Improving Dynamic Scripting for Adaptive Game AI with a Tactic Replacement Algorithm

Lucas Izumi de Oliveira[1], **Cristiano Grijó Pitangui**[2], **Alessandro Vivas Andrade**[1], **Luciana Pereira de Assis**[1], **Cristiano Maciel da Silva**[2]

[1]Universidade Federal dos Vales do Jequitinhonha e Mucuri, Minas Gerais, Brasil

[2]Universidade Federal de São João del-Rei, Minas Gerais, Brasil

lucasizumi@gmail.com, pitangui.cristiano@ufsj.edu.br,

alessandro.vivas@gmail.com, lupassis@gmail.com, cristiano@ufsj.edu.br

Abstract. *Artificial Intelligence (AI) plays an important role in digital 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 research consists on the analysis of an AAI technique known as Dynamic Scripting (DS) and in the development of a new algorithm (called Tactic Replacement) to improve it. Results show that, in comparison with the default DS algorithm, the proposed 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.*

Resumo. *A Inteligência Artificial (IA) possui um importante papel nos jogos digitais atuais. Com os jogadores cada vez mais exigentes, é vital fornecer 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. Esta pesquisa consiste na análise da técnica de IAA conhecida como Dynamic Scripting (DS) e no desenvolvimento de um novo algoritmo (chamado de Substituição de Tática) para aprimorá-la. Resultados experimentais mostram que, quando comparado com a técnica padrão de DS, o algoritmo proposto obtém uma redução de tempo de convergência de* $\approx$ *50% e reduz em 40% o número médio de rounds para a convergência.*

1. Introdução

A Inteligência Artificial (IA) é uma área de estudo que cobre muitos setores, como criação de poesia, demonstração de teoremas matemáticos, diagnósticos de doenças, simuladores de mercado, sistemas lógicos, dentre outros. A IA faz a sistematização e a automação de tarefas, tornando-se relevante para qualquer esfera da atividade intelectual humana [Russell 2003]. Entre essas áreas de interesse, destaca-se a de Jogos Digitais, que vem ganhando cada vez mais atenção [Wexler 2008].

A primeira vez que uma IA foi usada em um jogo digital foi com Pong (1972) [Wexler 2008]. Pong é um jogo que simula uma partida de tênis de mesa e consiste em duas barras, uma em cada lado da tela, e uma bola. O objetivo é bater na bola com as barras e fazê-la atravessar o campo do adversário.

A IA utilizada em Pong controla uma das barras e sempre tenta impedir o jogador de marcar um ponto, bloqueando a bola e enviando-a de volta ao campo adversário.

A IA determina para onde mover a barra usando uma equação simples que calcula com precisão o quão alto a bola iria cruzar seu campo. Em seguida, ela move a barra para esse ponto tão rápido quanto lhe for permitido [Wexler 2008]. Com base no nível de dificuldade selecionado (controlado pelo jogador), a velocidade da IA é alterada, o que afeta se a barra atinge ou não o ponto calculado em um determinado tempo. Há também a chance da barra mover-se na direção errada com certa probabilidade.

Durante muito tempo, a IA em jogos não era muito mais esperta que a IA do jogo Pong. Isto devido à simplicidade dos jogos e ao fato de que eles eram jogados, na maior parte do tempo, contra outro jogador humano. O uso da IA nos jogos evoluiu à medida que os jogos se tornaram mais complexos.

Criar uma boa IA não é uma tarefa fácil. Mesmo que uma IA inteligente seja desenvolvida, no sentido de que ela faz as melhores escolhas em uma determinada situação, isso não significa que ela é uma boa IA. Tudo o que importa é o que o jogador vê e acredita [Shafer 2012]. Este senso de credibilidade do mundo do jogo, transmitido ao jogador, está diretamente relacionado à imersão que um jogo pode proporcionar.

Imersão é uma condição psicológica em que uma pessoa tem toda a atenção focada em uma única atividade e seus sentimentos canalizados para esse ponto particular. Imersão define o quão confortável e envolvido um jogador se sente ao participar no contexto de um jogo. É o momento em que o jogador começa a se sentir "dentro"do jogo que lhe está sendo apresentado [Bateman 2007].

A fim de atingir este nível de imersão, os elementos do jogo precisam ser muito bem construídos, de modo que todos os aspectos do mundo do jogo sejam verdadeiros à natureza que eles devem representar. Um desses elementos é a Inteligência Artificial. Se um jogo tem qualquer agente de IA, é necessário trabalhar com cuidado para que este elemento não se destaque em relação ao que o jogo quer oferecer.

