Pontifícia Universidade Católica
do Rio de Janeiro

Adriano Brito Pereira

Q-Learning Pessimista: um algoritmo para geração de bots de jogos em turnos

Dissertação de Mestrado

Dissertação apresentada ao Programa de Pós-graduação em Informática da PUC-Rio como requisito parcial para obtenção do grau de Mestre em Informática.

Orientador: Prof. Ruy Luiz Milidiú

Rio de Janeiro
Julho de 2012

Pontifícia Universidade Católica
do Rio de Janeiro

Adriano Brito Pereira

Q-Learning Pessimista: um algoritmo para geração de bots de jogos em turnos

Dissertação apresentada ao Programa de Pós-graduação em Informática da PUC-Rio como requisito parcial para obtenção do grau de Mestre em Informática. Aprovada pela Comissão Examinadora abaixo assinada.

Prof. Ruy Luiz Milidiú
Orientador
Departamento de Informática - PUC-Rio

Prof. Eduardo Sany Laber
Departamento de Informática - PUC-Rio

Prof. Marcus Vinicius Soledade Poggi de Aragão
Departamento de Informática - PUC-Rio

Prof. Jose Eugenio Leal
Coordenador Setorial do Centro
Técnico Científico – PUC-Rio

Rio de Janeiro, 26 de Julho de 2012

Adriano Brito Pereira

Graduou-se em Ciência da Computação pela Universidade Federal do Rio de Janeiro em 2002. Realizou por dois anos iniciação científica em Algoritmos e Grafos e Inteligência Artificial pela COPPE. Estagiou por um ano e meio em uma grande empresa de tecnologia. Tem formação profissional como analista de sistemas especialistas para WEB nos últimos dez anos. Pesquisador e empreendedor autônomo de inteligência artificial e sistemas de aprendizado e cognição. Atualmente, é empreendedor na área de sistemas de aprendizado e inteligência artificial através da iniciativa Wisebots.

Ficha Catalográfica

Pereira, Adriano Brito

Q-Learning Pessimista: um algoritmo para geração de bots de jogos em turno / Adriano Brito Pereira; orientador: Ruy Luiz Milidiú; - Rio de Janeiro PUC, Departamento de Informática, 2012.

v., 63 f. : il. ; 30 cm

1. Dissertação (Mestrado em Informática) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática.

Inclui bibliografia.

1. Informática - Teses. 2. Inteligência artificial. 3. Jogos. 4. Aprendizado de Máquina. I. Milidiú, Ruy Luiz. II. Pontifícia Universidade Católica do Rio de Janeiro. III. Departamento de Informática. IV. Título.

CDD: 004

Para minha mãe, que sempre dedicou sua vida e apoio aos meus estudos

Para meu mestre Daisaku Ikeda, meu mestre de vida

Agradecimentos

Ao meu orientador Professor Ruy Luiz Milidiú pelo estímulo e apoio para a realização deste trabalho.

Aos professores Inês Dutra e Sheila Velloso, pelo apoio e admiração ao meu ingresso na pós-graduação e aos professores Marcus Vinicius Soledade Poggi de Aragão e Eduardo Sany Laber que estão acompanhando este trabalho

À PUC-Rio, pelos auxílios concedidos, sem os quais este trabalho não poderia ter sido realizado.

À minha mãe, pela educação, atenção e carinho de todas as horas.

Aos meus amigos Daniel Tornaghi e Marcos Vinícius Vasquez, por trocarem tantas idéias e filosofias que me inspiraram a concluir esse trabalho.

Aos meus parceiros de trabalho da Gazeus Games, João Felipe Assad e Guilherme Pereira e Oliveira pelo importante apoio e compreensão à realização deste trabalho.

À minha amiga e conselheira Maria Elisa Danna, pelo apoio e intensa orientação nos momentos de maiores dificuldades.

Ao meu respeitado amigo Fábio Ruiz, pelo apoio e amizade durante todo esse período.

A todos os professores, amigos e familiares que de alguma forma ou de outra me estimularam ou me ajudaram.

Resumo

Pereira, Adriano Brito; Milidiú, Ruy Luiz. **Q-Learning Pessimista: Um algoritmo para geração de bots de jogos em turnos**. Rio de Janeiro, 2012. 63p. Dissertação de Mestrado - Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Este documento apresenta um novo algoritmo de aprendizado por reforço, o Q-Learning Pessimista. Nossa motivação é resolver o problema de gerar bots capazes de jogar jogos baseados em turnos e contribuir para obtenção de melhores resultados através dessa extensão do algoritmo Q-Learning. O Q-Learning Pessimista explora a flexibilidade dos cálculos gerados pelo Q-Learning tradicional sem a utilização de força bruta. Para medir a qualidade do bot gerado, consideramos qualidade como a soma do potencial de vitória e empate em um jogo. Nosso propósito fundamental é gerar bots de boa qualidade para diferentes jogos. Desta forma, podemos utilizar este algoritmo para famílias de jogos baseados em turno. Desenvolvemos um framework chamado Wisebots e realizamos experimentos com alguns cenários aplicados aos seguintes jogos tradicionais: TicTacToe, Connect-4 e CardPoints. Comparando a qualidade do Q-Learning Pessimista com a do Q-Learning tradicional, observamos ganhos de 0,8% no TicTacToe, obtendo um algoritmo que nunca perde. Observamos também ganhos de 35% no Connect-4 e de 27% no CardPoints, elevando ambos da faixa de 50% a 60% para 90% a 100% de qualidade. Esses resultados ilustram o potencial de melhoria com o uso do Q-Learning Pessimista, sugerindo sua aplicação aos diversos tipos de jogos de turnos.

Palavras-chave

Q-learning pessimista; bots; aprendizado por reforço; jogos; inteligência artificial; aprendizado de máquina;

Abstract

Pereira, Adriano Brito. Milidiú, Ruy Luiz (Advisor). **Pessimistic Q-Learning: An algorithm to create bots for turn-based games**. Rio de Janeiro, 2012. 63p. MSc. Dissertation - Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

This document presents a new algorithm for reinforcement learning method, Q-Learning Pessimistic. Our motivation is to resolve the problem of generating bots able to play turn-based games and contribute to achieving better results through this extension of the Q-Learning algorithm. The Q-Learning Pessimistic explores the flexibility of the calculations generated by the traditional Q-learning without the use of force brute. To measure the quality of bot generated, we consider quality as the sum of the potential to win and tie in a game. Our fundamental purpose, is to generate bots with good quality for different games. Thus, we can use this algorithm to families of turn-based games. We developed a framework called Wisebots and conducted experiments with some scenarios applied to the following traditional games TicTacToe, Connect-4 and CardPoints. Comparing the quality of Pessimistic Q-Learning with the traditional Q-Learning, we observed gains to 100% in the TicTacToe, obtaining an algorithm that never loses. Also observed in 35% gains Connect-4 and 27% in CardPoints, increasing both the range of 60% to 80% for 90% to 100% of quality. These results illustrate the potential for improvement with the use of Q-Learning Pessimistic, suggesting its application to various types of games.

Keywords

Pessimistic Q-Learning; bots; reinforcement learning; games; artificial intelligence; machine learning;

Sumário

Lista de Figuras

Lista de Símbolos

α: Taxa de aprendizado. Determina o quão importante é uma informação nova adquirida em relação a uma informação anterior.

γ: Fator de desconto. Determina a importância de recompensas futuras. Um fator 0, por exemplo, consideraria apenas recompensas atuais. O fator de desconto, $0 <= \gamma < 1$ é o que faz o algoritmo Q-Learning convergir.

ξ: Taxa de exploração. Determina o quanto exploratório será o algoritmo.

δ: Taxa de falha do oponente. Taxa que determina falha nas escolhas das ações do oponente.

Dicionário de dados

Episódio: É uma seqüência de estados que representa uma partida completa de um jogo.

Q-values: Valores que representam um par estado-ação no algoritmo Q-Learning.

Função ação-valor: Refere-se a função Q do Q-Learning cujo valor depende do par estado-ação. Em ambientes não determinísticos, como no caso de jogos, considera-se a função ação-valor ao invés da função-valor. Na função-valor verifica-se apenas o valor de um estado.

Qualidade: Considera-se qualidade do resultado de testes sob um algoritmo de aprendizado por reforço, a porcentagem de soma de vitórias e empates sob o total de episódios verificados.

Fork: Estratégia de jogos utilizada para forçar o jogador adversário ou a perder o jogo ou a ficar numa situação de estado negativa, como por exemplo perdendo material, independente da ação do jogador.

Teste Clone: Estratégia de teste onde um bot criado por um algoritmo de aprendizado por reforço é confrontado por outro bot também criado por um algoritmo de aprendizado por reforço.

Teste Base: Estratégia de teste onde um bot criado por um algoritmo de aprendizado por reforço é confrontado por uma base de combinações de jogadas pré-estabelecidas.

MMORPG: Jogo online em massa para múltiplos jogadores.

"Aprender é a única coisa de que a mente nunca se cansa, nunca tem medo e nunca se arrepende".

Leonardo Da Vinci.

1. Introdução

Jogos e quebra-cabeças despertam o interesse das pessoas em virtude do seu entretenimento e também por causa de sua dificuldade, ou seja, exigem inteligência para resolvê-los. Jogos milenares como Xadrez, Gamão e Go são o foco de estudo de muitos matemáticos, e desde a metade do último século têm atraído cada vez mais a atenção dos cientistas da computação para resolver um problema cuja natureza é altamente complexa.

Bots, também conhecidos como web robots, são aplicações de software concebidas para simular ações humanas repetidas vezes e desempenhar tarefas rotineiras, como por exemplo num jogo de computador ser capaz de jogar contra um adversário através de recursos de inteligência artificial.

Nos últimos anos, com o aumento da utilização de redes sociais e a facilidade de acesso à internet, cresceu ainda mais o interesse de usuários a esse domínio de entretenimento: jogos online.

Motivados por esse contínuo interesse científico e comercial, cientistas, empresas de jogos e entusiastas buscam através de algoritmos de inteligência artificial implementar bots cada vez mais complexos e capazes de jogar contra seres humanos. Algumas razões para criação de bots capazes de jogar são: a geração de desafios que estimulam a competitividade, a busca por estratégias vencedoras, o estudo do comportamento de políticas adotadas pelos usuários, e a possibilidade de resolver quebra-cabeças que ainda não foram decifrados pelo homem.

Observamos portanto esse domínio de problema como vasto e promissor, e tomamos como nosso objetivo encontrar algoritmos e otimizações que possam contribuir cientificamente na geração de bots para jogos baseados em turno. Intencionalmente desejamos que esses bots tenham a maior qualidade possível,

porque desta forma conquistamos a capacidade de criar a dificuldade adequada ao jogador, para que este se mantenha motivado e estimulado a competir.

