UNIVERSIDADE FEDERAL DOS VALES DO JEQUITINHONHA E MUCURI

FACULDADE DE CIÊNCIAS EXATAS E TECNOLÓGICAS

CURSO DE SISTEMAS DE INFORMAÇÃO

APRIMORAMENTO DA TÉCNICA DE DYNAMIC SCRIPTING PARA IA ADAPTATIVA DE JOGOS COM UM ALGORITMO DE SUBSTITUIÇÃO DE TÁTICAS

LUCAS IZUMI DE OLIVEIRA

Trabalho de Conclusão de Curso

Diamantina/MG

2017

LUCAS IZUMI DE OLIVEIRA

APRIMORAMENTO DA TÉCNICA DE DYNAMIC SCRIPTING PARA IA ADAPTATIVA DE JOGOS COM UM ALGORITMO DE SUBSTITUIÇÃO DE TÁTICAS

Trabalho apresentado ao curso de Sistemas de Informação da Universidade Federal dos Vales do Jequitinhonha e Mucuri – UFVJM, como pré-requisito para obtenção do grau de bacharel, sob orientação do Prof. Dr. Alessandro Vivas Andrade.

Diamantina/MG

2017

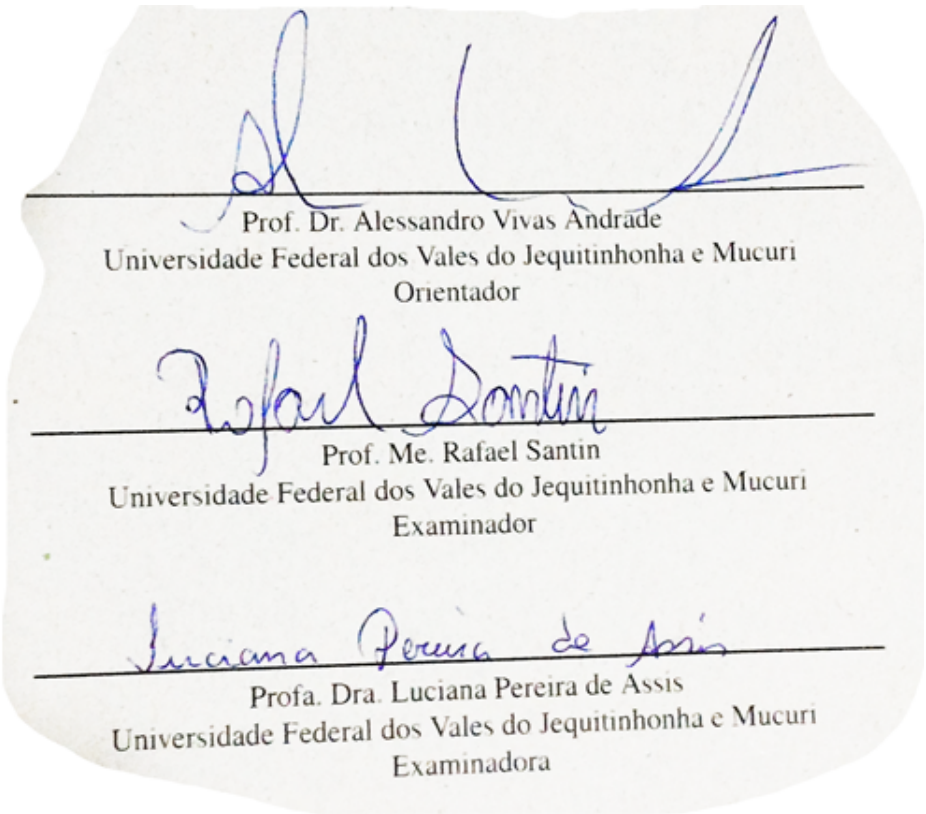
LUCAS IZUMI DE OLIVEIRA

APRIMORAMENTO DA TÉCNICA DE DYNAMIC SCRIPTING PARA IA ADAPTATIVA DE JOGOS COM UM ALGORITMO DE SUBSTITUIÇÃO DE TÁTICAS

Trabalho apresentado ao curso de Sistemas de Informação da Universidade Federal dos Vales do Jequitinhonha e Mucuri – UFVJM, como pré-requisito para obtenção do grau de bacharel, sob orientação do Prof. Dr. Alessandro Vivas Andrade.

COMISSÃO EXAMINADORA

DIAMANTINA, **3 DE MARÇO DE 2017**



Prof. Dr. Alessandro Vivas Andrade
Universidade Federal dos Vales do Jequitinhonha e Mucuri
Orientador

Prof. Me. Rafael Santin
Universidade Federal dos Vales do Jequitinhonha e Mucuri
Examinador

Profa. Dra. Luciana Pereira de Assis
Universidade Federal dos Vales do Jequitinhonha e Mucuri
Examinadora

À todos os familiares e amigos que me deram suporte durante essa jornada.

AGRADECIMENTOS

Aos meus pais, Jaqueline e Edmilson, pelo apoio e suporte oferecidos durante esta jornada. Ao meu padastro Amauri por sua disponibilidade e ajuda. Ao professor Claus Aranha pela orientação e motivação a estudar este tema. Aos professores Cristiano Grijó Pitangui e Alessandro Vivas Andrade pela orientação, paciência e compartilhamento de conhecimento. Aos meus amigos Gerri de Maio Faustino, Raiany Geovana Fernandes, Fernanda Macedo e Natália Leal por todos os anos de amizade e companheirismo.

“We all make choices, but in the end,
our choices makes us.”
(ANDREW RYAN)

OLIVEIRA, Lucas Izumi. **Aprimoramento da Técnica de Dynamic Scripting para IA Adaptativa de Jogos com um Algoritmo de Substituição de Táticas.** Diamantina, 2017. 53. Trabalho de Conclusão de Curso Superior em Sistemas de Informação. Faculdade de Ciências Exatas, Universidade Federal dos Vales do Jequitinhonha e Mucuri, Diamantina, 2017.

RESUMO

A Inteligência Artificial (IA) possui um importante papel nos jogos nos dias de hoje. Com os jogadores se tornando cada vez mais exigentes, é vital que se forneça uma IA que os desafie e os entretenha. O uso da Inteligência Artificial Adaptativa (IAA) mostrou potencial para se adaptar a cada jogador, aprendendo suas técnicas e oferecendo um desafio consistente. Este trabalho consiste na análise de uma técnica de IAA conhecida como Dynamic Scripting e no desenvolvimento de um novo algoritmo (chamado de Substituição de Tática) para melhorá-lo. A análise do método proposto é feita mediante a aplicação da técnica no jogo Neverwinter Nights (NWN), onde times de agentes lutam entre si. Embora a validação dos resultados seja feita no jogo Neverwinter Nights, o algoritmo pode ser aplicado a qualquer jogo que se encaixe nos requisitos do Dynamic Scripting e que possa ser dividido em estados. Os resultados mostram que o algoritmo de substituição tática alcançou uma redução de tempo de $\approx 50\%$ para alcançar a convergência. Além disso, foi capaz de reduzir em 40% o número médio de rodadas para alcançar a convergência. Ao melhorar a eficiência e a eficácia do Dynamic Scripting, criamos agentes "mais realistas", que aprendem e se adaptam mais rapidamente ao jogador e ajudam a aumentar a capacidade de imersão do jogo.

Palavras-chave: Dynamic Scripting. Inteligência Artificial. Aprendizado de Máquina.

OLIVEIRA, Lucas Izumi. **Aprimoramento da Técnica de Dynamic Scripting para IA Adaptativa de Jogos com um Algoritmo de Substituição de Táticas.** Diamantina, 2017. 53. Trabalho de Conclusão de Curso Superior em Faculdade de Ciências Exatas, Universidade Federal dos Vales do Jequitinhonha e Mucuri, Diamantina, 2017.

ABSTRACT

Artificial Intelligence (AI) plays an important role in games nowadays. With players becoming increasingly demanding, it is vital to provide an AI that challenges and entertains them. The use of Adaptive Artificial Intelligence (AAI) has shown potential to adapt to each player by learning their techniques and offering a consistent challenge. This work consists on the analysis of an AAI technique known as Dynamic Scripting and in the development of a new algorithm (called Tatic Replacement) to improve it. The analysis of the proposed method is made upon the application of the technique in the game Neverwinter Nights (NWN), where teams of agents battle against each other. Although the validation of results is made upon the Neverwinter Nights game, the algorithm can be applied to any game that fits in Dynamic Scripting requirements and that can be divided into states. Results show that the Tactic Replacement algorithm achieved a time reduction of $\approx 50\%$ to achieve convergence. Also, it was able to reduce by 40% the average number of rounds to reach the convergence. By improving the efficiency and effectiveness of Dynamic Scripting, we create "more realistic" agents that learn and adapt more quickly to the player and helps to increase the immersion-ability of the game.

Key-words: Dynamic Scripting. Artificial Intelligence. Machine Learning.

LISTA DE ILUSTRAÇÕES

Figura 1 – ***Pong*** 13
Figura 2 – ***Funcionamento do Aprendizado por Reforço*** 21
Figura 3 – ***Tela do jogo Neverwinter Nights*** 24
Figura 4 – ***Diagrama de Espaços Não-Expandidos*** 27
Figura 5 – ***Algoritmo de Substituição de Tática*** 33
Figura 6 – ***Gráfico da Média de Tempo dos Estados Críticos*** 36
Figura 7 – ***Gráfico da Média de Tempo dos Estados Críticos (Pós-Corte)*** 38
Figura 8 – ***Gráfico da Média de Tempo dos Estados Críticos Expandidos (Pós-Corte)*** 39

LISTA DE TABELAS

Tabela 1 – Média de tempo gasto em cada estado não-expandido 28
Tabela 2 – Média de tempo gasto nos estados expandidos 29
Tabela 3 – Média de Tempo dos Estados Expandidos Pré-Algoritmo 30
Tabela 4 – Média de Tempo dos Algoritmos 35
Tabela 5 – Frequência de Visita aos Estados Críticos em 50 iterações 37
Tabela 6 – Média de Tempo dos Algoritmos (Pós-Corte) 37
Tabela 7 – Média de Tempo dos Algoritmos com Estados Críticos Expandidos (Pós-Corte) 39
Tabela 8 – Estatísticas por Combate 40

SUMÁRIO

1 INTRODUÇÃO 13
1.1 JUSTIFICATIVA 15
1.2 OBJETIVOS 15
1.2.1 Objetivo geral 15
1.2.2 Objetivos específicos 15
1.3 ESTRUTURA DO TRABALHO 16
2 FUNDAMENTAÇÃO TEÓRICA 17
2.1 CONCEITOS COMPUTACIONAIS 17
2.1.1 Agente Computacional 17
2.1.2 Scripts 17
2.1.3 Espaço de Estados 17
2.1.4 Heurística 18
2.2 INTELIGÊNCIA ARTIFICIAL EM JOGOS 18
2.3 ALGORITMOS DE APRENDIZAGEM PARA IA EM JOGOS 18
2.3.1 Inteligência Artificial Adaptativa 19
2.3.2 Aprendizado por Reforço 20
2.3.3 *Dynamic Scripting* 21
3 FERRAMENTAS UTILIZADAS 23
3.1 Equipamento Utilizado 23
3.2 Linguagens Utilizadas 23
3.2.1 NWScript 23
3.3 Ambiente de Desenvolvimento 23
3.3.1 Aurora Toolset 23
3.4 Ambiente de Simulação 24
3.4.1 Neverwinter Nights 24
4 SISTEMA DESENVOLVIDO 25
4.1 Análise do Dynamic Scripting 25
4.1.1 Divisão por Estados 26
4.1.2 Análise dos Estados 27
4.2 Desenvolvimento do Algoritmo 29
5 TESTES COMPUTACIONAIS 35
5.0.1 Validação dos Resultados 40
6 CONCLUSÃO 41
6.1 CONSIDERAÇÕES FINAIS 41

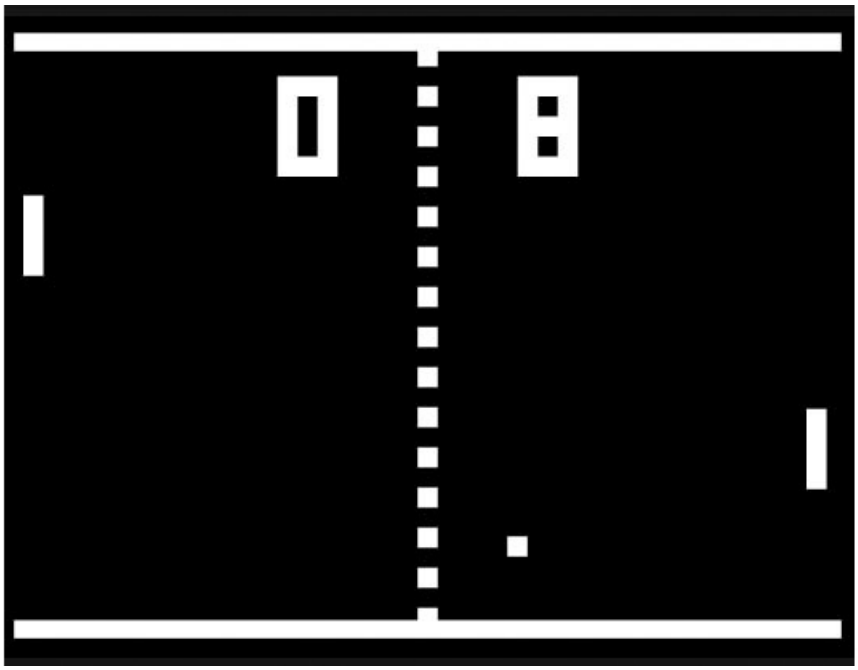
6.2 TRABALHOS FUTUROS 41
REFERÊNCIAS 43
APÊNDICE A Código fonte 47
A.1 Script *PseudoHeartbeat* 47

1 INTRODUÇÃO

A Inteligência Artificial (IA) é uma ciência que abrange vários setores. Criação de poesias, demonstração de teoremas matemáticos, diagnósticos de doenças, simuladores de mercado e sistemas lógicos são alguns dos diferentes campos que fazem uso da Inteligência Artificial. A IA faz a sistematização e a automação de tarefas, tornando-se relevante para qualquer esfera da atividade intelectual humana (RUSSELL; NORVIG, 2003). Dentre estas áreas de interesse, uma que vem utilizando de IA cada vez mais é a de jogos digitais.