Um agente de IA é considerado crível se ele puder oferecer imersão para o jogador, desempenhando seu papel de uma forma que não tire a atenção do foco principal do jogo. O realismo de um agente de IA é conseguido quando este se comporta de maneira plausível, de uma forma que pareça logicamente possível ao jogador [Champandard 2003]. É até possível criar um agente de IA com comportamento completamente aleatório, desde 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]. Nesse sentido, pesquisadores e *designers* de IA estudam e implementam novas técnicas para aumentar o poder da IA nos jogos.

Para criar uma IA mais forte e que gere uma experiência satisfatória para o jogador, Spronck [Spronck et al. 2004] sugeriu o uso dos chamados algoritmos de Aprendizado de Máquina [Mitchell 1997]. Em resumo, os algoritmos de Aprendizado de Máquina visam melhorar o desempenho de um sistema em uma determinada tarefa. O processo de aprendizagem pode acontecer enquanto o jogo está sendo jogado (*online*) ou usando dados coletados anteriormente [Muñoz-Avila et al. 2013].

Várias técnicas de Aprendizado de Máquina têm sido usadas para melhorar a IA

de jogos, e *Dynamic Scripting* (DS) [Spronck 2005] é uma destas. De forma geral, DS permite que os agentes de IA aprendam e se adaptem em tempo real, com grandes resultados em eficiência e eficácia [Spronck 2005]. Ao usar o jogo *Neverwinter Nights* [Bioware 2000] como um ambiente de simulação, Spronck [Spronck 2005] demonstrou que os resultados do DS podem ser reproduzidos em um jogo comercial.

Nesse sentido, a presente pesquisa apresenta um algoritmo para melhorar a técnica de DS. Para isso, realizou-se um estudo intensivo sobre o comportamento dos agentes de IA sob o efeito do DS no jogo *Neverwinter Nights*. O algoritmo proposto, denominado de Substituição de Tática (ST), utiliza o conceito de estados para identificar e melhorar situações em que o DS não é eficiente. Resultados experimentais mostraram que a aplicação do algoritmo ST reduziu o tempo necessário para alcançar a convergência em $\approx 50\%$ e melhorou a eficiência (número médio de *rounds* para a convergência) do DS em 40%.

Este trabalho se organiza como segue. A Seção 2 apresenta uma visão geral do DS. A Seção 3 explica a metodologia adotada neste trabalho para analisar a técnica de DS e para desenvolver o algoritmo de Substituição de Tática, que é discutido em detalhes na Seção 4. A Seção 5 apresenta os resultados experimentais e, finalmente, a Seção 6 apresenta as conclusões e discute trabalhos futuros.

2. Dynamic Scripting (DS)

Dynamic Scripting (DS) é uma técnica de Aprendizado por Reforço (AR) *online* para IA de jogos [Spronck 2005].

AR é uma técnica de Aprendizado de Máquina que visa mapear estados à ações [Sutton and Barto 1998]. Basicamente, a técnica é baseada em um agente que percebe seu ambiente e age sobre ele. As ações que o agente realiza afetam o estado do ambiente e geram recompensas, que são responsáveis por medir o desempenho do agente em uma determinada tarefa. O principal objetivo do agente é maximizar os valores das recompensas recebidas [Armstrong et al. 2006]. O agente do AR aprende por tentativa e erro. Em suma, dado um estado do ambiente, o agente escolhe uma ação e a executa. Como resultado desta ação, o agente muda para outro estado (ou permanece no mesmo estado) e recebe uma recompensa. Repetindo este processo, o agente eventualmente aprende quais são as melhores ações a tomar para receber as maiores recompensas [Sutton and Barto 1998].

O mecanismo de aprendizagem no DS é inspirado nas técnicas de AR. Técnicas habituais de AR não são suficientemente eficientes para a aprendizagem *online* em jogos, porque precisam de uma grande quantidade de tentativas a fim de produzir resultados [Rabin 2002]. Nos casos em que as tentativas ocorrem num curto intervalo de tempo, o AR é adequado. No entanto, quando uma tentativa toma um longo período de tempo, o AR demora muito tempo para ser particularmente eficiente. O DS, por outro lado, é capaz de aprender a partir de poucas tentativas [Spronck 2005] através do uso de *Scripts* tradicionais de IA, que limitam o tamanho do espaço de estados [Spronck 2005].

Scripts convencionais de IA para jogos consistem em uma sequência de regras. Cada uma dessas regras consiste em duas partes: uma parte condicional, que identifica um ou mais estados de jogo; e uma parte de ação, que descreve a ação a ser tomada

uma vez que a parte condicional seja satisfeita [Spronck 2005]. Para atribuir uma ação a um agente de IA, as regras em um *script* são avaliadas sequencialmente com base nas condições que correspondam ao estado atual do jogo.

