Atividade 2 – Desenvolvimento dos comportamentos propostos pelo tutorial 3

1) Comportamento vagueante
   Primeiramente será desenvolvido um tanque que vagueia pelo mapa em busca de objetos e evita obstáculos. Este tanque será a base para um tanque mais complexo.

   Serão desenvolvidos três operadores básicos:
   - *Move:* move para frente se não estiver bloqueado
   - *Turn:* gira para um lado não bloqueado e liga o radar
   - *Turn*-backwards: gira para a esquerda se se houver obstáculos tanto à esquerda quanto à direita
   - *Radar-off:* desliga o radar se estiver ligado e se não houver objetos ao alcance do mesmo. Junto com este operador, é proposto um operador que aumenta a prioridade de radar-off sobre as de move e turn.
   - move-radar-off: sempre que uma operação de turn ou move for selecionada e o radar estiver ligado sem identificar nenhum objeto de interesse, além de realizar a ação de turn ou move, desliga o radar

   Estes operadores, assim como no Eaters, serão enviados para o *output-link* pelo operador create-action-command e removidos pelo operador remove-command quando forem completos.

   a. Move

sp {tanksoar*propose*move	
(state <s> ^name tanksoar	Se o estado tem nome tanksoar
^io.input-link.blocked.forward no) -->	e não há bloqueio afrente
(<s> ^operator <o> +)	Adiciona um operador ‘o’ no estado
(<o> ^name move	com atributo nome ‘move’
^actions.move.direction forward)}	E atributo ‘ações’ contendo um objeto ‘move’ que por sua vez contém um objeto ‘direction’ cujo valor é ‘forward’

   b. Turn

sp {propose*turn	
(state <s> ^name tanksoar	Se o estado tem nome tanksoar
^io.input-link.blocked <b>)	e o input-link possui um objeto ‘blocked’, coloque o objeto em ‘b’
(<b> ^forward yes	se o valor forward de b for yes (se houver um bloqueio afrente)
^ { << left right >> <direction> } no)	Para todas as direções d em [left, right], cria variável ‘direction’ de valor d se b.d valer ‘no’

	(selectiona as direções entre direita e esquerda que não se encontram bloqueadas)
-->	
(<s> ^operator <o> + =)	adiciona ao estado um operador ‘s’
(<o> ^name turn	de nome ‘turn’
^actions <a>)	com ações ‘a’
(<a> ^rotate.direction <direction>	Adiciona em a um atributo ‘rotate’ com atributo ‘direction’ valendo ‘direction’ (cria uma ação ‘turn’ para cada direção que não estava bloqueada indicando o nome da direção)
^radar.switch on	Adiciona em ‘a’ um atributo ‘radar’ com atributo ‘switch’ valendo ‘on’ (liga o radar)
^radar-power.setting 13)}	adiciona em ‘a’ um atributo ‘radar-power’ com atributo ‘setting’ valendo 13

c. Turn-backwards

sp {propose*turn*backward	
(state <s> ^name tanksoar	
^io.input-link.blocked <b>)	
(<b> ^forward yes ^left yes ^right yes)	tanto esquerda quanto direita se encontram bloqueados
-->	
(<s> ^operator <o> +)	adiciona o operador ‘o’
(<o> ^name turn	de nome ‘turn’
^actions.rotate.direction left)}	e adiciona em suas ações a ação de turn com atributo direction valendo left

d. Radar-off

sp {propose*radar-off	Ação que desliga o radar
(state <s> ^name tanksoar	
^io.input-link <il>)	
(<il> ^radar-status on	se o radar está ligado
-^radar.<< energy health missiles tank >>)	e não há objetos de interesse
-->	
(<s> ^operator <o> +)	adiciona o operador o
(<o> ^name radar-off	de nome radar-off
^actions.radar.switch off)}	e adiciona em suas ações a ação de radar com atributo switch valendo off

sp {select*radar-off*move	Prioriza ‘radar-off’ sobre ‘turn’ e ‘move’
(state <s> ^name tanksoar	
^operator <o1> +	Se existe operador ‘o1’
^operator <o2> +)	e operador ‘o2’
(<o1> ^name radar-off)	onde nome de ‘o1’ é ‘radar-off’

(<o2> ^name << turn move >>)	E nome de ‘o2’ é ‘turn’ ou ‘move’
-->	
(<s> ^operator <o1> > <o2>)}	Aumenta a prioridade de ‘o1’ sobre ‘o2’

A aplicação destas regras criou com sucesso um tanque capaz de andar, virar e ligar e desligar o radar. O radar é sempre ligado quando o tanque se move e assim que não haja um objeto de interesse em seu alcance, desliga o radar.

