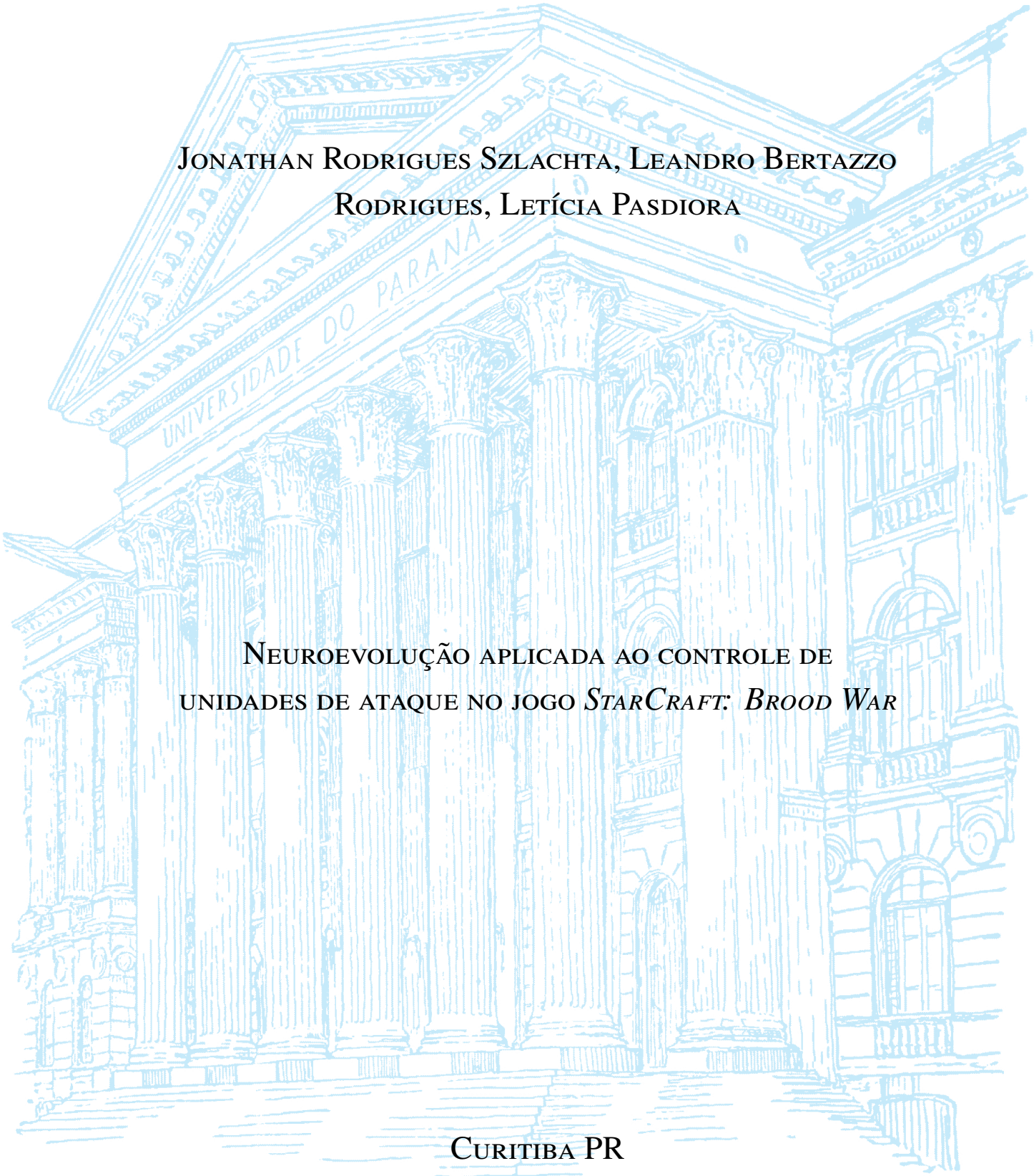
Universidade Federal do Paraná

Jonathan Rodrigues Szlachta, Leandro Bertazzo Rodrigues, Letícia Pasdiora



Neuroevolução aplicada ao controle de unidades de ataque no jogo *StarCraft: Brood War*

Curitiba PR
2017

Jonathan Rodrigues Szlachta, Leandro Bertazzo Rodrigues, Letícia Pasdiora

Neuroevolução aplicada ao controle de unidades de ataque no jogo *StarCraft: Brood War*

Trabalho apresentado como requisito parcial à conclusão do Curso de Bacharelado em Ciência da Computação, setor de Ciências Exatas, da Universidade Federal do Paraná.

Orientador: Eduardo Jaques Spinosa.

Curitiba PR
2017

"Se o sábio lhes der ouvidos,
aumentará seu conhecimento,
e quem tem discernimento
obterá orientação."
(Bíblia Sagrada, Provérbios 1, 5)

Agradecimentos

Agradecemos às nossas famílias e amigos por nos apoiarem em momentos difíceis, oferecendo palavras (e comida) de conforto. Em especial, agradecemos à mãe do Jhonny por nos acolher em sua casa com refeições deliciosas, revigorando nossos ânimos para continuar com força e persistência a desenvolver este trabalho.

Também gostaríamos de agradecer ao doutorando Jeovane Alves, nosso amigo que ajudou a tirar algumas dúvidas e deu sugestões interessantes para a nossa pesquisa.

Os agradecimentos também são dirigidos às gatinhas Celes e Megara, por permanecerem ao lado com explêndida fofura, acalmando momentos de tensão somente com suas existências. Ajudaram também a resolucionar problemas e sanar dúvidas, apenas ouvindo o que tínhamos a questionar, fazendo-nos perceber que as respostas não estavam tão distantes quanto o imaginado.

Os agradecimentos finais são dirigidos à todos aqueles que trabalham com amor na indústria de jogos digitais, fornecendo muito assunto do que falar e também estudar sobre.

Resumo

Nos jogos eletrônicos atuais, os personagens não-jogáveis (NPCs) ainda têm um comportamento muito precário, baseado somente em ações básicas a serem tomadas, tornando-se repetitivas e desse modo podendo oferecer com maior frequência pontos fracos para o jogador obter vantagem. Uma proposta para evitar esse cenário é o uso de redes neuroevolutivas (EANN). Deste modo, diferentes jogadores poderão ter experiências distintas, além de dificultar a percepção de um padrão no comportamento. Dado este cenário, nesta monografia propõe-se a utilização de EANNs no jogo *Starcraft: Brood War*, dada a complexidade que o jogo em questão oferece. Utilizou-se o rtNEAT e o *Novelty Search* para controlar as unidades de ataque. No caso do *Novelty Search*, acabou sendo inviável de utilizar esta implementação em um jogo, pois afetou o processamento, logo, em uma competição seria desclassificado por mau desempenho. Já no caso do rtNEAT, teve-se um resultado equilibrado, não sendo nem fácil nem difícil de derrotá-los quando colocados para lutarem contra outras inteligências artificiais já desenvolvidas para este jogo.

Palavras-chave: Personagem não-jogável, rtNEAT, Novelty Search, StarCraft: Brood War.

Abstract

In electronic games nowadays, non-playable characters still have dull or precarious behaviour, with only basic actions to be decided upon, making them repetitive and, in turn, easier for players to explore weak points and get the upper hand. One way to fix this is through evolutionary artificial neural networks (EANN). They make it possible for different players to have different experiences in the same game, also, they make it harder for players to find behaviour patterns and explore them for too long. Looking into this scenario, this text proposes a use of EANNs in the game *Starcraft: Brood War*, given its high complexity. The algorithms rtNEAT and Novelty Search were used to control and evolve the damage units. The Novelty Search algorithm implementation used has shown itself to be inviable for a game, it affected the processing time which in a competition would make it to be disqualified for below minimum performance. The rtNEAT algorithm has shown balanced results, not becoming too easy nor too hard to beat when in battle against other AIs developed for the game.

Keywords: Non-playable Character, rtNEAT, Novelty Search, StarCraft: Brood War.

Sumário

Lista de Figuras

Lista de Acrônimos

AN	Artificial Neuron
BWAPI	Brood War API
CIG	IEEE Conference on Computational Intelligence and Games
EANN	Evolutionary Artificial Neural Networks
FPS	First-person shooter
IA	Inteligência Artificial
IEEE	Instituto de Engenheiros Eletricistas e Eletrônicos
NEAT	NeuroEvolution of Augmented Topologies
NPC	Non-player Character
rtNEAT	Real time NeuroEvolution of Augmented Topologies
RTS	Real Time Strategy
SSCAIT	Student StarCraft AI Tournament

Capítulo 1

Introdução

Com a evolução da computação gráfica e com a popularização da realidade virtual, os jogos estão visualmente mais realistas, contudo, o desenvolvimento da inteligência artificial em jogos não acompanha o mesmo ritmo, parecendo estar estagnada (Asensio et al., 2014). Logo, os personagens não-jogáveis (*non-player character* - NPC) majoritariamente têm um comportamento simples e previsível, então, a experiência do jogador fica menos imersiva, além deste poder explorar fraquezas do NPC por conta da inteligência artificial limitada.