Dada a definição de *Scripts* de IA para jogos, nota-se que eles não apresentam características adaptativas. Seu funcionamento é baseado em uma série de regras, já conhecidas, que respondem aos estados do jogo. Por outro lado, o DS adiciona a capacidade de explorar a representação do espaço de estados (que os *scripts* produzem) para uma aprendizagem rápida e eficiente, ao mesmo tempo em que confia no conteúdo baseado em *scripts* para garantir que todo o comportamento adaptativo seja plausível e eficaz [Spronck 2005].

O mecanismo básico do DS pode ser definido em cinco etapas, como segue [Thawonmas and Osaka 2006]:

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

O DS usa várias bases de dados, uma para cada tipo de agente. Cada vez que uma nova instância de um agente é criada, as bases de dados são usadas para gerar um *script* e atribuí-lo a esse agente recém-criado. Na base de dados, cada regra tem um peso (um valor numérico) que determina a probabilidade de que a regra seja selecionada para um *script*. O processo de adaptação acontece mudando os pesos das regras com base no *feedback* coletado: a falha ou sucesso do agente [Spronck 2005].

No final de cada encontro entre o jogador e os agentes de IA, calcula-se um valor de *fitness*, que representa a eficácia do comportamento dos agentes. Normalmente, o valor de *fitness* é um valor real entre 0 e 1. Nos casos em que há um time de agentes trabalhando juntos, os valores de *fitness* são calculados para a equipe como um todo e também para os agentes individualmente.

A *fitness* da equipe é usada para indicar a eficácia dos agentes trabalhando como um grupo e pode ser usada para mensurar como eles estão se comportando contra um jogador ou outros agentes de IA. Por sua vez, a *fitness* de um agente individual é usada para atualizar o peso das regras associadas a esse agente específico.

A função de atualização de peso altera o peso das regras nos *scripts*, com base no valor de *fitness* que foi gerado por uma função de avaliação adequada. Valores elevados de *fitness* aumentam os pesos das regras e baixos valores de *fitness* diminuem tais pesos. Usando esta abordagem, as regras que fazem os agentes funcionarem bem serão associadas aos pesos maiores, o que significa que essas regras serão selecionadas com maiores probabilidades nos próximos combates [Spronck 2005]. Dessa forma, os agentes controlados pelo DS se adaptarão e se sairão melhor contra um determinado jogador.

3. Metodologia de Desenvolvimento

Para desenvolver e testar o algoritmo de Substituição Tática sobre o *Dynamic Scripting*, utilizou-se o jogo *Neverwinter Nights* como ambiente de simulação. *Neverwinter Nights* é um jogo de RPG (*Roling Player Game*) em terceira pessoa desenvolvido pela BioWare [Bioware 2000]. O módulo utilizado foi o *Online Adaptation Neverwinter Nights*, versão 3 [Spronck et al. 2006]. O módulo consiste em um encontro e combate entre duas equipes de agentes com a mesma composição de membros. A razão para utilizar este jogo e este módulo específico como ambiente de simulação é replicar o ambiente usado por Spronck [Spronck 2005] durante suas experiências com o *Dynamic Scripting*.

O DS é aplicado no combate entre essas duas equipes, cada uma representada por uma cor: branco e preto. A equipe branca é a equipe de agentes controlados pelo DS, chamada equipe dinâmica. A equipe preta é controlada pela IA padrão do *Neverwinter Nights* (versão 1.61) e é chamada de equipe estática. Esta versão da IA do *Neverwinter Nights* foi usada nos testes experimentais de Spronck [Spronck 2005], o qual este trabalho usa como base. Cada equipe é composta por quatro agentes, cada um com uma classe diferente. As classes dos agentes são: Clérigo (*Cleric*), Mago (*Mage*), Guerreiro (*Warrior*) e Ladino (*Rogue*), todos com o mesmo nível de experiência.

Em *Neverwinter Nights*, um combate acontece quando duas ou mais equipes, formadas por um ou mais personagens, se enfrentam em batalha. Um combate só é dado como concluído quando apenas os membros de uma equipe permanecem vivos no final. A equipe de agentes que permaneceu viva no final é considerada a equipe vencedora.

No final de cada combate, um valor médio de *fitness* da equipe é calculado e comparado com a outra equipe. A *fitness* da equipe, além de indicar a eficácia dos agentes, ajuda a identificar o ponto de convergência do algoritmo.