A primeira ocorrência de uso de Inteligência Artificial em um jogo digital foi com o *Pong* (1972) (WEXLER, 2008). *Pong*, mostrado na Figura 1, era um jogo que simulava uma partida de tênis de mesa e consistia em duas barras, uma em cada lado da tela, e uma bola. O objetivo era bater na bola com as barras e fazê-la atravessar o campo do oponente.

Figura 1 – *Pong*



Fonte: (GAMEINFORMER, 2013)

A Inteligência Artificial utilizada no jogo controlava uma das barras e tentava sempre impedir o jogador de marcar um ponto, bloqueando a bola e a enviando de volta ao campo adversário. A IA determinava para onde mover a barra utilizando uma equação simples que calculava, com precisão, a que altura a bola cruzaria o seu campo e então movia a barra até aquele ponto, o mais rápido que lhe fosse permitido (WEXLER, 2008). Dependendo da dificuldade selecionada, a velocidade da IA era alterada, o que impactava se ela conseguiria ou não chegar no ponto determinado a tempo. Havia também a chance dela ir na direção errada com uma certa

probabilidade.

Por muito tempo a IA em jogos não era muito mais inteligente que a IA do jogo *Pong*. Isso se dava pela simplicidade dos jogos e pelo fato deles serem jogados contra um outro jogador humano, na maioria das vezes. Enquanto a complexidade dos jogos não aumentava, a IA consistia, meramente, de uma busca em uma tabela, indicando o que estava acontecendo e procurando pela ação mais apropriada a se tomar (WEXLER, 2008).

O uso de Inteligência Artificial nos jogos evoluiu conforme os jogos foram se tornando mais complexos. De acordo com (SHAFER, 2012), a criação de uma boa IA é extremamente difícil. Fazer a IA obedecer os critérios mínimos estabelecidos pelo projeto não é o suficiente. Mesmo que uma IA inteligente seja desenvolvida, no sentido de ela fazer as melhores escolhas dada uma situação, isso não significa que ela seja considerada uma boa IA. Para (SHAFER, 2012), tudo o que importa é o que o jogador vê e acredita. Essa sensação de credibilidade do mundo do jogo, transmitida ao jogador, está diretamente relacionada com a imersão que um jogo pode proporcionar.

Imersão é uma condição psicológica em que uma pessoa tem toda a atenção voltada para uma única atividade e seus sentimentos canalizados àquele ponto em particular. A imersão define o quão confortável e engajado um jogador se sente enquanto participa do contexto de um jogo. É o momento em que o jogador passa a se sentir "dentro" do jogo que lhe está sendo apresentado (BATEMAN, 2007). Para permanecer neste estado o jogador precisa de motivação, até que alcance o ponto de imersão total, onde o cérebro passa a acreditar que o que está sendo mostrado a ele é a nova realidade. (MIRVIS; CSIKSZENTMIHALYI; CSIKZENTMIHALY, 1991) define este estado de total imersão como *flow* (fluxo).

Para que seja possível alcançar este nível de imersão, os elementos do jogo precisam estar bem trabalhados, de forma que todos os aspectos do mundo do jogo sejam verdadeiros à natureza que supõem-se a representar. Um destes elementos é a Inteligência Artificial. Se um jogo possui quaisquer agentes de IA é preciso trabalhar cuidadosamente para que este elemento não se sobressaia com relação ao que o jogo quer oferecer. Para que isto seja possível, pesquisadores e projetistas de IA estudam e implementam novas técnicas, com o objetivo de alavancar o poder da IA em jogos.

Desta forma, este trabalho busca desenvolver um algoritmo para aprimorar uma técnica de aprendizado de máquinas conhecida como *Dynamic Scripting* (SPRONCK, 2005), explicada detalhadamente na Seção 2.3.3, que permite que agentes de IA aprendam em tempo real. Para isso, foi realizado um estudo sobre o comportamento dos agentes de IA, sob o efeito do *Dynamic Scripting*, no jogo *Neverwinter Nights* (BIOWARE, 2000).

1.1 JUSTIFICATIVA

Um dos grandes desafios da Inteligência Artificial em jogos é a criação de agentes de IA que se comportem como um humano real, sem criar situações que possam quebrar a experiência imersiva que o jogador tem com o mundo do jogo (CHAMPANDARD, 2014). A habilidade de lidar com todos os tipos de jogadores, com todos os tipos de capacidades, também é um ponto importante nesta área. Os inimigos de um jogo não só ficam mais fortes com o passar do tempo, mas os jogadores também passam a aprender com o jogo a medida que jogam. Balancear a dificuldade de um inimigo dentro do jogo, enquanto o jogador progride é um grande desafio na área de Inteligência Artificial (CHAMPANDARD, 2014). A evolução da IA no jogo também deve seguir o ritmo definido pelo jogador. Enquanto uns podem demorar mais tempo, outros podem jogar em um ritmo acelerado e a IA deve ser capaz de acompanhar. Para que isto seja possível, é essencial que o aprendizado dos agentes de IA seja rápido e eficaz.

1.2 OBJETIVOS

Nesta seção são apresentados os objetivos do presente trabalho.

1.2.1 Objetivo geral

O objetivo deste trabalho é o aprimoramento do aprendizado de agentes que podem aprender em tempo real, podendo se adaptar às situações que o jogo e o jogador possam oferecer. O trabalho é feito em cima de uma técnica chamada *Dynamic Scripting*, que permite que agentes de Inteligência Artificial se adaptem aos seus arredores baseados em um conjunto de regras que são constantemente atualizadas.

1.2.2 Objetivos específicos

Os objetivos específicos deste trabalho são:

- Alcançar um ponto de convergência mais baixo para o algoritmo do *Dynamic Scripting* no jogo *Neverwinter Nights*, isto é, fazer com que os agentes de IA consigam se adaptar ao ambiente e ao jogador com um número de encontros (interações com o jogador) mais baixo;
- Diminuir o tempo necessário para que o algoritmo do *Dynamic Scripting* no jogo *Neverwinter Nights* alcance a convergência, isto é, aumentar a eficiência dos agentes de forma que os encontros com o jogador e/ou outros agentes de IA não se extenda mais que o necessário.

1.3 ESTRUTURA DO TRABALHO

O presente trabalho é composto por seis capítulos organizados da seguinte maneira:

- Capítulo 1: Introdução do trabalho e breve descrição do tema proposto;
- Capítulo 2: Apresentação dos conceitos básicos para o entendimento do trabalho: Conceitos Computacionais, Inteligência Artificial em Jogos e Algoritmos de Aprendizagem para IA em Jogos.
- Capítulo 3: Apresenta as ferramentas e linguagens utilizadas para a análise e desenvolvimento do algoritmo.
- Capítulo 4: Apresenta as etapas de análise e desenvolvimento do algoritmo.
- Capítulo 5: Serão apresentados os testes realizados para validação do algoritmo e os resultados alcançados.
- Capítulo 6: Serão apresentadas as conclusões do trabalho, considerações finais e projetos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados os conceitos necessários para o entendimento do trabalho.

A sessão 2.1 apresenta conceitos da computação que possuem alta relevância na área de Inteligência Artificial. A sessão 2.2 aborda conceitos sobre a IA aplicada a jogos e faz um paralelo entre a ela e a Inteligência Artificial Computacional. Na sessão 2.3 é apresentado um referencial em algumas técnicas de Inteligência Artificial que serviram como base no desenvolvimento deste trabalho.

2.1 CONCEITOS COMPUTACIONAIS

2.1.1 Agente Computacional

Um agente computacional é algo que age a fim de alcançar um determinado objetivo. Suas principais características são sua autonomia, percepção de ambiente, persistência por um período de tempo, adaptação a mudanças e capacidade de assumir metas de outros. Um agente computacional que age visando alcançar o melhor resultado, ou o melhor resultado esperado, é considerado um agente racional (RUSSELL; NORVIG, 2003).

2.1.2 Scripts

Scripts são uma série de comandos que permitem a identificação de passos a seguir para a resolução de um problema (WEINBERGER, 2003). (KOLLAR; FISCHER; HESSE, 2006) trata script como um outro termo para conhecimento estratégico, definido por (JONG; FERGUSON-HESSLER, 1996) como uma sequência de atividades em prol de uma solução.

2.1.3 Espaço de Estados

O estado de um sistema é um parâmetro, ou conjunto de parâmetros, que podem ser usados para descrever o sistema. Um sistema no qual os estados mudam é chamado de sistema dinâmico (GOSAVI, 2009). O estado de um sistema dinâmico é o menor conjunto de variáveis, que sumarizam o passado do sistema, tal que o conhecimento destas variáveis em conjunto com o conhecimento de entradas futuras, determinam por completo o comportamento do sistema (OGATA, 2010). O conjunto de todos os estados de um sistema que descreve todas as configurações possíveis deste é chamado de espaço de estados (VRANCX, 2011).

2.1.4 Heurística

Uma heurística refere-se a um dispositivo usado na resolução de um problema. Tal dispositivo pode ser, mas não limitado a, um programa, uma estrutura de dados, uma estratégia ou puro conhecimento aplicado (ROMANYCIA; PELLETIER, 1985). A característica principal de uma heurística é que o seu uso não garante a solucionabilidade de um problema. O uso de uma heurística se dá quando os métodos clássicos de resolução de problema são muito lentos, ou para encontrar uma solução aproximada quando os métodos clássicos não conseguem encontrar uma solução ótima.

2.2 INTELIGÊNCIA ARTIFICIAL EM JOGOS

A Inteligência Artificial em jogos digitais possui características que a difere da Inteligência Artificial computacional. Sua diferença principal vem do seu objetivo: a IA em um jogo não deve buscar o melhor resultado possível, mas o resultado que a faça parecer inteligente e que crie uma experiência atraente para o jogador (RABIN, 2013).

No desenvolvimento de jogos, (CHAMPANDARD, 2003) define que a IA tem o papel de controlar NPCs (*Non-Player Characters*) e produzir entretenimento, sendo este assistido pelo realismo e credibilidade. Um agente de IA é dito crível se puder oferecer imersão ao jogador, realizando seu papel de uma forma que não tire a atenção do foco principal do jogo. O realismo com um agente de IA é alcançado quando este se comporta de maneira plausível, de uma maneira que pareça lógicamente possível para o jogador.

(CHAMPANDARD, 2003) associa a IA em jogos digitais a aleatoriedade. É possível criar um agente de IA com um comportamento completamente aleatório, contanto que a imersão do jogador não seja quebrada. Manter o realismo e a credibilidade de um agente, em um jogo digital, é um dos grandes desafios no desenvolvimento de IA para jogos (RABIN, 2013). Visando a criação de uma IA mais forte e que gere uma experiência satisfatória para o jogador, (SPRONCK; SPRINKHUIZEN-KUYPER; POSTMA, 2004) sugere o uso de algoritmos de aprendizado.

