Agente inteligente Aplicado em Jogos Digitais

Patrick P. Rodrigues[1], **João L.T. da Silva**[1]

[1]Centro Universitário e Faculdade de Tecnologia (UNIFTEC)
95012-669 – Caxias do Sul – RS – Brasil

patrickprodrigues@hotmail.com, joaoluis.tavares@gmail.com

Abstract. *Currently the area of Digital Games tends to offer more and more realism and immersiveness, promoting a better experience to the player. Along with this the player expects an increasingly natural behavior of NPCs (Non-Player Character), looking for a more natural adaptation of the NPC's interactions in the context of the game. In this sense, Artificial Intelligence, in its area of Multiagent Systems, offers a promising technology to implement intelligent cognitive NPCs. However, specific features in game engines such as real-time, synchronization and communication aspects are not easily supported in terms of multiagent platforms, making it difficult to fully integrate AI techniques in this context. On the other hand, the agent autonomy feature may offer benefits for more convincing gameplay, and even for serious games, for instance. In this article, we focus on modeling a decisional component in an NPC agent to define an intelligent behavior of the NPC, which is adaptive according to the specific playing of each player. We intend, with these initial ideas and prototypes, to experiment the concept of intelligent agents applied to the area of digital games, using an artificial neural network as a decisional component.*

Resumo. *Atualmente a área de Jogos Digitais procura oferecer cada vez mais realismo e imersividade, promovendo uma melhor experiência ao jogador. E o jogador espera um comportamento cada vez mais natural dos NPCs (Non-Player Character), procurando uma adaptação mais natural das interações com o contexto do jogo. Neste sentido, a Inteligência Artificial, na sua área de Sistemas Multiagentes, oferece uma tecnologia promissora para implementar NPCs cognitivos inteligentes. No entanto, características específicas em motores de jogos, como aspectos de tempo real, sincronização e comunicação não são facilmente compatíveis em termos de plataformas multiagentes, dificultando a completa integração de técnicas de IA neste contexto. Por outro lado, a característica de autonomia dos agentes pode oferecer benefícios para uma jogabilidade mais convincente, sendo até necessária para jogos sérios, por exemplo. Neste artigo, vamos nos concentrar na modelagem de um componente decisional em um agente NPC para definir um comportamento inteligente do NPC, que seja adaptativo de acordo com a maneira de jogar específica de cada jogador. Esperamos, com estas ideias e protótipo iniciais, experimentar o conceito dos agentes inteligentes aplicado à área de jogos digitais, utilizando uma rede neural artificial como componente decisional.*

1. Introdução

Atualmente, os jogos digitais estão presentes na vida de muitas pessoas, sendo excelentes formas de entretenimento, possuindo as mais variadas temáticas e mecânicas. Indepen-

dente do estilo, normalmente a Inteligência Artificial (IA) é um componente presente em todos os gêneros. Seja ela aplicada no controle dos NPCs (*Non-Player Character*) ou nas dinâmicas contidas no cenário, seu uso possui uma alta relevância na experiência final proporcionada pelo jogo.

Desenvolver um componente de IA para um jogo pode representar um desafio, visto que a uma certa complexidade para que o produto final proporcione o desafio correto aos jogadores, pois cada um tem seu próprio estilo de jogo e a IA deve conseguir lidar com pelo menos a maioria desses perfis. Se a IA não for desenvolvida de uma forma satisfatória, afim de proporcionar o equilíbrio ideal, os jogadores podem acabar perdendo o interesse no jogo. Existem algumas vertentes que buscam maneiras de solucionar esse tipo de problema, como é o caso do trabalho em [Fogel et al. 2004] o qual visa desenvolver comportamentos adaptativos de vários NPCs através de abordagem evolutiva, aproximando de um comportamento mais humano. [Miikkulainen 2006] também usa algoritmos genéticos para a geração dos parâmetros e pesos de uma rede neural, para a adaptação sem um alvo explicito.