Os comportamentos dos NPCs são normalmente baseados em *scripts* e máquinas de estados (Kishimoto, 2004), principalmente por serem maneiras simples de criar um comportamento satisfatório, dado que NPCs dificilmente têm uma atuação complexa no jogo, além de não sobrecarregarem o processamento do jogo como um todo, visto que um bom desempenho de processamento é um fator importante em jogos. Entretanto, em jogos que exigem comportamento mais complexo do NPC, como jogos de estratégia, é muito mais viável para um jogador analisar este comportamento e saber reconhecer seus passos, tornando trivial derrotar o NPC.

Como a indústria de jogos está constantemente crescendo, aposta-se que o futuro para IA em jogos seja composto de algoritmos genéticos e redes neurais (Kishimoto, 2004), porém a maioria dos trabalhos publicados envolvendo jogos e algoritmos bioinspirados são resolucionadores de jogos ou adversários para jogos clássicos de tabuleiro, como xadrez e go (Moudřík et al., 2015).

Logo, a pesquisa para criar um comportamento para um NPC com base nesse tema é ainda escassa, em virtude de problemas como generalizar o algoritmo, que se torna muito específico para um tipo de jogo dado as suas regras e sua jogabilidade (Asensio et al., 2014). Outro fator importante a se considerar (Buckland, 2002), é o desempenho em questão de processamento das Redes Neurais Artificiais Evolutivas (*Evolutionary Artificial Neural Networks* ou EANN) , pois podem existir vários NPCs, cada um executando uma EANN, sobrecarregando ainda mais o processamento, tornando o jogo lento.

Para incentivar a pesquisa no campo de jogos digitais, existem competições de *bots* — algoritmos para controlar o jogador no lugar do humano — organizadas por universidades e também em eventos (Churchill et al., 2016). Muitos desses *bots* costumam controlar somente o jogador principal do jogo, não contribuindo para o desenvolvimento de NPCs. Contudo, o jogo *StarCraft: Brood War* (SC:BW) oferece um campo interessante para criar-se *bots* com ações mais complexas, podendo-se simular NPCs, pois tanto NPCs quanto jogador possuem os mesmos objetivos e ações no jogo.

Neste trabalho propõe-se a utilização de EANNs para controlar as unidades de ataque do jogo SC:BW, com o objetivo de avaliar seu comportamento e dificuldade durante o jogo. Espera-se que o comportamento não seja trivial de ser previsto mas não imbatível, com o

intuito de tornar o jogo mais equilibrado, sendo possível notar a evolução crescente dos *bots* em relação a quantidade de dano que causam. Sendo assim, no capítulo 2 são descritos os conceitos necessários para o entendimento do funcionamento das EANNs. Já no capítulo 3, apresenta-se o jogo SC:BW, suas mecânicas e a ferramenta que auxilia a implementação de outros módulos no jogo em questão. Em seguida, listam-se os trabalhos relacionados no capítulo 4, analisando não somente os trabalhos relacionados com SC:BW, mas também outras propostas para NPCs no geral. Já no capítulo 5, são descritas decisões de implementação e também a estrutura da rede neuroevolutiva desenvolvida. No capítulo 6 avalia-se o desempenho da rede, como também é descrita a metodologia para avaliação, além de apresentar-se sugestões para trabalhos futuros. Então, este trabalho é concluído no capítulo 7, descrevendo as conclusões tiradas principalmente sobre o capítulo 6.

Capítulo 2

Fundamentação Teórica

2.1 Algoritmos Bioinspirados

Algoritmos bioinspirados são algoritmos que tentam utilizar modelos da natureza para a resolução de problemas computacionais, sendo dividido em subáreas, como as redes neurais artificiais e a computação evolutiva. Um exemplo para algoritmos bioinspirados é sua aplicação para o controle em conjunto de robôs, sendo uma proposta para esse problema analisar o comportamento de grupos de animais, como enxame de abelhas e tentar modelá-lo para o problema computacional, esta área é conhecida como robótica de enxames. Nas seções a seguir serão apresentadas as definições relacionadas a computação evolutiva e redes neurais utilizadas neste trabalho.

2.2 Computação Evolutiva

Na computação evolutiva, um algoritmo genético é um algoritmo inspirado no processo de seleção natural, no qual o conceito principal é a sobrevivência dos mais aptos, ou seja, os organismos se reproduzem e aqueles que não se adequam às condições do ambiente, não sobrevivem. No modelo de seleção natural também existe a mutação, evento que acontece aleatoriamente.

Alguns conceitos que devem ser explicados antes de se explicar o algoritmo propriamente dito são:

- Organismo: Também chamado de indivíduo, é um candidato a solução para o problema ao qual está submetido, possuindo um conjunto de características, definido de acordo com o problema. Estas características também são chamadas de cromossomos ou genótipos;
- População: Conjunto de organismos que estão sendo analisados, dos quais os melhores serão usados para criar novos organismos para esta população;
- Ambiente: É o problema propriamente dito e as partes envolvidas;
- *Crossover*: Troca de características de dois indivíduos para gerar um novo. Pode-se comparar à reprodução no modelo da seleção natural.

Neste modelo, a evolução geralmente acontece com uma população gerada aleatoriamente, então será avaliada em uma quantidade de passos discretos, sendo este passo chamado de

geração. A avaliação da população dá-se por uma função chamada de *fitness*, que é decidida de acordo com o problema e com o ambiente. Esta função classifica se um organismo é uma solução melhor ou pior para o problema. Os indivíduos que forem classificados como piores para a solução serão removidos e substituídos pelo *crossover* de outros dois organismos com melhor *fitness*. Em cada geração, existe a probabilidade de algum organismo ter alguma das suas características mudada aleatoriamente, sendo chamada de mutação esta etapa (Engelbrecht, 2007).

2.3 Redes Neurais Artificiais

Redes neurais artificiais são inspiradas nas conexões que acontecem no cérebro, principalmente na relação de neurônios e suas transmissões de sinais. Um neurônio artificial (*Artificial Neuron* ou AN) recebe sinais do ambiente ou de outros ANs e, quando estimulado, transmite para os demais ANs que estão conectados consigo, como pode-se observar na Figura 2.1. Os sinais de entradas são ajustados através de pesos numéricos associados a cada conexão, além de cada AN possuir a sua própria função de ativação. Essa função de ativação é baseada nos seus sinais de entrada e indica se uma saída é ativa se o valor calculado está acima de um limite definido para esta rede, Engelbrecht (2007).

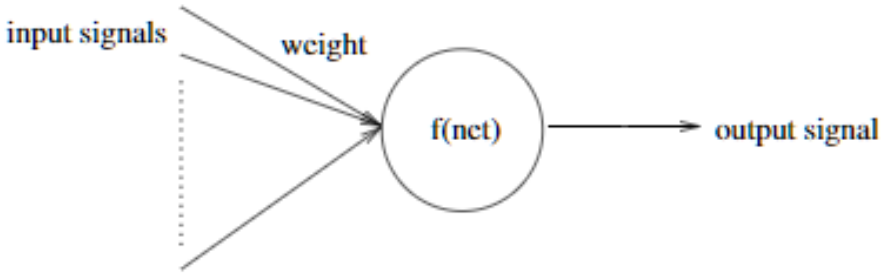


Figura 2.1: Exemplo de um neurônio artificial. Retirado de Engelbrecht (2007).

As redes neurais artificiais são conjuntos de ANs, tendo-se os ANs de entrada, que captam os sinais do ambiente, ANs de saída, que dão o resultado final dos cálculos através dos sinais, e os demais ANs que conectam as entradas e saídas, que se encontram em *hidden layers*.

Dada uma rede definida com os ANs e conexões, seu treinamento ocorre primeiramente inicializando seus pesos, podendo ser de forma aleatória. Esse treinamento pode ser realizado já sabendo qual a resposta esperada, conhecido como aprendizado supervisionado, se não forem fornecidas entradas e saídas esperadas se classifica como não-supervisionado e caso possua um avaliador externo que indique adequação da saída é conhecido como aprendizado por reforço. Dadas as informações de entrada e sua saída, é possível realizar a correção dos pesos observando o erro (calculado com os valores de entrada e comparado com um valor esperado de saída) dado em uma iteração da rede de forma a tentar resolver o problema corretamente (Engelbrecht, 2007).

As redes neurais artificiais se adequam a inúmeros problemas, dentre eles destacam-se aqueles onde deve-se classificar dados, reconhecer padrões, analisar informações e também na robótica.

2.4 Neuroevolução

Juntando os modelos de redes neurais artificiais e algoritmos genéticos, tem-se o modelo neuroevolutivo (EANN). Neste modelo, a representação do organismo é a rede, enquanto os dados a serem alterados pela mutação são as topologias das redes sendo avaliadas, sendo que esta mudança pode ocorrer nos pesos da rede, na criação ou remoção de um AN, ou ainda na criação ou remoção de uma conexão entre ANs.

No caso específico do NEAT e suas variações, o conceito de espécie é introduzida. Espécies são conjuntos de organismos que possuem estrutura semelhantes, sendo que o cálculo desta semelhança é descrita com maiores detalhes na seção 2.4.1. A inclusão de espécies faz com que a reprodução ocorra somente com organismos da mesma espécie. Este processo é chamado de especiação (Stanley e Miikkulainen, 2002).