2.3 ALGORITMOS DE APRENDIZAGEM PARA IA EM JOGOS

Em um jogo, uma IA tem três necessidades básicas: a habilidade de se movimentar pelo mundo do jogo, a habilidade de tomar decisões e a habilidade de pensar taticamente ou estrategicamente. Para que isso possa ser realizado existem várias técnicas que preenchem todos estes requerimentos (MILLINGTON; FUNGE, 2009). Baseado nestes requerimentos, (MILLINGTON; FUNGE, 2009) criou um modelo básico de IA para jogos, que consiste em três segmentos: movimentação, tomada de decisão e estratégia. Movimentação se refere a capacidade de um agente de IA transformar decisões em algum tipo de movimento. Tomada de decisão

envolve a avaliação do estado atual que um agente se encontra e a determinação, pelo próprio, de que ação realizar em seguida. Por fim, estratégia é a abordagem geral usada por um conjunto de agentes para coordenar ou influenciar o comportamento de outros agentes de IA.

Para a criação de agentes de IA realísticos os três conceitos básicos apresentados precisam ser expandidos, a fim de cumprir com todas as expectativas de um jogador. Nos jogos atuais, têm ficado cada vez mais comum encontrar agentes de IA com personalidades próprias. Para criar agentes desta maneira, um projetista de IA geralmente escreve os comportamentos ou scripts que descrevem as reações do agente a todas as circunstâncias imagináveis dentro do mundo do jogo (RAJU et al., 2012). Esta abordagem apresenta diversas dificuldades, uma vez que é difícil imaginar e planejar todos os cenários possíveis que podem acontecer, dada a natureza dinâmica de um mundo de jogo. Uma boa alternativa para lidar com este problema é o uso de Inteligência Artificial Adaptativa.

2.3.1 Inteligência Artificial Adaptativa

Inteligência Artificial Adaptativa tem o potencial de se adaptar a cada jogador, aprendendo suas técnicas e oferecendo um desafio consistente (MILLINGTON; FUNGE, 2009). (SPRONCK, 2005) define a IA Adaptativa como IA para jogo com a habilidade de se autocorrigir e a habilidade de ser criativa. Ela possui potencial para reduzir o esforço necessário para criar IA específica para jogos, uma vez que os agentes serão capazes de aprender sobre seus arredores e as opções táticas que eles oferecem. Existem diversas técnicas de aprendizado diferentes e elas podem ser classificadas em vários grupos, dependendo dos aspectos de aprendizado: Aprendizado Offline, Online e Supervisionado.

Aprendizado Offline de uma IA para jogos acontece enquanto o jogo não está sendo jogado por um humano (CHARLES; MCGLINCHEY, 2004). Esse aprendizado pode ser dado através de amostras ou self-play (SPRONCK, 2005). A alteração dos parâmetros da IA é um exemplo da aplicação do aprendizado offline. De acordo com (SPRONCK, 2005), aprendizado offline é uma técnica comumente usada em jogos analíticos e na pesquisa acadêmica de jogos comerciais, mas raramente utilizada por desenvolvedores de jogos profissionais, salvo a alteração de alguns parâmetros.

Aprendizado Online acontece enquanto o jogo está sendo jogado por um humano (CHARLES; MCGLINCHEY, 2004). Através do aprendizado online, a IA de um jogo automaticamente se adapta de acordo com o estilo e as táticas do jogador, oferecendo desafios mais consistentes. Para (MILLINGTON; FUNGE, 2009), aprendizado online pode ser usado para fazer com que inimigos se tornem um desafio contínuo, ou para oferecer ao jogador uma linha de história que ele goste de jogar. Na prática, porém, existem alguns problemas que deixam os produtores de jogos relutantes de usar aprendizado online. Um deles é o medo que a IA do jogo aprenda um comportamento que seja inferior (WOODCOCK, 2000). Também podem surgir problemas com

a previsibilidade do agente e com os testes. Se o jogo está em constante mudança, pode tornar-se difícil reproduzir bugs e problemas que surjam, eventualmente (MILLINGTON; FUNGE, 2009).

Aprendizado Supervisionado também acontece enquanto o jogo está sendo jogado por um humano. As mudanças realizadas no jogo são resultado direto do feedback coletado em cima de quaisquer das decisões que a IA do jogo faz. O feedback indica se uma decisão que foi feita é desejada ou não desejada (SPRONCK, 2005). Com o uso do aprendizado supervisionado, o projetista da IA consegue controlar o que exatamente está sendo aprendido pelo agente de IA, seja fornecendo amostras de comportamento ou recompensando e punindo determinados comportamentos que o agente apresente.

Dentro deste grupo, o aprendizado online é o único método que permite o aprendizado automático da IA de um jogo, consequentemente, o único que permite que agentes de IA aprendam em tempo real. Segundo (SPRONCK, 2005), para que o aprendizado online seja aplicável na prática ele precisa cumprir quatro requerimentos funcionais e quatro requerimentos computacionais.

Os quatro requerimentos computacionais são: velocidade, eficácia, robustez e eficiência. Uma vez que o aprendizado acontece durante o jogo, ele precisa ser computacionalmente rápido (MILLINGTON; FUNGE, 2009). O aprendizado online também precisa criar uma IA eficaz durante o processo de aprendizagem, evitando se tornar inferior à IA que foi projetada manualmente (MILLINGTON; FUNGE, 2009). O método precisa ser robusto, respeitando a aleatoriedade inerente na maioria dos jogos (LUDWIG; OREGON, 2008). Por fim, o aprendizado online precisa ser eficiente no que diz respeito ao número de tentativas necessárias para criar uma IA para o jogo com sucesso (SPRONCK, 2005).

Os quatro requerimentos funcionais são: clareza, variedade, consistência e escalabilidade. Os resultados produzidos por uma IA com aprendizado online precisam ser fáceis de se interpretar. Também é necessário que ela produza uma variedade de comportamentos diferentes. O número médio de tentativas necessárias para a IA adaptativa produzir resultados bem sucedidos devem ter alta consistência (SPRONCK, 2005). O aprendizado online também deve ser capaz de escalar o nível de dificuldade ao nível do jogador humano (LIDÉN; EINSTEIN, 2003).

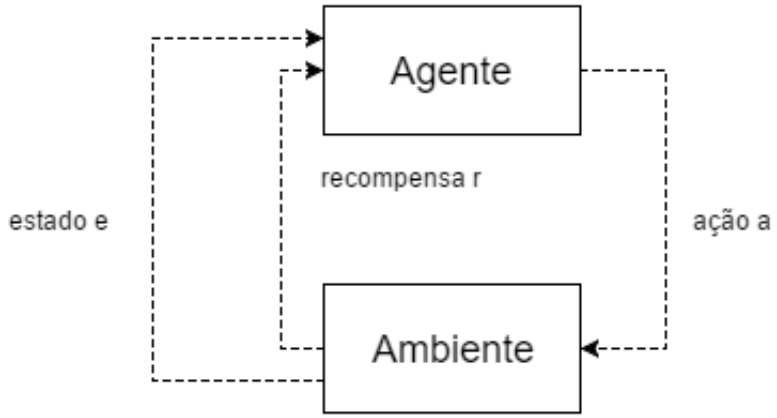
2.3.2 Aprendizado por Reforço

Aprendizado por Reforço é uma técnica de aprendizado de máquina que tem por objetivo mapear estados à ações (SUTTON; BARTO, 1998). A técnica consiste no agente percebendo o ambiente e agindo sobre ele. As ações que o agente executa afetam o estado do ambiente e geram recompensas, responsáveis por medir o quão bem o agente se saiu em determinada tarefa. O principal objetivo do agente é maximizar o ganho de recompensas (ARMSTRONG et al., 2006).

O agente do aprendizado por reforço aprende através da tentativa e erro. Dado um estado do ambiente o agente escolhe uma ação a realizar, que pode levar ou não a outro estado, e recebe

uma recompensa (Figura 2). Ao repetir este processo o agente eventualmente aprende quais são as melhores ações a tomar para obter a maior recompensa possível. Durante este processo o agente precisa adquirir experiências úteis sobre os estados, ações, transições de estados e recompensas, para que a avaliação do sistema aconteça ao mesmo tempo que o processo de aprendizagem (SUTTON; BARTO, 1998).

Figura 2 – ***Funcionamento do Aprendizado por Reforço***



Fonte: (SUTTON; BARTO, 1998)

O agente deve experimentar e explorar as ações possíveis de forma moderada. Um agente que não explora muito corre o risco de gerar uma política abaixo do esperado. Por outro lado, um agente que explora demais não se beneficiará de uma política ótima, caso passe por ela (ARMSTRONG et al., 2006). O agente deve aprender sobre quais ações maximizam seus ganhos, mas também agir visando essa maximização enquanto explora ações que ainda não foram executadas ou estados não vistos ou pouco visitados (CAMPONOGARA; TR, 2005).

2.3.3 *Dynamic Scripting*

Dynamic Scripting é uma técnica online de aprendizado por reforço para IA em jogos. Através do uso de scripts convencionais de IA, a técnica limita o tamanho do espaço de estados o suficiente para tornar a adaptação dentro do jogo possível.

O script de uma IA para jogos consiste em uma sequência de regras e cada uma dessas regras é composta por uma parte condicional opcional, que identifica um ou mais estados de jogo, e uma parte de ações (SPRONCK, 2005). Para atribuir uma ação à um agente de IA, as regras em um script são avaliadas em sequência e selecionadas baseadas na condição que combina com o estado atual do jogo.

Dada a definição de scripts, nota-se que os mesmos não apresentam características adaptativas. Seu funcionamento se baseia em uma série de regras, já conhecidas, e que respondem aos estados do jogo da forma que tais regras definiram. *Dynamic Scripting*, por outro lado,

adiciona a capacidade de explorar a representação do espaço de estados que os scripts produzem para a produção de aprendizado rápido e eficiente, enquanto conta com o conteúdo dos scripts para garantir que todo comportamento adaptativo é plausível e eficaz (SPRONCK, 2005).

(THAWONMAS; OSAKA, 2006) apresenta o algoritmo do processo de *Dynamic Scripting* de forma bem simples:

1. Uma base de dados, que consiste em um conjunto de regras, é atribuída a um grupo ou a um indivíduo de agente(s) de IA de um mesmo tipo.
2. Um conjunto de regras é selecionado da base de dados, de acordo com os pesos, para a construção do script de um agente.
3. O agente de IA batalha contra um personagem controlado pelo jogador (ou outro agente de IA) utilizando o conteúdo selecionado em seu script.
4. O peso de cada regra existente no script é atualizada de acordo com o resultado da batalha.
5. Vá para 2.

Dynamic Scripting utiliza várias bases de dados, uma para cada tipo de agente. Toda vez que uma nova instância de um agente é criada, as bases de dados são utilizadas para gerar um script e atribuí-lo àquele agente recém-criado. Na base de dados, cada regra possui um peso (valor numérico) que determina a probabilidade daquela regra ser selecionada para um script. O processo de adaptação acontece ao alterar o peso das regras baseado no feedback coletado: o fracasso ou sucesso do agente (SPRONCK, 2005).

Ao fim de cada encontro entre o jogador e os agentes de IA, um valor de fitness é calculado, que representa a eficácia do comportamento dos agentes. A função de atualização dos pesos altera o peso das regras nos scripts baseada no valor de fitness que foi gerado. Um valor de fitness alto aumenta os pesos e valores de fitness baixos diminuem os pesos. De acordo com (SPRONCK, 2005), ao fazer isto as regras que fazem com que os agentes se saiam bem ficarão associadas com os maiores pesos, o que significa que estas regras serão selecionadas com uma maior probabilidade. Desta forma, agentes que estejam sendo controlados por scripts passam a se adaptar e se sair melhor contra um jogador particular.

3 FERRAMENTAS UTILIZADAS

Neste trabalho foi proposto um algoritmo para aprimorar a técnica de *Dynamic Scripting*, que visa alcançar um tempo de convergência menor e, consequentemente, fazer com que os agentes de IA aprendam mais rapidamente. O seu desenvolvimento foi feito utilizando a linguagem NWScript, no software Aurora Toolset. A aplicação e os testes do algoritmo foram realizados no jogo Neverwinter Nights.

3.1 EQUIPAMENTO UTILIZADO

Todos os testes computacionais realizados neste trabalho foram feitos em um computador de mesa com sistema operacional Windows 10 64-bits, 8GB de Memória RAM, processador Intel *Core* i5 2.90 GHz e placa de vídeo NVIDIA GeForce 210.

3.2 LINGUAGENS UTILIZADAS

3.2.1 NWScript

NWScript é uma linguagem de script desenvolvida pela BioWare para o jogo *Neverwinter Nights*. Ela é baseada na linguagem de programação C e é implementada no software Aurora Toolset (NWN Lexicon, 2002).