Jogos resolvidos são todos aqueles que, em que qualquer estado do jogo, pode-se predizer o fim de jogo, ou seja, vitória, derrota ou empate, [Van den Herik H.Jaap, 2002]. Jogos como TicTacToe, Damas e Connect-4, entre outros, já foram resolvidos computacionalmente. TicTacToe foi resolvido de maneira trivial, entretanto Damas e Connect-4 utilizam algoritmos e heurísticas bastante complexas. Por outro lado, jogos cujo espaço de estados é ainda maior, como por exemplo Xadrez, Gamão, Go e Reversi, ainda não foram resolvidos computacionalmente.

Verificamos que o estado da arte concentra suas forças em resolver pontualmente o problema, ou seja, encontrar soluções para resolver jogos específicos. Todavia, dadas as características desse problema estocástico, e uma vez que exista um cenário de planejamento sob incerteza, algoritmos de aprendizado por reforço têm mostrado resultados bem interessantes como indica [Tesauro, 1995] e sua respectiva solução TD-Gammon. TD-Gammon se tornou um programa capaz de vencer os melhores jogadores de Gamão do mundo, utilizando o conceito de reforço de bots que aprendiam um com o outro.

Propomos a experimentação de jogos já resolvidos pelo estado da arte, utilizando este modelo de estratégia de bots que aprendem um com o outro, e revisitamos a base do algoritmo de aprendizado por reforço para buscar otimizações que possam ser úteis para jogos que possuem mais dificuldade. Utilizamos principalmente a trivialidade do jogo TicTacToe para rastrear possíveis otimizações no algoritmo.

Além disso, para aumentar a qualidade dos bots obtidos, propomos uma otimização do algoritmo Q-Learning, que intitulamos de Q-Learning Pessimista.

Diferente da implementação comum de aprendizado por reforço, que sugere algoritmos de força bruta como parte da solução para obtenção de bons resultados, nós isolamos o problema de aprendizado sem a utilização dessa natureza de algoritmos. A utilização dessa estratégia tornou viável a observação de uma aproximação que pudesse revisitar as células de aprendizado do método Q-Learning, e propor uma revisão do conjunto de estados que levam um bot à derrota.

Identificamos portanto que o Q-Learning, uma vez que tenta maximizar a utilidade dos estados seguintes a um estado em observação, valoriza caminhos da árvore de estados que deveriam ser evitados. O Q-Learning Pessimista estende essa característica do algoritmo Q-Learning e minimiza a probabilidade de transição para estados pertencentes a estes caminhos.

Alem disso, criamos um framework chamado Wisebots capaz de proporcionar uma infra-estrutura de desenvolvimento adequada à realização de treino, teste e validação para implementações de jogos, no intuito de facilitar o acompanhamento de resultados dos algoritmos de aprendizado por reforço suportados.

Este documento está organizado em seis capítulos: Introdução, Metodologia, Resultados, Discussões Gerais, Conclusão e Referências.

No Capítulo 2 abordamos o domínio do problema e de que forma identificamos a otimização do algoritmo Q-Learning, ou seja, como funciona o Q-Learning Pessimista e as técnicas de teste e validação adotadas para os jogos TicTacToe, Connect-4 e CardPoints.

No Capítulo 3 apresentamos a importância da criação de um framework para acompanhamento dos resultados, como este framework foi desenvolvido e os benefícios obtidos com sua implementação.

No Capítulo 4 comparamos os ganhos obtidos na utilização do Q-Learning Pessimista em relação ao Q-Learning, com o confronto entre bots que utilizam, tanto o primeiro jogador quanto o segundo jogador. Esses resultados estão minuciosamente detalhados para os jogos TicTacToe, Connect-4 e CardPoints.

No Capítulo 5 evidenciamos as dificuldades encontradas com a implementação do método de aprendizado e indicamos pontos de discussão que ainda estão em aberto: O problema do crescimento do espaço de exploração, a automatização da fase de testes e a utilização de jogos com alta taxa de entropia.

No Capítulo 6 apresentamos as conclusões e vantagens observadas com a utilização do Q-Learning Pessimista sob a perspectiva do Q-Learning tradicional. Nesta seção também contemplamos trabalhos futuros e possíveis caminhos para evolução da solução em famílias de jogos por turnos.

2. Metodologia

2.1. Domínio

O domínio de referência deste trabalho é a família dos jogos baseados em turnos. Turnos, para nosso modelo de jogador versus jogador, é o tempo disponível para um dos jogadores tomar uma decisão. Esse tempo deve ser suficientemente grande para que o jogador tome sua decisão.

Em jogos estilos MMORPG, por exemplo, embora existam adversários em combate, os turnos são caracterizados por micro-ações cujo tempo computacional para tomar uma ação é muito pequeno para ser respondido por nosso modelo de aprendizado de máquina.

Em teoria dos jogos, diz-se que um ramo da economia que visualiza qualquer ambiente multi-agente como um jogo, desde que o impacto de cada agente sobre o ambiente seja significativo, não importa se os agentes são cooperativos ou competitivos. Jogos normalmente são de um tipo bastante especializado, que os teóricos denominam sistemas determinísticos de revezamento de dois agentes com informações imperfeitas, [Russel, Norvig, 2005]. Isso significa ambientes determinísticos completamente observáveis em que existem dois agentes cujas ações devem se alternar e em que os valores de utilidade no fim do jogo são sempre iguais e opostos, ou simétricos. Por exemplo, se um jogadora ganha um jogo de xadrez então ganha 1 ponto, o outro jogador necessariamente perde 1 ponto. E completamente observáveis pois estes sempre sabem observar as condições do estado atual onde se encontram.

Em geral, jogos são interessantes porque despertam nossa competitividade, e podem se tornar muito difíceis de serem solucionados por seres humanos. Por exemplo, o xadrez tem um fator médio de ramificação de cerca de 35, e as partidas com frequência chegam até a 50 movimentos por cada jogador. A árvore de busca tem aproximadamente 10^{104} nós. Dessa forma, o simples cálculo da decisão ótima por árvore de busca é inviável, mesmo com otimizações com podas e utilização de heurísticas.

Particularmente, a escolha de jogos baseados em turno, refere-se a necessidade de determinar a utilidade de um par estado-ação de forma estática, ou seja, sem a interferência de variáveis temporais que alterem o estado antes mesmo que a função utilidade seja calculada. Jogos baseados em turno têm ações temporais bem definidas, que permitem que o agente tenha o tempo necessário para pensar como agir antes da próxima ação do adversário.

Neste domínio, o número de iterações necessárias para que um agente aprenda a jogar, geralmente, é muito grande. Quanto mais complexo o ambiente ou maior o número de ações possíveis, maior portanto a necessidade de capacidade computacional para resolver o problema. Assim, é de significativa importância contribuições que permitam otimizações nos algoritmos de aprendizado por reforço.

2.2. Aprendizado por Reforço

O problema de aprendizagem por reforço é o mais geral dos problemas de aprendizagem. Diferente das técnicas supervisionadas de aprendizado, ao invés de ser informado sobre o que fazer por um instrutor, um agente de aprendizado por reforço aprende a partir de reforço, também chamado de recompensa. No caso de jogos, por exemplo, uma vitória caracteriza uma recompensa e uma derrota caracteriza uma punição, ou uma recompensa negativa. Aprendizagem por reforço é a técnica em que examinamos como um agente pode aprender a partir do sucesso e do fracasso, da recompensa e do castigo.

Em geral, a aprendizagem por reforço inclui o subproblema de aprender como o ambiente funciona. Por exemplo, sabemos que um agente pode aprender a jogar TicTacToe por aprendizagem supervisionada, recebendo exemplos de estados com os melhores movimentos para cada um desses estados. Entretanto, no domínio dos jogos, não existe um tutor fornecendo exemplos que viabilizem essa técnica. Um agente jogador de aprendizado por reforço, via de regra, desconhece o ambiente pois não tem como prever a jogada do adversário e nem se sua ação o leva a uma vitória.

O problema é que sem uma referência sobre qual estado é bom e qual estado é ruim, o agente não tem nenhuma base para decidir qual movimento executar. Em jogos como TicTacToe, Connect-4, Xadrez e Gamão, o reforço é recebido apenas no fim do jogo. Em outros jogos ou até mesmo em outros modelos, as recompensas podem vir em maior frequência.

A tarefa de aprendizado por reforço consiste em usar recompensas observadas para aprender uma política ótima, ou quase ótima, para o ambiente.

Fundamentos de aprendizado por reforço:

Aprendizado iterativo: o agente toma uma ação no ambiente e aguarda pelo valor do reforço, ou recompensa, que o ambiente retorna, assimilando dessa forma o valor obtido para tomar decisões no futuro.

Retorno atrasado: Uma boa recompensa por uma ação não significa necessariamente que é a melhor ação possível. No aprendizado por reforço, o intuito é alcançar objetivos a longo prazo.

Orientado ao objetivo: Pode ser completamente desconhecido o conhecimento sobre o ambiente, como acontece no domínio dos jogos, uma vez que não se conhece os lances do adversário. O que existe portanto, é um agente que age no ambiente tentando alcançar um objetivo.

Investigação versus exploração: Esse ponto é muito importante pois é esta decisão que fará o algoritmo buscar por novas possibilidades na árvore de decisões. Este é o momento de decidir, respectivamente, entre utilizar o conhecimento já obtido pelo reforço no ambiente ou aprender mais sobre o ambiente. Resumindo, o agente deve aprender quais ações maximizam as recompensas obtidas, mas também deve ser capaz de explorar ações ainda desconhecidas ou regiões pouco visitadas no espaço de estados. O que se utiliza, geralmente, é mesclar os métodos de investigação e exploração.

Para jogos em que os domínios são complexos, aprendizagem por reforço é o único caminho possível para treinar um programa para execução em níveis elevados. Por exemplo, dado o número muito grande de estados possíveis de um jogo, é muito difícil para um ser humano fornecer avaliações precisas para cada um desses estados. Ou seja, ao invés de treinar uma função de avaliação baseada em exemplos, o programa pode ser informado de quando ganhou ou perdeu, podendo usar essa informação para aprender uma função de avaliação que forneça estimativas mais precisas das chances de ganhar a partir de qualquer posição dada.

Pode-se considerar que a aprendizagem por reforço abrange toda a IA: um agente é colocado em um ambiente e tem que aprender a se comportar com sucesso nesse ambiente, [Russel/Norvig, 2005].

Nesse documento consideramos o modelo onde um agente de aprendizado aprende uma função ação-valor, ou função Q, que fornece a utilidade esperada de se adotar uma dada ação em um estado específico.

