalterno). Veja o diagrama da Figura 10, ele não está representando as transições entre estados, mas apenas sua hierarquia.

Figura 10 - Diagrama de estados hierárquicos



Resumo

Foram muitos conceitos e ideias apresentadas nessa aula, não é mesmo? Pois é, você conheceu três diferentes alternativas para a **Máquina de Estados Finitos**. O uso de uma dessas alternativas não exclui necessariamente o uso da outra. Elas podem ser utilizadas em conjunto. Imagine, então, uma **Máquina de Estados Finitos Não-Determinística Fuzzy Hierárquica!** 😁

Mas antes de começar a programar algo assim, é importante que você tenha identificado em quais situações o uso de cada uma dessas técnicas é mais adequado. Não vale a pena programar algo complexo se o simples já resolve. Como já diria o grande amigo de Mogli, Balu, "Eu uso o necessário, somente o necessário. O extraordinário é demais". A filosofia desse urso sábio vale tanto na vida selvagem quando na vida tecnológica. Use com parcimônia (ação ou hábito de fazer economia, de poupar) as técnicas, procurando sempre adotar as soluções mais simples.

Assim, adote uma **MEF Não-Determinística** quando sua solução exige que não haja padrões de comportamento, quando for necessário "quebrar" a expectativa do jogador sobre os eventos que ocorrem no jogo. Adote uma **MEF Fuzzy** quando quiser adotar uma lógica mais natural (e, portanto, imprecisa) sobre os eventos do jogo. Por fim, adote uma **MEF Hierárquica** quando o número de estados e eventos começar a "explodir" a **MEF** em um grande número de possibilidades, tornando difícil gerenciá-la.

Apesar do grande número de conceitos, espero que esta aula tenha também aberto sua mente ao fato de que todas as técnicas apresentadas nesse curso são passíveis de serem adaptadas. Conhecer técnicas já desenvolvidas é essencial para não quebrar mais a cabeça sobre como resolver um determinado problema, porém, toda e qualquer técnica pode ser adaptada em função de suas necessidades.

Até a próxima! 😉

Referências

MILLINGTON, Ian; FUNGE, John. **Artificial intelligence for games.** 2. ed. Massachusetts: Morgan Kaufmann Eds, 2009.

MADHAV, Sanjay. **Game programming algorithms and techniques:** a plataform-agnostic approach. Massachusetts, Addison-Wesley, 2014.