

Raciocínio de Senso Comum para Agentes em AI Game Engines

Felipe Benezra, Luciano Silva

Faculdade de Computação e Informática

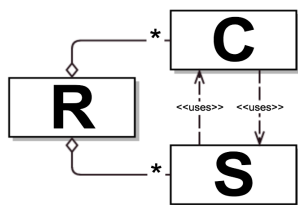
Universidade Presbiteriana Mackenzie

Caixa Postal 930 – 01302-907 – São Paulo – SP – Brazil

felipebenezra@hotmail.com, luciano.silva@mackenzie.br

Abstract. The exploration of new logics represents a field very wide to research in Artificial Intelligence applicated to games. The specification and implementation of plugins that allow to add intelligent behavior more and more complex to agents in game engines shows great interest both from computing and digital entertainment. In this context, this project proposes the specification and implementation of a plugin to intelligent behaviors based on Commonsense Reasoning, a form of mathematical logic, for agents in the UT3 game engine. This form of logic will allow exploration of agents behavior more related to human behavior in situations of conflict.

Resumo. A exploração de novas lógicas representa um campo muito fértil para pesquisa em Inteligência Artificial aplicada a jogos. A especificação e implementação de plugins que permitam adicionar comportamentos inteligentes cada vez mais complexos a agentes em game engines é de grande interesse tanto do ponto de vista computacional quanto do entretenimento digital. Neste contexto, este projeto propõe a especificação e implementação de um plugin para comportamentos inteligentes baseados em Raciocínio de Senso Comum, uma das formas de Lógica Matemática, para agentes no game engine UT3.



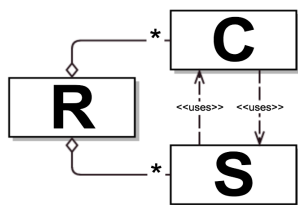
Esta forma de lógica permitirá a exploração de comportamentos de agentes mais relacionados com o comportamento humano frente a situações de conflito.

1. Introdução

A idéia central da Inteligência Artificial é ter computadores que entendem fatos simples sobre pessoas e sobre o dia-a-dia delas. Isto é chamado de senso comum. Uma abordagem para este problema é formalizar o Raciocínio de Senso Comum através da lógica matemática.

Esta formalização permite que se desenvolva um novo tipo de lógica. Consequentemente, também se desenvolve novas modalidades importantes para jogos, tais como a Inteligência Artificial dos agentes de um jogo. Por essas razões, é fundamental o estudo de novas lógicas para ser aplicado em jogos, no intuito de inovar as técnicas e possibilidades ao se programar um jogo, permitindo aos jogadores maiores desafios e possibilidades.

O objetivo deste trabalho é introduzir o Raciocínio de Senso Comum aos agentes inteligentes do *Game Engine* UT3, através do Cálculo de Eventos, do Cálculo Situacional e do Cálculo de Fluentes, que são ferramentas do Raciocínio de Senso Comum. Além disso, há também o uso da Lógica de Primeira Ordem, servindo como um pré-requisito para a realização dessa lógica. Com isto, será possível fazer uso de técnicas diversificadas de Inteligência Artificial para jogos e ter uma gama de opções inovadoras que podem ser utilizadas, dependendo da situação que o usuário deseja criar.



2. Raciocínio de Senso Comum

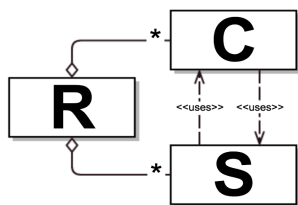
Este capítulo apresenta o Cálculo de Eventos, de Fluentes e Situacionais, respectivamente, utilizando-se formalismos da Lógica de Primeira Ordem.

De acordo com Mueller (2006), com o Cálculo de Eventos é possível se representar conhecimento de senso comum, representar cenários e até inferir algo sobre eles. Existem três conceitos importantes quando se trata do cálculo de eventos: eventos, fluentes e pontos temporais. Os eventos representam as ações que podem ocorrer no universo estudado. Um fluente é uma propriedade do mundo em que varia com o tempo, por exemplo, a localização de um determinado objeto em instantes de tempo distintos. Um ponto temporal se trata de um instante de tempo específico, uma data.