A primeira aplicação significativa de aprendizagem por reforço para jogos foi o jogo de damas escrito por [Arthur Samuel, 1967]. Este utilizou um programa que não observava recompensas nos estados finais, porém avaliava um estado qualquer conforme o peso da vantagem material. Isso foi suficiente para orientar o programa a espaços de estados correspondentes a um bom jogo de damas.

Já o sistema TD-Gammon de [Tesauro, 1995] ilustra o potencial da técnica de aprendizado por reforço. Simplesmente repetindo a função de avaliação tanto para o agente quanto para o agente adversário, fez com que o TD-Gammon aprendesse a jogar consideravelmente bem. Foram necessários cerca de 200.000 jogos de treinamento e duas semanas de tempo de processamento. Embora isso possa parecer uma quantidade grande de partidas, esta é apenas uma fração pequena do espaço de estados. Após 300.000 partidas de treinamento, o sistema foi capaz de chegar a uma padrão de jogo comparável ao dos três maiores jogadores de gamão do mundo.

2.3. Algoritmo Q-Learning

Aprendizado por reforço possui diversos métodos. Entre eles, o mais popular para trabalhar com aprendizado de jogos é o algoritmo Q-Learning.

Q-Learning [Watkins, 1989] é um método de aprendizado por reforço que não precisa de um modelo para representar o ambiente e pode ser usado online.

O algoritmo Q-Learning tem por característica atribuir um valor-utilidade ao par estado-ação, para que o agente de aprendizado possa escolher qual a melhor ação possível dado um estado do jogo. Na sua forma mais simples, ele consiste na atualização de valores Q descontados das recompensas esperadas por tomar uma ação a, em um estado s. Ainda assim, o algoritmo utiliza um fator de desconto γ, para garantir que os valores de Q sejam finitos, e uma constante de aprendizado α, para atribuir mais ou menos importância ao conhecimento recente, sendo: $0 < \alpha <= 1$ e $0 <= \gamma < 1$.

Uma outra característica desse algoritmo é que a função valor-ação Q aprendida se aproxima diretamente da função valor ótima Q*, sem depender da política que está sendo utilizada. Por isto, ele é dito off-policy.

Após executar a ação a, o agente passa do estado s e vai para um estado s', recebendo por esta ação uma recompensa imediata r. No estado s' obtido é feita uma busca, entre as ações disponíveis, para encontrar a ação a' que tenha o maior valor esperado, de forma a maximizar a expectativa do agente. Ainda assim, uma vez escolhida a ação a', o algoritmo utiliza uma taxa de exploração, geralmente de 30%, que escolhe se usa uma ação aleatória do universo de ações possíveis. Isso garante, que estados não explorados pela técnica sejam aprendidos.

Notação:

A(s): Conjunto de ações disponíveis no estado s.
R(s): Recompensa adquirida no estado s.
Q(s, a): Valor de atualização do q-learning para o par estado-ação.
Episódio: Período de tempo referente a uma partida.
α: Taxa de aprendizagem.
γ: Fator de desconto.

Algoritmo Q-Learning

Inicialize Q(s, a) arbitrariamente
Repita (para cada episódio)
    Inicialize s
    Repita para cada estado s do episódio
        Escolha a ação a ∈ A(s,a)
        Execute a ação a no estado s
        Observe os valores s'e r
        $Q(s, a) = Q(s, a) + \alpha[r + \gamma*(\max Q(s',a')) - Q(s, a)]$
        s = s'
    até que s seja terminal
até que acabe o número de episódios definidos

Uma propriedade importante desse algoritmo é que as ações usadas no processo de iteração da função Q, podem ser escolhidas usando qualquer critério de exploração, desde que cada par estado-ação seja visitado muitas vezes, a fim de garantir a convergência. A convergência dos valores de Q é garantida [MITCHELL, 1997], entretanto esta é extremamente lenta.

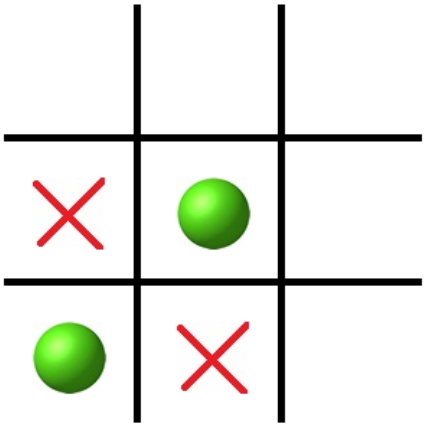
Para melhor entendimento do algoritmo Q-Learning, apresentaremos o nosso caso base e mostraremos abaixo o funcionamento do algoritmo para o jogo TicTacToe.

2.3.1. O Jogo TicTacToe

TicTacToe, conhecido popularmente no Brasil como Jogo da Velha, é um jogo de tabuleiro 3x3 para 2 jogadores, onde o objetivo é formar uma linha de 3 X ou 3 O na horizontal, vertical ou diagonal. O jogador que começa marca o X e assim alterna com o adversário jogando Bola. Se acabarem os espaços vazios é declarado empate ou velha.

O que torna este jogo muito interessante como caso de estudo é a facilidade de testar os algoritmos de aprendizado por reforço, uma vez que seu espaço de estados é pequeno, em torno de 5000. É também um jogo facilmente resolvido por um algoritmo de busca em árvore, como o MinMax, de profundidade 9. TicTacToe possui uma característica estática de início de jogo, um tabuleiro vazio, diferente de jogos de cartas por exemplo, o que facilita ainda mais sua definição.

Para seguirmos com nossa representação durante todo esse documento,



utilizaremos o jogo TicTacToe como base, representado da seguinte forma:

Figura 1: Exemplo de um estado cuja notação é [0, 0, 0, 1, 2, 0, 2, 1, 0]

Considere um array de 1 a 9 representando cada posição do tabuleiro 3x3 a começar pela parte superior esquerda até a parte inferior direita, seguindo a ordem por linha. Considere também 0 como uma posição vazia, 1 como uma jogada do jogador X e 2 como uma jogada do jogador O. Por exemplo, o array [1, 0, 0, 0, 1, 0, 0, 0, 1] representa o tabuleiro do jogo com a diagonal principal preenchida pelo X.

O estado [1, 0, 0, 1, 0 ,0 ,0 , 1] não é um estado válido, uma vez que não contempla as jogadas do jogador O. Um episódio de jogo, ou partida, é por exemplo a seguinte sequência:

Um episódio E válido:

[1, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 0, 0, 2, 0, 0, 0, 0, 0]
[1, 1 0, 2, 0, 0, 0, 0, 0]
[1, 1, 0, 2, 0, 0, 0, 0, 2]
[1, 1, 1, 2, 0, 0, 0, 0, 2] - Fim de jogo e vitória do jogador X

O conjunto A de ações que representam a sequência do episódio de jogo anterior é: A={1, 4, 2, 9, 3}, sendo A(X)={1, 2, 3} e A(O)={4, 9}.

2.3.2. Executando o Q-Learning

Inicializamos os vetores conforme o estado inicial do jogo. Considere o conjunto Q(s, a) como a representação dos valores ação-estado, também chamados de q-values, que são atualizados conforme o algoritmo Q-Learning.

s = [0, 0, 0, 0, 0, 0, 0, 0, 0], E={ }, A={ }, Q(s, a) ={ }.

Recompensas por fim de jogo:

R(s'') = + 100, se s'' for um estado de vitória para o jogador X.

R(s'') = -50, se s''for um estado de derrota para o jogador X.

R(s'') = 0, se s'' for um estado terminal de e mpate.

Para um Episódio E:

a) Escolhe-se uma ação a, dentro do conjunto de possibilidades do estado s. Essa ação pode ser tanto uma ação do conjunto Q que tenha um bom valor, por exemplo o maior por estratégia gulosa, ou pode ser uma ação aleatória, dada a necessidade de exploração do ambiente. Uma vez que Q é vazio, a princípio, o algoritmo gerará uma ação a, aleatória.

a = 1, A = {1}

b) Executa-se a=1 em s.

s = [1, 0, 0, 0, 0, 0, 0, 0, 0]

c) Observa-se os estados s' e as recompensas R(s').

Cada estado s' é uma possível ação do jogador X, uma vez que já houve uma jogada do ambiente através do jogador O. No exemplo abaixo, o ambiente poderia jogar em a=3 ou a=5 e portanto teríamos 2 possíveis estados de jogo para o jogador X:

S' = { [1, 1, 2, 0, 0, 0, 0, 0, 0, 0], [1, 0, 1, 0, 2, 0, 0 ,0 ,0], ...}

Como nenhum desses estados representa um fim de jogo, a recompensa será 0.

r = 0

d) Atualiza-se Q(s, a) conforme recompensa observada em s'.

$Q(s, a) = Q(s, a) + \alpha[r + \gamma * Q(s', a') - Q(s, a)]$

Como todos os Q(s', a') estão zerados inicialmente, então Q(s, a) permanece como 0.

Q([0, 0, 0, 0, 0, 0, 0, 0, 0], 1) = 0

e) Atualiza-se s com o estado s’ e repete-se o procedimento.

s = [1, 0, 0, 2 ,0 ,0, 0, 0 ,0], considerando A= {1, 4}.

E = { [1, 0, 0, 0, 0, 0, 0, 0, 0], [1, 0, 0, 2 ,0 ,0, 0, 0 ,0] }

Observamos que o algoritmo Q-Learning é um método de propagação de trás para frente, pois ele precisa chegar no final de um episódio para receber uma recompensa.

Para ficar ainda mais claro, considere para um episódio E, uma iteração cujo estado é pré-terminal.

s = [1, 0, 1, 2, 0, 2, 0, 0, 0].

Certamente que se o jogador X fizer a ação a=2, ganhará o jogo. Portanto no passo d do algoritmo, a recompensa R(s, a) será +100.

Entretanto, se o jogador X fizer a ação a=9, dará a chance do oponente jogar a’=5 e vencer o jogo, portanto R(s, a’) é -50.

Um outro fator interessante é considerar que, já que estamos lidando com um ambiente onde não se pode prever a reação do ambiente, uma vez que existe um oponente tentando vencer o agente, então verifica-se a importância da utilização de uma função ação-valor. Em outras palavras, um estado não tem um valor próprio, pois ele depende da ação que será tomada. Veja no exemplo:

Estado s1= [1, 0, 2, 0, 0, 0, 0, 0, 0]

Estado s2= [0, 0, 2, 1, 0, 0, 0, 0, 0]

Ambos os estados tem por consequência o estado [1, 0, 2, 1, 0, 0, 0, 0, 0] caso sejam tomadas respectivamente as ações a=4 em s1 e a=1 em s2, mas vem de estados diferentes.