2.4.1 NEAT

O algoritmo *NeuroEvolution of Augmenting Topologies* (NEAT) é um algoritmo de neuroevolução de redes neurais artificiais (EANNs). O diferencial desta EANN é o fato de alterar a sua própria estrutura e pesos durante a evolução, como mostrado em Stanley e Miikkulainen (2002).

Neste modelo, existe mais de uma rede para comparar e testar quais possuem melhor desempenho. Após uma quantidade discreta de passos, essas redes são avaliadas e as que têm um *fitness* baixo, são eliminadas e substituídas pelo *crossover* de outras duas redes com *fitness* maior, segundo Stanley e Miikkulainen (2002).

Observa-se em Stanley e Miikkulainen (2002) que a criação de novas ligações acontece aleatoriamente, adicionando-se uma conexão entre dois ANs — também chamados de nodos, ou no caso do NEAT, genes — aleatórios. Já a criação de um novo gene dá-se pela divisão de uma conexão já existente, ou seja, essa conexão é desativada e no lugar coloca-se um novo gene que irá se conectar com os outros dois genes que antes possuíam uma ligação, como pode-se observar na figura 2.2.

Para o *crossover* ser possível, são adicionados marcadores históricos nos genes, ou seja, quando um novo gene é criado, as conexões anteriores só são desativadas e a nova conexão é introduzida no histórico do genoma, sendo que um genoma da rede é a descrição de um gene (Figura 2.3). Assim, é possível analisar os genes em comum entre duas redes, de acordo com Stanley e Miikkulainen (2002).

No contexto de comparação de duas redes, podem existir genes correspondentes, genes em excesso e genes disjuntos. Genes correspondentes possuem os mesmos marcadores históricos e mesmas conexões. Um gene disjunto é um gene pertencente à rede n, mas que não pertence à rede m e o seu marcador histórico é menor que o maior marcador histórico tanto da rede n quanto da m. Por outro lado, um gene em excesso é um gene que pertence somente à rede n com um marcador histórico maior que o maior marcador histórico da rede m, como visto em Stanley e Miikkulainen (2002).

As diferenças entre redes são preservadas com a especiação. Cada uma das redes (ou fenótipos) são classificados pela semelhança topológica entre elas, tendo-se assim um número variável e finito de espécies. A semelhança é calculada pela equação 2.1, onde a distância δ entre dois genomas é calculada pela combinação linear da quantidade de genes em excesso (E) e disjuntos (D) e pelo peso médio de genes correspondentes ($\overline{W}$). Os coeficientes c_1, c_2 e c_3 ajustam a importância dos fatores E, D e W, enquanto a divisão pelo número de genes no maior

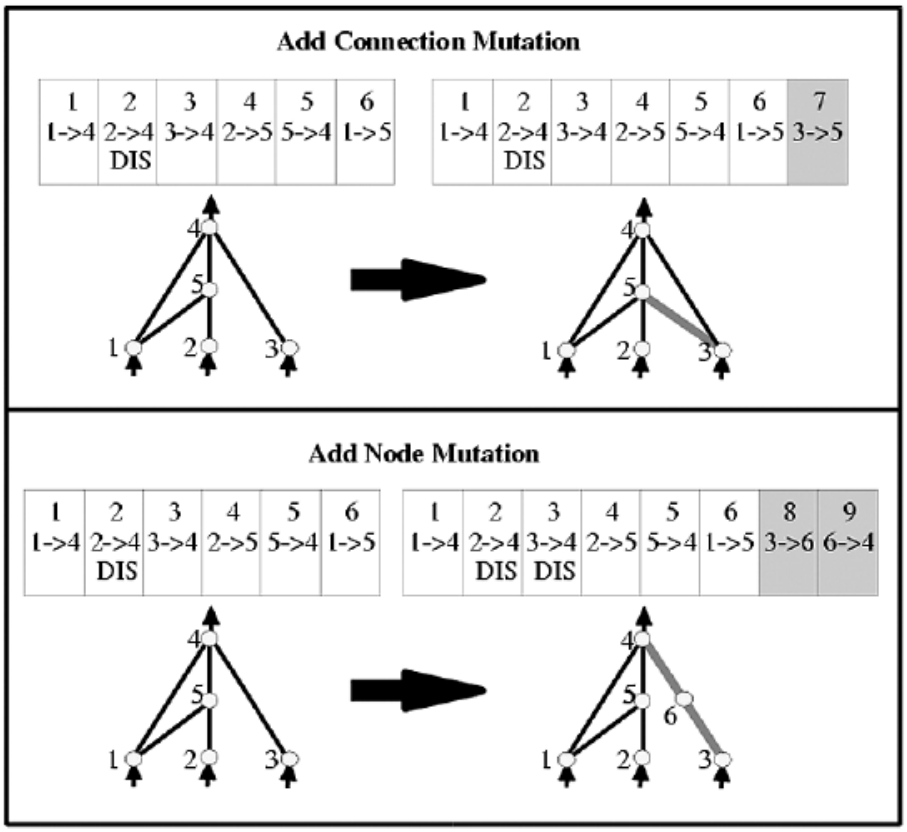


Figura 2.2: Os dois tipos de mutações possíveis no NEAT. A primeira representa mutação de conexão, enquanto a segunda mutação de adição de gene. Nos quadros o número em cima é o marcador histórico na inovação, seguido pela ligação entre nodos e se está ativa (em branco) ou desativada (DIS). Retirado de Stanley et al. (2005).

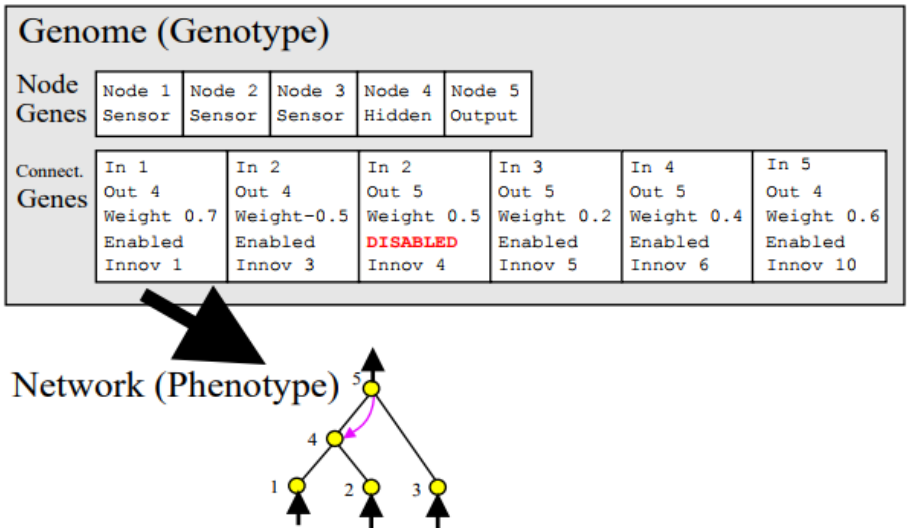


Figura 2.3: Mapeamento de um genótipo (genoma) para um fenótipo (rede). Retirado de Stanley e Miikkulainen (2002).

genoma (N) evita valores discrepantes entre as redes, como proposto em Stanley e Miikkulainen (2002).

$$\delta = \frac{c_1 \cdot E}{N} + \frac{c_2 \cdot D}{N} + c_3 \cdot \overline{W} \tag{2.1}$$

Deste modo, o NEAT pode ser aplicado em diversos problemas, principalmente de classificação. Também facilita o desenvolvimento de EANNs, pois não é necessário desenvolver a estrutura completa da rede, somente definir suas entradas e saídas. Além disso, com os marcadores históricos, crescimento da topologia e proteção da inovação com especiação, produz-se uma solução com estrutura pequena, como constatado em Stanley e Miikkulainen (2002).

2.4.2 rtNEAT

O algoritmo *Real Time NeuroEvolution of Augmenting Topologies* (rtNEAT) é baseado no NEAT, a principal diferença entre os dois está na evolução da rede, no rtNEAT ela acontece durante a execução da resolução do problema de acordo com os resultados dos organismos até o momento. Já no NEAT ocorre somente depois de uma iteração do problema. No exemplo do SC:BW, no NEAT a evolução só aconteceria ao final de cada partida, enquanto no rtNEAT, aconteceria durante a partida. Além disso no rtNEAT apenas um organismo é substituído por vez, ao contrário do NEAT que pode eliminar mais de um organismo por iteração, como proposto em Stanley et al. (2005).

Assim, a população é avaliada constantemente, analisando qual organismo na população tem o pior *fitness* e que já tenha passado do tempo mínimo de vida, logo, o organismo com pior desempenho é substituído por um novo gerado a partir de dois outros organismos aleatórios Stanley et al. (2005).

O rtNEAT se adequa à aplicação em jogos porque a evolução pode ocorrer ao decorrer do jogo, como foi originalmente proposto em Stanley et al. (2005). Como exemplos de aplicações em jogos pode-se citar o NERO utilizado em Stanley et al. (2005) e também com o SC:BW, que já foi utilizado nos trabalhos Zhen e Watson (2013) e Gabriel et al. (2012).