Cada conceito é representado por um tipo distinto. O tipo de evento, representado pela letra *e*, o de fluente pela letra *f*, e o de ponto temporal, que é um subtipo do tipo de um número real, pela letra *t*. Esses tipos servem para definir os predicados no cálculo de eventos, mostrados na Tabela 1:

Tabela 2. Predicados do Cálculo de Eventos.

Predicado	Descrição
<i>Happens (e, t)</i>	Evento <i>e</i> acontece ou ocorre em um ponto temporal <i>t</i> .
<i>HoldsAt (f, t)</i>	Fluente <i>f</i> é verdadeiro no ponto temporal <i>t</i> . Se $\neg HoldsAt (f, t)$, então dizemos que <i>f</i> é falso em <i>t</i> .
<i>ReleasedAt (f, t)</i>	Fluente <i>f</i> é liberado pela lei da inércia do senso comum em um ponto temporal <i>t</i> . Se $\neg ReleasedAt (f, t)$, então <i>f</i> não é liberado.

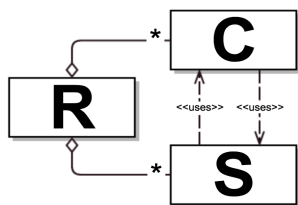


<i>Initiates</i> (e, f, t)	Evento e inicia um fluente f no ponto temporal t .
<i>Terminates</i> (e, f, t)	Evento e termina um fluente f no ponto temporal t .
<i>Releases</i> (e, f, t)	Evento e solta o fluente f no ponto temporal t .
<i>Trajectory</i> ($f1, t1, f2, t2$)	Se o fluente $f1$ é iniciado por um evento que ocorre no ponto temporal $t1$, e $t2 > 0$, então o fluente $f2$ será verdadeiro no ponto temporal $t1+t2$.
<i>AntiTrajectory</i> ($f1, t1, f2, t2$)	Se o fluente $f1$ é terminado por um evento que ocorre no ponto temporal $t1$, e $t2 > 0$, então o fluente $f2$ será verdadeiro no ponto temporal $t1+t2$.

Um fluente f pode ter 4 estados em um dado ponto temporal t . Eles são mostrados na Tabela 2:

Tabela 1. Estados de um fluente.

Estados	Predicados
Verdadeiro e liberado	$HoldsAt(f, t) \wedge ReleasedAt(f, t)$
Verdadeiro e não liberado	$HoldsAt(f, t) \wedge \neg ReleasedAt(f, t)$
Falso e liberado	$\neg HoldsAt(f, t) \wedge ReleasedAt(f, t)$
Falso e não liberado	$\neg HoldsAt(f, t) \wedge \neg ReleasedAt(f, t)$



A principal diferença entre o cálculo situacional e o cálculo de eventos é que o cálculo situacional usa ramificação de tempo, enquanto que o cálculo de eventos usa tempo linear. No cálculo de eventos, há uma única linha de tempo na qual os eventos ocorrem, e todos os eventos são considerados como reais. No cálculo situacional, o tempo é representado por uma árvore, e cada evento pode dar origem a um diferente futuro; portanto, os eventos são considerados como hipotéticos. Um ponto no tempo é representado por uma situação, isto é, se em uma situação N ocorrer uma ação, no final desta estará em uma situação $N+1$.

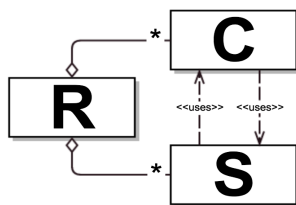
O cálculo de fluentes acrescenta a noção de estados com as noções de ações, fluentes e situações do cálculo situacional.

Um estado representa um conjunto de fluentes e é definido por indução, como mostrado a seguir:

- \emptyset é um estado.
- Se f é um fluente, então f é um estado.
- Se z_1 e z_2 são estados, então $z_1 \circ z_2$ é um estado.
- Nada mais é um estado.