Este trabalho apresenta um conceito de como melhorar a experiência do jogador ao interagir com NPCs, utilizando agente inteligente adaptativo, com uma Rede Neural Artificial (RNA) como componente decisional do agente. Consideramos aqui, que um agente inteligente é uma entidade contínua e autônoma capaz de atuar em um determinado ambiente, tomando suas próprias decisões [Leyton-Brown and Shoham 2008]. Com o uso da RNA esse agente pode adquirir a capacidade de aprender com a sua experiência de jogo, assim como um jogador humano faz e dessa forma oferecer ao *player* uma experiência diversificada e mais imersiva.

2. Projeto de RNA para um Agente adaptativo de jogo

Para demonstração desse projeto foi desenvolvido um jogo onde o jogador possui dois comandos. Os comandos são representados por *0* e *1*, em um deles o jogador dispara um míssil na posição *0* e no outro na posição *1*, o papel da RNA é o de fazer o agente inteligente (*bot*) prever qual será a próxima jogada realizada pelo jogador, a partir dos comandos utilizados anteriormente. A previsão é exibida ao jogador de forma visual, com um objeto aparecendo na posição que o *bot* previu para a jogada seguinte. Os componentes utilizados no jogo podem ser vistos na Figura 1.

Figura 1. Protótipo do jogo

3. Protótipo

Para realização do projeto, foi desenvolvido um protótipo com três agentes com funções específicas: *Engine*, *API* e *RNA*. Essa divisão possibilita a comunicação entre as diferentes abordagens. A divisão é necessária pois os diferentes agentes possuem diferentes tecnologias e diferentes características, cada uma com seu propósito, necessitando assim de uma *API* para comunicação.

3.1. Decisão Adaptativa

O módulo de rede neural utiliza dois modos: o modo de treinamento (*training*), onde recebe o log em *JSON* gerado pela *engine* através da *API*. O treinamento realiza todas as interações necessárias visando minimizar os erros e ajustar seus pesos sinápticos. O modo de previsão (*prediction*), recebe apenas as entradas e tenta prever a saída esperada com os conhecimentos adquiridos no modo de treinamento. A estrutura do *JSON* (figura 2) contém quatro entradas para uma saída onde o arquivo de treinamento é a lista de golpes que o jogador executou durante o jogo, separados em grupos de cinco. Os primeiros quatro ataques são considerados como entradas e o último, a saída. Usando esses dados a RNA é capaz de treinar para prever o quinto ataque do próximo grupo de ataques que será executado pelo jogador.

```
[
  {
    "inputs": [[0, 0, 1, 1],
               [0, 1, 1, 0],
               [1, 0, 1, 0],
               [1, 1, 1, 1]],
    "outputs": [[0],
                [1],
                [1],
                [0]]
  }
]
```

Figura 2. Exemplo de JSON de treinamento

A RNA do agente é dividida em três camadas: a camada *l0*, contendo as representações dos comandos, que podem ser *0* ou *1*. A segunda camada, *l1*, onde ficam os pesos sinápticos da primeira camada. Após cada interação, a camada *l2* recebe as entradas *l0* multiplicadas pelos pesos sinápticos *syn0*. Após, o resultado em *l2* é multiplicado pelo valor correspondente a sua sinapse de saída *syn1*, e esses valores passam pela função de ativação sigmoide, gerando o *output*. Na etapa de treinamento é calculado o erro dessa saída em comparação com as saídas fornecidas pelo usuário. A estrutura básica da RNA, pode ser vista na figura 3

Após todo ciclo e suas repetições, o agente é capaz de atingir um valor próximo ao ideal para seus pesos sinápticos. A Figura 4 ilustra as sinapses (*syn0* e *syn1*) sendo inicializadas com valores aleatórios, a multiplicação das entradas pelos valores contidos nas sinapses e os cálculos visando minimizar o erro. No final de todo processo os valores contidos nas sinapses são salvos em JSON e estão prontos para serem utilizadas pelo método de previsão.