2.4.3 Novelty Search

A *Novelty Search* é uma nova estratégia aplicada a neuroevolução proposta por Lehman e Stanley (2011) que não se baseia em procurar objetivos. Nesse modelo a importância se encontra em sempre buscar novidades, ou seja, encontrando diferentes maneiras de resolver um problema. Isso ocorre de forma contrária ao de outras técnicas que bonificam organismos que apresentam um melhor desempenho, isto é, que se encontram mais próximos de um objetivo. Ao invés de utilizar uma função de *fitness* como no NEAT, é utilizada uma função que realiza um cálculo mostrando a distância de um organismo da população para seus vizinhos mais próximos. Dessa maneira, a *Novelty Search* pode sair de casos de ótimo local que ocorreriam com outras técnicas, por buscar um organismo que apresente mais diferenças em relação aos seus vizinhos.

Mais do que procurar as soluções, com este procedimento tem-se uma modelagem mais próxima da evolução natural, onde as diversidades que surgem geram maior complexidade da população.

Os autores utilizam-se do NEAT adaptado com essa técnica para demonstrar seu potencial. Suas aplicações são voltadas para os problemas onde é difícil encontrar uma função de *fitness* que os represente. Na Figura 2.4 pode-se observar um exemplo de aplicação da *Novelty Search* no problema de um labirinto, onde uma função de *fitness* não sai de ótimos locais, enquanto a busca por novidades consegue encontrar uma solução.

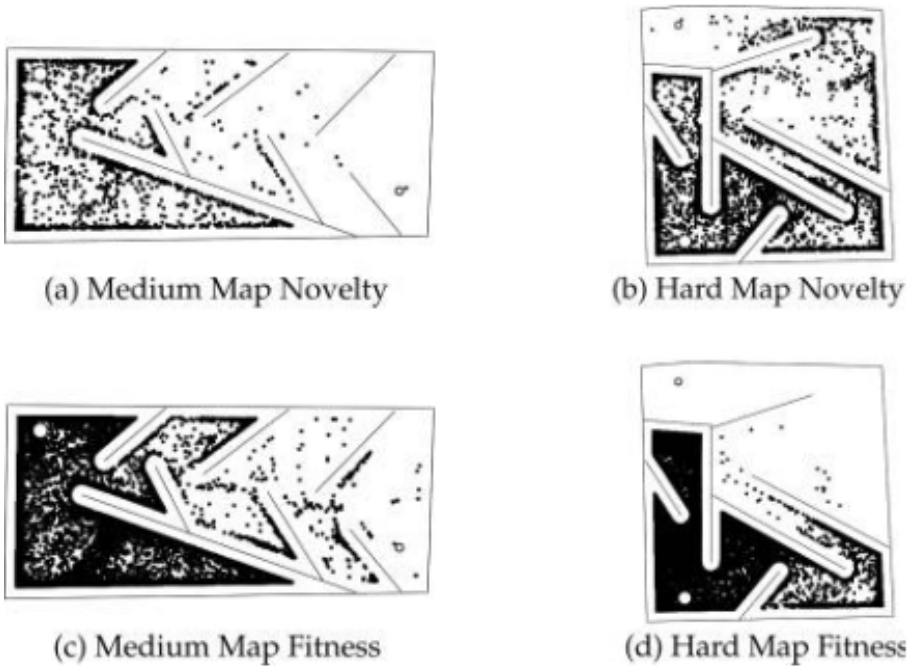


Figura 2.4: Exemplo de *Novelty Search* aplicado ao problema de um labirinto. Os pontos representam os lugares onde os resolvedores pararam. Em *a* e *b* observa-se a *Novelty Search* precisando de menos tentativas (número menor de pontos de parada) para conseguir resolver o problema, ao contrário do uso de *fitness* em *c* e *d*. Fonte: Lehman e Stanley (2011).

2.5 Conclusão

A computação bioinspirada mostra-se eficaz para a resolução de problemas diversos, como pode-se citar classificação e mineração de dados. Essa abordagem é eficaz por se adaptar ao problema, aprendendo o comportamento necessário para o problema, diferenciando da IA clássica, na qual o comportamento de um agente é programado previamente (Engelbrecht, 2007).

Capítulo 3

Cenário do Problema

3.1 StarCraft: Broodwar

Starcraft é um jogo do gênero *Real Time Strategy* (RTS), um jogo prestigiado para a criação de *bots*, oferecendo um campo interessante de pesquisa de Inteligência Artificial em jogos. O pacote de expansão SC:BW foi lançado em 1998, adicionando novas funcionalidades, como novas unidades a serem controladas e também novas histórias para o jogo.

A ideia principal de SC:BW é destruir as bases de controle — ponto de início dos jogadores, com uma estrutura inicial para armazenamento de recursos — de todos os inimigos. Para desenvolver o seu território e a força de ataque ser maior, é necessário a coleta de recursos como cristais e gás. Estes recursos são matéria prima para construção de novos edifícios, unidades de ataque e unidades de coleta.

Unidades de coleta têm como função coletar os dois tipos de matéria prima e também construir novos edifícios. As unidades de ataque têm como função proteger e patrulhar a base de controle e outras estruturas, atacando quem tenta atacar sua base, além de atacar e destruir as bases inimigas. Já os edifícios possuem funções diversificadas, como pode-se citar armazenamento de recurso, aprimoramento e desenvolvimento de novas armas e também construção de novas unidades, tanto de coleta quanto de ataque. [1]

O jogo possui três raças para o jogador escolher:

- *Zerg*: raça que exige menor consumo de matéria prima para criar unidades e estruturas. Estas possuem baixa complexidade funcional, cada uma tendo uma função bem definida. A estratégia básica e mais utilizada dessa raça é construir a maior quantidade de unidades de ataque possível para destruir a base inimiga enquanto ainda não possui grande força de ataque e nem estruturas complexas;

- *Terran*: raça com consumo de matéria prima medianos, possuindo diversidade média de opções de construção, com complexidade estratégica média, sendo que as funções das unidades não são tão limitadas quanto a *Zerg*;

- *Protoss*: raça com o maior consumo de matéria prima, com a possibilidade de evoluir suas unidades para as unidades das outras duas raças, oferecendo a maior complexidade estratégica dentre as três raças.

[1] Mais informações em: `http://starcraft.wikia.com/wiki/StarCraft_Wiki`

Figura 3.1: Exemplo de partida com a raça *Terran*. Recursos indicados em azul, unidades de coleta em verde, unidades de ataque em vermelho e edifícios em amarelo.

A raça mais popular para a construção de *bots* é a *Terran*, com 27 *bots* já desenvolvidos para competições ou com fins acadêmicos, seguido pela raça *Protoss*, com 22 *bots* e, por último, *Zerg* com somente 14 *bots* listados. [2]

3.2 Brood War API

O *Brood War API* (BWAPI)[3] é uma ferramenta para auxiliar o desenvolvimento de programas que interagem com o SC:BW, como por exemplo a criação de *bots*, facilitando tanto o controle das unidades quanto a análise do mapa.

Oferece principalmente informações gerais do estado do jogo e de todas as unidades, como quantos pontos de vida a unidade ainda possui, sendo esta inimiga ou não. Oferece também uma lista de ações possíveis com as próprias unidades.

Também possui funções pré-determinadas para o desenvolvimentos de *bots* como `onStart()` para configuração inicial do jogo antes de começar, `onFrame()` para ditar o que deve ser feito a cada quadro em que se passa e também `onUnitDestroy()` para realizar alguma ação depois que uma unidade foi destruída, dentre outras.

[2]Mais informações em: `http://www.starcraftai.com/wiki/Main_Page`
[3]Mais informações em: `https://github.com/bwapi/bwapi`

Capítulo 4

Trabalhos relacionados

A quantidade de trabalhos envolvendo redes neuroevolutivas está crescendo anualmente, podendo-se observar esse fato na *IEEE Conference on Computational Intelligence and Games* (CIG), uma conferência anual realizada pela IEEE. Entretanto, ainda foca-se principalmente no quesito de jogos de estratégia clássicos que possuam somente um adversário, entrando na categoria de resolucionadores de jogos, como analisado em Moudŕík et al. (2015).

Um trabalho notável é Asensio et al. (2014), onde foi definida uma rede neural e utilizaram algoritmos genéticos para criar *bots* para o jogo *Unreal Tournament 2004*, um jogo de tiro em primeira pessoa (*first-person shooter* - FPS). Para analisar a eficiência dos *bots* desenvolvidos, foi aplicado um teste inspirado no teste de Turing, analisando o quão humano era o comportamento dos *bots*. Por mais que o resultado tenha sido satisfatório, a rede neural não era treinada durante o jogo, limitando o comportamento dos *bots* por não aprenderem e não encontrarem novas estratégias durante o jogo.

Outro trabalho que merece destaque é Schrum e Miikkulainen (2009), pois propõe o desenvolvimento de personagens que possuam modos de comportamento diferentes dadas situações diferentes (*multi-modal behavior*). Para definir esse comportamento, implementa-se uma rede neural com mutações, que aprende durante o jogo também. Entretanto, os NPCs definidos trabalham como um time contra o jogador, além de possuírem somente duas tarefas e quatro objetivos básicos, mostrando-se simplista nesse ponto. É proposto para trabalhos futuros desenvolver entradas mais complexas, ou ainda generalizar a rede definida. Nota-se também que o jogo desenvolvido não tem prestígio comercial, sendo desenvolvido exclusivamente para a pesquisa.