A linguagem dispõe de uma biblioteca de funções que permite ao usuário programar, com precisão nos detalhes, as ações de vários objetos e cenários de jogo, como: monstros, NPCs, itens, entre outros. O NWScript herda algumas funções e tipos de dados da linguagem C, porém ainda é muito limitada, servindo particularmente ao propósito de funcionar dentro do ambiente do jogo *Neverwinter Nights*.

3.3 AMBIENTE DE DESENVOLVIMENTO

3.3.1 Aurora Toolset

Aurora Toolset é um conjunto de ferramentas, criada pela BioWare, que permite ao usuário criar seus próprios módulos para serem utilizados no jogo *Neverwinter Nights*. Através do uso do Toolset o usuário é capaz de mudar vários aspectos do jogo e criar novos módulos usando recursos visuais próprios do *Neverwinter Nights*. Esta é a mesma ferramenta utilizada pelos desenvolvedores do jogo para criar as campanhas oficiais (NWN Wiki, 2005).

3.4 AMBIENTE DE SIMULAÇÃO

3.4.1 Neverwinter Nights

Neverwinter Nights é um jogo de *role-play* (RPG) em terceira pessoa desenvolvido pela BioWare (BIOWARE, 2000). O jogo foi lançado para Windows em Junho de 2002 e para Linux e Mac OS em Junho de 2003. No jogo, o jogador se aventura pelo mundo de fantasia de *Forgotten Realms* e pode escolher diferentes classes de personagens para jogar, como: ladino, bárbaro, clérigo, paladino, mago, entre outras. O jogo utiliza os sistemas da terceira edição do conceituado jogo de mesa *Advanced Dungeons & Dragons* (WIZARDS, 1995), concedendo centenas de magias e habilidades para os jogadores experimentarem e mais de duzentas criaturas para enfrentarem em combates.

Neverwinter Nights é jogado através de uma perspectiva isométrica[1], como mostrado na Figura 3, onde o jogador controla o seu personagem através de comandos enviados com o *mouse*.

Figura 3 – *Tela do jogo Neverwinter Nights*

Fonte: (NWNCOMMUNITY, 2002)

[1] Posicionamento de câmera que simula um ambiente tridimensional em jogos 2D e enriquece a visualização de um mapa em 3D real. O posicionamento isométrico é um intermediário entre a visão lateral e do teto. (DOZZI; DANIEL, 1987)

4 SISTEMA DESENVOLVIDO

De forma resumida as etapas realizadas para o desenvolvimento do trabalho são apresentadas a seguir:

1. Detecção e análise de problema no algoritmo do *Dynamic Scripting*;

2. Proposta e elaboração de um algoritmo para solucionar o problema.

4.1 ANÁLISE DO DYNAMIC SCRIPTING

Neste trabalho o ambiente de simulação para aplicação do algoritmo será o mesmo utilizado por (SPRONCK, 2005). O módulo utilizado será o *Online Adaptation Neverwinter Nights module, version 3* (SPRONCK et al., 2006). O módulo consiste no encontro e combate entre dois times de agentes, um dinâmico e um estático com a mesma composição de membros.

O algoritmo do *Dynamic Scripting* é aplicado no combate entre estes dois times, cada um representado por uma cor: branco e preto. O time branco é o time de agentes controlados pelo *Dynamic Scripting*, enquanto o time preto será controlado pela a IA padrão do jogo. Neste trabalho a IA utilizada é a da versão 1.61. Cada time é composto por quatro agentes, cada um com uma classe diferente. As classes dos agentes são: Clérigo, Mago, Guerreiro e Ladino, todos eles com o mesmo nível de experiência.

Nos testes realizados utilizando a versão 1.61 da IA, (SPRONCK, 2005) obteve resultados que comprovaram que o *Dynamic Scripting* alcança todos os requerimentos indicados na Seção 2.3. Com 31 testes realizados, o algoritmo alcançou o ponto de convergência com, em média, 35 combates. Cada teste, também chamado de iteração, consiste em cem combates entre os times de agentes. O algoritmo alcança o ponto de convergência quando a média da fitness do time de agentes dinâmico (time branco) é maior que a média da fitness do time estático (time preto) nos últimos dez combates. O ponto de convergência é o número do primeiro combate após o time dinâmico superar o time estático por, pelo menos, dez combates consecutivos (SPRONCK, 2005).

Dado que a fitness obtida mediante o resultado de um combate é que determina o ponto de convergência do algoritmo, é preciso observar como esta é calculada e destacar os pontos que são relevantes a ela. A função que calcula a fitness de um time é definida em 4.1.

$$F(g) = \begin{cases} 0, \{g \text{ lost}\} \\ \frac{1}{5} + \sum_{c \in g, h_T(c) > 0} \frac{2}{5N_g} \left(1 + \frac{h_T(c)}{h_0(c)}\right), \{g \text{ won}\} \end{cases} \tag{4.1}$$

Na Equação 4.1, c representa um agente, g representa um time de agentes. $h_t(c) \in \mathbb{N}$ representa a vida de um agente c no tempo t. $N_g \in \mathbb{N}$ é o número total de agentes em um time g. Nota-se que os principais fatores para determinação de uma fitness alta são: número de agentes vivos ao fim do combate e quantidade de vida restante dos agentes vivos. O tempo de combate também é um fator importante, pois está indiretamente ligado aos outros dois citados. Quanto mais se estende um combate, maiores são as chances da vida e do número de agentes diminuir.

Visando analisar em detalhes os efeitos do algoritmo sobre os agentes, dividiremos um combate em uma série de estados. Estes estados serão definidos e explicados na Seção 4.1.1.

4.1.1 Divisão por Estados

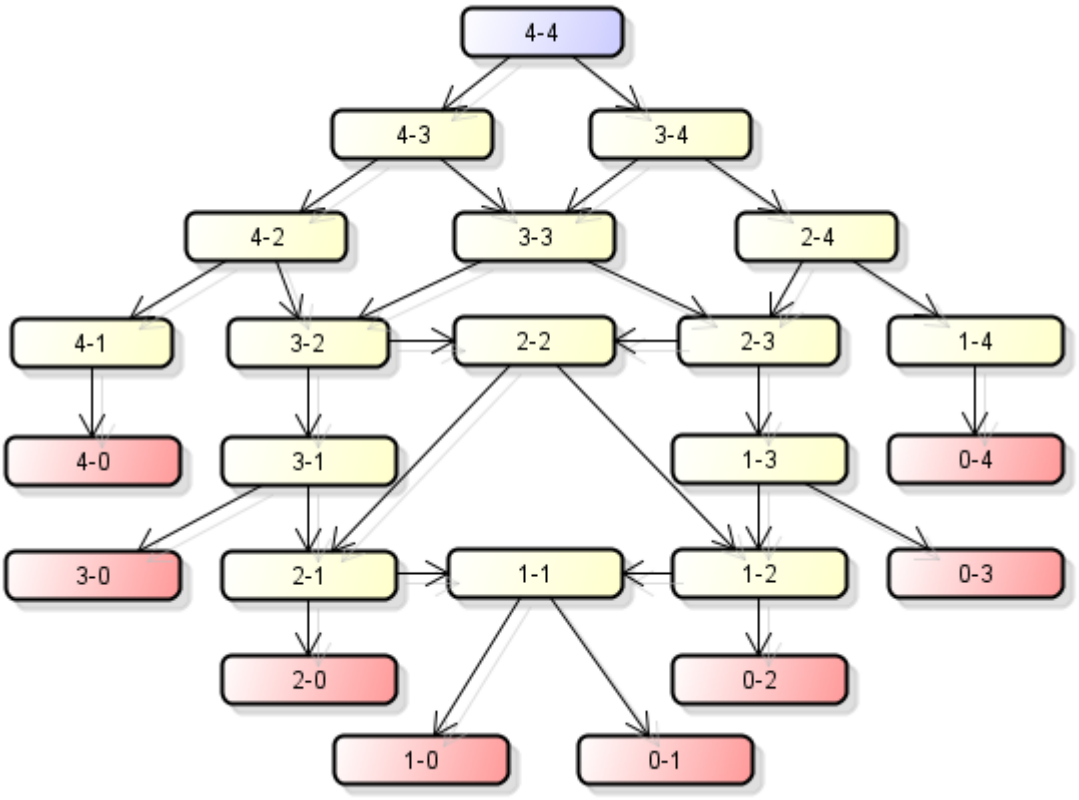
A principal proposta da divisão por estados é facilitar a identificação de pontos em que o algoritmo não seja eficiente e que possam estar prejudicando o resultado em geral. Cada combinação possível de agentes vivos, em um determinado instante de tempo, representa um estado.

A representação escrita de um estado será dada pelo número de agentes vivos no time branco e o número de agentes vivos no time preto, separados por um sinal de menos. Depois disso há um espaço em branco e então a primeira letra da classe dos agentes vivos do time branco e a primeira letra das classes dos agentes vivos do time preto, separados por um sinal de menos. Por exemplo, o time branco possui dois agentes vivos, um Clérigo e um Mago, e o time preto possui três agentes vivos, um Guerreiro, um Ladino e um Clérigo. A representação escrita deste estado seria 2-3 CM-GLC.

Todo combate é iniciado no estado 4-4 CGLM-CGLM, uma vez que todos os membros de ambos os times estão vivos. Com isto, este estado será definido como Estado Inicial. Um estado em que o número de agentes vivos do time branco é maior que o número de agentes vivos do time preto, será definido como Estado Positivo. Sua contraparte, onde o número de agentes vivos pretos é maior que número de agentes vivos brancos, será definida como Estado Negativo. Um estado no qual, em qualquer um dos times, o número de agentes vivos seja igual a zero será definido como Estado Final.

Um estado não-expandido não exibe as combinações de classes de personagens vivas em ambos os times. Ele pode ser representado apenas pela quantidade de agentes vivos de ambos os lados. Esta representação indica o espaço de estados de maneira geralizada, exibindo quais estados podem gerar os próximos. A Figura 4 apresenta o diagrama de estados não-expandidos.

Figura 4 – ***Diagrama de Espaços Não-Expandidos***



4.1.2 Análise dos Estados

Uma vez que a representação de um estado já indica a quantidade de agentes vivos e mortos em cada time, é necessário agora analisar o tempo gasto em cada um eles. Esta análise ajudará a indicar a eficiência do algoritmo sobre determinadas combinações de agentes.

Para cada um dos estados foi calculado quanto tempo, em segundos, demorava-se para passar para um próximo estado. Os resultados listados na Tabela 1 são referentes a cinquenta iterações do algoritmo.

Tabela 1 – Média de tempo gasto em cada estado não-expandido

Estado	Média de Tempo (s)
4-3	5
4-2	6
4-1	17
3-4	5.5
3-3	5.75
3-2	7.5
3-1	20
2-4	6.16
2-3	6.5
2-2	8.33
2-1	18.33
1-4	6.5
1-3	9.5
1-2	13.5
1-1	22.25

Nota-se que, nos estados mais próximos dos Estados Finais, o tempo gasto aumenta consideravelmente. Este fenômeno, porém, só acontece nos Estados Positivos, ou seja, aqueles em que o time dinâmico está na vantagem. O estado 1-4, por exemplo, corresponde a uma configuração em que existem quatro agentes do time estático vivos e apenas um agente do time dinâmico vivo. A média de tempo gasto neste estado é de 6.5 segundos. Sua contraparte, o estado 4-1 tem média de tempo gasto de 17 segundos, quase o triplo de tempo do estado 1-4.

Para compreender melhor este fenômeno foi realizada a análise de tempos dos estados expandidos. Todos os estados que apresentaram média de tempo acima do limiar de 10 segundos foram selecionados. O estado 1-3 também será expandido devido à proximidade do seu tempo com o limiar definido. A tabela 2 exibe os resultados obtidos. Os resultados listados são referentes a cinquenta iterações do algoritmo.

Tabela 2 – Média de tempo gasto nos estados expandidos

Estado	Média de Tempo (s)
4-1 CMGL	17
3-1 CMG	16
3-1 CML	20
3-1 CGL	23
3-1 MGL	21
2-1 MG	10
2-1 ML	23
2-1 MC	10
2-1 LC	30
2-1 LG	22
2-1 CG	15
1-3 C	19
1-3 G	6
1-3 M	5
1-3 L	8
1-2 C	30
1-2 G	9
1-2 M	5
1-2 L	10
1-1 G	13
1-1 C	27
1-1 M	17
1-1 L	32

Com os resultados da Tabela 2 é possível identificar alguns estados responsáveis por elevar a média geral. Tomando o estado 1-3 como exemplo, o estado expandido 1-3 C possui média de tempo muito maior que o 1-3 G, 1-3 L e 1-3 M. Isso se repete em outros estados como o 1-2 C, 2-1 ML e 2-1 LC.