O algoritmo Q-Learning, no seu formato mais simples, apresentado acima, usa uma tabela de q-values para armazenar os valores dos pares estado-ação obtidos. Entretanto, o algoritmo perde viabilidade muito rapidamente, conforme aumenta o nível de complexidade do jogo. O estado da arte, por sua vez, utiliza diversas estratégias para resolver este problema. Conforme demonstrado por [Tesauro, 1995], a utilização de uma rede neural adaptada como função de aproximação, ajudou o sistema TD-Gammon a contornar essa questão.

É comum encontrar também, diversas soluções de aceleradores por heurística para o Q-Learning, para mapear mais rapidamente estados que sejam mais significativos na representação genérica necessária para um bom agente.

Neste trabalho, vamos desconsiderar a importância de resolver esse problema, para focar numa otimização do próprio Q-Learning. Dessa forma, nosso intuito é de solucionar alguns casos críticos que não são considerados pelo algoritmo tradicional.

2.4. Algoritmo Q-Learning Pessimista

Seja qualidade o percentual de vitórias e empates de um jogo. Verificado por experimentação, o algoritmo Q-Learning, na sua forma mais simples, mesmo para o caso base do jogo TicTacToe, não atinge 100% de qualidade. Esse valor, estabiliza em aproximadamente 99% com apenas 12.000 episódios.

Desenvolvendo heurísticas, através de simples estratégias, conseguimos obter resultados de qualidade total num agente de inteligência artificial. Podemos listar algumas bem simples como: Não deixar um jogador formar uma linha, coluna ou diagonal com 3 peças, dado que este já tem 2 peças alinhadas, tentar formar uma linha de 3 peças, dado que o agente tem 2 peças alinhadas, evitar que o adversário realize um fork e buscar realizar um fork no adversário.

Existem poucas estratégias para serem aprendidas no TicTacToe. Mesmo aumentando consideravelmente o número de episódios, ou até mesmo alterando os parâmetros de aprendizado, não conseguimos resolver essa região crítica de 1% para conduzir o agente ao sistema de jogo perfeito usando o Q-Learning.

Analisamos portanto os episódios que faziam nosso agente de aprendizado falhar, a seguir:

Episódio falho 1:

[0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 0, 0, 0]
[2, 1, 0, 0, 0, 0, 0, 0, 0]
[2, 1, 0, 0, 0, 0, 0, 1, 0] - Essa ação ingênua leva necessariamente derrota *
[2, 1, 0, 0, 2, 0, 0, 1, 0]
[2, 1, 0, 0, 2, 0, 0, 1, 1]

[2, 1, 0, 0, 2, 0, 2, 1, 1] - O jogador adversário cria um fork
[2, 1, 0, 0, 2, 1, 2, 1, 1]
[2, 1, 0, 2, 2, 1, 2, 1, 1] - O jogador adversário ganha na diagonal inversa

* Consideramos esta ação como ingênua porque não é fácil considerar que, já no segundo movimento do agente, faltando ainda 5 movimentos para o fim do jogo, a derrota já é certa.

Episódio falho 2:

[0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 1, 2, 0, 0, 0, 0, 0]
[0, 0, 1, 2, 0, 1, 0, 0, 0]
[0, 0, 1, 2, 0, 1, 0, 0, 2]
[0, 1, 1, 2, 0, 1, 0, 0, 2] - Essa ação ingênua leva necessariamente derrota **
[2, 1, 1, 2, 0, 1, 0, 0, 2] - O jogador adversário cria um fork
[2, 1, 1, 2, 0, 1, 1, 0, 2]
[2, 1, 1, 2, 2, 1, 1, 0, 2] - O jogador adversário ganha na diagonal principal

** Repare que aparentemente a ação é boa, uma vez que é uma tentativa do agente de formar a primeira linha do tabuleiro. Entretanto a ação força o adversário a criar o fork no agente e portanto ganhar o jogo.

Episódio falho 3:

[0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0, 2]
[0, 0, 0, 0, 1, 0, 0, 1, 2]
[0, 2, 0, 0, 1, 0, 0, 1, 2] - Mais uma ação que força receber um fork

[0, 2, 0, 0, 1, 0, 1, 1, 2]
[0, 2, 2, 0, 1, 0, 1, 1, 2] - O jogador adversário cria um fork
[0, 2, 2, 1, 1, 0, 1, 1, 2]
[0, 2, 2, 1, 1, 2, 1, 1, 2] - O adversário ganha na última coluna

Reparamos portanto que a atualização da função-valor para estados não triviais de derrota, como os casos dos forks listados acima, não refletem a real condição do estado do jogo.

Intuitivamente, é fácil o entendimento do conceito de fork. Porém, para o Q-Learning, que é um algoritmo iterativo de programação dinâmica de atualização de valores para os pares estado-ação, esta não é uma tarefa fácil. E existe uma razão matemática e determinística para isso.

Observe, por exemplo, o Episódio falho 3:

A ação a=7 para o estado [0, 2, 0, 0, 1, 0, 0, 1, 2] é o exato instante onde o agente falha. Todavia esta não é uma ação baseada em exploração. É uma ação baseada em investigação, ou seja, na atualização da função-valor. O agente escolheu essa ação por um motivo bastante simples. A função de atualização do Q-Learning visa maximizar a recompensa dos próximos pares de estado s'e ação a' do jogo.

Repare que, se o ambiente responder com a ação a=1, o estado s' passa a ser s' = [2, 2, 0, 0, 1, 0, 1, 1, 2]. Para este estado, existe uma ação a', tal que a'= 3, tal que a recompensa obtida pelo estado s = [0, 2, 0, 0, 1, 0, 0, 1, 2] e ação a = 7, é máxima. Afinal, tomando a'= 3, s''= [2, 2, 1, 0, 1, 0, 1, 1, 2] e o jogador 1 ganha na diagonal inversa, recebendo a recompensa por sua vitória.

Em outras palavras, a função de maximização do Q-Learning foi muito

otimista ao aceitar maximizar a possibilidade de recompensa de um estado perdido, uma vez que o estado s combinado com a ação a=7 força o fork do adversário, derrotando o agente.

A noção intuitiva dessa investigação do algoritmo, nos fez entender que caminhos que conduzem nosso agente à derrota inevitável não deveriam ser maximizados. Todavia, devemos mexer com cuidado na função de atualização do Q-Learning, de tal forma que esta mantenha suas características de viabilidade e convergência.

Foi então que criamos o algoritmo Q-Learning Pessimista. Pessimista porque, ao invés de usar sempre uma função de maximização no cálculo dos q-values, ele utiliza uma função de minimização quando o agente é conduzido a derrota.

A vantagem da simples substituição de uma função de minimização no lugar da função de maximização, é a possibilidade de manter todas as propriedades de convergência e viabilidade, uma vez que as funções de mínimo são complementares às funções de máximo.

Essa atualização pessimista, proporciona uma suavização nos valores maximizados pelo Q-Learning na região crítica, conforme podemos evidenciar via experimentação científica. Essa suavização, desprioriza esses estados problemáticos, evitando portanto os forks. Atingimos portanto, com a otimização proposta o resultado de 100% de qualidade com menos de 3000 episódios.

Algoritmo Q-Learning Pessimista

Inicialize Q(s, a) arbitrariamente
Repita (para cada episódio)
    Inicialize
    Armazene s em P
    Repita para cada estado s do episódio
        Escolha a ação a E A(s)
        Execute a ação a no estado s
        Observe os valores s'e r
        $Q(s, a) = Q(s, a) + \alpha[r + \gamma*(\max Q(s',a')) - Q(s, a)]$
        s = s'
        Armazene s em P
    até que s seja terminal
    Repita para cada estado s de P
        $Q(s, a) = Q(s, a) + \alpha[r + \gamma*(\min(s',a')) - Q(s, a)]$
    até que não existam mais estados em P
até que acabe o número de episódios definidos

O algoritmo Q-Learning Pessimista, rapidamente consegue identificar os estados falhos no Q-Learning Tradicional e minimizar a chance do agente escolher as ações que o levam àqueles estados.

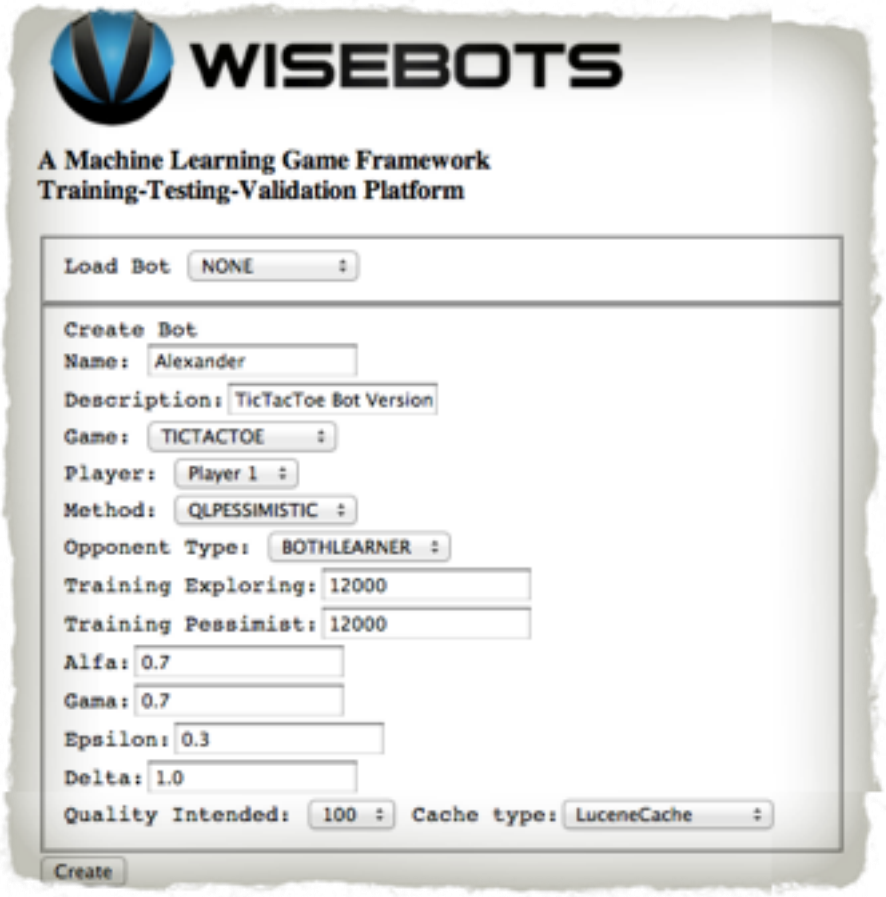
Repare que, principalmente no TicTacToe, bastava a utilização de um algoritmo de força bruta, como um MinMax, dentro da estratégia de exploração do algoritmo, para em poucos níveis resolver esse problema. Ele identificaria a derrota por fork mais a frente da árvore de decisão e evitaria tomar aquela ação. Entretanto, como temos por objetivo utilizar esse aperfeiçoamento do Q-Learning para famílias de jogos mais complexos, a utilização de algoritmos de força bruta dificultaria encontrar tais otimizações.