Em Stanley et al. (2005) é proposto o algoritmo rtNEAT, um algoritmo de redes artificiais neuroevolutivas (EANNs), sendo que a aplicação exemplo é um jogo. O algoritmo mostra-se bastante eficaz para a evolução e controle de NPCs, entretanto, o jogo utilizado como exemplo foi criado para exemplificar o funcionamento do rtNEAT, logo, é um jogo que não possui tanto prestígio comercial (aproximadamente 18.200 resultados no Google, enquanto SC:BW conta com 885.000 resultados).

O trabalho mais próximo deste é Zhen e Watson (2013), no qual é feita a comparação de desempenho entre o algoritmo NEAT e o rtNEAT aplicado em *micromanagement* em SC:BW. O algoritmo é dividido em duas partes: as unidades e o controlador NEAT. As unidades são responsáveis por obter os dados, enviá-los para e rede neural e interpretar as saídas. O controlador NEAT gera as unidades e faz interface com as implementações do NEAT e rtNEAT. As redes são aplicadas apenas em unidades de ataque, fazendo-as tomar a decisão de atacar ou recuar, como em uma estratégia de bater e correr, que segundo os autores já apresenta comportamento sofisticado suficiente. Seu objetivo foi avaliar as diferenças dos algoritmos em cenários de

batalha. Contudo, o artigo não define como controlar as outras unidades, enquanto neste trabalho este comportamento é definido. As métricas utilizadas para avaliar os *bots* do artigo não são tão eficientes, baseando-se principalmente no porcentual de vitórias. Logo, neste trabalho avalia-se também outras métricas, como dano feito e inimigos mortos.

Um trabalho anterior a este que também busca uma estratégia neuroevolutiva em *micromanagement* para o SC:BW é Gabriel et al. (2012). Neste trabalho, os autores também apresentam uma abstração realizada para o controle geral, que é divida em três partes: o *Abstractor* que é responsável pelo fornecimento das informações gerais do jogo que não possam ser acessadas diretamente pelas unidades; o *Spawn Agent* que controla a criação de novas unidades, inicialização, avaliação e evolução delas; as *Unit Agents* representa cada unidade presente no jogo. Sua rede tem como entradas tanto informações vindas da própria unidade como do *Abstractor*, e busca realizar a avaliação de uma unidade com base no número de unidades que matou unido ao seu nível de agressividade (em função da quantidade de dano causado e tomado), buscando controlar as ações de avançar ou recuar, movimentação e ataque.

Capítulo 5

Implementação

No jogo há diferentes tipos de unidades de ataque que aumentam a sua complexidade, como unidades terrestres ou voadoras, que atacam ao estarem diretamente ao lado de uma unidade inimiga ou a uma determinada distância. Deste modo, é preferível aplicar as redes somente em unidades semelhantes, por exemplo unidades terrestres que atacam estando a uma mesma distância de uma unidade inimiga, pois colocar mais de um tipo de unidade aumenta consideravelmente as entradas e saídas da rede, prejudicando o desempenho do algoritmo e tornando a execução mais lenta.

Inspirado em Zhen e Watson (2013), optou-se por adicionar redes somente às unidades de terrestres que atacam a distância, por poderem ser abstraídas para somente duas ações de saída (atacar ou recuar).

As unidades também se diferenciam entre raças, como explicado na seção 3.1. Portanto para facilitar o controle através do BWAPI e seguindo as recomendações das competições de inteligência artificial para o jogo, como Churchill et al. (2016), optou-se por escolher somente uma das raças. A raça escolhida foi a *Terran*, por ser a mais simples de controlar entre as três, por isso também é a mais utilizada para fins acadêmicos.

Entretanto, para realizar o treinamento (os parâmetros do treinamento são explicados em detalhes no capítulo 6), as demais unidades (de coleta de recursos e construção da unidades de ataque) precisam de controle, seguindo o algoritmo básico a seguir e conforme o comportamento básico do jogo no modo *melee*, utilizado nas competições do jogo:

- Construir mais 4 unidades de coleta, para que coletem rapidamente recurso, além das 4 inicialmente disponibilizadas. Isso permite utilizar os outros espaços de bots (limitado a 200 pelo jogo) para geração de unidades de ataque;
- Ordenar alguma unidade construir um *barracks*, (estrutura *Terran* que constrói unidades de ataque);
- Unidades de coleta constroem mais *supply depots*, para que o máximo de unidades seja aumentado até o necessário;
- *Barracks* constrói a quantidade necessária de soldados, e caso um seja abatido, produz um novo para substituir;
- Unidades de coleta coletam cristais normalmente, salvo exceções citadas acima;
- Unidades de ataques agem conforme a saída da rede.

Para os testes, foram criados mapas de combate que só possuem unidades de ataque em ambos os lados, para que se possa avaliar seu aprendizado sem interefências de outras unidades. Neste caso o funcionamento do algoritmo é dado apenas por determinar qual ação as unidades de ataque realizarão.

A diferença em relação ao treinamento e testes existe devido ao interesse em aprender o comportamento das unidades de forma completa durante toda a partida, visto que os as unidades são criadas com o passar do tempo. Entretanto nos testes tem-se um número pré-definido e limitado de unidades e busca-se apenas avaliar como elas se portam em um ambiente de batalha.

5.1 Estrutura da Rede Neuroevolutiva

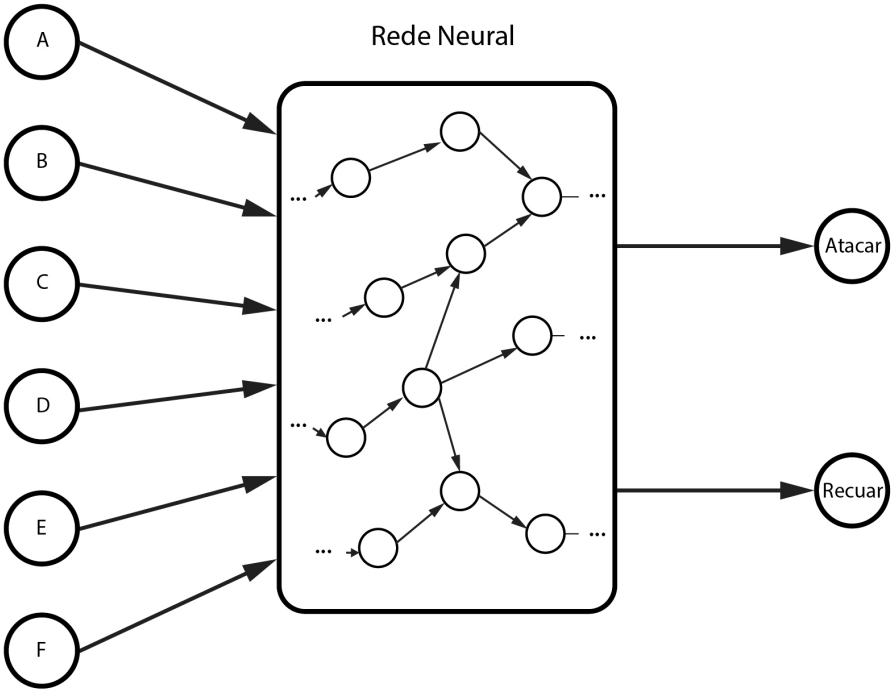


Figura 5.1: Estrutura de exemplo da rede, com 6 entradas, duas saídas (atacar e recuar) e nós internos gerados pelos algoritmos neuroevolutivos.

Para facilitar a implementação das redes, utilizaram-se as bibliotecas em C++ providas pelos próprios criadores do rtNEAT e do *Novelty Search*. Como ambas implementações são semelhantes, todas as decisões tomadas nessa seção aplicam-se para ambas.

A estrutura básica da rede está representada na Figura 5.1, com pesos iniciais aleatorizados e inicialmente sem nenhum nó interno (*hidden*), pois estes fatores serão evoluídos conforme a EANN é utilizada. Inspirado em Zhen e Watson (2013), as duas saídas possíveis são atacar e recuar, enquanto as entradas são:

- Quantidade de aliados ao redor (num raio do alcance máximo da sua própria arma);
- Quantidade de inimigos ao redor (num raio do alcance máximo da sua própria arma);
- Alcance máximo da arma da unidade;

- Alcance máximo da arma do inimigo;
- Tempo de recarga da arma da unidade, já que uma unidade não pode atacar tão rapidamente em uma sequência;
- Pontos de vida restantes da unidade.

Deste modo, as entradas são informações mistas tanto da própria unidade quanto das unidades inimigas, assim, a rede possui entradas que abrangem tanto seus dados e limitações como os do inimigo, ficando equilibrada. Por exemplo, é esperado que se o alcance da própria arma é maior que a do inimigo, a unidade ataque de uma distância maior para não ser atingida pelo ataque do inimigo. As saídas são implementadas de maneira que ataque-se o inimigo mais próximo e em caso de recuar, faça o caminho de volta até a base. Um ponto importante é que as ações são decididas a cada 50 quadros, número escolhido por não ter um atraso na ação, nem sobrecarregar o processamento desnecessariamente, testando-se alguns números num intervalo de 0 a 150. Caso a unidade decida recuar e logo em seguida atacar, esta não irá realizar o caminho completo até a base, ou seja, as ações não são inseridas numa fila e sim, todas as ações da lista são canceladas para realizar-se a nova ação decidida pela rede.