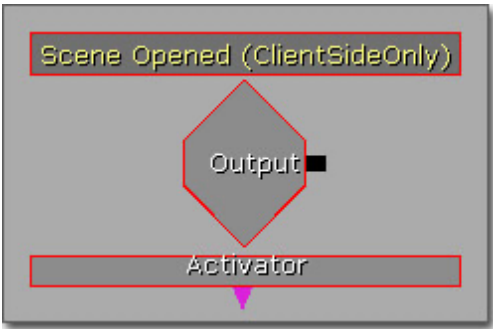
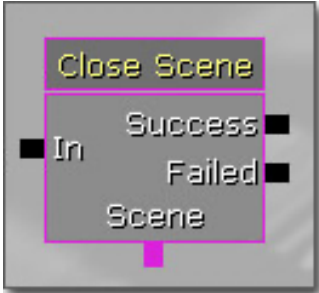

3. Proposta

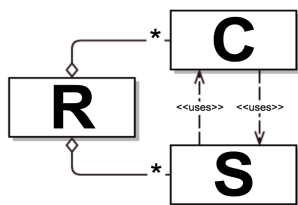
A proposta deste trabalho é desenvolver um padrão para a implementação da lógica de um jogo no *UnrealKismet*, através do mapeamento do Raciocínio de Senso Comum para o *UnrealKismet*. Este mapeamento é utilizado para descrever o modo de como será implementado os comportamentos no *UnrealKismet*. Este mapeamento tem como objetivo formalizar a implementação no *UnrealKismet*, pois não há um padrão para tal.

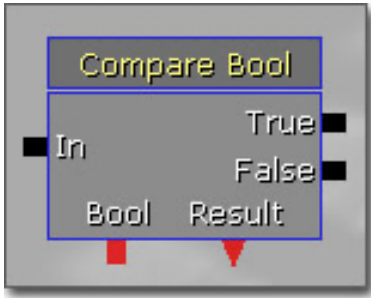



O mapeamento é mostrado pela Tabela 3:

Tabela 3. Mapeamento do Raciocínio de Senso Comum para o *UnrealKismet*.

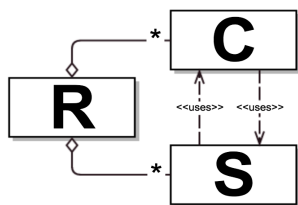
Raciocínio de Senso Comum	<i>UnrealKismet</i>
Eventos do Cálculo de Eventos	<p>Eventos</p>  <p>ou</p> <p>Ações</p> 
Fluentes do Cálculo de Eventos	<p>Variáveis</p> 
Pontos Temporais do Cálculo de Eventos	Tempo do jogo (<i>gameplay</i>)



Tempo do Cálculo Situacional	Condições 
Estados do Cálculo de Fluentes	Variáveis 

Com o auxílio desta tabela, é possível mapear um exemplo para o raciocínio de senso comum e deste para o *UnrealKismet*. Por exemplo, uma personagem teletransportar para trás, sempre que colidir com uma parede. No Raciocínio de Senso Comum, isso ficaria

Initiates (ColidirParede, Teletransportar, t).



Passando para o *UnrealKismet*:

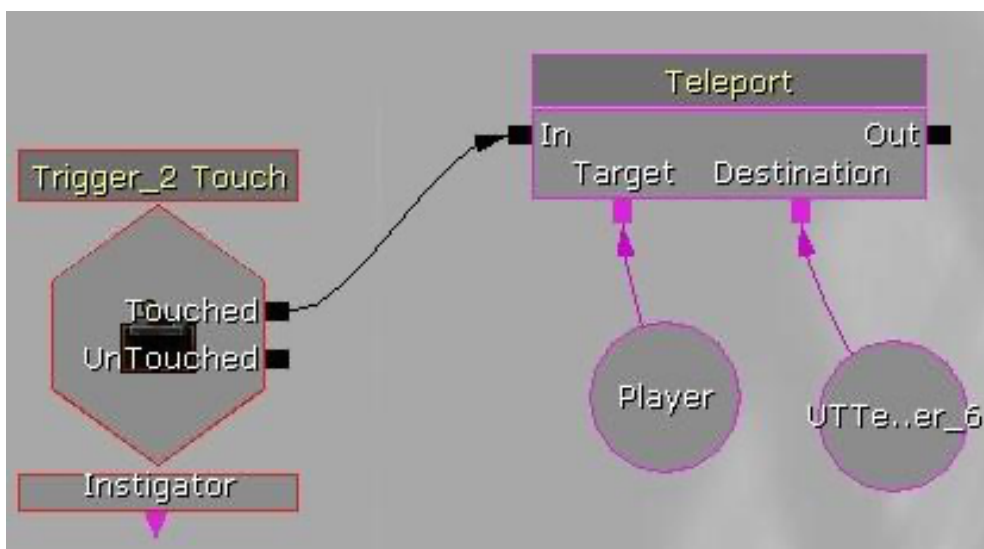


Figura 1. Personagem se teletransportando ao encostar na *trigger*, no *UnrealKismet*.

O Trigger_2 Touch indica um evento de *Trigger* do tipo *Touch*, que, ao tocá-lo, o alvo será teletransportado. Outro exemplo seria a personagem atirar quando vir um inimigo. No Raciocínio de Senso Comum ficaria assim

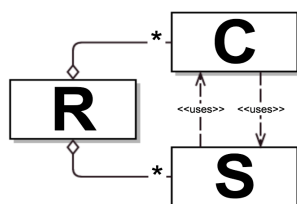
$$Poss (AtirarInimigo) \equiv ver (inimigo, s) \wedge equipada(arma, s)$$

que diz que uma personagem só pode atirar no inimigo se, e somente se, estiver vendo o inimigo em uma situação *s* e estiver equipada com alguma arma nesta mesma situação.

Este exemplo no *UnrealKismet* é mostrado pela Figura 2:



Figura 2. Personagem atirando no inimigo, no *UnrealKismet*.



Outro exemplo que pode ser aplicado no Raciocínio de Senso Comum é

State (Do) Perseguir (inimigo), s = (State (s) – Parada (p)) + Movimento (p) + Olhando (inimigo)

que diz que se uma personagem começa a perseguir um inimigo, então esta personagem não está mais parada. Agora ela está em movimento e olhando para o inimigo. Este exemplo no *UnrealKismet* é mostrado pela Figura 3:

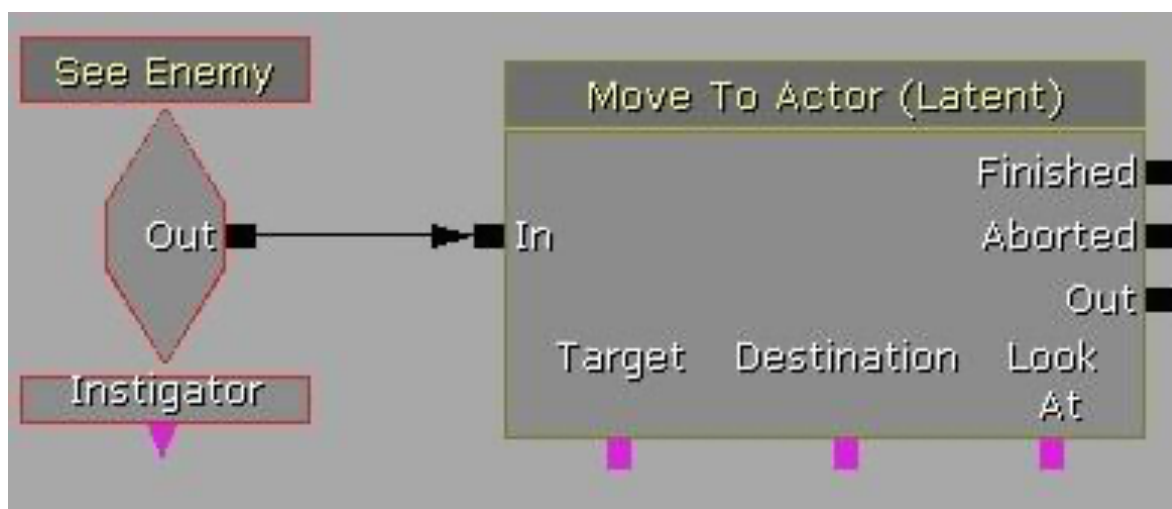
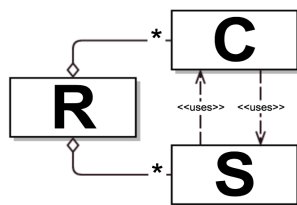


Figura 3. Personagem perseguindo o inimigo, no *UnrealKismet*.