2) Comportamentos como operações independentes

O comportamento de vaguear (*wander*) por si só não é muito útil para a sobrevivência de um tanque no simulador. Para que um tanque seja bem sucedido, ele deve ser capaz de adotar comportamentos como perseguir e atacar um tanque inimigo e recuar de uma ameaça, por exemplo. Simplesmente adicionar mais operadores para que haja emergência destes diferentes comportamentos pode ser complicado, uma vez que os operadores são tarefas de baixo nível e os comportamentos tarefas de alto nível. Para selecionar as tarefas corretas para cada comportamento, muitos operadores de seleção precisariam ser aplicados e o código se tornaria confuso, altamente acoplado e difícil de manter.

Sendo assim, serão criadas ‘unidades’ diferentes para cada comportamento com as regras necessárias para que este emerja. Em SOAR, isto é feito utilizando-se sub estados. Haverá ‘operadores de alto nível’ e ‘operadores de baixo nível’. Os operadores de alto nível serão aplicados para selecionar o comportamento esperado. Enquanto este operador estiver selecionado, operadores de baixo nível associados a ele serão aplicados. Eventualmente, um outro operador de alto nível será selecionado por causa de mudanças no input ou por aquelas causadas pelos operadores de baixo nível do comportamento atual.

Pensando no *TankSoar*, podemos imaginar que se não houver inimigos, ameaça ou objetos de interesse para o tanque, o operador *wander* será selecionado. Enquanto ele estiver selecionado, os operadores *turn, move e radar-off,* por exemplo, farão que o tanque se mova pelo mapa e detecte outro tanque. Quando isto acontecer, o operador *wander* poderá ser removido e um operador para perseguição do tanque inimigo poderá ser selecionado. Este operador terá seus próprios operadores de baixo nível modificando o estado do tanque.

a) O operador *wander*

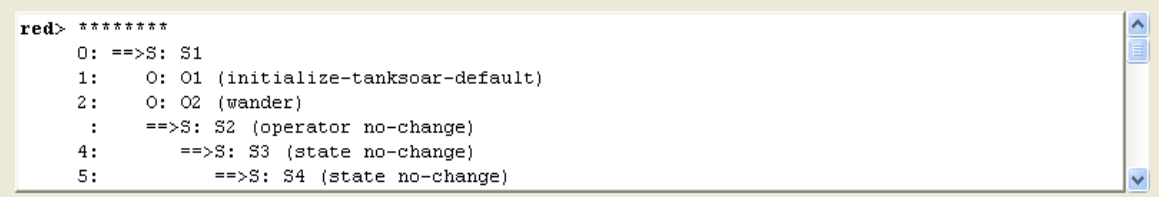
Serão utilizados os inputs dos sensores *incoming, radar and sound* para o desenvolvimento do comportamento de vaguear. A proposição deste operador ocorrerá da seguinte forma:

“Se não há tanques detectados pelo radar, não há som e não há mísseis vindo em direção ao tanque, proponha o operador de vaguear”

sp {propose*wander	
(state <s> ^name tanksoar	
^io.input-link <io>)	

(<io> ^sound silent	se o valor do input ‘sound’ for ‘silent’
-^radar.tank	e não há um atributo ‘tank’ no input ‘radar’
-^incoming.<dir> yes)	e não há um nenhum atributo no input ‘incoming’ com valor ‘yes’
-->	
(<s> ^operator <o> +)	adiciona ao estado um operador o
(<o> ^name wander)}	cujo nome é *wander*

Se criarmos um agente apenas com esta regra, ele entrara em um empasse que nunca será resolvido. O operador *wander* será selecionado mas não existe nada que faça com que ele mude de estado, uma vez que não estamos aplicando nenhuma ação sobre o tanque utilizando operadores de baixo nível. Abaixo uma imagem que mostra o log de execução de um agente desta forma.