A evolução tanto do rtNEAT quanto do *Novelty Search*, como já citado na seção 2.4.2, acontece constantemente, analisando-se quais unidades podem ser eliminadas. O tempo de vida mínimo n em quadros segue a equação 5.1, proposto em Stanley et al. (2005). O valor de m representa o tempo mínimo em quadros para que uma unidade realize alguma ação significativa. Considerando que do momento da criação de uma unidade até encontrar um inimigo demora pelo menos 500 quadros e dada a diversidade de mapas que pode-se ter, o valor de m foi definido em 1000 quadros. O valor de I é a fração da população que ainda é considerada muito jovem para ser substituída. Seguindo-se Stanley et al. (2005) o valor I é igual a 50%. Já o valor de $|P|$ é o tamanho da população, fixada em 100 unidades de ataque para o treinamento.

$$n = \frac{m}{|P|I} \tag{5.1}$$

Para a decisão da função de *fitness* seguiu-se Zhen e Watson (2013), que propôs a equação 5.2. A *fitness* da unidade i é definida com a diferença do dano já tomado por essa unidade (HPL_i) pelo total de dano feito (TDD_i), normalizada pela divisão do máximo de pontos de vida daquela unidade (IHP_i). Com a soma de mais um, evita-se que a *fitness* fique negativa, sendo o seu mínimo em 0. Com essa função, prioriza-se unidades que fizeram mais dano do que sofreram, sendo adequada para o problema em questão. Em específico para o caso do *Novelty Search*, acrescenta-se à função de *fitness* a diferença entre as redes, sendo esta equação já dada pela implementação do *Novelty Search*.

$$F_i = \frac{TDD_i - HPL_i}{IHP_i} + 1 \tag{5.2}$$

Capítulo 6

Experimentos e Resultados

Inicialmente, as redes foram treinadas em partidas comuns, contra a IA provida pelo próprio jogo e somente um jogador, já que dependendo do mapa, pode-se jogar contra vários inimigos. Entretanto, esse cenário não é interessante pois os demais jogadores (controlados pela IA do jogo) acabam indiretamente formando uma aliança ao focar o ataque no jogador principal. Estas partidas comuns podem ser descritas como partidas em que se utiliza o algoritmo descrito na seção 5, para controle das demais unidades, além de ser contra alguma raça escolhida aleatoriamente. Esta decisão foi tomada para as redes não se especializarem somente contra uma raça, além de também aprenderem a jogar em partidas comuns utilizadas em competições, por mais que o teste de desempenho seja em um tipo de partida específico para o problema.

Para analisar-se a efetividade, seguiu-se algumas sugestões de Uriarte e Ontañón (2015), como não só analisar se o *bot* venceu ou perdeu, mas também contabilizando dado total feito pelas unidades, pontuação de inimigos mortos (pontuação fornecida originalmente pelo próprio SC:BW) e tempo decorrido de cada partida, em segundos. Contudo, para os resultados serem mais significativos, também foi feito a relação de dano realizado e quantidade de inimigos mortos por segundo.

Foi considerado analisar a efetividade dos algoritmos jogando-se contra um humano e este dizer suas impressões. Contudo, este método foi descartado por não oferecer uma medida precisa e exata, sendo que o jogador humano não é confiável, além de não ser possível quantificar com precisão o quão humano o *bot* é.

No treinamento do rtNEAT, jogaram-se 350 partidas, mas com dezenas já era possível analisar-se um comportamento mais inteligente por parte das unidades de ataque. Observa-se que todas as partidas foram perdidas, contudo pode-se explicar como complexidade baixa do *bot* desenvolvido como um todo, pois enquanto a complexidade do inimigo era baixa, as partidas ficavam equilibradas. A partir do momento em que o inimigo iniciava a desenvolver unidades mais complexas e custosas, a partida chegava ao fim.

Observou-se também na Figura 6.1 que a relação dano por segundo aumentou em relação o início, explicando o comportamento das unidades recuarem após receberem muito dano. Entretanto, não pode-se afirmar o mesmo das mortes de inimigos ao analisar-se a Figura 6.2, pois como as unidades aprenderam o comportamento de recuar, deixavam unidades inimigas vivas, estas tendo a chance de também voltar e curar um tanto dos seus pontos de vida.

Já no caso do *Novelty Search*, com base na implementação utilizada, os testes acabaram mostrando um processamento lento, chegando a apenas 8 quadros por segundo no modo extra rápido no melhor caso, e em média 2 quadros por segundo. Já no rtNEAT, no modo extra rápido em alguns momentos chegava-se a 2000 quadros por segundo. A lentidão em questão

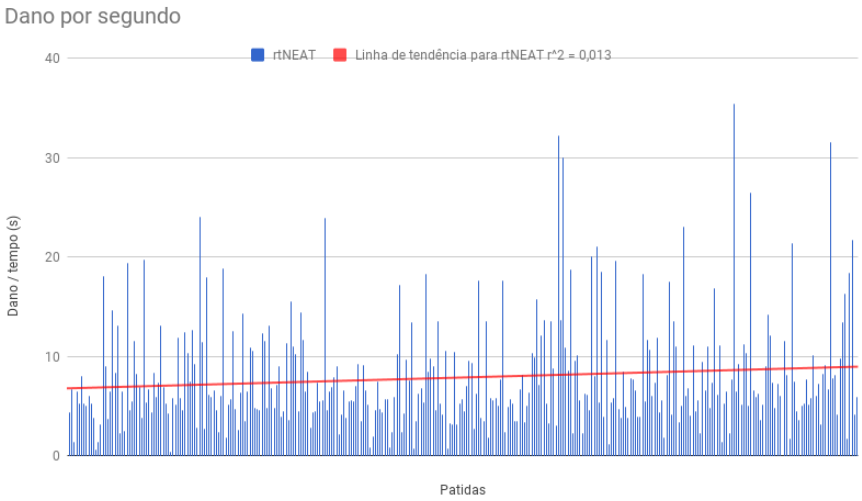


Figura 6.1: Gráfico com a relação de dano feito por segundo, em cada partida.

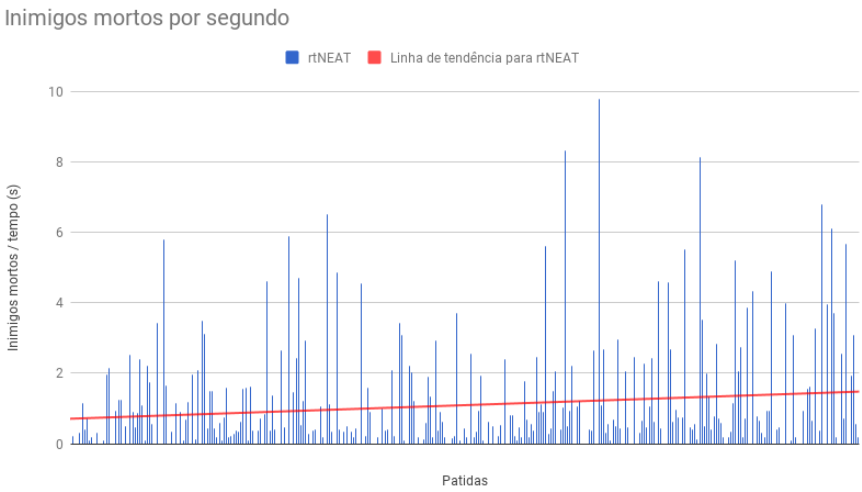


Figura 6.2: Gráfico com a relação de pontuação de inimigos mortos por segundo, em cada partida.

desclassificaria o *bot* de uma competição como a *Student StarCraft AI Tournament* (SSCAIT), por exemplo.

Para serem partidas mais justas e tomando como base Zhen e Watson (2013), os testes foram feitos fixando-se o número de unidades de ataque, dependendo da raça, tanto para o lado do *bot* quanto para os inimigos. Os mapas testados foram inspirados em Uriarte e Ontañón (2015), já que estes mapas foram criados com o intuito de testar *bots*, como o da Figura 6.3. O *layout* do mapa foi preservado como em Uriarte e Ontañón (2015) para ser uma arena em formato de losango, com as unidades iniciando em pontas opostas. As quantidades iniciais de soldados mais básicos são distintas dada a diferença de raças. Para a raça *Terran* são 12 unidades de ataque e para a raça *Zerg* 16 unidades, que são os oponentes. Esses números são baseados no custo de criação de cada unidade, na quantidade de dano aplicado e na frequência de ataque, para manter um combate mais justo.