4. Testes e Resultados

Os testes do primeiro cenário são focados a ativação da *trigger* conectada a parede, com o objetivo de verificar se a personagem realmente se afastava ao encostar na *trigger*.



O resultado é mostrado na Figura 9:

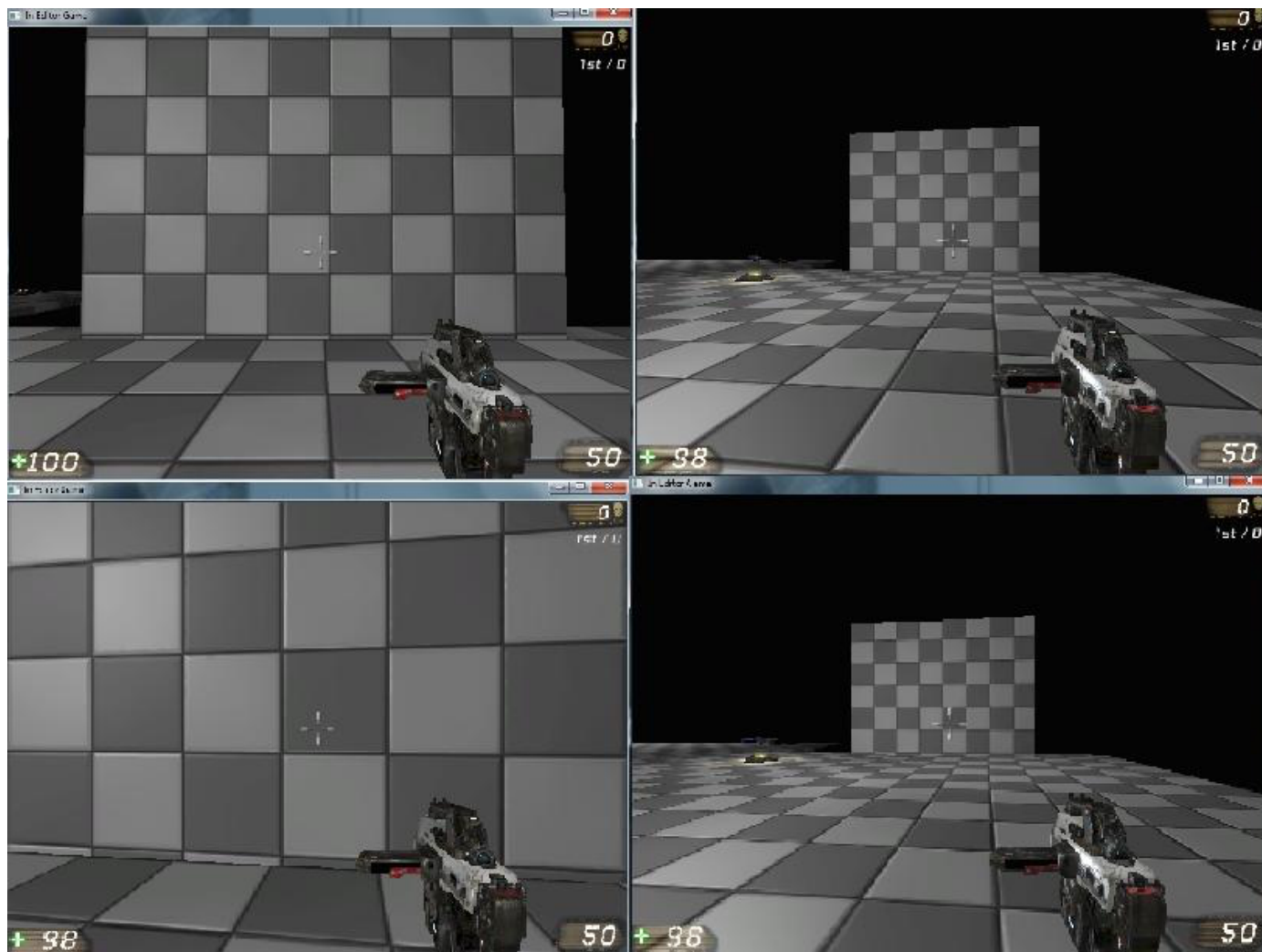
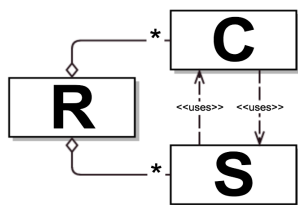