Identificamos que para todos os jogos implementados, o algoritmo Q-Learning Pessimista desvalorizou os caminhos que levam a derrota iminente do agente, solucionando assim o problema do fork. Uma característica bastante interessante do algoritmo é que, embora exista a desvalorização do estados pertencente a um episódio com cenário de derrota, isso não inviabilizou os estados do caminho. Em outras palavras, as características de viabilidade do fator de desconto do algoritmo Q-Learning foram mantidas, garantindo assim a consistência do algoritmo.

3. Framework Wisebots

Conforme a evolução na implementação dos algoritmos de aprendizado por reforço, cada vez mais demonstrou-se a importância de uma ferramenta de aprendizado que controlasse as seguintes características:

A configuração de parametrização do sistema
A facilidade de integração de novos jogos
A observação visual da fase de treino
A criação e manutenção da memória do bot gerado



F

3.1. A configuração de parametrização do sistema

Para garantir a facilidade e velocidade na criação de novos bots ou até mesmo a criação destes em paralelo, foi preciso criar uma ferramenta WEB, desenvolvida em JAVA e HTML5, que permitisse essa rápida configuração.

Criamos 3 grupos principais de configuração.

O primeiro grupo de parametrizações refere-se às características do bot: Nome do bot, descrição do bot, o jogo que ele pretende aprender a jogar e se este pretende ser o primeiro jogador ou o segundo jogador.

O segundo grupo de parametrizações refere-se às características primárias do algoritmo Q-Learning Pessimista: O método de aprendizado, a quantidade de episódios de treinamento, a configuração das variáveis do algoritmo, α, γ, ξ e δ.

O terceiro grupo é exclusivamente para a escolha do tipo de oponente.. Inicialmente utilizamos um oponente aleatório, que funcionava bem no jogo TicTacToe. Entretanto este oponente demonstrou ser muito fraco no Connect-4 e em qualquer jogo cujo espaço de estados possua um tamanho significativo.

A posteriori utilizamos um algoritmo especialista. Essa solução utilizava um algoritmo que jogava TicTacToe de forma perfeita, com a implementação de um MinMax com profundidade 9. Entretanto, o algoritmo de aprendizado aprendia a melhor forma de vencer o especialista e ficava limitado à capacidade de aprendizado de exploração do próprio especialista. Essa característica inviabilizava os testes de validação realizados contra humanos.

Por último, seguindo a estratégia do TD-Gammon [Tesauro, 1995], utilizamos um oponente que também aprendia com o algoritmo Q-Learning, e também com sua própria taxa de exploração. Esse mostrou ser o melhor cenário, e portanto tornou-se o foco exclusivo de estudo para o aprendizado.

3.2. A facilidade de integração de novos jogos

Outra preocupação iminente era a dificuldade anterior de integração de novos jogos no algoritmo. Criamos portanto interfaces que representam unicamente as regras de um jogo, que é o que exatamente um jogo significa, suas regras. Uma vez implementadas, essas interfaces são integradas ao sistema, e o jogo está preparado para criar seus agentes de aprendizado.

Essas interfaces, embora escritas em Java, podem ser escritas em qualquer linguagem de programação, pois esta implementação foi desenvolvida como webservice. A interface exige apenas que o seguinte contrato seja respeitado:

A inicialização do estado do jogo.
A obtenção das possíveis ações de um estado do jogo.
A obtenção de um novo estado de jogo ao executar uma ação.
A verificação de vitória de um jogador.
A verificação de fim de jogo.

Entretanto, havia uma outra dificuldade. Os testes de validação humana. Para concretizar esse objetivo, implementamos clientes web que, ainda que o agente esteja na fase de treinamento, a memória do agente já seja capaz de fornecer jogabilidade contra humanos.

A implementação dessas interfaces podem ser feitas com qualquer plugin compatível com web: HTML5, javascript, flash, ActiveX. Esta fase é essencial para que a validação do bot seja monitorada por seres humanos e testada de forma simples e rápida.

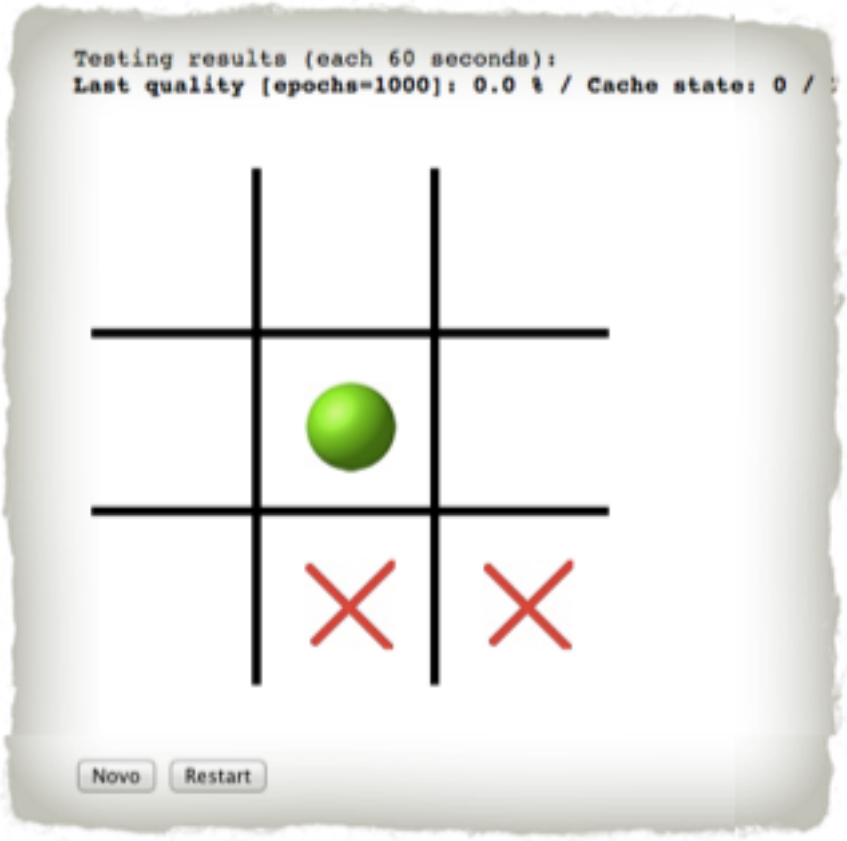


Figura 3: Exemplo de cliente HTML5 implementado para validação do jogo TicTacToe

3.3. A criação e manutenção da memória do bot e a inspeção visual

Outro aspecto importante é o acompanhamento visual por meio de gráficos da evolução da fase de treino. Uma vez que o crescimento do aprendizado pode ser muito lento, pode demorar dias em virtude de um alto número de episódios, esse acompanhamento demonstrou ser fundamental.

Uma vez que faz parte da característica do Q-Learning armazenar uma tabela com os q-values dos pares estado-ação, essa tabela é considerada então a memória do bot. Essa memória define a qualidade do bot gerado juntamente com um algoritmo de busca, seja guloso ou usando qualquer outra estratégia.

O mais importante é que essa memória fosse portável, para que pudesse ser testada com os mais diversos clientes de jogos possíveis. Para tanto utilizamos algumas estratégias de cacheamento de informações, como MemBase da [CouchBase] e Lucene da [Apache]. No nível de treinamento, por motivos óbvios de performance, é utilizada a memória RAM, e que para efeito de funcionamento dos algoritmos, exigiu uma máquina 8-core com 24GB RAM de memória.

3.4. Jogos implementados

Além do TicTacToe, já explicado anteriormente, concluimos nossos testes com mais 2 jogos significativos. São eles o Connect-4 e o CardPoints.

O Connect-4 consiste num jogo de tabuleiro para 2 jogadores, geralmente de dimensões 6x7, onde o objetivo é formar uma linha de 4 peças da mesma cor na horizontal, vertical ou diagonal.

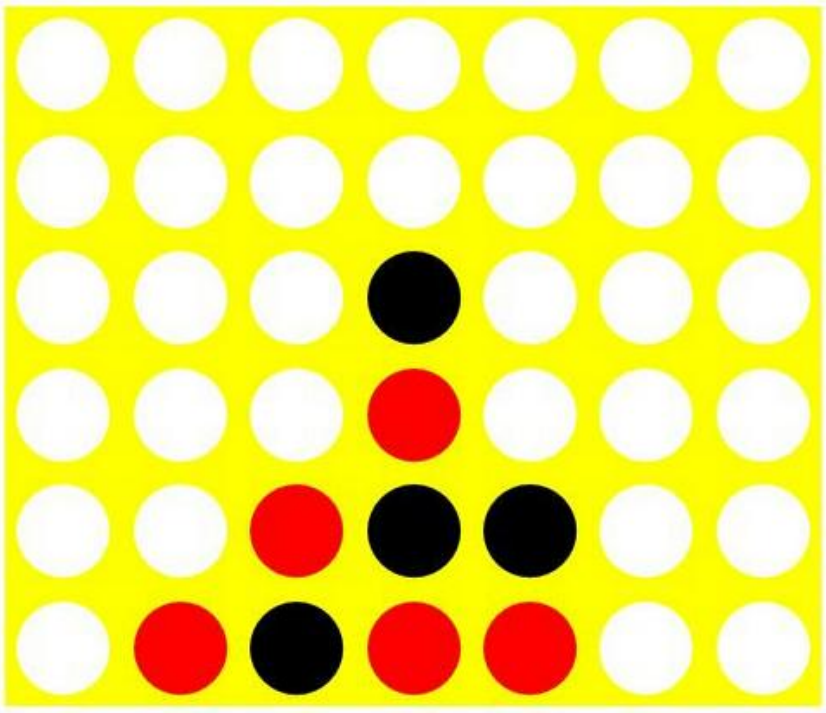


Figura 4: Connect-4 numa posição de aparente vantagem da cor vermelha

O interessante na escolha do Connect-4 é que, mesmo com características semelhantes ao do TicTacToe, o espaço-estado de complexidade é muito maior. Comparativamente, como log na base 10, o espaço-estado de complexidade do TicTacToe é 3 enquanto o do Connect-4 é 13.

O Connect-4 possui uma solução baseada na escolha de heurísticas complexas e redes neurais. Esse algoritmo é muito difícil de ser implementado. O que os desenvolvedores do jogo costumam fazer na prática é utilizar mecanismos de busca mais simples como MinMax ou Alfa-Beta Prunning, combinados com heurísticas triviais, como por exemplo não permitir formar uma linha, coluna ou diagonal. Essa estratégia torna o jogo suficientemente competitivo para jogar contra o ser humano. Conforme verificamos, existem poucas implementações do jogo com o algoritmo perfeito, e são muito difíceis de entendimento.