Através dos tempos coletados é possível observar que os agentes dinâmicos possuem dificuldades em se adaptar a alguns estados com configurações específicas. O tempo extra gasto em cada um desses estados pode ser um grande influenciador na forma como os agentes aprendem e, consequentemente, na velocidade em que o algoritmo converge. A Seção 4.2 propõe um algoritmo para tratar estes casos e melhorar a eficiência e eficácia dos agentes dinâmicos.

4.2 DESENVOLVIMENTO DO ALGORITMO

Este algoritmo tem por objetivo tratar apenas os casos em que os estados apresentem um valor de tempo elevado. Dos estados apresentados na Tabela 2 foram selecionados todos com tempos iguais ou acima 13 segundos. Este valor de tempo foi definido com base no tempo médio gasto entre todos os estados, onde tempos abaixo de 13 segundos foram considerados aceitáveis.

Os estados selecionados foram então expandidos, exibindo agora sua informação completa: número de membros vivos em ambos os times e classes dos membros vivos de ambos os times. O tempo gasto em cada um desses estados também foi calculado, os dados completos podem ser encontrados na Tabela 3. Os resultados listados são referentes a cinquenta iterações do algoritmo.

Tabela 3 – Média de Tempo dos Estados Expandidos Pré-Algoritmo

Estado	Média de Tempo (s)	Estado	Média de Tempo (s)
4-1 CMGL vs C	23.5	2-1 CG vs G	Sem Dados
4-1 CMGL vs M	3.36	2-1 LG vs C	27
4-1 CMGL vs G	6	2-1 LG vs L	27
4-1 CMGL vs L	4.7	2-1 LG vs M	6
3-1 CML vs C	25.2	2-1 LG vs G	6.6
3-1 CML vs L	4.5	1-3 C vs CGL	9.5
3-1 CML vs M	4	1-3 C vs LCM	11
3-1 CML vs G	9.5	1-3 C vs LMG	Sem Dados
3-1 MGL vs C	25.2	1-3 C vs GMC	14
3-1 MGL vs L	3	1-2 C vs CL	10
3-1 MGL vs M	5.11	1-2 C vs GC	18
3-1 MGL vs G	5	1-2 C vs LM	Sem Dados
3-1 CMG vs C	24.3	1-2 C vs GL	1.5
3-1 CMG vs L	1	1-2 C vs MG	Sem Dados
3-1 CMG vs M	3.2	1-2 C vs MC	8.75
3-1 CMG vs G	5	1-1 G vs C	34
3-1 CGL vs C	25.5	1-1 G vs L	10.33
3-1 CGL vs L	4.5	1-1 G vs M	6
3-1 CGL vs M	5.25	1-1 G vs G	7.54
3-1 CGL vs G	3.66	1-1 M vs C	27.5
2-1 ML vs C	34.2	1-1 M vs L	7
2-1 ML vs L	5	1-1 M vs M	12
2-1 ML vs M	2	1-1 M vs G	6
2-1 ML vs G	18	1-1 C vs C	12.66
2-1 LC vs C	32.11	1-1 C vs L	28.5
2-1 LC vs L	Sem Dados	1-1 C vs M	21
2-1 LC vs M	17	1-1 C vs G	14.5
2-1 LC vs G	1	1-1 L vs C	25.57
2-1 CG vs C	22.66	1-1 L vs L	Sem Dados
2-1 CG vs L	Sem Dados	1-1 L vs M	4.5
2-1 CG vs M	Sem Dados	1-1 L vs G	11.5

Estados que apresentam "Sem Dados"como média são estados que nunca foram visitados durante os testes. Devido a baixa frequência com que aparecem, estes estados foram deixados de fora do tratamento do algoritmo.

Os estados com tempos iguais ou acima do limiar, definido como 13 segundos, agora compõem um novo conjunto denomidado Estados Críticos. Um Estado Crítico é todo estado,

em sua forma expandida, cuja média de tempo gasto nele é igual ou superior a 13 segundos. Os seguintes estados compõem o conjunto de Estados Críticos:

- 4-1 CMGL-C;
- 3-1 CMG-C;
- 3-1 CML-C;
- 3-1 CGL-C;
- 3-1 MGL-C;
- 2-1 ML-C;
- 2-1 LC-C;
- 2-1 LC-M;
- 2-1 LG-C;
- 2-1 LG-L;
- 2-1 CG-C;
- 1-3 C-GMC;
- 1-2 C-GC;
- 1-1 C-L;
- 1-1 C-M;
- 1-1 C-G;
- 1-1 G-C;
- 1-1 L-C;
- 1-1 M-C.

Os Estados Críticos funcionam como uma condição para o algoritmo. Sempre que o algoritmo detectar que o jogo entrou em um destes estados, novas táticas serão aplicadas sobre os agentes. A tática escolhida para todas as classes de agentes é a mesma: executar um ataque básico no inimigo. Esta tática interrompe quaisquer ações que os agentes realizariam de seu script gerado pelo algoritmo do *Dynamic Scripting* e focariam apenas em atacar os inimigos restantes, visando encerrar o combate o mais rápido possível.

A detecção de um Estado Crítico é realizada no script *PseudoHeartbeat* (Apêndice A.1). Este *script* roda em um *loop* a cada um segundo e é responsável unicamente por fazer estas checagens. Para cada um dos Estados Críticos há uma condição, como mostra o Código 4.1.

Código 4.1 – Detecção de Estado Crítico

```
if (WhiteMembers == 4 && BlackMembers == 1 && GetLocalInt(oTarget, "state3") == 0)
{
        PrintString("4-1"); //Imprime no Arquivo de log
        PrintString(classAlive); //Imprime no Arquivo de log
        PrintString(classAliveB); //Imprime no Arquivo de log
        PrintString(IntToString (nHeartbeat)); //Imprime no Arquivo de log
        SetLocalInt(oTarget, "state3", 1); //Define que esse estado foi visitado
        SetLocalInt(oTarget, "EstadoCritico", 1); //Atualiza a variavel EstadoCritico

        if (classAliveB == "C")
                ActionSpeakString("Estado Critico: 4-1 C"); //Exibe uma mensagem na tela
}
```

As variáveis *WhiteMembers* e *BlackMembers* representam o número de agentes vivos nos times branco e preto, respectivamente. No Código 4.1 a verificação é feita sobre o estado 4-1. As demais condições possuem a mesma estrutura, mudando apena a os valores checados para as variáveis *WhiteMembers* e *BlackMembers*. A variável *EstadoCritico* indica se o estado atual é ou não um Estado Crítico. Ela recebe o valor 1 quando é um Estado Crítico e 0 caso contrário.

Uma vez que um Estado Crítico é detectado, os agentes devem parar de checar seus *scripts* e passar a seguir a nova tática. Essa verificação é feita no *script in_learn_generic*. Este *script* está sempre atualizando o valor da variável *EstadoCritico*, conforme mostra o Código 4.2.

Código 4.2 – Atualização da variável EstadoCritico

```
//Pegando o estado atual e salvando na variavel
object oTarget = GetObjectByTag( "ArcheryTarget" );
int isEstadoCritico = GetLocalInt(oTarget, "EstadoCritico");
```

No mesmo *script* também há a substituição da tática utilizada. O Código 4.3 mostra como essa mudança é realizada.

Código 4.3 – Substituição de Tática

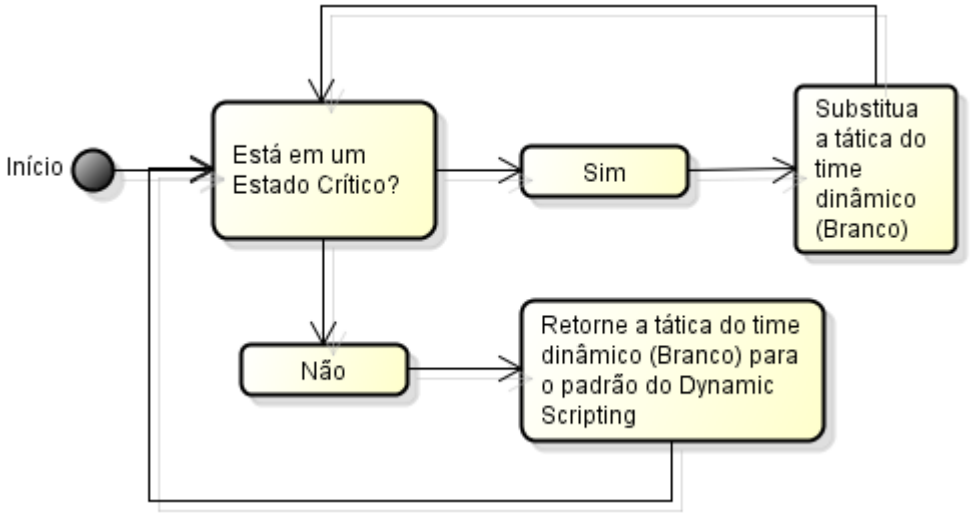
```
if (isEstadoCritico == 1) { //Se o jogo esta em um Estado Critico
        if ( !IsPriest(oCreature) && !IsMage(oCreature) ) //Verifica a classe
        {       if (iJustCheck)
                        return TRUE;
                sRuleText = "Estao Critico: ExecuteMeleeAttack(oNearest)";
                return ExecuteMeleeAttack( oNearest ); //Altera a tatica
        }

        if ( IsPriest (oCreature) ) //Verifica a classe
        {       if (iJustCheck)
                        return TRUE;
                sRuleText = "Estado Critico: TalentMeleeAttack(oNearest)";
                return TalentMeleeAttack( oNearest ); //Altera a tatica
```

```
        }
}
```

Identificado um Estado Crítico o agente entrará na condição da Figura 4.3. As condições internas então verificarão a classe do agente e enviarão o comando para executar um ataque básico. O algoritmo completo é definido abaixo e no diagrama da Figura 5.

1. A cada segundo atualize a variável *EstadoCritico* indicando se o estado atual é um Estado Crítico ou não.
2. Se o estado atual é um Estado Crítico, substitua a tática do time dinâmico (time branco). Se não, retorne a tática do time dinâmico para o padrão do *Dynamic Scripting*. Vá para 1.

Figura 5 – *Algoritmo de Substituição de Tática*



5 TESTES COMPUTACIONAIS

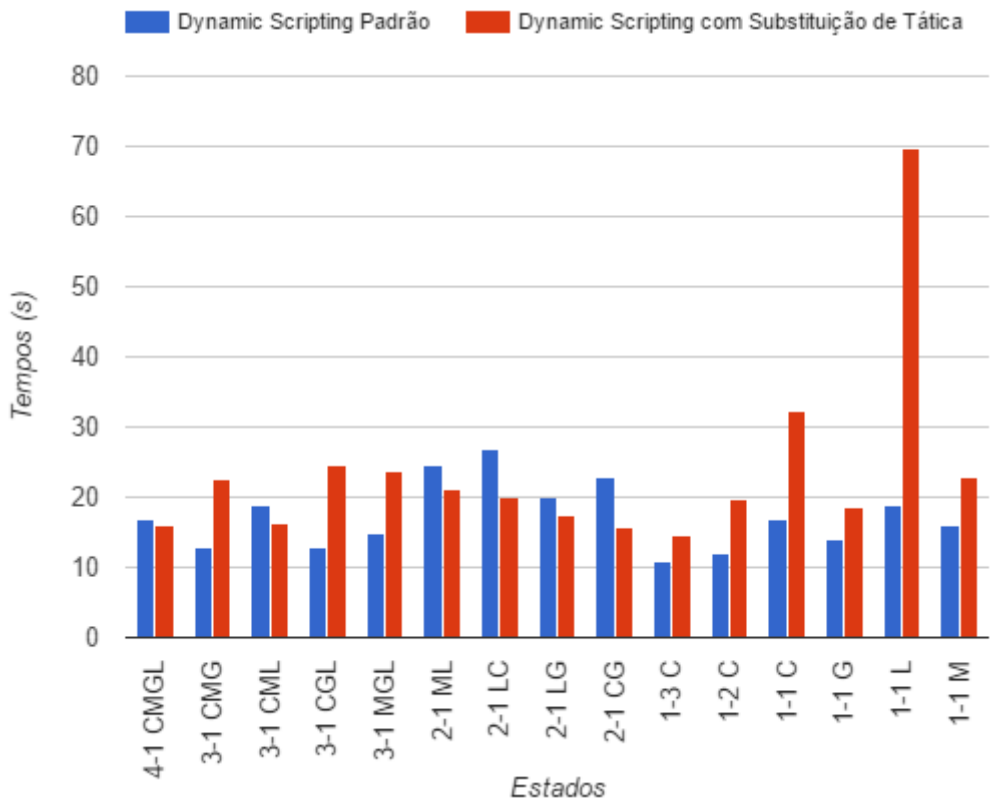
Nos testes realizados, cada iteração do algoritmo corresponde à 50 combates entre os times de agentes. A versão do jogo utilizada para rodar os testes foi a *Neverwinter Nights Diamond Edition*. O módulo utilizado durante os testes foi o *Online Adaptation Neverwinter Nights module, version 3*.

