Desenvolvimento de um Sistema Baseado em Regras em um motor de jogos

Aluno: Rodrigo Volpe Battistin
Orientador: Dr. Wilson Kazuo Mizutani
Profa. Responsável: Nina Sumiko Tomita Hirata

Inteligência Artificial em jogos

Dentre as inúmeras aplicações de Inteligência Artificial (IA) como uma ferramenta dentro de sistemas de *software*, temos o seu uso para a criação de personagens que atuam sozinhas em cenários de jogos (chamadas comumente de "*bots*"[1], abreviação de *robots*). Esses agentes são responsáveis por dar vida a uma grande variedade de *games*, desde os mais simples *Platformers*, como **Super Mario Bros.** [Nintendo, 1985], até os mais recentes *First Person Shooters*, como **Valorant** [Riot Games, 2020].

Nesse contexto, o papel da IA é reger as ações das personagens não controladas por um jogador, de forma a proporcionar interações entre a(s) pessoa(s) jogando o jogo e as entidades que habitam seu espaço virtual. Essas interações podem ser desafios, com a intenção de aumentar a dificuldade da experiência, ou auxílios, que buscam diminuí-la. Vemos que a primeira opção é a mais comum, o que é refletido na ampla utilização de *bots* como inimigos do jogador.

Se olharmos historicamente, talvez o primeiro jogo no qual temos inimigos com algum nível de inteligência atuando contra o jogador é **Pac-Man** [Bandai Namco, 1980]. Nele, os fantasmas correndo atrás do *Pac-Man* eram programados com uma Máquina de Estados, uma técnica extremamente simples e comum, que ainda é usada até hoje na indústria de jogos (MILLINGTON e FUNGE, 2009).

Porém, o modelo de Máquina de Estado não apresenta robustez o suficiente para implementarmos todo tipo de comportamento desejado para as personagens. Há contextos que exigiriam um número muito grande de estados, com uma quantidade exponencial de transições entre eles, o que significa um grande consumo de memória e de processamento. Dessa forma, outras técnicas de IA para jogos foram sendo desenvolvidas ao longo dos anos, incluindo a que discutiremos a seguir.

---

[1] Para saber mais sobre a definição de *bot* no contexto da indústria de jogos, visite: https://www.techopedia.com/definition/10459/bot-software-robot

Sistemas Baseados em Regras

Um Sistema Baseado em Regras (SBR) é em um tipo de sistema especialista, o que é definido na área de Inteligência Artificial como um *software* que visa emular a inteligência de um especialista em um determinado domínio, por meio da representação e utilização de conhecimentos fornecidos manualmente pelos projetistas, ou inferidos pelo sistema. Embora não seja a abordagem mais popular em desenvolvimento de jogos atualmente, sistemas especialistas são utilizados há quase três décadas nessa área (MILLINGTON e FUNGE, 2009).

A estrutura básica de um SBR é dada por um banco de dados contendo fatos relevantes, um conjunto de regras “se-então” e um Árbitro, para resolver conflitos entre regras. Tendo isso, um ciclo básico de funcionamento do sistema é como se segue: fazemos uma varredura periódica para detectar se os fatos conhecidos satisfazem a condição (parte “se”) de alguma regra; coletamos todas as regras com as quais isso ocorre; pedimos para o Árbitro decidir qual delas será acionada e executamos a parte “então” (ou ações) da escolhida.

Com isso, vemos que existem três principais desafios ao se implementar um sistema especialista desse tipo: **i)** como representar a estrutura de dados que compõe as regras; **ii)** como avaliar se os fatos satisfazem a condição de alguma regra, o que é especialmente complexo quando temos *variáveis* na definição das condições e **iii)** como fazer a arbitragem entre as regras satisfeitas, para escolher qual delas será acionada.

Superando-se esses desafios, a principal vantagem de se usar regras declarativas para definir a IA em um jogo é a transparência, praticidade e velocidade com as quais é possível definir o comportamento de uma entidade. Escrever as regras segue uma lógica muito próxima de como as descreveríamos em linguagem natural (“se esta condição for verdade, então estas ações ocorrem”) e sua estrutura modular nos permite realizar modificações manuais de forma simples, mudando completamente a estratégia de uma personagem com apenas algumas alterações em suas regras (CALIMERI *et al.*, 2018; BOURG e SEEMANN, 2014).

Porém, os SBR também apresentam uma desvantagem relevante: sua dificuldade de implementação, algo que gerou uma reputação negativa na indústria de jogos e desmotiva desenvolvedores a usarem essa técnica (MILLINGTON e FUNGE, 2009). Sua complexidade está associada aos desafios supracitados, que

trazem a necessidade de um alto grau de abstração e da manipulação interna de formalismos declarativos, sendo esse último tópico algo que está além da formação de um típico desenvolvedor de jogos.