Depois da inicialização (1) o operador *wander* é selecionado. Como não há mais regras que sejam ativadas uma vez que *wander* está selecionado, ocorre um impasse do tipo *operator no-change* e um sub estado S2 é criado. A seguir, como nenhum operador é proposto para o estado atual, ocorre um impasse do tipo *state no-change* e o estado S3 é gerado assim como o S4 em seguida. Este comportamento seguiria adiante indefinidamente até a memória do processo se esgotar.

Para obter o comportamento desejado, as regras de proposição de operadores serão ligeiramente modificadas de modo que os operadores de baixo nível para o comportamento de ‘*wander’* sejam propostos quando o operador ‘*wander’* estiver selecionado.

Primeiramente, a verificação inicial para cada regra garantia que havia um estado s cujo nome valia ‘tanksoar’. Agora, o que precisamos é que se verifique se o operador ‘wander’ está selecionado no super-estado (*superstate*) do estado atual. Isto pode ser feito através da seguinte verificação: `state <s> ^superstate.operator.name` wander

Outro problema é o atributo *input-link*, de onde são coletados os inputs do sistema. Originalmente, a regra de seleção dos operadores buscava um estado que tivesse um input-link dentre seus atributos. Agora, seria necessário buscar esta augmentação no superestado do estado atual. Contudo, se fizermos isso será um problema quando outro sub-estado surgir: precisaremos buscar em todos os super-estados até encontrarmos um com o *input-link*.

Para superar este problema, criaremos uma elaboração que adiciona em todo estado uma augmentação *input-link* com o valor do *input-link* original. Esta elaboração será como se segue:

sp {elaborate*state*io	
(state <s> ^superstate.io <io>)	Se houver um estado ‘s’ cujo super-estado possui um atributo io, copie io para uma variável ‘io’

-->	
(<s> ^io <io>)}	Crie uma augmentação ‘io’ no estado ‘s’ com o valor copiado na variável ‘io’

Esta abordagem permite uma abstração maior e um código mais inteligível. Sendo assim, podemos aplicar uma elaboração semelhante para a verificação do super-operador. Em vez de verificar se o super-estado tem um operador *wander*, podemos modificar o nome do estado atual quando este operador estiver selecionado. De forma geral, podemos modificar o nome do estado atual para o nome do operador selecionado no super-estado, como se segue:

sp {elaborate*state*name	
(state <s> ^superstate.operator.name <name>)	Se houver um estado com um super-estado e o super-estado possuir um operador selecionado que tenha um nome, copie o nome para a variável ‘nome’
-->	
(<s> ^name <name>)}	O nome do estado passa a ser o nome armazenado na variável ‘nome’

As regras modificadas ficarão como se segue:

```
sp {wander*propose*move
   (state <s> ^name wander
              ^io.input-link.blocked.forward no)
-->
   (<s> ^operator <o> + =)
   (<o> ^name move
        ^actions.move.direction forward)}


sp {wander*propose*turn
   (state <s> ^name wander
              ^io.input-link.blocked <b>)
   (<b> ^forward yes
        ^ { << left right >> <dir> } no)
-->
   (<s> ^operator <o> + =)
   (<o> ^name turn
        ^actions <a>)
   (<a> ^rotate.direction <dir>
        ^radar.switch on
        ^radar-power.setting 13)}


sp {wander*propose*turn*backward
   (state <s> ^name wander
              ^io.input-link.blocked <b>)
   (<b> ^forward yes
        ^left yes
        ^right yes)
-->
```

```
(<s> ^operator <o> +)
(<o> ^name turn
     ^actions.rotate.direction left)}
```

b) O operador *Chase*

Em linhas gerais, é desejável que operador de alto-nível *chase* deve ser selecionado quando um tanque inimigo for percebido e o tanque agente tiver em boas condições (disponibilidade de mísseis e tiver uma quantidade razoável de energia para se defender com o escudo. Além disso, este não é o comportamento desejado se um tanque já está no radar (quando isso acontecer, é desejável que o agente ataque o inimigo, não que o persiga)