O DS atinge o ponto de convergência quando a *fitness* média da equipe de agentes dinâmicos (equipe branca) é maior do que a *fitness* média da equipe estática (time preto) por dez combates consecutivos [Spronck 2005]. Assim, o ponto de convergência é o número do primeiro combate depois que a equipe dinâmica ultrapassa a equipe estática por pelo menos dez combates consecutivos.

Em testes realizados usando a IA do *Neverwinter Nights* (versão 1.61), Spronck [Spronck 2005] obteve resultados que provaram que o DS atende a todos os requisitos para ser aplicado em um jogo comercial real. O algoritmo atingiu o ponto de convergência com, em média, 35 combates. Cada teste, também chamado de iteração, consistia de 100 combates entre as equipes de agentes. Spronck [Spronck 2005] executou o DS 31 vezes, totalizando 3100 *rounds* de combate.

3.1. Usando Divisão por Estados para Analisar o DS

Para analisar em detalhes os efeitos do DS sobre os agentes, dividiu-se um combate em uma série de estados. O objetivo da divisão por estados é facilitar a identificação de pontos onde o DS não é eficiente e pode estar prejudicando seu próprio comportamento.

Definiu-se um estado como uma possível combinação de agentes vivos em um dado instante de tempo em um combate. Assim, um estado representa o número de agentes vivos nas equipes branca e preta em um dado momento. Optou-se por usar as primeiras letras das classes dos agentes (em inglês) para representar um estado, portanto,

C para *Cleric*, M para *Mage*, W para *Warrior* e R para *Rogue*. Por exemplo, se a equipe branca tem dois agentes vivos, um *Cleric* e um *Mage* e a equipe preta tem três agentes vivos, um *Warrior*, um *Rogue* e um *Cleric*, então a representação deste estado é dada por [2-3 CM-WRC].

Todos os combates começam no estado [4-4 CWRM-CWRM], já que todos os membros de ambas as equipes estão vivos. Assim, este estado é definido como o *Estado Inicial*. Um estado em que o número de agentes vivos na equipe branca é maior que o número de agentes vivos na equipe preta é definido como *Estado Positivo*. Sua contrapartida, onde o número de agentes pretos vivos é maior do que o número de agentes brancos vivos, é definido como *Estado Negativo*. Um estado em que, em qualquer equipe, o número de agentes vivos é igual a zero, é definido como *Estado Final*.

Um *Estado Não-Expandido* não exibe as combinações de classes de personagens vivos em ambas as equipes. Ele representa apenas o número de agentes vivos em ambos os lados. Assim, um *Estado Não-Expandido* representa o espaço de estados de uma maneira geral, apontando quais estados podem ser gerados a partir dele. Por exemplo, [4-3] é um *Estado Não-Expandido* que representa quatro membros vivos na equipe branca e três membros vivos na equipe preta. Este *Estado Não-Expandido*, uma vez expandido, pode gerar os seguintes *estados positivos*: [4-3 CWRM-CWR], [4-3 CWRM-CWM], [4-3 CWRM-CRM] ou [4-3 CWRM-WRM].

Como a representação de um estado já indica o número de agentes vivos e mortos em cada equipe, é necessário analisar o tempo que o DS permanece em cada um deles. Esta análise indica a eficiência do algoritmo sobre certos estados. Assim, para cada estado, calcula-se quanto tempo, em segundos, o algoritmo DS levou para passar para o próximo estado. Em outras palavras, calcula-se quanto tempo o algoritmo DS realmente permaneceu em cada estado. A Tabela 1 apresenta os resultados desta análise. Este resultado refere-se a cinquenta iterações[1] do algoritmo.

Tabela 1. Tempo médio gasto em cada estado *Não-Expandido*

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

Nota-se que nos estados mais próximos aos *Estados Finais*, o tempo de permanência aumenta consideravelmente. Esse fenômeno, no entanto, só acontece nos estados positivos, isto é, aqueles em que a equipe dinâmica está na vantagem. O estado [1-4], por exemplo, corresponde a uma configuração na qual há quatro agentes vivos na equipe estática e apenas um agente vivo na equipe dinâmica. O tempo médio gasto neste estado é 6.5s. Por outro lado, o estado [4-1] tem um tempo médio gasto de 17s, $\approx$ 3x o tempo gasto no estado [1-4].

[1]Acima de cinquenta iterações não foram identificadas mudanças significativas no tempo médio de permanência nos estados.

4. Algoritmo Proposto: Substituição de Tática (ST)

O algoritmo proposto, denominado Substituição de Tática (ST), tem como objetivo tratar os estados em que o algoritmo DS não é eficiente, isto é, os estados em que o DS permanece uma quantidade de tempo que pode ser considerada "alta" em relação ao tempo de permanência em outros estados.