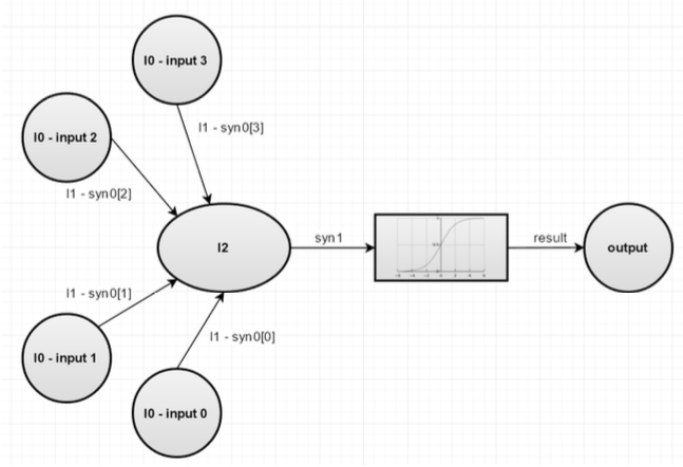

Figura 3. Estrutura da RNA


```
[
  {
    "syn0": [[2.1527945143164984, 1.3328690694820238, -2.4662028383481283, -2.824410357116697, -0.7805288111175995],
            [-2.85059313369365, -2.14888078981325, -0.13428335506776606, 3.2653073976903024, 0.14077593354945456],
            [-0.8757825358301803, -0.3943343828932792, 0.9457808716655521, 1.3036759132368, -1.0350996747507961],
            [5.136615479835274, 3.505712793504951, -2.4098894988745876, -6.210780982518584, -0.4371286576833272]],
    "syn1": [[6.595451483743302],
             [4.2945636133835166],
             [-3.8599390791562476],
             [-8.676074066650129],
             [-0.5614033160011409]]
  }
]
```


Figura 4. Exemplo de JSON das sinapses

No modo de previsão, o agente captura apenas quatro ataques feitos pelo usuário e envia-os para a API que realiza a chamada do método de previsão na RNA. O método de previsão recebe as quatro entradas do usuário e as sinapses que foram calculadas anteriormente no método de treinamento, onde são multiplicados pelos pesos sinápticos e o resultado obtido será a previsão da quinta jogada do usuário. Esse resultado será então enviado para o agente executor (*engine*), que será responsável por executar a ação correspondente de defesa do *bot*. Tudo isso acontece em tempo de jogo.

3.2. Testes e resultados

Os experimentos dividem-se em teste de conceito e teste de ambiguidade, considerando a ordem de execução dos padrões, suas entradas e saídas, valor previsto pela RNA e erro de treinamento do padrão da jogada. A previsão pode ser considerada correta quando o valor da saída coincide com o valor previsto pela RNA.

3.2.1. Teste de conceito

Para o teste de conceito do sistema, foram executadas dez rodadas, com diversos padrões sem ambiguidade, onde a RNA apresentou um bom desempenho, mesmo com um número relativamente baixo de execuções. Na Tabela 5, observa-se que 80% dos padrões foram identificados corretamente. Quanto mais acontece a repetição de um padrão, mais

o agente infere a previsão da saída do padrão, fazendo assim com que seu erro de treinamento diminua, mesmo que esse padrão passe por uma futura ambiguidade seu valor de previsão será mantido, até que o padrão com a nova saída ultrapasse as repetições do padrão antigo.

Execução	Entradas	Saídas	Previsto	Erro	Resultado
1	0,1,0,1	1	0	1,44%	Erro
2	0,1,0,0	0	1	2,42%	Erro
3	0,1,0,1	1	1	1,89%	Acerto
4	0,1,1,1	1	1	1,53%	Acerto
5	0,0,0,0	0	0	1,96%	Acerto
6	0,1,1,1	1	1	1,71%	Acerto
7	0,1,0,0	0	0	1,52%	Acerto
8	0,1,1,1	1	1	1,40%	Acerto
9	0,0,0,0	0	0	1,38%	Acerto
10	1,0,0,0	0	0	1,27%	Acerto

Figura 5. Teste de conceito

3.2.2. Teste de ambiguidade

Com o intuito de demonstrar a capacidade de adaptação do agente inteligente, realizou-se um teste com padrões ambíguos, onde na primeira e segunda execução foram executados o padrão *(0,1,0,1)* com a saída sendo definida como *(0)*, para as demais jogadas o mesmo padrão *(0,1,0,1)* foi utilizado, porém com a saída alterada para *(1)*. A Tabela 6 ilustra a assimilação da modificação do padrão em três execuções. Mesmo com erro de 44,44%, devido as ambiguidades causadas pelas saídas anteriores, o agente conseguiu assimilar a mudança, prevendo corretamente a nova saída esperada na sexta execução.