Para facilitar a leitura e modificação do código do agente, criaremos uma elaboração que defina se a energia do agente ou sua quantidades de mísseis está baixa. Como para qualquer um dos casos desejamos que o tanque o tanque não persiga um inimigo, usaremos a mesma augumentação para ambas as condições, como se segue.

sp {elaborate*state*missiles*low	Elaboração para o estado dos mísseis
(state <s> ^name tanksoar	Apenas para o estado superior
^io.input-link.missiles 0)	Se o agente não possui mísseis
-->	
(<s> ^missiles-energy low)}	Cria uma augumentação indicando baixa quantidade de mísseis ou energia

sp {elaborate*state*energy*low	Elaboração para o estado da energia
(state <s> ^name tanksoar	Apenas para o estado superior
^io.input-link.energy <= 200)	
-->	
(<s> ^missiles-energy low)}	Cria uma augumentação indicando baixa quantidade de mísseis ou energia

Assim, a proposição do operador *chase* deve ser como se segue:

sp {propose*chase	
(state <s> ^name tanksoar	Operador 'de alto nível'
^io.input-link <io>	
-^missiles-energy low)	Se nem a energia nem os mísseis estão em estado baixo
(<io> ^sound <> silent	e se o valor do input de som é diferente de 'silent'
-^radar.tank)	e se não há um tanque detectado no radar
-->	
(<s> ^operator <o> +)	Adiciona um operador no estado superior
(<o> ^name chase)}	com nome 'chase'

A aplicação deste operador é semelhante às aplicações do operador *wander*, já que ambas são relacionadas à locomoção do agente. A diferença é que este operador tenta se locomover em direção ao som que está percebendo. Além disso, durante uma perseguição o agente manterá seu radar sempre ligado. Esta é claramente uma abordagem gulosa da solução já que existem obstáculos e seguir a direção até o som provavelmente não é o caminho correto para o tanque que está causando o som.

Antes de implementar as regras devidamente, criaremos uma elaboração que facilitará a leitura do input do sensor de som. Esta elaboração insere no estado do agente uma augmentação *'sound-direction'* que tem o mesmo valor do sensor de som, como se segue.

sp {chase*elaborate*state*sound-direction	
(state <s> ^name chase	Se o estado é 'chase'
^io.input-link.sound <sound>)	copia o input para <sound>
-->	
(<s> ^sound-direction <sound>)}	augmenta o estado com o valor

O radar será ligado se o agente se encontra em uma perseguição:

sp {chase*elaborate*radar	
(state <s> ^name chase	
^operator.actions <a>	
^io.input-link.radar-status off)	Se o radar estiver desligado
-->	
(<a> ^radar.switch on	Adiciona uma ação nas ações do operador para ligar o radar
^radar-power.setting 13)}	Adiciona uma ação nas ações do operador para definir a potencia do radar para 13

O operador *move* deve ser aplicado se há um som afrente e se não está bloqueado. Esta também é uma abordagem gulosa para a solução do problema.

sp {chase*propose*move	
(state <s> ^name chase	
^sound-direction forward	se há um som afrente
^io.input-link.blocked.forward no)	e o agente não está bloqueado afrente
-->	
(<s> ^operator <o> +)	Adiciona um operador
(<o> ^name move	com nome 'move'
^actions.move.direction forward)}	adiciona atributo move com um atributo direction de valor forward

O operador ´turn' simplesmente cria uma ação de rotação na direção de onde há um som vindo (entre direita e esquerda apenas, já que um tanque só pode girar nestas direções)

sp {chase*propose*turn	
(state <s> ^name chase	

^sound-direction {<< left right >> <direction>})	Para todas as direções d em [left, right], cria variável 'direction' de valor d se o o atributo sound-direction tiver uma augumentação d
-->	
(<s> ^operator <o> + =)	adiciona operador
(<o> ^name turn	com nome 'turn'
^actions.rotate.direction <direction>)}	adiciona atributo 'turn' com um atributo direction de valor d

O operador '*backward*' simplesmente vira o tanque para a esquerda se houver um som vindo de trás. Isso é necessário porque o tanque só pode virar para a esquerda e para a direita. No próximo passo, o tanque deverá utilizar o operador '*turn*' em vez deste se os inputs se mantiverem.