Para cada estado, calculou-se o tempo médio de permanência do DS. Por exemplo, no ambiente de simulação usado neste trabalho (*Neverwinter Nights*), um combate começa sempre no estado [4-4 CMWR-CMWR]. Quando o combate é iniciado, um temporizador também se inicia. Uma vez que ocorra uma mudança de estado, que pode ser para os estados [4-3] ou [3-4], o temporizador pára e registra o tempo total de permanência no estado [4-4 CMWR-CMWR]. Este método é usado para calcular o tempo médio de permanência em todos os estados.

Com base em todos os tempos médios coletados, definiu-se um valor de corte para separar os estados nos quais o tempo médio de permanência é considerado aceitável e estados nos quais notou-se espaço para melhoria.

A partir dos estados apresentados na Tabela 1, selecioaram-se os estados com tempos iguais (ou superiores) a 13s. Este valor de tempo foi definido com base no tempo médio de permanência em todos os estados, onde tempos abaixo de 13s foram considerados aceitáveis[2].

Os estados selecionados foram então expandidos para exibir suas informações completas, isto é, o número de membros vivos em ambas as equipes e suas classes. O tempo gasto em cada um desses estados expandidos foi também calculado. Estados com tempos iguais ou superiores ao limite de 13s foram selecionados e denominados de *Estados Críticos*. Assim, um *Estado Crítico* é todo estado, em sua forma expandida, cujo tempo médio de permanência do DS é igual ou superior a 13s. Os 20 estados a seguir (com seus respectivos tempos de permanência entre parêntesis) são considerados *Estados Críticos*[3]: [4-1 CMWR-C] (23.5s), [3-1 CMR-C] (25.2s), [3-1 MWR-C] (25.2s), [3-1 CMW-C] (24.3s), [3-1 CWR-C] (25.5s), [2-1 MR-C] (34.2s), [2-1 MR-W] (18s), [2-1 RC-C] (32.1s), [2-1 RC-M] (17s), [2-1 CW-C] (22.6s), [2-1 RW-C] (27s), [2-1 RW-R] (27s), [1-3 C-WMC] (14s), [1-2 C-WC] (18s), [1-1 W-C] (34s), [1-1 M-C] (27.5s), [1-1 C-R] (28.5s), [1-1 C-W] (14.5s) [1-1 C-M] (21s) e [1-1 R-C] (25.5s).

Estados Críticos funcionam como um "gatilho" para o algoritmo de Substituição de Tática. Sempre que o ST detecta que o jogo entrou em um desses estados, uma nova tática é aplicada aos agentes. A tática escolhida para todas as classes de agentes é a mesma: realizar um ataque básico ao inimigo. Esta tática será identificada no restante deste trabalho como tática de "Foco no Ataque". Ela interrompe todas as ações (por exemplo, beber uma poção, lançar uma magia, mover-se para um local) que os agentes teriam quando controlados pelo DS, e os fazem atacar os inimigos restantes com o objetivo de finalizarem o combate o mais rapidamente possível.

[2]Outros valores de tempo foram testados e melhores resultados foram obtidos com o uso do "limite" de 13s.

[3]A média de tempo de um estado *Não-Expandido* é formada pela média de tempo de seus estados expandidos. Por esse motivo, um estado *Não-Expandido* que apresente 13s de média, por exemplo, pode conter um estado expandido com uma média de tempo bem maior que 13s e outro estado expandido com média de tempo bem menor que 13s.

O algoritmo ST é definido na Figura 1 e seu funcionamento é explicado a seguir:

1. A cada segundo, atualize a variável "EstadoCrítico" que indica se o estado atual é um *Estado Crítico* ou não.
2. Se o estado atual for um *Estado Crítico*, substitua a tática de combate do DS da equipe dinâmica (equipe branca) pela tática "Foco no Ataque". Caso contrário, retorne a tática de combate da equipe dinâmica ao DS padrão. Vá para 1.

Figura 1. Algoritmo de Substituição de Tática

O algoritmo de Substituição de Tática usa o algoritmo DS como base e atua modificando as táticas de combate da equipe dinâmica em situações onde o DS apresenta "baixa" eficiência.

A técnica de DS pode ser aplicada a qualquer IA de jogos que atenda a três requisitos [Spronck 2005]: i) a IA do jogo pode ser scriptada; ii) conhecimento de domínio sobre as características de um *script* bem-sucedido pode ser coletado; e iii) uma função de avaliação pode ser projetada para avaliar o sucesso do *script*. Dessa forma, o algoritmo ST pode ser aplicado a qualquer IA de jogos que atenda aos mesmos requisitos do DS.