O CardPoints, por sua vez, é um jogo simples de cartas, onde coloca-se um número par de cartas sorteadas ao acaso em aberto numa linha numa mesa. É um jogo de 2 jogadores cujo objetivo é somar o maior valor de pontuação possível , sendo que só é possível pegar a carda mais à esquerda ou mais à direita.

Figura 5: CardPoints e uma sequência sorteada ao acaso. As cartas A e K são de maior valor.

Para os nossos testes de aprendizado esse jogo é bastante importante, pois além de não ser um jogo de tabuleiro propriamente dito, é um jogo que conta também com o fator entropia. Esse fator entropia é determinado no sorteio na inicialização do jogo.

Vale ressaltar que, o jogo CardPoints, é facilmente resolvido por programação dinâmica. A vantagem na utilização desse jogo, é além do fator aleatório, é poder aumentar o espaço de estados do jogo aumentando o número de cartas dispostas.

4. Resultados

Para realizar os testes nós utilizamos uma máquina com as seguintes características:

8 processadores Intel
24 GB RAM
1 TB memória física
Sistema operacional: Ubuntu 11.10

Servidor Web: Jetty 6.1.24
Servidor Banco de dados NOSQL: CouchBase 1.8

4.1. Técnicas de teste

Para compreender os avanços obtidos com a aproximação do algoritmo Q-Learning Pessimista, aplicamos algumas técnicas de teste que pudessem dar uma garantia razoável de ganho com os resultados.

Em virtude disso, sugerimos 3 técnicas de teste: O teste contra o próprio agente de aprendizado, o teste contra um banco de dados de estados pré-definidos e o teste de viabilidade por inspeção.

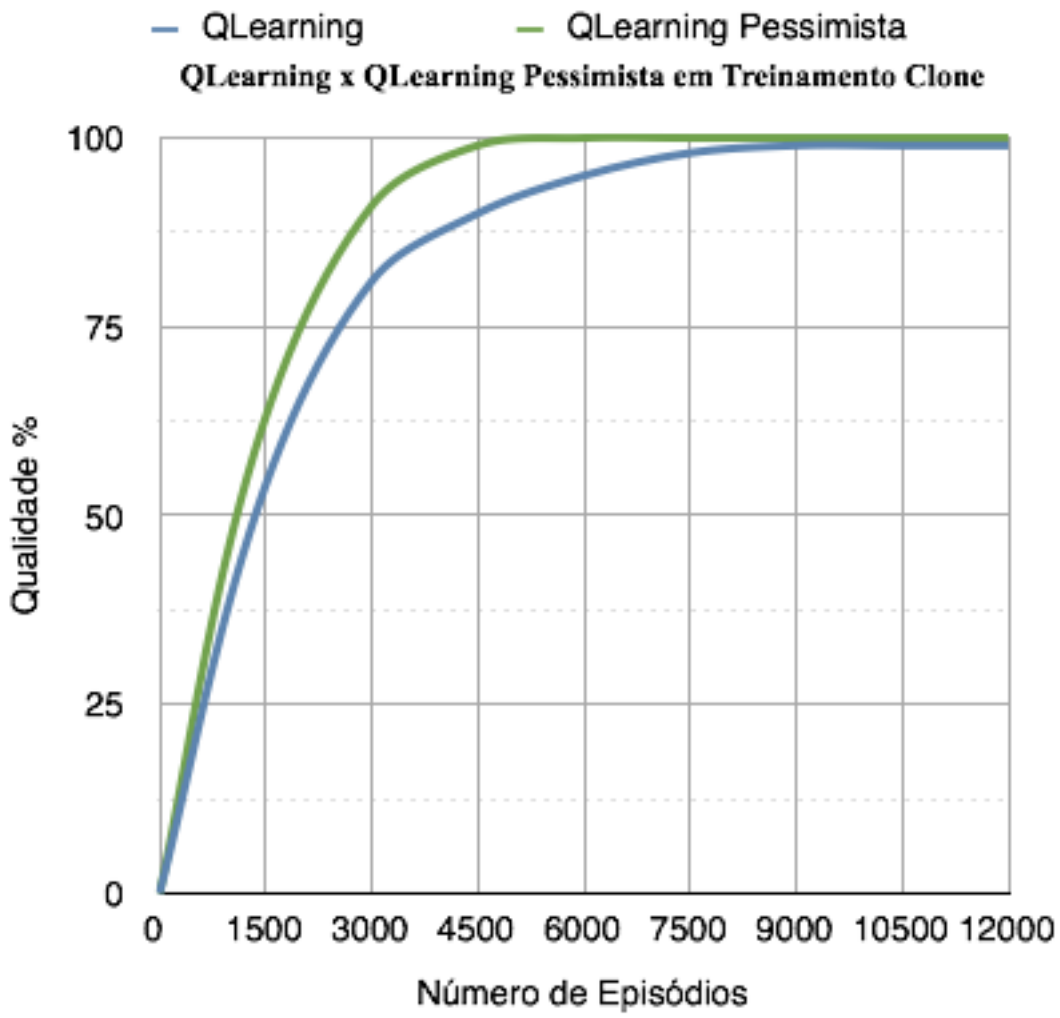
Teste clone, ou teste contra o mesmo agente de aprendizado, significa usar a base de conhecimento do primeiro jogador obtida com o armazenamento dos q-values originados do Q-Learning contra a base de armazenamento dos q-values gerados pelo mesmo algoritmo para o segundo jogador.

Para fazer a comparação das técnicas, aplicamos o Q-Learning Pessimista apenas à base do primeiro jogador e, por sua vez, confrontamos esta nova base à base de q-values obtidas apenas pelo Q-Learning do segundo jogador.

Teste base, ou teste contra o banco de dados funciona como uma espécie de teste automatizado para verificação. Ele consiste da geração, à priori, de conjuntos de estados que sejam interessantes de serem testados. No caso do jogo TicTacToe, em virtude do baixo número de estados, é possível verificar todos os estados do jogo em tempo computacional baixo. Entretanto, em jogos como Connect-4, montamos uma base com 1000 episódios. Esses episódios contemplam verificação de busca pela vitória, de não deixar ser derrotado e de não permitir o fork. Além disso, como temos conhecimento de boas heurísticas para o jogo, observamos se os lances iniciais casam com o que essas heurísticas sugerem. A mais óbvia de todas, por exemplo, é que a vantagem por jogar no centro do tabuleiro, ao começar, é maior. Por medida de nomenclatura, vamos chamar esse teste de base.

O teste de viabilidade por inspeção consta da verificação humana da qualidade do agente no jogo. Jogamos contra o nosso próprio agente de aprendizado e observamos como se comporta.

4.2. Resultados para o Jogo TicTacToe



a) Teste com taxas medianas

alfa = 0.7

gama = 0.7

epsilon = 0.3

número de episódios = 12 mil

tempo de execução = 5 segundos

número de estados explorados = 4136

Algoritmo	Teste aleatório	Teste clone	Teste base	Inspeção
Q-Learning	54%	88%	99%	Forte
Q-Learning Pessimista	N/D	100%	100%	Invencível

Esses resultados ilustram o ganho de 1% do Q-Learning Pessimista sobre o Q-Learning. Embora essa porcentagem seja pequena, ela é um ganho considerável, pois atua sobre uma área crítica de aprendizado que, sem mecanismos de busca, inviabilizam a obtenção de um bot incapaz de ser derrotado.

O teste clone também tem um importante significado. Ele mostra que o comportamento de dois agentes de aprendizado de reforço um contra o outro, proporciona 100% de empates, não permitindo que nenhum dos dois ganhe durante o próprio aprendizado.

No teste de inspeção, jogamos contra o agente de TicTacToe e conseguimos vencê-lo em apenas uma combinação de estados quando o algoritmo é apenas o Q-Learning. Quando aplicado o Q-Learning Pessimista, não conseguimos mais vencer o adversário.

O teste aleatório é executado antes do início do aprendizado, executando 100 mil episódios, apenas para ilustrar a taxa média de vantagem entre o jogador 1 e o jogador 2. Salientamos que se esse número for muito alto para o agente aprendiz, isso afeta diretamente no resultado do aprendizado. Por exemplo, se a chance inicial de um agente for 99%, certamente que este é um limite inferior para os resultados de qualidade.

b) Teste com baixo fator de convergência e alto valor da taxa de aprendizado

alfa = 0.9
gama = 0.1
epsilon = 0.3
número de episódios = 12 mil
tempo de execução = 6 segundos
número de estados explorados = 6591

Algoritmo	Teste aleatório	Teste clone	Teste base	Inspeção
Q-Learning	54%	85%	95%	Forte
Q-Learning Pessimista	N/D	94%	99%	Forte

Quando reduzido o fator de convergência, o número de 12 mil testes não são suficientes para garantir 100% de qualidade no Q-Learning Pessimista. Realizamos ainda testes com 15000, 20000 e 30000 episódios, e apenas com 30000 episódios o algoritmo Q-Learning Pessimista chegou ao resultado de 100%, que foram confirmados no teste de inspeção.

c) Teste com alto fator de convergência e baixo valor da taxa de aprendizado

alfa = 0.3
gama = 0.9
epsilon = 0.3
número de episódios = 12 mil
tempo de execução = 28 segundos
número de estados explorados = 5586

Algoritmo	Teste aleatório	Teste clone	Teste base	Inspeção
Q-Learning	54%	75%	97%	Forte
Q-Learning Pessimista	N/D	100%	100%	Invencível

Observamos que com um alto fator de convergência e baixo valor de aprendizado, os resultados são semelhantes aos cuja taxa é mediana como em [a].

d) Teste com médio fator de convergência, valor médio da taxa de aprendizado e baixa taxa de exploração

alfa = 0.7

gama = 0.7

epsilon = 0.01

número de episódios = 12 mil

tempo de execução = 7 segundos

número de estados explorados = 3888

Diminuímos a taxa de exploração para analisar o comportamento do algoritmo e constatamos que ele garante um combate em tempo de treinamento acirrado, de 100% a cada 1000 épocas medidas das 12000 treinadas. Esse resultado, mais uma vez mostra a consistência da técnica de treinamento que garante que os dois agentes estão de fato aprendendo.

Os resultados finais, embora tenham diminuído o número de estados explorados, garante a mesma qualidade obtida em [a].