Todas as análises são feitas em comparação com os resultados obtidos com algoritmo do *Dynamic Scripting* de (SPRONCK et al., 2006). Este algoritmo será definido como *Dynamic Scripting* Padrão. Os resultados do algoritmo foram coletados através de 50 iterações. O Algoritmo de Substituição de Tática foi executado o mesmo número de vezes, para fins comparativos. Os resultados são exibidos na Tabela 4 e na Figura 6.

Tabela 4 – Média de Tempo dos Algoritmos

Estado	Média de Tempo (s) Dynamic Scripting Padrão	Média de Tempo (s) Dynamic Scripting com Substituição de Tática
4-1 CMGL	17	16
3-1 CMG	13	22.5
3-1 CML	19	16.42
3-1 CGL	13	24.46
3-1 MGL	15	23.77
2-1 ML	24.5	21.1
2-1 LC	27	20
2-1 LG	20	17.33
2-1 CG	23	15.83
1-3 C	11	14.64
1-2 C	12	19.66
1-1 C	17	32.25
1-1 G	14	18.46
1-1 L	19	69.66
1-1 M	16	23

Figura 6 – *Gráfico da Média de Tempo dos Estados Críticos*



Fonte: Captura de Tela - Planilhas Google

Com os resultados obtidos foi possível observar a diminuição dos tempos em certos estados e o aumento do tempo em outros. A fim de identificar o impacto das mudanças no tempo total, foi realizada uma análise da frequência com que os times entram nos Estados Críticos (Tabela 5).

Tabela 5 – Frequência de Visita aos Estados Críticos em 50 iterações

Estado	Frequência
4-1 CMGL	390
3-1 CMG	30
3-1 CML	105
3-1 CGL	32
3-1 MGL	67
2-1 ML	25
2-1 LC	7
2-1 LG	7
2-1 CG	15
1-3 C	35
1-2 C	30
1-1 C	10
1-1 G	32
1-1 L	7
1-1 M	20

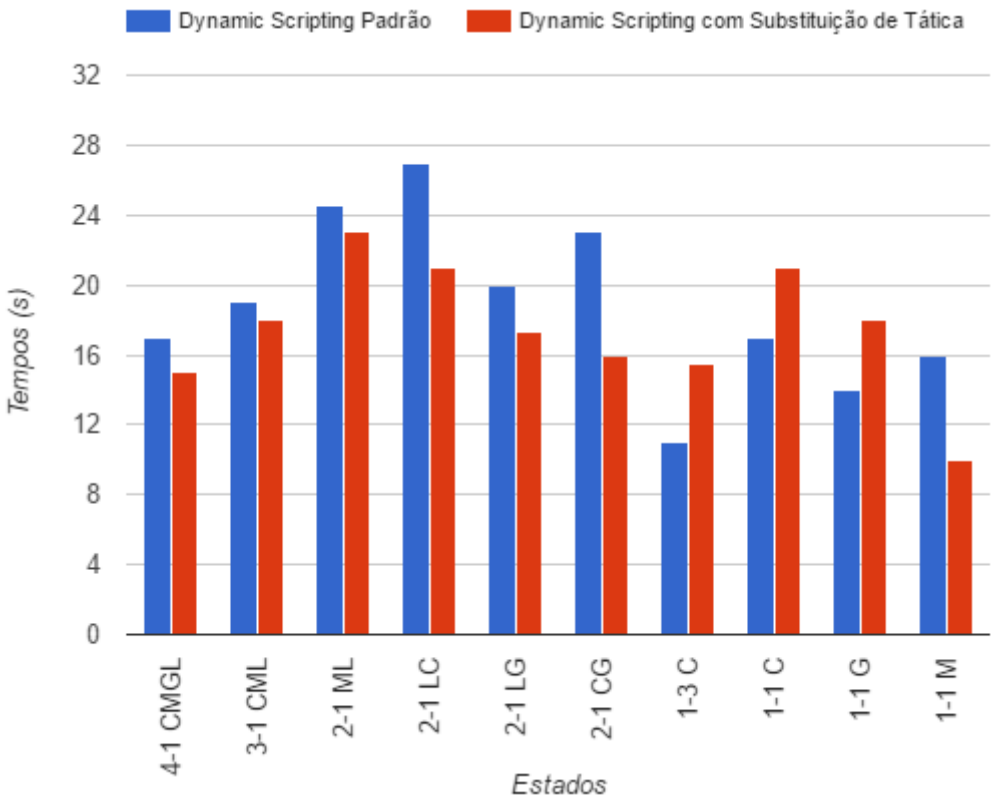
Estados em que o tempo aumentou consideravelmente após a aplicação do Algoritmo de Substituição de Tática, e que possuem uma frequência alta, foram removidos do tratamento do algoritmo. Isso se aplica aos seguintes estados: 3-1 CMG, 3-1 CGL, 3-1 MGL, 1-2 C e 1-1 L.

Com o corte destes estados, novos testes foram realizados. O algoritmo foi executado 110 vezes e os resultados comparativos são exibidos na Tabela 6 e na Figura 7.

Tabela 6 – Média de Tempo dos Algoritmos (Pós-Corte)

Estado	Média de Tempo (s) Dynamic Scripting Padrão	Média de Tempo (s) Dynamic Scripting com Substituição de Tática
4-1 CMGL	17	15
3-1 CML	19	18
2-1 ML	24.5	23
2-1 LC	27	21
2-1 LG	20	17.33
2-1 CG	23	16
1-3 C	11	15.5
1-1 C	17	21
1-1 G	14	18
1-1 M	16	10

Figura 7 – ***Gráfico da Média de Tempo dos Estados Críticos (Pós-Corte)***



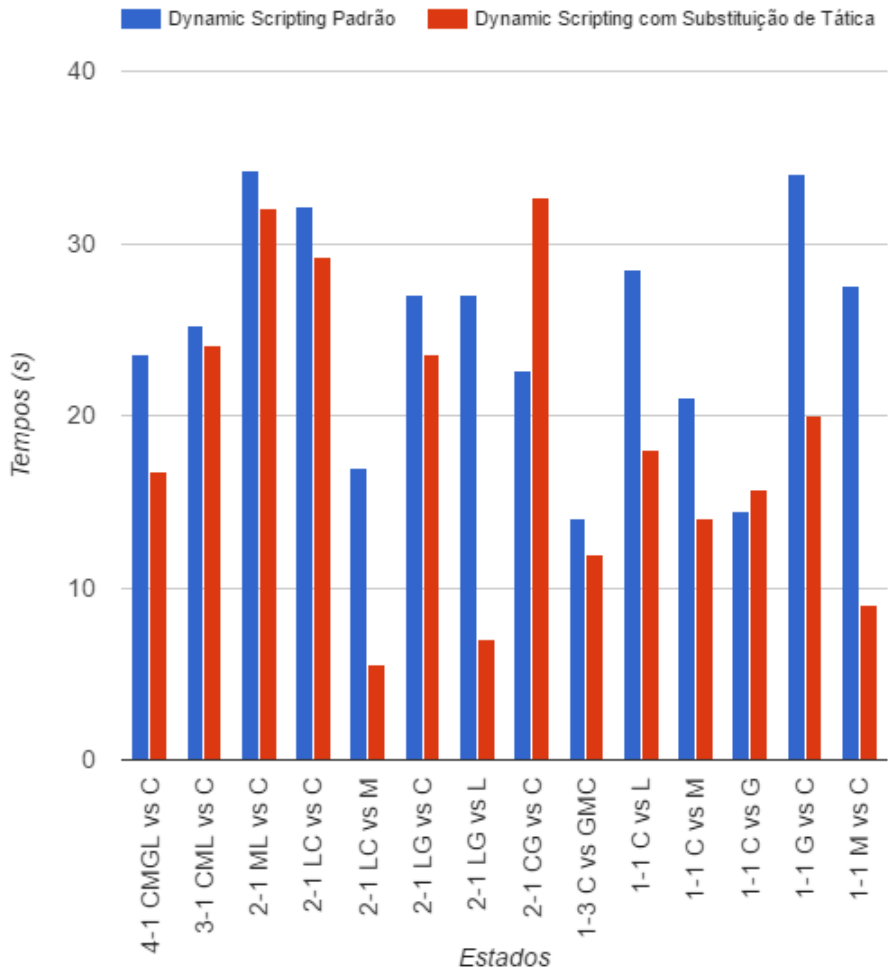
Fonte: Captura de Tela - Planilhas Google

A diminuição nos tempos dos Estados Críticos pode ser melhor visualizada ao expandí-los. A Tabela 7 e a Figura 8 exibem estes resultados.

Tabela 7 – Média de Tempo dos Algoritmos com Estados Críticos Expandidos (Pós-Corte)

Estado	Média de Tempo (s) Dynamic Scripting Padrão	Média de Tempo (s) Dynamic Scripting com Substituição de Tática
4-1 CMGL-C	23.52	16.79
3-1 CML-C	25.22	24
2-1 ML-C	34.2	32
2-1 LC-C	32.11	29.22
2-1 LC-M	17	5.5
2-1 LG-C	27	23.6
2-1 LG-L	27	7
2-1 CG-C	22.66	32.66
1-3 C-GMC	14	11.9
1-1 C-L	28.5	18
1-1 C-M	21	14
1-1 C-G	14.5	15.75
1-1 G-C	34	20
1-1 M-C	27.5	9

Figura 8 – ***Gráfico da Média de Tempo dos Estados Críticos Expandidos (Pós-Corte)***



Fonte: Captura de Tela - Planilhas Google

5.0.1 Validação dos Resultados

Através dos resultados obtidos pode-se constatar uma melhora em relação ao tempo gasto nos Estados Críticos. É necessário, porém, analisar este resultado e demonstrar os ganhos reais provenientes da aplicação do algoritmo. A Tabela 8 exibe um comparativo das estatísticas do *Dynamic Scripting* Padrão e do *Dynamic Scripting* com Substituição de Tática.

Tabela 8 – Estatísticas por Combate

	DS Padrão	DS com Substituição de Tática
Média de Visita a Estados por Combate	4.7	4.5
Média de Duração do Combate (s)	53.4	46.04
Round de Convergência	35	21

A duração de cada combate foi calculada através do *script* PseudoHeartbeat (Apêndice A.1), onde um contador é iniciado quando um combate começa e uma contagem é realizada segundo a segundo. Este mesmo *script* é responsável por criar um *log*[1] que lista a ordem de visita a cada estado dentro de um combate. O *Round* de Convergência foi calculado segundo os critérios estabelecidos por (SPRONCK, 2005) e discutidos na Seção 4.1.

Após análise dos resultados dos dois algoritmos verifica-se que cada combate do *Dynamic Scripting* Padrão dura em média 53.4 segundos e cada combate do algoritmo sugerido dura em média 46.04 segundos, uma redução de 7.36 segundos (13,78%).

A média do ponto de convergência alcançado no *Dynamic Scripting* Padrão é de 35 *rounds*, enquanto no algoritmo sugerido foi de 21 *rounds*, uma redução de 40%.

O tempo gasto até alcançar a convergência, em média, no algoritmo proposto é de 966.84 segundos contra 1869 segundos do *Dynamic Scripting* Padrão. Uma redução de 48,26%.

Através dos dados apresentados, verifica-se que o algoritmo sugerido ganha do algoritmo padrão do *Dynamic Scripting* em velocidade e tempo.

[1] Arquivo de texto produzido por um software contendo informações auxiliares (VALDMAN, 2001).

6 CONCLUSÃO

Neste capítulo é apresentada uma síntese das conclusões obtidas no desenvolvimento do trabalho juntamente a sugestões para os trabalhos futuros.

6.1 CONSIDERAÇÕES FINAIS

Neste trabalho foram discutidos pontos do *Dynamic Scripting* que podem ser melhorados, aumentando a eficácia e eficiência da técnica. Através do algoritmo sugerido foi observado que o tratamento destes pontos acrescenta à técnica e permite um aprendizado mais veloz dos Agentes de Inteligência Artificial. Apesar da especificidade do tratamento devido ao ambiente de testes ao qual foi aplicado, abre-se um precedente para a análise e tratamento do *Dynamic Scripting* também em outros meios. Uma vez que se identifique os estados em que a técnica é menos eficiente, pode-se aplicar um tratamento adequado ao ambiente e contexto.

Durante o desenvolvimento percebeu-se a dificuldade em trabalhar com o NWScript e o Aurora Toolset. A dificuldade se deu, predominantemente, com relação à implementação de novo código e no *feedback* enviado pelo Toolset, devido às poucas fontes de consulta disponíveis e a uma comunidade inativa. Através da documentação da linguagem (NWN Lexicon, 2002) e de estudos do código criado por (SPRONCK et al., 2006) foi possível solucionar os problemas existentes e atingir os objetivos propostos.