5. Seção Experimental

5.1. Metodologia Experimental

A metodologia utilizada para avaliar experimentalmente o algoritmo de Substituição de Tática foi baseada em [Spronck 2005]. Assim, nos testes realizados, cada iteração do ST corresponde a 50 combates entre as equipes de agentes. Executou-se o algoritmo 110 vezes, totalizando, portanto, 5500 *rounds* de combate[4].

A versão do jogo usada para executar os testes foi a *Neverwinter Nights Diamond Edition* [Bioware 2000]. O módulo utilizado durante os testes foi o módulo *Online Adaptation Neverwinter Nights*, versão 3.

[4]A Metodologia de Spronck, discutida na Seção 3, também foi testada, mas com o uso da metodologia apresentada obteve-se uma maior variação de combates e, consequentemente, uma média mais precisa. Nota-se, ainda, que os resultados originais do *Dynamic Scripting* [Spronck 2005] não variaram muito usando a metodologia adotada. O ponto de convergência médio para o algoritmo DS, usando a configuração de 50 combates e 110 iterações, foi de 34 *rounds* de combate. [Spronck 2005], com a configuração de 100 combates e 31 iterações, obteve um ponto de convergência de 35 *rounds* de combate.

A duração de cada combate foi calculada usando um *script* onde um contador é iniciado quando um combate começa e uma contagem é realizada segundo a segundo. O *round* de convergência foi calculado de acordo com o critério estabelecido por Spronck [Spronck 2005] (explicado na seção 3).

Todas as análises são feitas em comparação com os resultados obtidos com o algoritmo do *Dynamic Scripting* de Spronck [Spronck et al. 2006] (referenciado agora por *Dynamic Scripting* Padrão (DSP)). Os resultados do algoritmo DS foram coletados por de 110 iterações, onde cada iteração corresponde a 50 *rounds* de combate. O algoritmo *Dynamic Scripting* com Substituição de Tática (DSST) foi executado usando as mesmas configurações, 110 iterações com 50 *rounds* de combate cada, para fins comparativos.

Realizou-se, inicialmente, a comparação entre o DSP e o DSST para verificar o tempo médio gasto em cada *Estado Crítico*. Os resultados são apresentados na Tabela 2.

Tabela 2. Comparação do tempo médio gasto em cada *Estado Crítico*

Estado	Tempo Médio (s) DSP	Tempo Médio (s) DSST	Estado	Tempo Médio (s) DSP	Tempo Médio (s) DSST
[4-1 CMWR-C]	23.5	19.3	[2-1 RW-C]	27	23.6
[3-1 CMR-C]	25.2	20.5	[2-1 RW-R]	27	7
[3-1 MWR-C]	25.2	27.7	[1-3 C-WMC]	14	20.5
[3-1 CMW-C]	24.3	33.8	[1-2 C-WC]	18	19.7
[3-1 CWR-C]	25.5	28.5	[1-1 W-C]	34	21.8
[2-1 MR-C]	34.2	25.6	[1-1 M-C]	27.5	30.5
[2-1 MR-W]	18	23	[1-1 C-R]	28.5	18
[2-1 RC-C]	32.1	31.5	[1-1 C-W]	15.5	15.7
[2-1 CW-C]	22.6	32.6	[1-1 R-C]	25.5	69.6
[2-1 RC-M]	17	5.5	[1-1 C-M]	21	14

Os resultados na Tabela 2 mostram que o algoritmo DSST diminuiu o tempo gasto em certos *Estados Críticos* (por exemplo, estado [2-1 RC-M]), mas aumentou em outros (por exemplo, estado [1-1 R-C]). Para identificar o impacto das mudanças no tempo total do combate, foi realizada uma análise da frequência com que as equipes de agentes entram nos *Estados Críticos*. Esta frequência é definida como número de vezes que cada *Estado Crítico* é visitado durante um combate.

Para analisar a relação entre o tempo gasto em cada *Estado Crítico* e a frequência deste estado, utilizou-se a Equação 1,

$$f(n) = (Tdsst_n - Tdsp_n) * F_n \tag{1}$$

onde: n é um *Estado Crítico*, $Tdsst_n$ é o tempo médio gasto no estado n usando o algoritmo DSST, $Tdsp_n$ é o tempo médio gasto no estado n usando o algoritmo DSP e F_n é a frequência do estado n.