Figura 9. Sequência de figuras do primeiro cenário em execução.

Nesta figura, nota-se que quando a personagem se aproxima da parede, além de ser teletransportado para trás, há um dano causado pela ação. O dano foi utilizado apenas como teste.



5. Conclusões e Trabalhos Futuros

O Raciocínio de Senso Comum é um tipo de raciocínio que permite afirmar qualquer tipo de situação, seja quando uma pessoa entra na cozinha ou quando ela está comendo. Isto é possível devido à facilidade de se criar inferências sobre um determinado cenário, possibilitando realizar operações muito complexas.

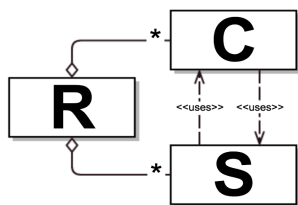
Esses tipos de lógicas não convencionais são de suma importância para jogos, por diversificar a lógica de criação da Inteligência Artificial dos jogos, gerando novas possibilidades de movimentação, de ações e até de conceitos.

Tendo-se como objetivo essas novas possibilidades, foi feito um mapeamento do Raciocínio de Senso Comum para a ferramenta *Unreal*, no intuito de mostrar o que se pode adicionar ao *Unreal* com a inclusão deste raciocínio.

O resultado se direcionou para a criação de um padrão para a construção da lógica no *UnrealKismet*, através de uma tabela que mapeia do Raciocínio de Senso Comum para o *UnrealKismet*.

Porém, o que se observou é que a parte de Inteligência Artificial na ferramenta *Unreal* é um pouco limitada e imprecisa quanto aos movimentos e ações que o agente realiza. O que se poderia fazer a respeito é construir uma lógica para os agentes inteligentes do *Unreal* que fosse mais precisa e desse suporte a diversos tipos de manipulação requerida pelo usuário que utiliza a ferramenta.

Outro trabalho futuro que poderia ser feito seria construir uma espécie de *debug* na execução do *Unreal*, de forma que desse para visualizar a lógica de Inteligência Artificial de alguma forma enquanto se testa a aplicação. Dessa



maneira, ficaria mais fácil de entender como está funcionando a lógica implementada.

Referências Bibliográficas:

Busby, J.; Parrish, Z.; Van Eenwyk, J. Mastering Unreal Technology: The Art of Level Design. New York: Sams Publishing, 2005a.

_____. Mastering Unreal Technology Volume II: Mastering Unreal Technology: The Art of Level Design. New York: Sams Publishing, 2005b.

_____. Mastering Unreal Technology Volume III: Introduction to UnrealScript with Unreal Engine 3. New York: Sams Publishing, 2005c.

Epic Games (2001). UDN – Three – UIEditorUserGuide. Disponível em: <http://udn.epicgames.com/Three/UIEditorUserGuide.html>. Acesso em: 15/10/2010.

Epic Games (2008). Unreal Technology. Disponível em: <http://www.unreal.com/features.php?ref=kismet>. Acesso em: 15/10/2010.

Millington, I. Artificial Intelligence for Games. New York: Morgan Kaufmann, 2006.

Mueller, E. Commonsense Reasoning. Boston: The MIT Press, 2006.

Reynolds, C. W. (1987) Flocks, Herds, and Schools: A Distributed Behavioral Model, in Computer Graphics, 21(4) (SIGGRAPH '87 Conference Proceedings).