sp {chase*propose*backward	
(state <s> ^name chase	
^sound-direction backward)	se há um som vindo de trás
-->	
(<s> ^operator <o> +)	Cria um operador o
(<o> ^name turn	De nome 'turn'
^actions.rotate.direction left)}	Com uma ação re rotação cujo atributo direção vale 'left'

c) O operador *Attack*

Um tanque deve adotar um comportamento ofensivo quando detecta um inimigo em seu radar. Contudo, nem sempre um tanque detectado está diretamente afrente do agente. Se o tanque estiver diretamente afrente (no centro do radar) o agente simplesmente atira. Se o tanque estiver à esquerda ou à direita o tanque deve se mover para estas posições antes de poder atirar o míssil (já que este segue sempre em trajetória retilínea). Precisamos levar em consideração que o agente pode estar bloqueado nestas direções de movimento. Se este for o caso, o tanque deve se mover para frente apenas. No próximo ciclo, pode ser que a direção não esteja mais bloqueada e o tanque tenha o posicionamento certo para efetuar o disparo. Pode ser necessário executar esta movimentação até que o agente fique imediatamente adjacente ao tanque inimigo. Quando isto ocorrer, o agente deve girar em direção ao inimigo e atirar na mesma ação. A seguir, seguem as implementações destas regras na linguagem SOAR.

sp {attack*propose*fire-missile	Atira caso o inimigo esteja no centro do radar
(state <s> ^name attack	
^io.input-link <il>)	
(<il> ^radar.tank.position center	Se o tanque detectado estiver no centro do radar
^missiles > 0)	e se o agente tiver mísseis
-->	
(<s> ^operator <o> + >)	Adiciona um operador com a ação para o disparo
(<o> ^name fire-missile	

^actions.fire.weapon missile)}	

sp {attack*propose*slide	Move para a direção do tanque detectado se esta não se encontrar bloqueada
(state <s> ^name attack	
^io.input-link <input>)	
(<input> ^blocked.<dir> no	Para cada direção <dir> do agente não bloqueada (entre esquerda ou direita)
^radar <r>)	
(<r> ^tank.position { << left right >> <dir> }	se a posição do tanque inimigo for igual a <dir>
-^tank.position center)	E não houver um tanque no centro
-->	
(<s> ^operator <o> + =)	Cria um operador com ação para se mover na direção <dir> com indiferença caso haja um tanque tanto à direita quanto à esquerda
(<o> ^name slide	
^actions.move.direction <dir>)}	

sp {attack*propose*move-forward	Move o agente para frente se a direção onde está o tanque inimigo estiver bloqueada
(state <s> ^name attack	
^io.input-link <input>)	
(<input> ^blocked.<dir> yes	seja <dir> uma direção bloqueada
^radar <r>)	seja <r> o radar do input
(<r> ^tank <t>	seja <t> um tanque detectado por <r>
-^tank.position center)	e não há tanque no centro do radar
(<t> ^position { << left right >> <dir> }	se a <t> se encontrar na direção <dir> bloqueada
^distance <> 0)	e não estiver adjacente ao agente (caso contrário queremos que o tanque gire e atire)
-->	
(<s> ^operator <o> + =)	cria um operador com uma ação para mover o agente para frente
(<o> ^name move-forward	
^actions.move.direction forward)}	

Obs: Esta abordagem proposta pelo tutorial pode ser melhorada. Se houver um tanque à esquerda e afrente e um à direta e afrente e, por exemplo, a direita estiver bloqueada mas a esquerda não, o agente deveria se mover para a esquerda (como proposto no operador ‘slide’. Podemos criar a seguinte regra para gerar esta preferência:

sp {select*attack*slide*move-forward	Prioriza 'slide' sobre 'move- forward
(state <s> ^name attack	
^operator <o1> + =	Se existe operador 'o1'
^operator <o2> + =)	e operador 'o2'
(<o1> ^name slide)	onde nome de 'o1' é 'slide'
(<o2> ^name turn move-forward)	E nome de 'o2' é 'move-forward'
-->	
(<s> ^operator <o1> > <o2>)}	Aumenta a prioridade de 'o1' sobre 'o2'

sp {attack*propose*turn	Gira o agente na direção onde há um tanque adjacente e faz com que ele dispare um míssil
(state <s> ^name attack	
^io.input-link.radar.tank <tank>)	
(<tank> ^distance 0	seja <tank> um tanque inimigo adjacente
^position { << left right >> <dir> })	seja <dir> a direção onde se encontra <tank>
-->	
(<s> ^operator <o> + =)	Cria um operador com uma ação para que o tanque gire na direção do tanque inimigo e atire. A ordem destas ações é girar e atirar, mas porque o simulador é definido desta forma e não por causa da ordem das ações
(<o> ^name turn	
^actions <a>)	
(<a> ^rotate.direction <dir>	
^fire.weapon missile)}	

d) O operador 'retreat'