A Equação 1 avalia o tempo de permanência em um *Estado Crítico* pelo DSST, em

comparação ao DSP. A equação calcula um valor que corresponde ao número de segundos que um determinado *Estado Crítico*, sendo tratado pelo DSST, acrescentou ou removeu do tempo de combate total. Caso o resultado da equação seja negativo para um dado *Estado Crítico*, significa que o uso do algoritmo diminuiu o tempo total de combate. De forma oposta, um valor positivo significa que o uso do algoritmo naquele estado aumentou o tempo total de combate.

A equação foi aplicada sobre os resultados de 110 iterações de 50 *rounds* cada (5500 *rounds* no total). Em 5500 *rounds*, o número total de estados visitados foi de $\approx$ 25000. A frequência dos *Estados Críticos* é obtida a partir destes resultados e é exibida em percentual ao lado de cada *Estado Crítico* para melhor visualização: [4-1 CMWR-C] (6.85%), [3-1 CMR-C] (1.04%), [3-1 MWR-C] (2.72%), [3-1 CMW-C] (0.84%), [3-1 CWR-C] (0.45%), [2-1 MR-C] (0.32%), [2-1 MR-W] (0.33%), [2-1 RC-C] (0.10%), [2-1 RC-M] (0.15%), [2-1 CW-C] (0.20%), [2-1 RW-C] (0.08%), [2-1 RW-R] (0.07%), [1-3 C-WMC] (0.39%), [1-2 C-WC] (0.26%), [1-1 W-C] (0.24%), [1-1 M-C] (0.08%), [1-1 C-R] (0.05%), [1-1 C-W] (0.03%) [1-1 C-M] (0.07%), [1-1 R-C] (0.13%).

Para se obter o valor de F_n, utilizado na equação (1), basta multiplicar o número total de estados visitados pelo percentual do estado respectivo. Dado o número de *rounds*, os resultados da equação estarão na mesma "proporção", logo, caso aponte que um estado aumentou o tempo de combate, significa que ele aumentou o tempo total de combate nos 5500 *rounds*.

Após a aplicação da equação, estados em que o tempo aumentou mais de 800s, após o uso do algoritmo DSST, foram escolhidos para serem cortados do tratamento do algoritmo. Optou-se por manter estados em que o tempo aumentou, mas ficaram abaixo de 800s, pois eles possuem baixa frequência (uma frequência inferior à 0.35%) ou a diferença entre $Tdsst_n - Tdsp_n$ é inferior a 5 segundos. Dessa forma, os seguintes estados não são mais considerados *Estados Críticos*: [3-1 CMW-C], [3-1 CWR-C], [3-1 MWR-C], [2-1 MR-W], [1-2 C-WC] e [1-1 R-C].

Após a remoção dos estados mencionados, os seguintes estados são considerados *Estados Críticos* pelo DSST: [4-1 CMWR-C], [3-1 CMR-C], [2-1 MR-C], [2-1 RC-C], [2-1 RC-M], [2-1 RW-C], [2-1 RW-R], [2-1 CW-C], [1-3 C-WMC], [1-1 C-R], [1-1 C-M], [1-1 C-W], [1-1 W-C] e [1-1 M-C]. Assim, o DSST utiliza, em todos os *Estados Críticos*, a tática "Foco no Ataque", enquanto em todos os outros estados possíveis a tática utilizada é selecionada pelo DS.

5.2. Resultados Experimentais

O algoritmo DSST foi aplicado a todos os estados *Estados Críticos*. O algoritmo DSP também foi aplicado aos mesmos estados com objetivo de avaliar ambos os algoritmos.

Todos os resultados apresentados nesta sessão seguem a metodologia discutida na seção 5.1. Os resultados são exibidos na Tabela 3.

De 14 estados, o algoritmo de Substituição de Tática se saiu melhor que o *Dynamic Scripting* Padrão em 12 deles. Devido à dinâmica do jogo, os estados [2-1 CW-C] e [1-1 C-W] podem ter sofrido um aumento no tempo dado que os estados anteriores podem influenciar os próximos.

Os resultados mostram uma melhora em relação ao tempo gasto nos *Estados*

Tabela 3. Comparação do tempo médio gasto em cada *Estado Crítico*

Estado	Tempo Médio (s) DSP	Tempo Médio (s) DSST	Estado	Tempo Médio (s) DSP	Tempo Médio (s) DSST
[4-1 CMWR-C]	23.52	16.79	[2-1 CW-C]	22.66	32.66
[3-1 CMR-C]	25.22	24	[1-3 C-WMC]	14	11.9
[2-1 MR-C]	34.2	32	[1-1 C-R]	28.5	18
[2-1 RC-C]	32.11	29.22	[1-1 C-M]	21	14
[2-1 RC-M]	17	5.5	[1-1 C-W]	14.5	15.75
[2-1 RW-C]	27	23.6	[1-1 W-C]	34	20
[2-1 RW-R]	27	7	[1-1 M-C]	27.5	9

Críticos. No entanto, é necessário analisar esses resultados e demonstrar os ganhos reais da aplicação do algoritmo proposto. A tabela 4 compara as estatísticas relativas ao tempo e convergência do DSST e DSP.