Ao final do trabalho tem-se como resultado um algoritmo, aplicado ao *Dynamic Scripting*, capaz de melhorar e acelerar o aprendizado de Agentes de IA. Os resultados obtidos são referentes à sua aplicação no jogo *Neverwinter Nights*, porém é possível adaptá-lo para qualquer outro jogo que possa ser dividido em estados.

6.2 TRABALHOS FUTUROS

A partir dos resultados apresentados neste trabalho o *Dynamic Scripting* torna-se uma técnica ainda mais convidativa para o uso em jogos.

Como proposta para outros projetos, sugere-se a aplicação do *Dynamic Scripting* e do algoritmo proposto em outros estilos de jogos, onde o aprendizado do agente requeira velocidade e consistência.

Uma melhoria que pode ser realizada é a alteração das táticas a serem substituídas pelo algoritmo. Neste trabalho optou-se por uma tática ofensiva, visando encerrar o combate rapidamente. O uso de diferentes táticas, porém, pode criar resultados interessantes e que sejam benéficos para a técnica.

REFERÊNCIAS

ARMSTRONG, W. et al. Dynamic Algorithm Selection Using Reinforcement Learning. *2006 International Workshop on Integrating AI and Data Mining*, p. 18–25, 2006. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4030708>. Citado 2 vezes nas páginas 20 e 21.

BATEMAN, C. M. *Game writing : narrative skills for videogames*. Cengage Learning; 1 edition (July 3, 2006), 2007. xxvii, 308 p. p. ISBN 1584504900 (pbk. alk. paper)\r9781584504900 (pbk. alk. paper). Disponível em: <http://www.loc.gov/catdir/toc/ecip0614/2006017150.html>. Citado na página 14.

BIOWARE. *Neverwinter Nights*. 2000. Disponível em: <http://nwn.bioware.com/>. Citado 2 vezes nas páginas 14 e 24.

CAMPONOGARA, E.; TR, C. D. Aprendizagem por Reforço : Uma Primeira Introdução. 2005. Citado na página 21.

CHAMPANDARD, A. J. *AI Game Development: Synthetic Creatures with Learning and Reactive Behaviors*. New Riders, 2003. (NRG Series). ISBN 9781592730049. Disponível em: <https://books.google.com.br/books?id=ZpuR8GnBSGcC>. Citado na página 18.

CHAMPANDARD, A. J. *Challenges in AI for Games: Speaker Spotlight for Game/AI Conf. 2014*. 2014. Disponível em: <http://aigamedev.com/open/upcoming/spotlight-challenges-2014/>. Citado na página 15.

CHARLES, D.; MCGLINCHEY, S. *The Past, Present and Future of Artificial Neural Networks in Digital Games*. 2004. Citado na página 19.

DOZZI, A.; DANIEL, F. *Desenho Técnico*. São Paulo: Escola de Engenharia Mackenzie, 1987. Citado na página 24.

GAMEINFORMER. *Researcher From Duke University Creates Mind-Controlled Pong*. 2013. Disponível em: <http://www.gameinformer.com/b/news/archive/2013/06/23/researcher-from-duke-creates-mind-controlled-pong.aspx>. Citado na página 13.

GOSAVI, A. A Tutorial for Reinforcement Learning. *Science And Technology*, n. 716, p. 1–12, 2009. Disponível em: <http://web.mst.edu/{~}gosavia/tutorial.pdf>. Citado na página 17.

JONG, T. de; FERGUSON-HESSLER, M. G. *Types and qualities of knowledge*. 1996. 105–113 p. Citado na página 17.

KOLLAR, I.; FISCHER, F.; HESSE, F. W. Collaboration Scripts – A Conceptual Analysis. *Educational Psychology Review*, v. 18, n. 2, p. 159–185, 2006. ISSN 1573-336X. Disponível em: <http://dx.doi.org/10.1007/s10648-006-9007-2>. Citado na página 17.

LIDÉN, L.; EINSTEIN, A. Artificial Stupidity : The Art of Intentional Mistakes. *AI Game Programming Wisdom*, v. 2, n. 5, p. 41–48, 2003. ISSN 00010782. Citado na página 20.

LUDWIG, J. R.; OREGON, U. of. *Extending Dynamic Scripting*. University of Oregon, 2008. ISBN 9781109016192. Disponível em: <https://books.google.com.br/books?id=TOXkuZ6OYW4C>. Citado na página 20.

MILLINGTON, I.; FUNGE, J. *Artificial Intelligence for Games, Second Edition*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. 2009, 2009. 895 p. ISSN 0269-8889. ISBN 9780123747310. Citado 3 vezes nas páginas 18, 19 e 20.

MIRVIS, P. H.; CSIKSZENTMIHALYI, M.; CSIKZENTMIHALY, M. Flow: The Psychology of Optimal Experience. *Academy of Management Review*, v. 16, n. 3, p. 636–640, 1991. ISSN 0363-7425. Citado na página 14.

NWN Lexicon. *Neverwinter Nights Lexicon*. 2002. Disponível em: <http://www.nwnlexicon.com/index.php?title=Main{_}Page>. Citado 2 vezes nas páginas 23 e 41.

NWN Wiki. *Neverwinter Nighs Wiki*. 2005. Disponível em: <http://nwn.wikia.com/wiki/Toolset>. Citado na página 23.

NWNCOMMUNITY. *Neverwinter Nights Community Site*. 2002. Disponível em: <http://www.neverwinternights.info/>. Citado na página 24.

OGATA, K. *Engenharia de Controle Moderno*. 5ª edição. ed. New York: Pearson, 2010, 2010. 912 p. ISBN 9788576058106. Citado na página 17.

RABIN, S. *Game AI Pro: Collected Wisdom of Game AI Professionals*. Taylor & Francis, 2013. ISBN 9781466565968. Disponível em: <https://books.google.com.br/books?id=jDb6AwAAQBAJ>. Citado na página 18.

RAJU et al. Artificial Intelligence in Games. *International Journal of Soft Computing & Engineering*, n. International Journal of Soft Computing & Engineering, 2012. Disponível em: <http://www.ijsce.org/attachments/File/v2i5/E1050102512.pdf>. Citado na página 19.

ROMANYCIA, M. H. J.; PELLETIER, F. J. What is a heuristic? *Computational Intelligence*, v. 1, n. 1, p. 47–58, 1985. ISSN 0824-7935. Disponível em: <http://doi.wiley.com/10.1111/j.1467-8640.1985.tb00058.x>. Citado na página 18.

RUSSELL, J.; NORVIG, P. *Artificial Intelligence: A Modern Approach*. Pearson; 3 edition (December 11, 2009), 2003. 1132 p. ISSN 00206539. ISBN 0137903952. Disponível em: <http://amazon.de/o/ASIN/0130803022/>. Citado 2 vezes nas páginas 13 e 17.

SHAFER, J. *The Recipe for Good AI*. 2012. Disponível em: <http://www.gamasutra.com/blogs/JonShafer/20120910/177436/The{_}Recipe{_}for{_}Good{_}AI.php>. Citado na página 14.

SPRONCK, P. *Adaptive Game AI*. Tese (Doutorado), 2005. Disponível em: <http://www.cs.unimaas.nl/p.spronck/Pubs/ThesisSpronck.pdf>. Citado 7 vezes nas páginas 14, 19, 20, 21, 22, 25 e 40.

SPRONCK, P. et al. Adaptive game AI with dynamic scripting. *Machine Learning*, v. 63, n. 3, p. 217–248, 2006. ISSN 08856125. Citado 3 vezes nas páginas 25, 35 e 41.

SPRONCK, P.; SPRINKHUIZEN-KUYPER, I.; POSTMA, E. Difficulty scaling of game AI. *Proceedings of the 5th International Conference on Intelligent Games and Simulation (GAMEON'2004)*, p. 33–37, 2004. Disponível em: <http://www.dcc.ru.nl/{~}idak/publications/papers/SpronckGAMEON2004.pdf>. Citado na página 18.

SUTTON, R.; BARTO, A. Reinforcement Learning: An Introduction. *IEEE Transactions on Neural Networks*, v. 9, n. 5, p. 1054–1054, 1998. ISSN 1045-9227. Disponível em: <http://dl.acm.org/citation.cfm?id=551283$\delimiter"026E30F$nhttp://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=712192>. Citado 2 vezes nas páginas 20 e 21.

THAWONMAS, R.; OSAKA, S. A method for online adaptation of computer-game AI rulebase. *International Conference on Advances in Computer Entertainment Technology 2006*, p. 16, 2006. Disponível em: <http://portal.acm.org/citation.cfm?doid=1178823.1178843>. Citado na página 22.

VALDMAN, J. Log file analysis. *Department of Computer Science and Engineering (FAV UWB)., Tech. Rep. DCSE/TR-2001-04*, 2001. Disponível em: <http://www.kiv.zcu.cz/vyzkum/publikace/technicke-zpravy/2001/tr-2001-04.pdf>. Citado na página 40.

VRANCX, P. *Decentralised Reinforcement Learning in Markov Games*. VUB University Press, 2011. ISBN 9789054877158. Disponível em: <https://books.google.com.br/books?id=e0cD6N7SXUYC>. Citado na página 17.

WEINBERGER, A. *Scripts for Computer-Supported Collaborative Learning*. Tese (Doutorado), 2003. Disponível em: <http://nbn-resolving.de/urn:nbn:de:bvb:19-11206>. Citado na página 17.

WEXLER, J. Artificial Intelligence in Games: A look at the smarts behind Lionhead Studio's "Black and White" and where it can and will go in the future. *Spring Simulation Multiconference*, 2008. Disponível em: <http://books.google.com/books?hl=en{&}lr={&}id=O06LoDbb48kC{&}oi=fnd{&}pg=PA98{&}dq=Artificial+Intelligence+in+Games+Evolution{&}ots=QBpB05LioY{&}sig=gxXGUkmp0qEdqRWYE9Y7Dv7HBTU>. Citado 2 vezes nas páginas 13 e 14.

WIZARDS. *Advanced Dungeons and Dragons*. 1995. Disponível em: <http://dnd.wizards.com/>. Citado na página 24.

WOODCOCK, S. AI Roundtables Moderators' Report. In: . San Jose, California: Game Developers Conference, May 17, 2000. Disponível em: <http://www.gameai.com/cgdc00notes.html>. Citado na página 19.