Algoritmo	Teste aleatório	Teste clone	Teste base	Inspeção
Q-Learning	54%	99%	99%	Forte
Q-Learning Pessimista	N/D	100%	100%	Invencível

e) Teste com médio fator de convergência, médio valor de aprendizado e alta taxa de exploração

alfa = 0.7

gama = 0.7

epsilon = 0.9

número de episódios = 12 mil

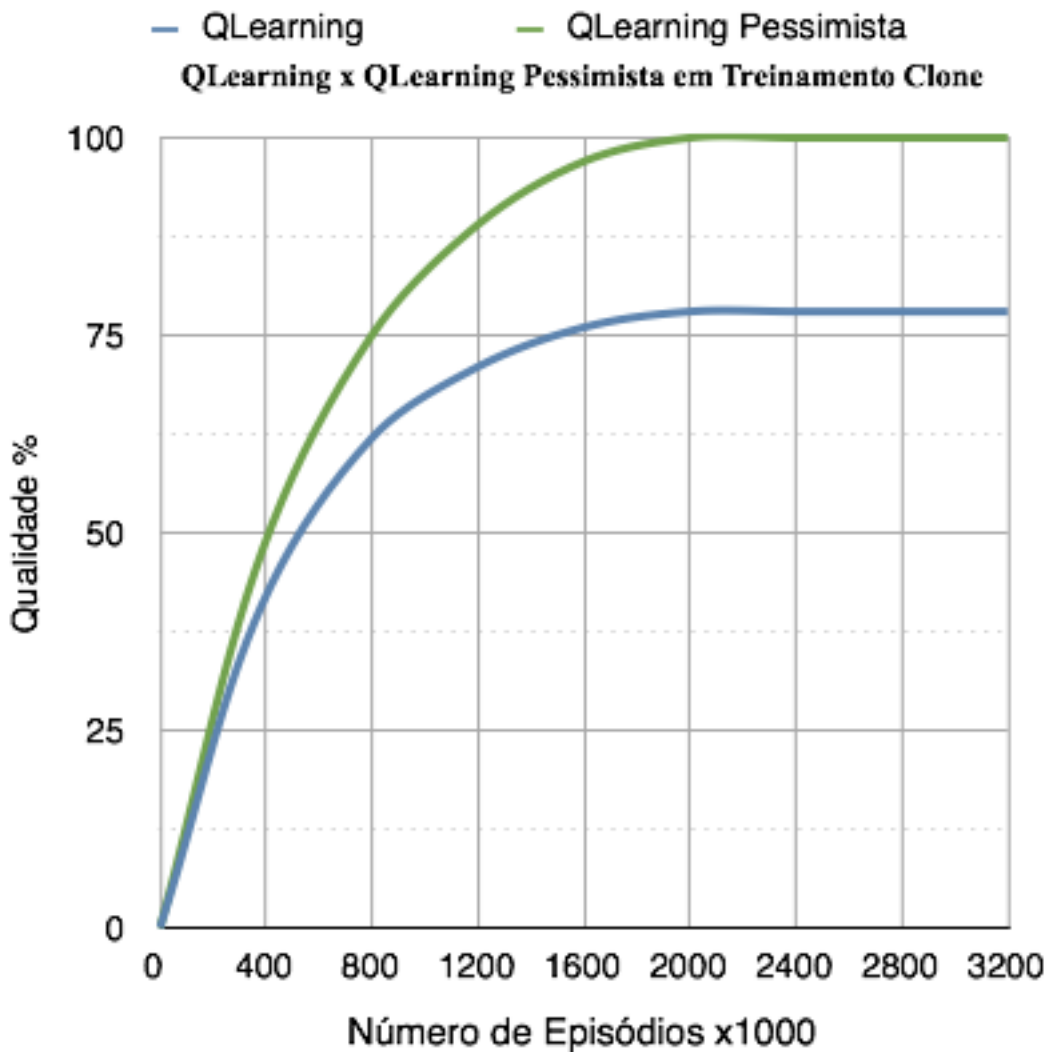
tempo de execução = 7 segundos

número de estados explorados = 8395

Algoritmo	Teste aleatório	Teste clone	Teste base	Inspeção
Q-Learning	54%	100%	99%	Forte
Q-Learning Pessimista	N/D	100%	100%	Invencível

Verificamos que o aumento da taxa de exploração não foi significativo para alterar os resultados já obtidos para o TicTacToe.

4.3. Resultados para o Connect-4



a) Teste com taxas medianas

alfa = 0.7

gama = 0.7

epsilon = 0.3

número de episódios = 4 milhões

tempo de execução = 27 horas e 44 minutos

Algoritmo	Teste aleatório	Teste clone	Teste base	Inspeção
Q-Learning	55%	77%	64%	Fraca
Q-Learning Pessimista	N/D	100%	92%	Mediana

Assim como no TicTacToe, o teste clone obteve resultados de 100% quando utilizado o Q-Learning Pessimista, o que para o Connect-4 representou um ganho de 23%.

Já o teste base, falhou para o Q-Learning em muitos dos cenários esperados, como por exemplo não iniciar no meio e falhar em movimentos simples como preparar uma linha de 3 peças sem nenhuma outra bloqueando o lado esquerdo ou direito.

O Q-Learning Pessimista, por sua vez, conseguiu aprender o movimento mais importante: Começar no meio. Esta é considerada uma heurística de vantagem básica para este jogo.

Ressaltamos que, para a quantidade de episódios utilizada, a profundidade de medição do agente foi apenas até o nível 20 de profundidade da árvore. Isso mostra que, embora o Q-Learning Pessimista proporcione ganhos interessantes em estados críticos do jogo, é indispensável a mesclagem do aprendizado por reforço com a utilização de mecanismos de busca, redes neurais ou outro.

b) Teste com baixo fator de convergência e alto valor da taxa de aprendizado

alfa = 0.9
gama = 0.1
epsilon = 0.3
número de episódios = 4 milhões
tempo de execução = 27 horas e 49 minutos

Algoritmo	Teste aleatório	Teste clone	Teste base	Inspeção
Q-Learning	55%	78%	58%	Fraca
Q-Learning Pessimista	N/D	100%	86%	Mediana

Observamos com essa amostragem que os testes de base pioram, embora o teste clone fique mais competitivo. Isso ocorre provavelmente porque o valor alto da taxa de aprendizado mantém.

Os resultados finais de inspeção não se alteram significativamente.

c) Teste com alto fator de convergência e baixo valor da taxa de aprendizado

alfa = 0.1

gama = 0.9

epsilon = 0.3

número de episódios = 4 milhões

tempo de execução = 27 horas e 45 minutos

Algoritmo	Teste aleatório	Teste clone	Teste base	Inspeção
Q-Learning	54%	75%	64%	Fraca
Q-Learning Pessimista	N/D	100%	90%	Mediana

Observamos que com um alto fator de convergência e baixo valor de aprendizado, os resultados são semelhantes aos cuja taxa é mediana como em [a].

d) Teste com médio fator de convergência, valor médio da taxa de aprendizado e baixa taxa de exploração

alfa = 0.7

gama = 0.7

epsilon = 0.01

número de episódios = 4 milhões

tempo de execução = 27 horas e 40 minutos

Algoritmo	Teste aleatório	Teste clone	Teste base	Inspeção
Q-Learning	54%	78%	74%	Fraca
Q-Learning Pessimista	N/D	100%	84%	Mediana

Embora o teste clone tenha sido alto, observamos que com uma baixa taxa de exploração, os resultados são semelhantes aos cuja taxa é mediana como em [a].

e) Teste com médio fator de convergência, médio valor de aprendizado e alta taxa de exploração

alfa = 0.7

gama = 0.7

epsilon = 0.9

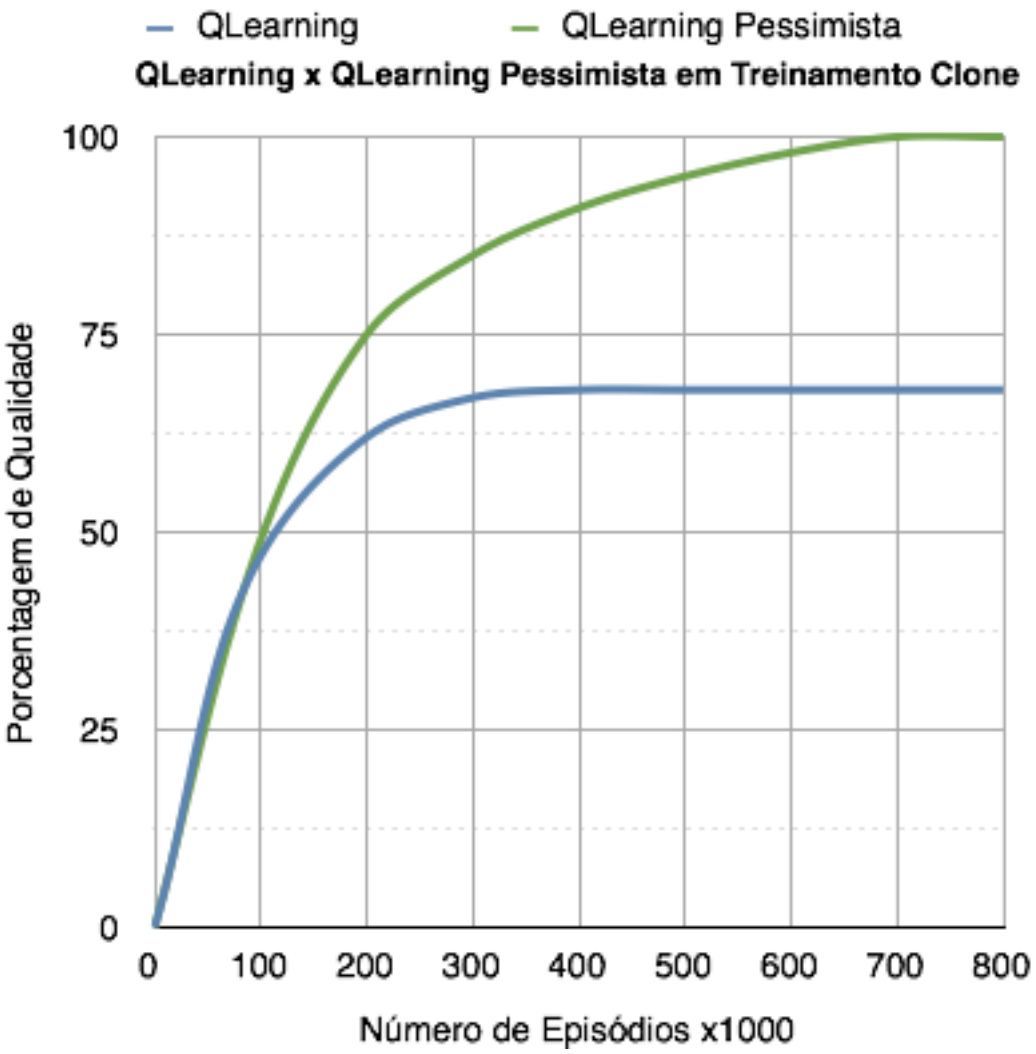
número de episódios = 4 milhões

tempo de execução = 27 horas e 33 minutos

Algoritmo	Teste aleatório	Teste clone	Teste base	Inspeção
Q-Learning	54%	63%	74%	Mediana
Q-Learning Pessimista	N/D	100%	92%	Mediana

Quando aumentada a taxa de exploração, o comportamento do teste de base tanto para o Q-Learning quanto para o Q-Learning Pessimista aumentaram. A inspeção, por sua vez, também demonstrou melhores resultados.