Este operador é por natureza o mais complicado pois vários cenários devem ser considerados para seu tratamento. Vamos considerar que um tanque deve recuar nas seguintes situações:

1. O agente está com baixa energia e
    i. Detecta outros tanques por som **ou**
    ii. Detecta um tanque pelo radar **ou**
    iii. Um míssil for disparado em sua direção

2. Existe um míssil vindo em direção ao agente **e**
    i. Não há tanque detectado pelo radar **e**
    ii. Não há tanque detectado por som

Sendo assim, criaremos quatro regras que selecionarão o operador 'retreat' para aplicação, como se segue.

sp {propose*retreat*sound	Recua se há som detectado e agente com baixa energia ou mísseis
(state <s> ^name tanksoar	
^missiles-energy low	Energia ou mísseis baixos
^io.input-link.sound {<direction> <> silent})	E som detectado em qualquer direção <direction>
-->	
(<s> ^operator <o> + =)	Propõe o operador 'retreat'
(<o> ^name retreat)}	

sp {propose*retreat*radar	Recua se há um tanque detectado pelo radar e pouca energia ou mísseis
(state <s> ^name tanksoar	
^missiles-energy low	
^io.input-link.radar.tank)	
-->	
(<s> ^operator <o> + =)	
(<o> ^name retreat)}	

sp {propose*retreat*incoming	Recua se há um míssil indo em direção ao agente e este possui pouca energia ou mísseis
(state <s> ^name tanksoar	
^missiles-energy low	
^io.input-link.incoming.<dir> yes)	Existe um míssil vindo em qualquer direção <dir>
-->	
(<s> ^operator <o> + =) (<o> ^name retreat)}	

sp {propose*retreat*incoming*not-sensed	Recua se o houver míssil vindo em direção ao agente e se nenhum tanque foi percebido por seus sensores
(state <s> ^name tanksoar	
^io.input-link <io>)	
(<io> ^incoming.<dir> yes	Existe um míssil vindo de alguma direção <dir>
-^radar.tank	Não há tanque no radar
^sound silent)	Não há som no sensor de som
-->	
(<s> ^operator <o> + =) (<o> ^name retreat)}	

A aplicação do operador para recuar é por natureza uma operação de movimentação. Obviamente esta movimentação deve ser feita de tal forma que evite uma direção onde foram detectadas ameaças. Contudo, é possível que não haja uma movimentação boa para se efetuar. Neste caso, a abordagem será o agente se manter em silêncio na esperança de que o inimigo não o perceba.

Para se saber para onde recuar é necessário manter no estado quais são as direções ameaçadas. Para isto, utilizaremos elaborações no estado '*retreat*' que computarão as direções de onde provém ameaças. As ameaças são detecção tanque por som, detecção de um ranque pelo radar e detecção de míssil vindo na direção do agente:

sp {elaborate*retreat*sound*direction	Guarda todas as direções de onde se detectou sons
(state <s> ^name retreat	
^io.input-link.sound { <> silent <direction> })	para toda direção <direction> de onde se detectou um som
-->	
(<s> ^direction <direction>)}	guarda <direction>

sp {elaborate*retreat*radar*front	Guarda a direção 'forward se um tanque foi detectado pelo radar
(state <s> ^name retreat	
^io.input-link.radar.tank)	se há um tanque no radar
-->	
(<s> ^direction forward)}	Guarda 'forward'

sp {elaborate*retreat*incoming*direction	Guarda toda direção de onde provém misséis
(state <s> ^name retreat	
^io.input-link.incoming.<dir> yes)	Para toda direção <dir> de onde se detectou mísseis
-->	
(<s> ^direction <dir>)}	guarda dir

sp {elaborate*retreat*radar*direction	Marca todas as direções não centrais onde se detectou um tanque pelo radar
(state <s> ^name retreat	
^io.input-link.radar.tank.position { <dir> <> center })	Para toda direção <dir> não central onde se detectou um tanque pelo radar
-->	
(<s> ^avoid-direction <dir>)}	Marca <dir> para ser evitada