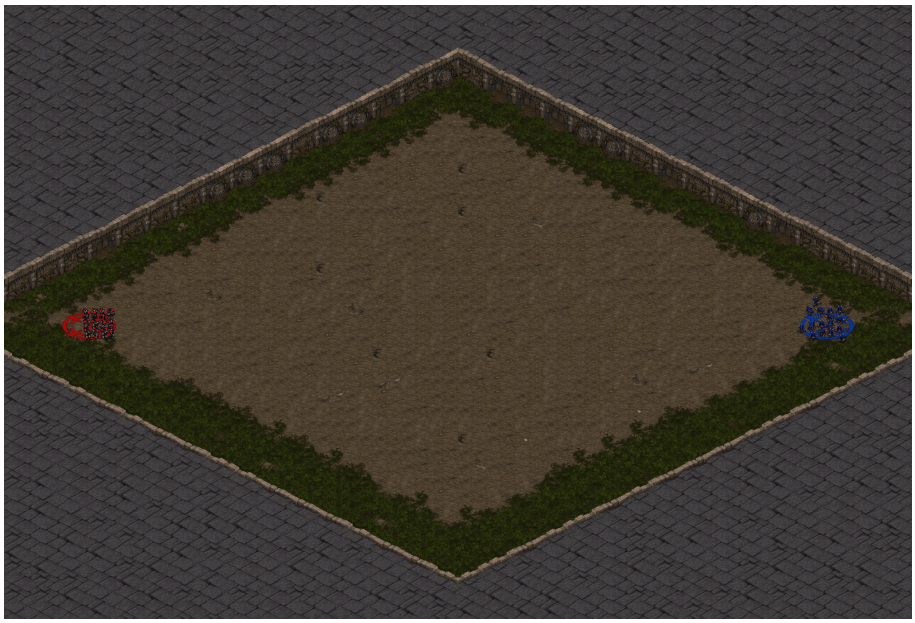


Figura 6.3: Mapa criado para testar os *bots*. Em vermelho estão as unidades controladas pela rtNEAT e em azul as unidades controladas pelo oponente.

Como oponente, optou-se por utilizar também outros *bots* já desenvolvidos, sendo eles o ZZZKbot[1] e o AILien[2], sendo ambos da raça *Zerg*. Estes *bots* foram escolhidos por motivos de compatibilidade de versão do BWAPI utilizada neste trabalho. Ambos participaram no torneio SSCAIT, voltado para a criação de *bots* para o jogo *StarCraft*. O ZZZKbot foi codificado de forma que as regras são feitas especificamente para cada oponente, de forma não clara. Entretanto, não foi possível descobrir mais sobre a lógica por detrás das unidades de ataque do AILien, pois foi disponibilizada somente a biblioteca já compilada.

Os gráficos a seguir são os resultados efetivos e individuais para cada variável analisada contra os outras IAs. Foram realizadas 15 partidas contra cada IA.

Observou-se que a melhor tática neste mapa era esperar no lugar de início até que o inimigo se aproximasse do *bot*. Como esta tática acabou sendo empregada pelos *bots* de ambos participantes, algumas partidas tiveram que ser encerradas manualmente, já que neste caso teriam duração infinita. Logo, neste caso não há lógica em utilizar métricas baseadas no tempo, como foi utilizado no treinamento.

Analisando o gráfico da Figura 6.4, é possível observar que o inimigo que recebeu menos dano foi o AILien, por empregar em todas as partidas a tática citada no parágrafo anterior. Já contra a IA do próprio jogo é quando teve-se os melhores resultados de maior dano, contra a raça *Zerg*, mesmo o maior valor absoluto sendo contra *Terran*, com 2.178 pontos de dano.

Observa-se na Figura 6.5 que em todas as partidas contra *Terrans* controlados pela IA do jogo houve eliminação total das unidades controladas pelo rtNEAT. Isto deve-se ao fato da configuração da IA do SC:BW estar configurada no modo agressivo. Contudo, em um caso pode-se observar que 75% da população do adversário foi eliminado e, em outro, nenhuma unidade adversária fora eliminada, não havendo necessariamente um padrão e nem uniformidade nas partidas.

[1]Mais informações em: `https://sscaitournament.com/index.php?action=botDetails&bot=ZZZKBot`

[2]Mais informações em: `https://sscaitournament.com/index.php?action=botDetails&bot=AILien`

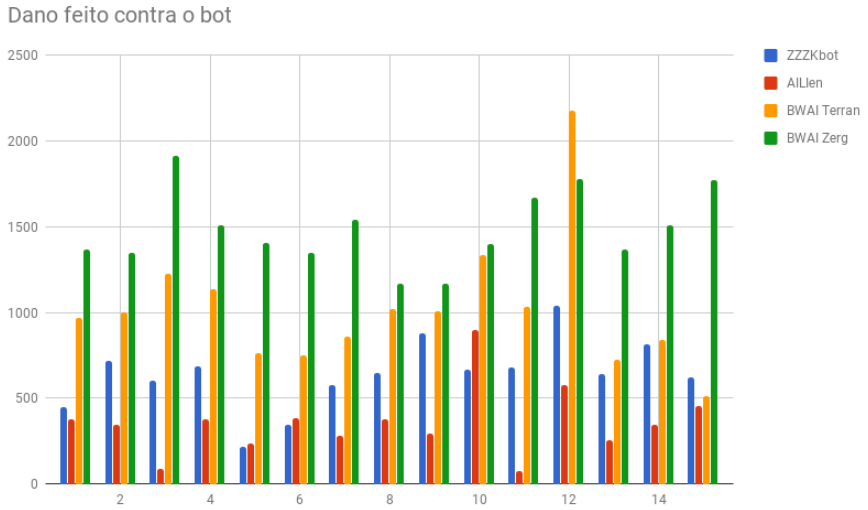


Figura 6.4: Gráfico com a relação de dano total feito pelo rtNEAT por partida contra cada *bot*.

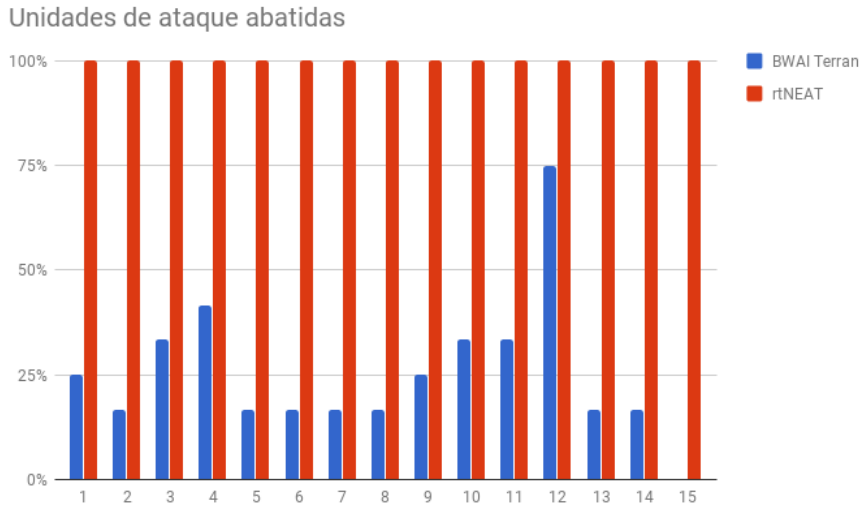


Figura 6.5: Gráfico com a relação de porcentagem de unidades de ataque que cada lado perdeu nas partidas, os oponentes sendo o rtNEAT contra a IA do próprio jogo jogando com a raça *Terran*.

Contra a IA do próprio jogo usando a raça *Zerg*, pode-se observar mais uniformidade nos resultados. Como pode-se ver na Figura 6.6, a porcentagem da população destruída dos *Zergs* manteve-se constante entre 25% e 50%, enquanto o rtNEAT fora massacrado, sendo que a configuração do modo agressivo estava ativa.

Já testando o rtNEAT contra o AILien, observa-se no gráfico da Figura 6.7 que sempre uma maior parte da população do rtNEAT fora eliminada do que a população do AILien. Contudo, em nenhuma das partidas o rtNEAT fora completamente eliminado.

É possível observar na Figura 6.8 que o ZZZKbot teve maior percentual de unidades eliminadas em 10 partidas. Em todas as partidas, menos de 50% da população do rtNEAT fora eliminado, demonstrando um comportamento mais agressivo.

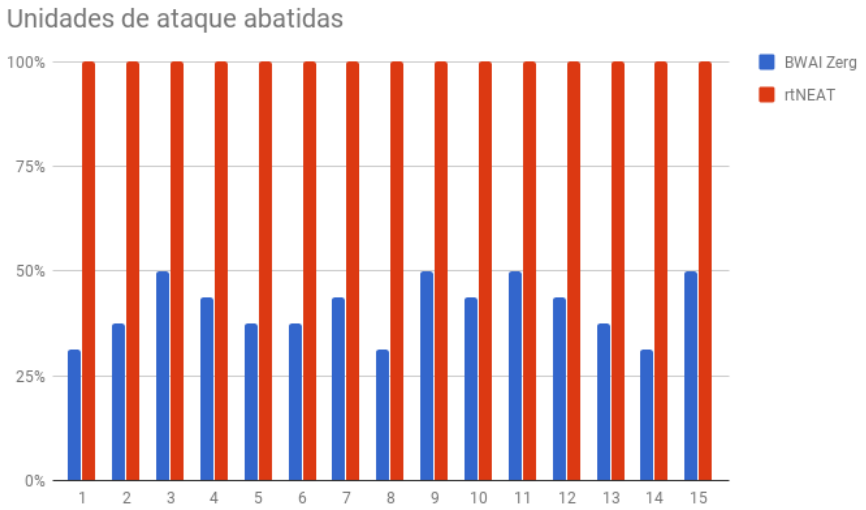


Figura 6.6: Gráfico com a relação de porcentagem de unidades de ataque que cada lado perdeu nas partidas, os oponentes sendo o rtNEAT contra a IA do próprio jogo jogando com a raça *Zerg*.