Consequentemente, essa complexidade de implementação ajuda a explicar a raridade com que vemos esses sistemas implantados em jogos; porém, isso não nos impede de encontrar casos interessantes de aplicação, como os que são mencionados a seguir.

Exemplos de SBR em jogos

Na indústria de *games*, mesmo com o uso de Sistemas Baseados em Regras não sendo tão comum, há casos interessantes nos quais eles são utilizados para dar ao jogador a liberdade de programar o comportamento de seus companheiros — quando esses não são controlados por ele. Vale notar que enquanto o uso mais comum de IA em jogos é definir as ações dos inimigos, os SBR aparecem aqui como uma opção para os aliados, mas é fácil ver que a mesma tecnologia poderia ser usada pelos desenvolvedores para definir os inimigos do jogador.

Um ponto de interesse nesses exemplos é o fato da declaração de regras ser considerada intuitiva o suficiente para ser deixada nas mãos dos jogadores, um público que de maneira geral não possui experiência em programação. Isso nos dá indícios de que a dificuldade de implementação não é transferida para a usabilidade desses sistemas, já que manipular as declarações é complexo no nível de desenvolvimento, mas apenas as descrever é considerado fácil para o usuário.

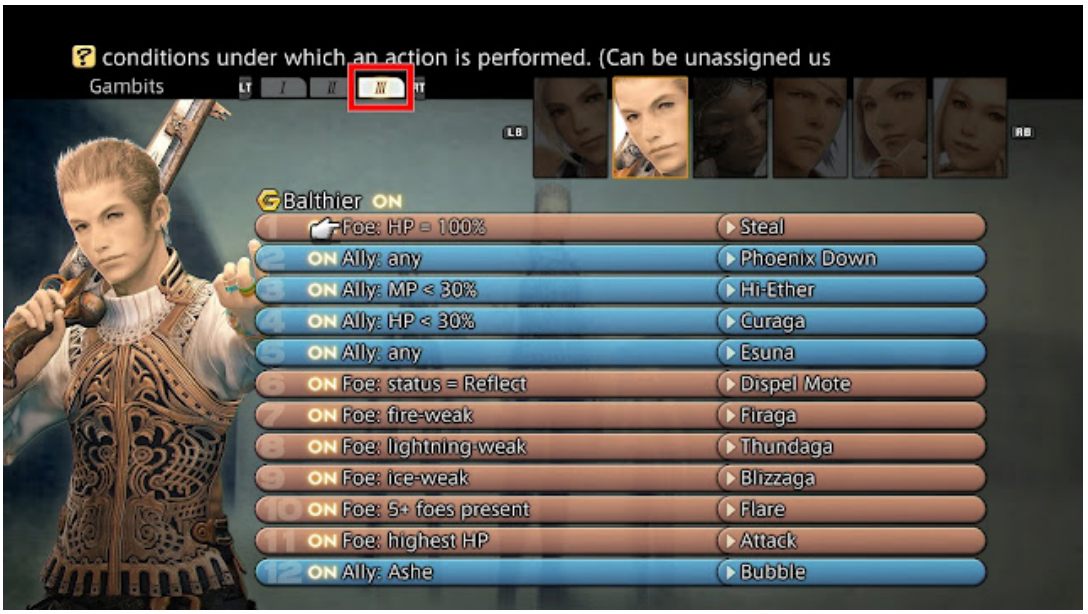
Um exemplo dessa aplicação está na série de jogos *Dragon Age*, especificamente em **Dragon Age: Origins** [Electronic Arts, 2009] e **Dragon Age II** [Electronic Arts, 2011], nos quais há o menu de “Táticas”, que pode ser visualizado na Figura 1 abaixo:

Figura 1 - Menu de Táticas em Dragon Age II

Fonte: Dragon Age Wiki.
Disponível em: https://dragonage.fandom.com/wiki/Tactics_(Dragon_Age_II)
Acesso em: 21 abr. 2023

Outro exemplo de uso dessa mecânica é o sistema de “artimanhas” (*gambits*) em **Final Fantasy XII** [Square Enix, 2006] e em seu *remake* **Final Fantasy XII: The Zodiac Age** [Square Enix, 2017], o qual pode ser visto no menu da Figura 2 abaixo:

Figura 2 - Menu de *Gambits* em Final Fantasy XII: The Zodiac Age



Fonte: Vinícius Rutes em Nintendo Blast.
Disponível em: https://www.nintendoblast.com.br/2019/05/analise-final-fantasy-xii-zodiac-age.html
Acesso em: 21 abr. 2023

Em ambos os jogos, são utilizadas regras definidas por uma condição à esquerda e a ação a ser tomada à direita, sendo que as condições são relacionadas à própria personagem, aos outros aliados ou aos inimigos e as ações geralmente representam habilidades da personagem, ou o uso de itens. Também notamos que as regras estão ordenadas de cima para baixo, pois nesses sistemas a arbitragem é feita com o princípio de prioridade: aquelas declaradas mais para cima na lista têm prioridade de serem acionadas.