APÊNDICE A CÓDIGO FONTE

A.1 SCRIPT *PSEUDOHEARTBEAT*

```
#include "nw_o0_itemmaker"
void PseudoHeartbeat();

void main()
{
        ActionSpeakString("Script ativado.");
        PseudoHeartbeat();
}

void PseudoHeartbeat()
{
        // oWhite e o oponente branco
        object oWhite = GetNearestObjectByTag( "BLANCHE" );
        // oBlack e o oponente preto
        object oBlack = GetNearestObjectByTag( "NERA" );
        object oSign;
        string classAlive = "";
        string classAliveB = "";
        int iSum = 0; // Soma da vida de todos os membros do time branco
        // Round atual
        int nRound = GetLocalInt( GetObjectByTag( " IGOR " ), "ROUND_NUMBER" );
        int  nNumWhiteAlive = 0;
        int  nNumBlackAlive = 0;
        int  WhiteMembers; // Numero de brancos vivos no momento
        int  BlackMembers; // Numero de pretos vivos no momento
        object oTarget = GetObjectByTag( "ArcheryTarget" );
        if (!GetIsObjectValid( oTarget ))
                return;

        int nHeartbeat = GetLocalInt( OBJECT_SELF, "heartbeatCount" );
        // Conta o numero de heartbeats da batalha (cada heartbeat e 1 segundo)
        nHeartbeat++;
        SetLocalInt( OBJECT_SELF, "heartbeatCount", nHeartbeat );

        {
                int i = 1;
                while (GetIsObjectValid( oWhite ))
                {
                if (GetResRef(oWhite) == "blanche004")
                        classAlive = classAlive + "G";
                if (GetResRef(oWhite) == "blanche010")
                        classAlive = classAlive + "L";
                if (GetResRef(oWhite) == "blanche011")
                        classAlive = classAlive + "C";
                if (GetResRef(oWhite) == "blanche013")
                        classAlive = classAlive + "M";

        if (i==1)
                SetLocalInt( OBJECT_SELF, "CURRENTHP_P1", GetCurrentHitPoints( oWhite ) );
```

```
else if (i==2)
        SetLocalInt( OBJECT_SELF, "CURRENTHP_P2", GetCurrentHitPoints( oWhite ) );
else if (i==3)
        SetLocalInt( OBJECT_SELF, "CURRENTHP_P3", GetCurrentHitPoints( oWhite ) );
else if (i==4)
        SetLocalInt( OBJECT_SELF, "CURRENTHP_P4", GetCurrentHitPoints( oWhite ) );
iSum += GetCurrentHitPoints( oWhite );  //TEAM_HITPOINTS_REMAINING
++i;
// Passa para o proximo personagem do time branco
oWhite = GetNearestObjectByTag( "BLANCHE", OBJECT_SELF, i );
        }
nNumWhiteAlive = i-1;
WhiteMembers = nNumWhiteAlive; // TEAM_MEMBERS_REMAINING
}

{
        int iSumB = 0; // Soma da vida de todos os membros do time preto
        int i = 1;
        while (GetIsObjectValid( oBlack ))
        {
        if (GetResRef(oBlack) == "nera004")
                classAliveB = classAliveB + "G";
        if (GetResRef(oBlack) == "nera010")
                classAliveB = classAliveB + "L";
        if (GetResRef(oBlack) == "nera011")
                classAliveB = classAliveB + "C";
        if (GetResRef(oBlack) == "nera013")
                classAliveB = classAliveB + "M";

        iSumB += GetCurrentHitPoints( oBlack );
        ++i;
        // Passa para o proximo personagem do time preto
        oBlack = GetNearestObjectByTag( "NERA", OBJECT_SELF, i );
        }
nNumBlackAlive = i-1;
BlackMembers = nNumBlackAlive;
}

// Estados Positivos
if (WhiteMembers == 4 && BlackMembers == 3 && GetLocalInt(oTarget, "state1")==0)
{
        PrintString("4-3");
        PrintString(classAlive);
        PrintString(classAliveB);
        PrintString(IntToString (nHeartbeat));
        SetLocalInt(oTarget, "state1", 1);
        SetLocalInt(oTarget, "EstadoCritico", 0);
}

if (WhiteMembers == 4 && BlackMembers == 2 && GetLocalInt(oTarget, "state2")==0)
{
        PrintString("4-2");
        PrintString(classAlive);
        PrintString(classAliveB);
        PrintString(IntToString (nHeartbeat));
        SetLocalInt(oTarget, "state2", 1);
        SetLocalInt(oTarget, "EstadoCritico", 0);
```

```
}

// ESTADO CRITICO
if (WhiteMembers == 4 && BlackMembers == 1 && GetLocalInt(oTarget, "state3")==0)
{
        PrintString("4-1");
        PrintString(classAlive);
        PrintString(classAliveB);
        PrintString(IntToString (nHeartbeat));
        SetLocalInt(oTarget, "state3", 1);
        SetLocalInt(oTarget, "EstadoCritico", 1);
        ActionSpeakString("Estado Critico: 4-1");
}

if (WhiteMembers == 4 && BlackMembers == 0 && GetLocalInt(oTarget, "state4")==0)
{
        PrintString("4-0");
        PrintString(classAlive);
        PrintString(classAliveB);
        PrintString(IntToString (nHeartbeat));
        SetLocalInt(oTarget, "state4", 1);
        SetLocalInt(oTarget, "EstadoCritico", 0);
}

if (WhiteMembers == 3 && BlackMembers == 2 && GetLocalInt(oTarget, "state5")==0)
{
        PrintString("3-2");
        PrintString(classAlive);
        PrintString(classAliveB);
        PrintString(IntToString (nHeartbeat));
        SetLocalInt(oTarget, "state5", 1);
        SetLocalInt(oTarget, "EstadoCritico", 0);
}

// ESTADO CRITICO
if (WhiteMembers == 3 && BlackMembers == 1 && GetLocalInt(oTarget, "state6")==0)
{
        PrintString("3-1");
        PrintString(classAlive);
        PrintString(classAliveB);
        PrintString(IntToString (nHeartbeat));
        SetLocalInt(oTarget, "state6", 1);

        if      ((classAlive == "CML" && classAliveB == "C") ||
                (classAlive == "CLM" && classAliveB == "C") ||
                (classAlive == "LMC" && classAliveB == "C") ||
                (classAlive == "LCM" && classAliveB == "C") ||
                (classAlive == "MLC" && classAliveB == "C") ||
                (classAlive == "MCL" && classAliveB == "C"))
        {
                SetLocalInt(oTarget, "EstadoCritico", 1);
                ActionSpeakString("Entrou Estado Critico: 3-1");
        }
}

if (WhiteMembers == 3 && BlackMembers == 0 && GetLocalInt(oTarget, "state7")==0)
{
```

```
        PrintString("3-0");
        PrintString(classAlive);
        PrintString(classAliveB);
        PrintString(IntToString (nHeartbeat));
        SetLocalInt(oTarget, "state7", 1);
        SetLocalInt(oTarget, "EstadoCritico", 0);
}

// ESTADO CRITICO
if (WhiteMembers == 2 && BlackMembers == 1 && GetLocalInt(oTarget, "state8")==0)
{
        PrintString("2-1");
        PrintString(classAlive);
        PrintString(classAliveB);
        PrintString(IntToString (nHeartbeat));
        SetLocalInt(oTarget, "state8", 1);

        if      ((classAlive == "LC" && classAliveB == "C") ||
                (classAlive == "CL" && classAliveB == "C") ||
                (classAlive == "LC" && classAliveB == "M") ||
                (classAlive == "CL" && classAliveB == "M") ||
                        (classAlive == "ML" && classAliveB == "C") ||
                (classAlive == "LM" && classAliveB == "C") ||
                        (classAlive == "LG" && classAliveB == "C") ||
                (classAlive == "GL" && classAliveB == "C") ||
                        (classAlive == "LG" && classAliveB == "L") ||
                (classAlive == "GL" && classAliveB == "L") ||
                        (classAlive == "CG" && classAliveB == "C") ||
                (classAlive == "GC" && classAliveB == "C"))
        {
                SetLocalInt(oTarget, "EstadoCritico", 1);
                ActionSpeakString("Entrou Estado Critico: 2-1");
        }
}

if (WhiteMembers == 2 && BlackMembers == 0 && GetLocalInt(oTarget, "state9")==0)
{
        PrintString("2-0");
        PrintString(classAlive);
        PrintString(classAliveB);
        PrintString(IntToString (nHeartbeat));
        SetLocalInt(oTarget, "state9", 1);
        SetLocalInt(oTarget, "EstadoCritico", 0);
}

if (WhiteMembers == 1 && BlackMembers == 0 && GetLocalInt(oTarget, "state10")==0)
{
        PrintString("1-0");
        PrintString(classAlive);
        PrintString(classAliveB);
        PrintString(IntToString (nHeartbeat));
        SetLocalInt(oTarget, "state10", 1);
        SetLocalInt(oTarget, "EstadoCritico", 0);
}

// Estados Negativos
if (BlackMembers == 4 && WhiteMembers == 3 && GetLocalInt(oTarget, "state11")==0)
```

```
{
        PrintString("3-4");
        PrintString(classAlive);
        PrintString(classAliveB);
        PrintString(IntToString (nHeartbeat));
        SetLocalInt(oTarget, "state11", 1);
        SetLocalInt(oTarget, "EstadoCritico", 0);
}

if (BlackMembers == 4 && WhiteMembers == 2 && GetLocalInt(oTarget, "state12")==0)
{
        PrintString("2-4");
        PrintString(classAlive);
        PrintString(classAliveB);
        PrintString(IntToString (nHeartbeat));
        SetLocalInt(oTarget, "state12", 1);
        SetLocalInt(oTarget, "EstadoCritico", 0);
}

if (BlackMembers == 4 && WhiteMembers == 1 && GetLocalInt(oTarget, "state13")==0)
{
        PrintString("1-4");
        PrintString(classAlive);
        PrintString(classAliveB);
        PrintString(IntToString (nHeartbeat));
        SetLocalInt(oTarget, "state13", 1);
        SetLocalInt(oTarget, "EstadoCritico", 0);
}

if (BlackMembers == 4 && WhiteMembers == 0 && GetLocalInt(oTarget, "state14")==0)
{
        PrintString("0-4");
        PrintString(classAlive);
        PrintString(classAliveB);
        PrintString(IntToString (nHeartbeat));
        SetLocalInt(oTarget, "state14", 1);
        SetLocalInt(oTarget, "EstadoCritico", 0);
}

if (BlackMembers == 3 && WhiteMembers == 2 && GetLocalInt(oTarget, "state15")==0)
{
        PrintString("2-3");
        PrintString(classAlive);
        PrintString(classAliveB);
        PrintString(IntToString (nHeartbeat));
        SetLocalInt(oTarget, "state15", 1);
        SetLocalInt(oTarget, "EstadoCritico", 0);
}

// ESTADO CRITICO
if (BlackMembers == 3 && WhiteMembers == 1 && GetLocalInt(oTarget, "state16")==0)
{
        PrintString("1-3");
        PrintString(classAlive);
        PrintString(classAliveB);
        PrintString(IntToString (nHeartbeat));
        SetLocalInt(oTarget, "state16", 1);
```

```
        if      ((classAlive == "C" && classAliveB == "GMC") ||
                (classAlive == "C" && classAliveB == "GCM") ||
                        (classAlive == "C" && classAliveB == "CMG") ||
                (classAlive == "C" && classAliveB == "CGM") ||
                        (classAlive == "C" && classAliveB == "MCG") ||
                (classAlive == "C" && classAliveB == "MGC"))
        {
                SetLocalInt(oTarget, "EstadoCritico", 1);
                ActionSpeakString("Entrou Estado Critico: 1-3");
        }
}

if (BlackMembers == 3 && WhiteMembers == 0 && GetLocalInt(oTarget, "state17")==0)
{
        PrintString("0-3");
        PrintString(classAlive);
        PrintString(classAliveB);
        PrintString(IntToString (nHeartbeat));
        SetLocalInt(oTarget, "state17", 1);
        SetLocalInt(oTarget, "EstadoCritico", 0);
}

if (BlackMembers == 2 && WhiteMembers == 1 && GetLocalInt(oTarget, "state18")==0)
{
        PrintString("1-2");
        PrintString(classAlive);
        PrintString(classAliveB);
        PrintString(IntToString (nHeartbeat));
        SetLocalInt(oTarget, "state18", 1);
}

if (BlackMembers == 2 && WhiteMembers == 0 && GetLocalInt(oTarget, "state19")==0)
{
        PrintString("0-2");
        PrintString(classAlive);
        PrintString(classAliveB);
        PrintString(IntToString (nHeartbeat));
        SetLocalInt(oTarget, "state19", 1);
        SetLocalInt(oTarget, "EstadoCritico", 0);
}

if (BlackMembers == 1 && WhiteMembers == 0 && GetLocalInt(oTarget, "state20")==0)
{
        PrintString("0-1");
        PrintString(classAlive);
        PrintString(classAliveB);
        PrintString(IntToString (nHeartbeat));
        SetLocalInt(oTarget, "state20", 1);
        SetLocalInt(oTarget, "EstadoCritico", 0);
}

// Estados de Empate
if (WhiteMembers == 3 && BlackMembers == 3 && GetLocalInt(oTarget, "stateE1")==0)
{
        PrintString("3-3");
        PrintString(classAlive);
```

```
                PrintString(classAliveB);
                PrintString(IntToString (nHeartbeat));
                SetLocalInt(oTarget, "stateE1", 1);
                SetLocalInt(oTarget, "EstadoCritico", 0);
        }

        if (WhiteMembers == 2 && BlackMembers == 2 && GetLocalInt(oTarget, "stateE2")==0)
        {
                PrintString("2-2");
                PrintString(classAlive);
                PrintString(classAliveB);
                PrintString(IntToString (nHeartbeat));
                SetLocalInt(oTarget, "stateE2", 1);
                SetLocalInt(oTarget, "EstadoCritico", 0);
        }

        // ESTADO CRITICO
        if (WhiteMembers == 1 && BlackMembers == 1 && GetLocalInt(oTarget, "stateE3")==0)
        {
                PrintString("1-1");
                PrintString(classAlive);
                PrintString(classAliveB);
                PrintString(IntToString (nHeartbeat));
                SetLocalInt(oTarget, "stateE3", 1);

                if      ((classAlive == "M" && classAliveB == "C") ||
                        (classAlive == "G" && classAliveB == "C") ||
                        (classAlive == "C" && classAliveB == "L") ||
                        (classAlive == "C" && classAliveB == "M") ||
                        (classAlive == "C" && classAliveB == "G"))
                {
                        SetLocalInt(oTarget, "EstadoCritico", 1);
                        ActionSpeakString("Entrou Estado Critico: 1-1");
                }
        }

        classAlive = "";
        // Recursao. Depois de 1 segundo, a funcao sera chamada novamente
        DelayCommand(1.0, PseudoHeartbeat());
}
```