4.4. Resultados para o CardPoints



a) Teste com taxas medianas

alfa = 0.7

gama = 0.7

epsilon = 0.3

número de episódios = 800.000

tempo de execução = 3 horas e 12 minutos

Algoritmo	Teste aleatório	Teste clone	Teste base	Inspeção
Q-Learning	50%	63%	82%	Mediana
Q-Learning Pessimista	N/D	100%	90%	Mediana

Assim como no TicTacToe, o teste clone obteve resultados de 100% quando utilizado o Q-Learning Pessimista, o que para o CardPoints representou um ganho de 37%.

Já o teste base, falhou para o Q-Learning em muitos dos cenários esperados.

O Q-Learning Pessimista, por sua vez, conseguiu aprender um movimento importante: Escolher cartas menores para obter vantagens em cartas maiores a posteriori.

b) Teste com baixo fator de convergência e alto valor da taxa de aprendizado

alfa = 0.9
gama = 0.1
epsilon = 0.3
número de episódios = 800.000
tempo de execução = 3 horas e 57 minutos

Algoritmo	Teste aleatório	Teste clone	Teste base	Inspeção
Q-Learning	50%	63%	82%	Mediana
Q-Learning Pessimista	N/D	100%	90%	Mediana

Os resultados não se alteram significativamente.

c) Teste com alto fator de convergência e baixo valor da taxa de aprendizado

alfa = 0.1

gama = 0.9

epsilon = 0.3

número de episódios = 800.000

tempo de execução = 3 horas e 37 minutos

Algoritmo	Teste aleatório	Teste clone	Teste base	Inspeção
Q-Learning	50%	64%	85%	Mediana
Q-Learning Pessimista	N/D	100%	90%	Mediana

Os resultados não se alteram significativamente.

d) Teste com médio fator de convergência, valor médio da taxa de aprendizado e baixa taxa de exploração

alfa = 0.7

gama = 0.7

epsilon = 0.01

número de episódios = 800.000

tempo de execução = 3 horas e 12 minutos

Algoritmo	Teste aleatório	Teste clone	Teste base	Inspeção
Q-Learning	50%	80%	82%	Mediana
Q-Learning Pessimista	N/D	100%	89%	Mediana

Os resultados não se alteram significativamente.

e) Teste com médio fator de convergência, médio valor de aprendizado e alta taxa de exploração

alfa = 0.7

gama = 0.7

epsilon = 0.9

número de episódios = 800.000

tempo de execução = 3 horas e 52 minutos

Algoritmo	Teste aleatório	Teste clone	Teste base	Inspeção
Q-Learning	50%	75%	84%	Forte
Q-Learning Pessimista	N/D	100%	91%	Forte

Os resultados não se alteram significativamente.

4.5. Resultados Gerais

a) Ganhos utilizando o Teste Clone

	TicTacToe	Connect-4	CardPoints
Q-Learning	99%	78%	80%
Q-Learning Pessimista	100%	100%	100%

b) Ganhos utilizando o Teste Base

	TicTacToe	Connect-4	CardPoints
Q-Learning	99%	74%	84%
Q-Learning Pessimista	100%	92%	91%

Esses resultados, mais uma vez, mostram que a estratégia do Q-Learning Pessimista consegue aproximar positivamente a qualidade do agente, indicando o quanto essa técnica pode ser promissora quando combinada com outros algoritmos.

5. Discussões Gerais

5.1. O problema do espaço de estados para exploração

O legado do Q-Learning é que em jogos cujo espaço de estados é muito grande, por exemplo em jogos complexos como Xadrez e Go, o tempo de convergência dos q-values é muito demorado. Muitas vezes esse tempo é até mesmo inviável computacionalmente.

O estado da arte utiliza, através de tentativas de encontrar abstrações ou aceleradores de heurísticas, respostas para esse problema. A expectativa é que o Q-Learning Pessimista contribua para aperfeiçoar o conhecimento explorado, porém, ele não resolve o problema do crescimento do espaço de estados da exploração.

5.2. O problema do cacheamento na fase de treino

Durante a implementação do sistema, ficou evidente a importância da obtenção e gravação rápida dos q-values na tabela. Nenhuma das soluções que não utilizassem apenas memória RAM foram satisfatórias para ter um desempenho significativo.

Analisamos que a tabela representa o espaço de estados. Mais uma vez, como o espaço de estados cresce muito rapidamente conforme a complexidade do jogo, isso exige muita memória RAM para trabalhar com o desempenho esperado. A melhor solução para persistência em disco pareceu trabalhar diretamente com o Lucene, embora tenhamos iniciado a solução com o Membase.

Entretanto, existe uma discussão em aberto, de como trabalhar com os q-values de maneira otimizada. Existe também a possibilidade de trabalhar parcialmente com memória RAM e parcialmente em disco. Mas nenhuma solução desse tipo foi implementada.

5.3. Jogos baseados em entropia

Nosso intuito é criar uma generalização do algoritmo Q-Learning para jogos baseados em turno. Porém, existe uma classe de jogos de turno, que é altamente dependente da entropia, como sorteio ou geração de configurações iniciais. É o caso por exemplo de jogos de baralho como poker ou buraco.

5.4. A combinação com algoritmos de força-bruta

Seguindo o caminho das técnicas utilizadas pelo estado da arte, aparentemente é inevitável não utilizar algoritmos de força-bruta como substituição ao algoritmo aleatório da fase exploração da técnica.

Certamente isso aumenta a possibilidade de otimizar os resultados, e diminuir a capacidade de percorrer estados pouco interessantes. Entretanto, deixamos em aberto a discussão de como pode se comportar a proposta do algoritmo Q-Learning Pessimista com essa combinação.

5.5. A combinação com aceleradores por heurísticas

Aceleradores de heurísticas exigem conhecimento prévio das regras do jogo, ou a criação de abstrações que limitam de alguma forma a quantidade de jogos que podem ser resolvidos pela técnica.

Em geral, o que se faz é combinar heurísticas com a fase de exploração, para buscar estados ainda mais significativos de jogo.

O ponto em questão é a perca de generalização desta técnica, uma vez que para desenvolver heurísticas, é necessário ter conhecimento a priori das regras do jogo.

Não sabemos como seria a combinação do Q-Learning Pessimista com aceleradores por heurísticas, embora intuitivamente pareça ter uma contribuição relevante comparativamente ao Q-Learning.

5.6. Q-Learning Pessimista para jogos de grande porte

Uma vez que não implementamos uma solução para o problema do crescimento de espaços, não sabemos como será o comportamento do Q-Learning Pessimista para jogos de grande porte, como xadrez.

Sabemos que, para este tipo de jogos, utiliza-se uma extensa base de dados anotada por valores, da mesma forma que se faz no aprendizado supervisionado. Exceto para o jogo de gamão, [Tesauro, 1995], intencionamos seguir o mesmo caminho de implementação, utilizando ainda como otimização, o Q-Learning Pessimista.

6. Conclusões

A utilização do algoritmo Q-Learning tradicional, sob muitas iterações, sempre apresenta algum ganho de aprendizado no domínio de jogos baseados em turno. Todavia, quando se aumenta a complexidade dos jogos sem a utilização de algoritmos de força bruta, esses ganhos são poucos.

A aproximação realizada com a adição do algoritmo Q-Learning Pessimista, mostrou ser uma técnica bastante eficaz para criação de bots de alta qualidade. O Q-Learning Pessimista apresentou grande eficiência numa região muito importante da árvore de exploração do método de aprendizado: o foco em minimizar as derrotas do bot contra do adversário.

O Q-Learning Pessimista conseguiu aumentar a qualidade do jogo TicTacToe de 99% para a perfeição, resolvendo o problema da região crítica causada pelo fork do adversário. Também conseguiu aumentar a qualidade do Q-Learning para o jogo Connect-4 de 60% para 90% e do jogo de cartas CardPoints de 55% para 90% aproximadamente.

As técnicas de aprendizado por reforço possuem muitas variantes e apresentam muitos ajustes em parametrização. A criação do framework Wisebots demonstrou ter sido essencial no acompanhamento e depuração da técnica, e portanto fundamental na garantia de bons resultados para este trabalho.

Acreditamos que, por analogia experimental, possamos no futuro utilizar a técnica do Q-Learning Pessimista em jogos ainda mais complexos e não resolvidos, e comparar estes resultados com o estado da arte. Pretendemos também, uma vez já descoberta a otimização, reintroduzir o conceito de exploração por força bruta, ou seja, combinar o Q-Learning Pessimista com algoritmos de exploração em árvores e redes neurais. Esperamos com isso garantir resultados ainda mais positivos.

Como trabalhos futuros, esperamos conseguir classificar mais jogos com a implementação da técnica, gerando bots cada vez mais genéricos, ou seja, que não

dependam das características exclusivas dos jogos e heurísticas adotadas para resolvê-los.

7. Referências

[1] Van den Herik H.Jaap, Uiterwijk J.W.H.M., Van Rijswijck J.
Games solved: Now and in the future
(2002) Artificial Intelligence, 134 (1-2), pp. 277-311.

[2] WIKIPÉDIA. Desenvolvido pela Wikimedia Foundation.
Solved game
(2012) Disponível em http://en.wikipedia.org/wiki/Solved_game

[3] Tesauro, Gerry
Temporal difference learning and td-gammon
(1995) Communications of the ACM, 38(3):58-68

[4] Richard S. Sutton, Andrew G. Barto
Reinforcement Learning: An Introduction
(1998) MIT Press, Cambridge, MA, A Bradford Book

[5] Russel Stuart, Norvig Peter
Artificial Intelligence – A modern approach
(1995) New Jersey: Prentice Hall Series in Artificial Intelligence

[6] Cristopher J. C. H. Watkins, Peter Dayan
Q-Learning, Technical Note
(1992) Kluwer Adacemic Publishers, Boston, Machine Learning, 8, 279-292

[7] Csaba Szepesvári
Algorithms for Reinforcement Learning
(2009) Draft of the lecture published in the Sinthesis Lectures on Artificial Intelligence and Machine Learning series by Morgan & Claypool Publishers