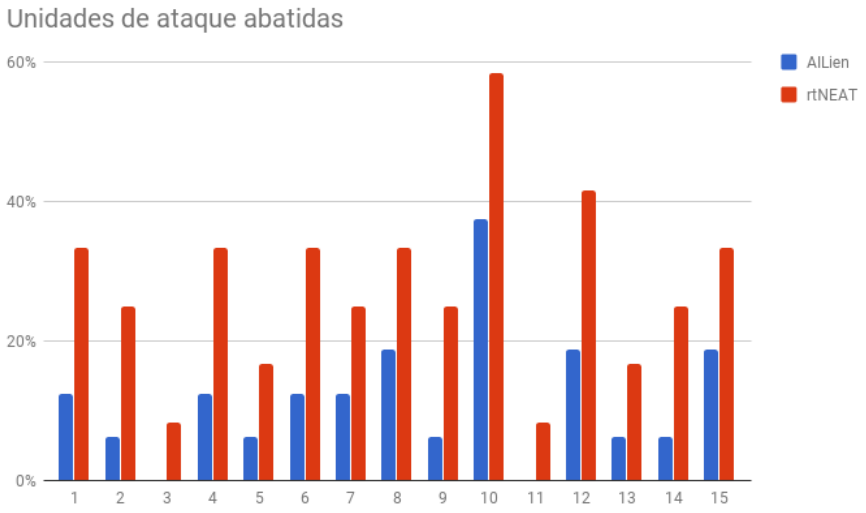


Figura 6.7: Gráfico com a relação de porcentagem de unidades de ataque que cada lado perdeu nas partidas, os oponentes sendo o rtNEAT contra o *bot* AILien.

Contra todos os adversários foi possível observar que todos tinham um comportamento fixo. A variabilidade dos resultados deu-se pelo rtNEAT. Por mais que tenha aprendido um comportamento, foi possível analisar que em cada partida alguns detalhes acabavam sendo um pouco distintos, como quantidade de unidades de ataques se agrupavam para atacar ou esperar no ponto. Assim, o rtNEAT mostrou-se eficaz para o problema, não seguindo necessariamente um padrão e nem obtendo vitória ou derrota absoluta, já que neste caso, o NPC não pode ser invencível, mas não ser uma tarefa trivial vencê-lo.

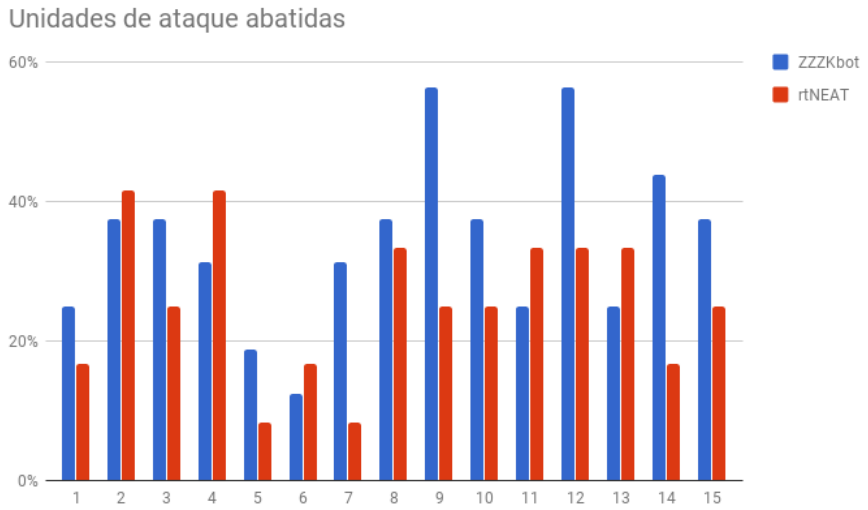


Figura 6.8: Gráfico com a relação de porcentagem de unidades de ataque que cada lado perdeu nas partidas, os oponentes sendo o rtNEAT contra o *bot* ZZZKbot.

6.1 Trabalhos Futuros

Para trabalhos futuros, sugere-se mudar a raça utilizada como teste, como por exemplo, focar na raça *protoss* já que é a que possui maior complexidade de controle. Inclusive pode-se generalizar o problema para não depender da raça a ser utilizada.

Também é possível utilizar outras EANNs, diversificando ainda mais o resultado. Sugere-se principalmente neste caso utilizar uma implementação do *Novelty Search* mais otimizada, focando-se na parte da necessidade de acontecer a evolução em tempo real, já que o *Novelty Search* geralmente resolve o problema em menos etapas, todavia, estas etapas acabam sendo mais lentas.

Recomenda-se também alterar a função de *fitness*, esta podendo mudar ao longo do tempo, como exemplo, um *fitness* para ataque e outro para defesa. Deste modo, comportamentos mais diversificados poderiam ser observados.

Pode-se analisar os dados de jogo em uma perspectiva mais subjetiva, onde avalia-se nesse caso a proximidade do comportamento dos *bots* com o de um humano. Nesse caso existem dois problemas relacionados: encontrar a *fitness* ideal e o esforço necessário para realizar a avaliação. Porém isso pode tornar o jogo mais interessante do ponto de vista do jogador, pois oferece um nível maior de dificuldade e imprevisibilidade de ações.

Além disso, pode-se utilizar outras heurísticas para avaliar os *bots* criados. No caso das EANNs, é interessante treinar separadamente contra cada raça também, para analisar o desempenho contra cada raça e se existem estratégias que sejam mais eficazes dependendo da raça.

Outra sugestão é aplicar EANNs em outras unidades, como por exemplo, nas unidades de coleta de recursos. Note que também é possível modelar EANNs para edifícios, todavia teriam saídas mais complexas como qual unidade construir e quando fazer uma pesquisa para aprimoramento de armas.

Capítulo 7

Conclusão

Conclui-se que o rtNEAT, dados os experimentos realizados, é um algoritmo adequado para o problema, pois não sobrecarrega o processamento do jogo. Também ofereceu um comportamento menos previsível do que os *bots* analisados, mas também não imbatível, como pode-se observar nos testes. Este é um ponto de equilíbrio interessante procurado para os jogos atuais.

Contudo, não se pode afirmar o mesmo do *Novelty Search*, pois com a implementação utilizada inviabilizou a utilização em tempo real em jogos, sobrecarregando o processamento.

Nota-se também que o jogo SC:BW oferece um campo interessante para estudos de IA em jogos, ainda existindo campos a serem explorados, como modelar EANNs para outras unidades do jogo.

Referências Bibliográficas

Asensio, J. M. L., Donate, J. P. e Cortez, P. (2014). Evolving artificial neural networks applied to generate virtual characters. Em *Computational Intelligence and Games (CIG), 2014 IEEE Conference on*, páginas 1–5. IEEE.

Buckland, M. (2002). *AI Techniques for Game Programming*. Premier Press, first edition.

Churchill, D., Preuss, M., Richoux, F., Synnaeve, G., Uriarte, A., Ontañnón, S. e Čertickỳ, M. (2016). Starcraft bots and competitions. *Encyclopedia of Computer Graphics and Games*, páginas 1–18.

Engelbrecht, A. P. (2007). *Computational intelligence: an introduction*. John Wiley & Sons.

Gabriel, I., Negru, V. e Zaharie, D. (2012). Neuroevolution based multi-agent system for micromanagement in real-time strategy games. Em *Proceedings of the Fifth Balkan Conference in Informatics*, BCI '12, páginas 32–39. ACM.

Kishimoto, A. (2004). Inteligência artificial em jogos eletrônicos. *Academic research about Artificial Intelligence for games*.

Lehman, J. e Stanley, K. O. (2011). Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation*, 19(2):189–223.

Moudříik, J., Baudiš, P. e Neruda, R. (2015). Evaluating go game records for prediction of player attributes. Em *Computational Intelligence and Games (CIG), 2015 IEEE Conference on*, páginas 162–168. IEEE.

Schrum, J. e Miikkulainen, R. (2009). Evolving multi-modal behavior in npcs. Em *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium On*, páginas 325–332. IEEE.

Stanley, K. O., Cornelius, R., Miikkulainen, R., D'Silva, T. e Gold, A. (2005). Real-time learning in the nero video game. Em *AIIDE*, páginas 159–160.

Stanley, K. O. e Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127.

Uriarte, A. e Ontañón, S. (2015). A benchmark for starcraft intelligent agents. Em *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*.

Zhen, J. S. e Watson, I. D. (2013). Neuroevolution for micromanagement in the real-time strategy game starcraft: Brood war. Em *Australasian Conference on Artificial Intelligence*, páginas 259–270. Springer.