Execução	Entradas	Saídas	Previsto	Erro	Resultado
1	0,1,0,1	0	0	1,87%	Acerto
2	0,1,0,1	0	0	1,56%	Acerto
3	0,1,0,1	1	0	44,44%	Erro
4	0,1,0,1	1	0	50%	Erro
5	0,1,0,1	1	0	48%	Erro
6	0,1,0,1	1	1	44,44%	Acerto

Figura 6. Teste de ambiguidade

4. Discussão e Conclusão

Existem diversas técnicas para o desenvolvimento de agentes inteligentes capazes de atuar em um ambiente dinâmico. Na área de jogos uma das técnicas é a *dynamic scripting*, que utiliza um conjunto de regras e aprendizagem não supervisionada para adaptar o comportamento do *bot* ao contexto do jogo de forma rápida e eficiente. Esse tipo de abordagem facilita o contexto da implementação, pois é um método mais ágil de processamento, entretanto a ação da IA, embora adaptativa fica restrita a um conjunto de regras de base

[Spronck et al. 2003]. Métodos supervisionados de aprendizagem como as RNAs, normalmente não são utilizados em jogos, devido a demanda na performance de processamento, pois os requisitos de tempo são difíceis de atingir. Entretanto, o uso desse tipo de técnica sob certos critérios, como quantidade controlada de variáveis, pode ser utilizado em jogos, adequando-se ao requisito de tempo de processamento rápido necessário para uma jogabilidade fluida. Também pode proporcionar um certo nível de adaptação ao agente, já que não necessita seguir um conjunto pré concebido de regras, como o *dynamic scripting*, podendo ser baseada exclusivamente nos resultados gerados através da interação dos jogadores.

Técnicas de aprendizagem supervisionada podem ser utilizadas também em jogos tradicionais, sendo capazes também de criar um jogador "perfeito", como no caso do *Google Go*, uma IA capaz de jogar Go vencendo grandes jogadores humanos. Para isso a equipe de desenvolvimento do Google utilizou redes neurais, responsáveis por avaliar as posições e a possível eficácia dos movimentos realizados no tabuleiro. Além disso foi utilizado o método de Monte Carlo, para realizar as simulações necessárias das milhares de jogadas possíveis de ocorrem no jogo [Silver et al. 2016].

A utilização de uma RNA como componente decisional no agente inteligente, no presente trabalho, apresentou resultados que demonstram a adaptabilidade do agente, conseguindo aprender a maneira com que o jogador interage com o sistema. Isso faz com que o jogo ofereça uma experiência diferenciada, capaz de reconhecer a maneira de jogar de cada um. Porém as técnicas utilizadas demandam um certo poder de processamento, algo que não é muito evidente em um jogo simples, como no caso do desenvolvido para o projeto. Por isso essa metodologia deve ser utilizada preferencialmente em jogos com um número controlado de possibilidades, dessa forma pode ser apresentado um diferencial ao jogo, melhorando a jogabilidade e a diversão proporcionada, conseguindo assim despertar maior interesse por parte dos jogadores.

Referências

Fogel, D. B., Hays, T. J., and Johnson, D. R. (2004). A platform for evolving characters in competitive games. In *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No.04TH8753)*, volume 2, pages 1420–1426 Vol.2.

Leyton-Brown, K. and Shoham, Y. (2008). *Essentials of Game Theory: A Concise, Multidisciplinary Introduction*. Morgan and Claypool Publishers, 1st edition.

Miikkulainen, R. (2006). Creating intelligent agents in games. *The Bridge*, pages 5–13.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503.

Spronck, P., Sprinkhuizen-kuyper, I., and Postma, E. (2003). Online adaptation of game opponent ai in simulation and in practice. In *In Proceedings of the Fourth International Conference on Intelligent Games and Simulation*, pages 93–100.