Como podemos ver por esses exemplos, existem casos em que é interessante incluir um SBR na produção de um *videogame*, então seria útil haver mais ferramentas que auxiliassem na implementação desses sistemas.

Objetivos do trabalho

Portanto, o principal objetivo do trabalho é criar uma ferramenta que forneça aos desenvolvedores de jogos o alicerce de um SBR, facilitando sua implementação em projetos específicos. Isso será feito por meio de um arcabouço (ou *framework*), que disponibilizará: a estrutura de dados para representar as regras; um sistema de inferência e unificação, para verificar quais condições são satisfeitas em um dado

momento; e um ou mais Árbitros, já implementados — resolvendo assim os principais desafios apontados na seção “**Sistemas Baseados em Regras**”.

O foco dessa ferramenta será auxiliar equipes de desenvolvedores de jogos, as quais são multidisciplinares e reúnem pessoas de formações distintas, a definirem a inteligência das personagens em seu jogo. Como essa tarefa costuma ser responsabilidade de um *game designer*, alguém que não possui necessariamente experiência em programação, é relevante que a ferramenta possua uma interface gráfica que permita a edição das regras de maneira fácil. Isso porque os exemplos dados sugerem que esse tipo de interface permite que pessoas sem conhecimento técnico construam comportamentos interessantes.

Outro objetivo é tornar a tecnologia de SBR o mais acessível e transparente possível, então será disponibilizado o código-fonte do *framework* para que outros possam estudá-lo e entender como o sistema foi implementado. O principal fator que influenciou essa decisão foi o Movimento *Software* Livre[2], que acredita que o compartilhamento de código é benéfico para o desenvolvimento de *software* como um todo, portanto, será dada prioridade por tecnologias de código aberto na hora de desenvolver o projeto.

Por fim, além dos objetivos práticos há também o estudo de conhecimentos teóricos. Em especial, será analisada uma técnica de IA pouco abordada em outros contextos, algo que irá permear todo o trabalho e espera-se que traga experiência relevante para a formação de um especialista em inteligência artificial. A outra vertente teórica-prática a ser estudada é o processo de construção de *softwares* para desenvolvimento de jogos, o que será feito conforme a metodologia explicada a seguir e permitirá um contato mais próximo com o sistema de um motor de jogos.

Metodologia

Para se cumprir tais objetivos, o projeto que será desenvolvido é a construção de um *plugin* para a Godot *game engine*[3] que implemente um Sistema Baseado em Regras genérico e com interface gráfica. Esse *plugin* irá adicionar ao motor de jogos os componentes mencionados anteriormente, por meio de novos tipos de *nós* e

[2] Para saber mais sobre o Movimento *Software* Livre, visite: https://www.gnu.org/philosophy/free-software-intro.pt-br.html

[3] Site oficial do motor de jogos Godot: https://godotengine.org/

cenas e da inclusão de *scripts* especiais, assim construindo um arcabouço que poderá ser usado de maneira genérica em vários jogos.

Dentre os motivos da escolha pela Godot, podemos citar que seu repositório[4] é o número um no GitHub dentre *engines open-source*, sendo talvez a opção de código livre mais conhecida atualmente. Além disso, os seus *plugins* são desenvolvidos utilizando o próprio sistema da *engine*, algo que facilita tanto a implementação, quanto o entendimento do usuário que os importa em seu projeto.

Em decorrência dessa escolha, o projeto irá possuir a mesma licença MIT[5] presente no motor de jogos Godot, para manter compatibilidade e garantir o princípio de código livre adotado. Além desses motivos, temos que é necessário haver uma licença para submeter o *plugin* na biblioteca oficial de recursos, a *Godot Asset Library*[6], algo que será uma meta para a etapa final do trabalho.