Tabela 4. Estatísticas de Combate

	DSP	DSST
Média de visitas a Estados por Combate	4.7	4.5
Duração Média do Combate (s)	53.4	46
Round de Convergência Média	35	21

Ao se analisar os resultados dos dois algoritmos, verifica-se que cada combate do *Dynamic Scripting* Padrão dura, em média, 53.4s e cada combate do algoritmo proposto dura, em média, 46s, uma redução de 7.3s (13.7%).

O *Dynamic Scripting* Padrão atingiu, em média, a convergência em 35 *rounds*, enquanto o DSST atingiu a convergência em 21 *rounds*, uma notável redução de $\approx 40\%$.

Considerando o tempo de combate e o número de *rounds* até a convergência, o tempo gasto para alcançar a convergência, em média, no algoritmo proposto é de 966.8s versus 1869s no *Dynamic Scripting* Padrão - uma considerável redução de $\approx 50\%$.

Os resultados apontam que o DSST é superior ao algoritmo DSP em velocidade (número de *rounds* necessários para alcançar a convergência) e tempo (duração do combate em segundos).

6. Conclusões

Este trabalho discutiu alguns pontos do *Dynamic Scripting* que podem ser melhorados a fim de aumentar sua eficácia e eficiência. Através do algoritmo sugerido, observou-se que o tratamento destes pontos permite uma aprendizagem mais rápida dos agentes de IA.

Como resultado, tem-se o algoritmo de Substituição de Tática que é capaz de melhorar e acelerar o aprendizado de agentes de IA controlados por *Dynamic Scripting*. Os resultados obtidos estão relacionados à sua aplicação no jogo *Neverwinter Nights*, no entanto, é possível adaptá-lo a qualquer outro jogo que possa ser dividido em estados.

Apesar da especificidade do tratamento devido ao ambiente de teste ao qual foi aplicado, este trabalho abre um precedente para a análise e tratamento do *Dynamic Scripting* quando aplicado em outros meios. Neste sentido, uma vez que identifiquem-se os estados nos quais a técnica é menos eficiente, tratamentos adequados (corretivos) podem ser aplicados.

Referências

Armstrong, W., Christen, P., McCreath, E., and Rendell, A. (2006). Dynamic Algorithm Selection Using Reinforcement Learning. *2006 International Workshop on Integrating AI and Data Mining*, pages 18–25.

Bateman, C. M. (2007). *Game writing : narrative skills for videogames*. Cengage Learning; 1 edition (July 3, 2006).

Bioware (2000). Neverwinter Nights.

Champandard, A. J. (2003). *AI Game Development: Synthetic Creatures with Learning and Reactive Behaviors*. NRG Series. New Riders.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition.

Muñoz-Avila, H., Bauckhage, C., Bida, M., Congdon, C., and Kendall, G. (2013). Learning and Game AI. *Artificial and Computational Intelligence in Games*, 6:33–43.

Rabin, S. (2002). *AI Game Programming Wisdom*. Charles River Media; 1 edition (April 3, 2002), Boston, first edit edition.

Rabin, S. (2013). *Game AI Pro: Collected Wisdom of Game AI Professionals*. Taylor & Francis.

Russell, J.S. e Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*. Pearson; 3 edition (December 11, 2009).

Shafer, J. (2012). *The Recipe for Good AI*. Disponível em: http://ubm.io/2j7DXFY. Acesso em: 31 de Outubro de 2016.

Spronck, P. (2005). *Adaptive Game AI*. PhD thesis.

Spronck, P., Ponsen, M., Sprinkhuizen-Kuyper, I., and Postma, E. (2006). Adaptive game AI with dynamic scripting. *Machine Learning*, 63(3):217–248.

Spronck, P., Sprinkhuizen-Kuyper, I., and Postma, E. (2004). Difficulty scaling of game AI. *Proceedings of the 5th International Conference on Intelligent Games and Simulation (GAMEON'2004)*, pages 33–37.

Sutton, R. and Barto, A. (1998). Reinforcement Learning: An Introduction. *IEEE Transactions on Neural Networks*, 9(5):1054–1054.

Thawonmas, R. and Osaka, S. (2006). A method for online adaptation of computer-game AI rulebase. *International Conference on Advances in Computer Entertainment Technology 2006*, page 16.

Wexler, J. (2008). 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*.