Para evitar estas ameaças, iremos adotar a abordagem de mover o agente para um quadrado de forma que evite ficar em uma linha reta em relação à direção de onde vem a ameaça. Por exemplo, se existe uma ameaça à frente do tanque, queremos que ele se mova para a direita ou para a esquerda, pois se for para trás pode continuar na linha de tiro do inimigo afrente e o

mesmo se for para trás. Esta é uma abordagem muito simplista mas será utilizada para o propósito do treinamento.

Para não precisarmos fazer contas complicadas a todo momento para determinar quais são as direções que o agente deve se mover a dada uma direção ameaçada, manteremos estes valores fixos no estado superior com uma simples elaboração que determina para cada direção, qual o paço que tira o agente de sua linha reta:

```
sp {elaborate*sidestep-directions
   (state <s> ^name tanksoar)
-->
   (<s> ^side-direction <sd>)
   (<sd> ^forward right left ^backward right left
         ^right forward backward ^left forward backward)}
```

Sendo assim, a aplicação do operador *'retreat' é como se segue:*

`sp {retreat*propose*move`	
`(state <s> ^name retreat`	
`^direction <dir>`	Para toda direção <dir> marcada como ameaçada
`^superstate.side-direction.<dir> <ndir>`	seja <ndir> cada uma das direções ideais para evitar uma ameaça vinda de <dir>
`-^direction <ndir>`	se <ndir> não está marcada como ameaçada
`-^avoid-direction <ndir>`	nem deve ser evitada
`^io.input-link.blocked.<ndir> no)`	e não está bloqueada
`-->`	
`(<s> ^operator <o> + =)`	Cria um operador com uma ação de movimento para esta direção
`(<o> ^name move`	
`^actions.move.direction <ndir>)}`	

Para completar o mecanismo de defesa do agente, é necessário que ele ative seu escudo caso esteja sob ataque. Para fazer isto sem gastar energia à toa, faremos o agente desativar seu escudo se não estiver sob ataque. Estas regras são bem simples de compreender e são descritas abaixo:

```
sp {elaborate*shields-on
   (state <s> ^operator.actions <a>
              ^io.input-link <il>)
   (<il> ^incoming.<dir> yes
         ^shield-status off)
-->
   (<a> ^shields.switch on)}

sp {elaborate*shields-off
   (state <s> ^operator.actions <a>
              ^io.input-link <il>)
   (<il> -^incoming.<dir> yes
          ^shield-status on)
-->
```

```
   (<a> ^shields.switch off)}
```

e) Operador Wait

Em muitos dos casos que descrevemos, cairemos em um impasse onde nenhum operador será selecionado. Para que isso não gere uma cascata infinita de sub estados no sistema, criaremos um operador 'wait' que não realiza ação nenhuma. Este é um operador muito útil e genérico que pode ser utilizado em diversas aplicações. Basicamente, se nenhum operador pode ser selecionado e wait não está selecionado o operador 'wait' será selecionado. No próximo ciclo, este a proposta ira retrair pois o operador foi selecionado. No ciclo seguinte, ele poderá ser selecionado de novo e assim por diante.

```
sp {top-state*propose*wait
   (state <s> ^attribute state
              ^choices none
             -^operator.name wait)
-->
   (<s> ^operator <o> +)
   (<o> ^name wait)}
```

Conclusão

Esta atividade cobriu diversos problemas simples mas importantes de serem compreendidos no desenvolvimento de um projeto em SOAR. Em especial, a compreensão de como utilizar estados e sub estados para criar abstrações e facilitar a compreensão do e elaboração do código foi de fundamental importância e sem dúvidas é uma peça fundamental para criação de agentes complexos. Além disso, por mais simples que seja, foi importante compreender o funcionamento de um operador como o *wait*, que deve ser utilizado sempre que possível para evitar uma cascata infinita de sub estados por impossibilidade de decisão.