Porém, antes de se começar a escrever código será feita uma pesquisa por material acadêmico e exemplos práticos da aplicação de Sistemas Baseados em Regras em jogos, fazendo assim um levantamento bibliográfico de referências e inspirações relevantes. Esse estudo irá se intercalar com o desenvolvimento do *software* em etapas posteriores do trabalho, mas é essencial haver um entendimento básico do assunto antes de se começar a programar.

Tendo uma boa noção da estrutura do sistema, será iniciado o desenvolvimento do *plugin*, seguindo um modelo inspirado em métodos ágeis[7] e com entregas parciais, sendo as principais versões: *Alpha*, quando houver usabilidade mínima, o que no nosso caso já irá incluir uma interface gráfica simples; *Beta*, quando todas as funcionalidades já tiverem sido incluídas, mesmo que com problemas; e versão de produção 1.0.0, quando o sistema estiver estável e teoricamente pronto para ser lançado na *Asset Library*.

Ademais, ao longo do desenvolvimento será criado um ambiente de testes, semelhante a um *videogame* muito simples de uma cena, para emular o uso de um usuário comum do *plugin*, ou seja, um desenvolvedor de jogos que utiliza Godot. Serão realizados testes de unidade no sistema de inferência e unificação, para verificar se ele detecta que as condições certas estão sendo satisfeitas, e também

---

[4] Acesse o repositório da Godot em: https://github.com/godotengine/godot
[5] Licença MIT disponível em: https://opensource.org/license/mit/
[6] Visite a Godot Asset Library em: https://godotengine.org/asset-library/asset
[7] Leia mais sobre métodos ágeis em: https://agilemanifesto.org/iso/ptbr/manifesto.html

nos Árbitros implementados, averiguando que eles estão acionando a regra correta de acordo com seu critério específico.

Por fim, haverá testes de integração nos quais será feita a simulação de uma pessoa comum jogando a cena criada, com o intuito de analisar se o comportamento das entidades automatizadas é interessante no contexto de um jogo. Esse tipo de teste será subjetivo e irá contar com a experiência prévia do desenvolvedor com inúmeros *videogames*.

Com isso, será feita uma avaliação contínua do progresso do sistema, utilizando-se os testes para detectar problemas no código (e assim eliminá-los) e os materiais de apoio para confirmar que todas as funcionalidades relevantes de um SBR foram incluídas. Essa natureza contínua de desenvolvimento e avaliação é refletida no cronograma, que está incluso a seguir.

Cronograma

Serão realizadas reuniões quinzenais às sextas-feiras, para que seja avaliado o progresso feito durante as últimas duas semanas, além de se planejar o que será feito até o próximo encontro. A primeira reunião se deu dia 31 de março de 2023 e o último dia foi delimitado como 15 de dezembro, então foram calculadas 36 semanas de trabalho, marcadas de sexta a sexta.

As datas importantes para o cronograma estão na Tabela 1 abaixo:

Tabela 1 - Cronograma inicial do projeto

31 de março	**Reunião inicial**
31 de março a 30 de abril	Procurar e estudar material bibliográfico, escrever a proposta
30 de abril	**Entrega desta proposta**
30 de abril a 26 de maio	Alternar entre desenvolver o *plugin* e procurar referências bibliográficas
26 de maio	**Término do levantamento bibliográfico**
26 de maio a 14 de julho	Focar no desenvolvimento do *plugin*
14 de julho	**Entrega da versão Alpha do *plugin***
14 de julho a 01 de setembro	Alternar entre desenvolver o *plugin* e estudar o material levantado

01 de setembro	**Entrega da versão Beta do *plugin* e início da escrita da monografia (junto da montagem da apresentação e pôster)**
01 de setembro a 03 de novembro	Alternar entre escrever a monografia e desenvolver o *plugin*
03 de novembro	**Lançamento da versão 1.0.0 do *plugin* na *Asset Library* da Godot (se for possível)**
03 de novembro a 01 de dezembro	Focar na escrita da monografia (junto da montagem da apresentação e pôster)
01 de dezembro	**Entrega da monografia para o orientador**
01 a 15 de dezembro	Revisar e finalizar a monografia, apresentação e o pôster
15 de dezembro	**Finalização do Trabalho de Conclusão de Curso**

Bibliografia

- MILLINGTON, Ian e FUNGE, John David. **Artificial Intelligence for Games**. 2. ed. Burlington: Morgan Kaufmann, 2009.
- CALIMERI, Francesco *et al.* Integrating Rule-Based AI Tools into Mainstream Game Development. **Rules and Reasoning — Second International Joint Conference**, Luxembourg, RuleML+RR, p. 310–317, September, 2018.
- BOURG, David M. e SEEMANN, Glenn. **AI for Game Developers**. O'Reilly, 2004.