Aprendizado por Reforço sobre Algoritmos em Jogos de Estratégia em Tempo Real

Autor: Marcelo Luiz Harry Diniz Lemos
Orientador: Luiz Chaimowicz
Coorientadores: Anderson Rocha Tavares, Leandro Soriano Marcolino

[1]Departamento de Ciência da Computação
Universidade Federal de Minas Gerais (UFMG) – Belo Horizonte, MG

{marcelolemos,chaimo}@dcc.ufmg.br

Resumo. *Jogos de estratégia em tempo real são bastante desafiadores para o aprendizado por reforço devido aos seus grandes espaços de estados e ações. No entanto, existem várias estratégias que foram desenvolvidas com o objetivo de selecionar a melhor ação dado um estado. Neste trabalho, vamos avaliar o desempenho de um agente que realiza seu aprendizado sobre estas estratégias colocando ele para jogar contra adversários bem consolidados de microRTS.*

1. Introdução

Aprendizado por Reforço (RL) é uma abordagem para se entender e automatizar aprendizado direcionado a objetivos e tomada de decisões [Sutton and Barto 2018]. Nesta abordagem, o agente deve aprender como se comportar, por meio de tentativa e erro, através de interações com o ambiente.

Uma das limitações atuais do aprendizado por reforço é que à medida que o conjunto de estados e ações cresce, o agente tende a ter dificuldades em manter uma boa performance. Por outro lado, existem diversos domínios que possuem algoritmos que foram desenvolvidos especificamente para solucionar o problema em mãos, e um agente poderia delegar esses algoritmos para agir em seu lugar [Rice 1976].

Uma forma de tentar minimizar o impacto desta limitação é realizar o treinamento do aprendizado por reforço sobre algoritmos, a fim de determinar qual algoritmo é capaz de obter o melhor resultado em cada estado do ambiente. Esta estratégia reduz o tamanho do espaço de ações que o agente precisa explorar, facilitando o aprendizado em problemas complexos. Mesmo assim, RL ainda pode ter dificuldades quando o conjunto de estados possíveis é muito grande, e um dos exemplos disto são os jogos de *Estratégia em Tempo Real* (RTS), que são especialmente desafiadores na área de *Inteligência Artifical* [Ontanón et al. 2013]. Este outro problema pode ser mitigado com o uso de métodos de aproximação por função, que são capazes de generalizar diferentes estados semelhantes em uma única representação, permitindo que o agente realize seu aprendizado de forma mais rápida.

Neste trabalho nós analisamos, por meio de métodos empíricos, qual o desempenho de um agente que se utiliza destas técnicas quando colocado para jogar contra inimigos do estado da arte de um jogo RTS. Além disto, também propomos algumas estratégias que utilizam de *sticky actions* e *curriculum learning* a fim de maximizar a performance deste agente.

2. Referêncial Teórico

Aprendizado por reforço é um dos paradigmas básicos de *aprendizado de máquina*. O framework de *Processos de Decisão de Markov* (MDP) é utilizado para modelar o RL e é definido como a tupla (S, A, T, R), onde S é o conjunto de estados existentes, A é o conjunto de ações possíveis para cada estado, T é uma função de transição $T(s, a, s')$ que mapeia a probabilidade de se chegar ao estado s' ao se executar a ação a a partir do estado s, e R é uma função que define o valor da recompensa que deve ser recebida ao se alcançar o estado s. O objetivo em um MDP é encontrar a política ótima, uma função $\pi(s)$ que define qual ação um agente deve executar no estado s a fim de maximizar seu desempenho. Mais referências sobre modelos, algoritmos e aplicações de RL são apresentadas em [Sutton and Barto 2018].

A questão de se realizar aprendizado por reforço sobre ações ou sobre algoritmos é abordada em [Tavares et al. 2018], que é a principal referência para este trabalho. Neste artigo, o aprendizado sobre algoritmos é explorado a fim de se estabelecer quando ele pode ser útil, quais as condições necessárias para seu correto funcionamento, e suas limitações em relação ao aprendizado sobre ações.

O uso de jogos para a realização de pesquisas em IA e seus desafios já vêm sendo explorados a bastante tempo, e podemos ver em [Bellemare et al. 2013] como o aprendizado por reforço profundo pode ser utilizado para solucionar os jogos do clássico Atari utilizando o *Arcade Learning Environment* (ALE). Um uso bem sucedido de sticky actions no ALE foi realizado em [Machado et al. 2018]. Além disto, vários trabalhos e desafios quanto uso de IA em RTS são abordados em [Ontanón et al. 2013].

Bots são amplamente desenvolvidos e utilizados para avaliar o desempenho de outros em pesquisas envolvendo jogos RTS. Alguns exemplos são os seguintes trabalhos: AHTN [Ontanón and Buro 2015] que combina *hierarchical-task network* (HTN) com um algoritmo semelhante a uma arvore de busca minimax. PuppetSearch combina comportamentos de scripts com *game-tree search*. Existem duas variações deste algoritmo: PuppetAB [Barriga et al. 2015] e PuppetMCTS [Barriga et al. 2017b]. StrategyTactics [Barriga et al. 2017a] utiliza uma rede neural convolucional para prever a saída do PuppetSearch, aumentando o tempo que um algoritmo de busca tática tem para executar. NaiveMCTS [Ontanón 2017] aplica *Monte Carlo Tree Search* mas utiliza uma estratégia de amostragem baseada em *multi-armed bandits*.

Humanos aprendem com maior facilidade quando são gradualmente apresentados a conceitos e desafios mais complexos. [Bengio et al. 2009] discute como isto também pode ser aplicado a aprendizado de máquina a fim de melhorar o desempenho dos agentes.

3. Metodologia

3.1. Ambiente

Este trabalho foi realizado no ambiente *microRTS*[1], que é um RTS simplificado, voltado para pesquisas, mas que mantem a complexidade de um RTS tradicional. Na Figura 1 podemos ver um exemplo de partida em microRTS.

[1]https://github.com/santiontanon/microrts

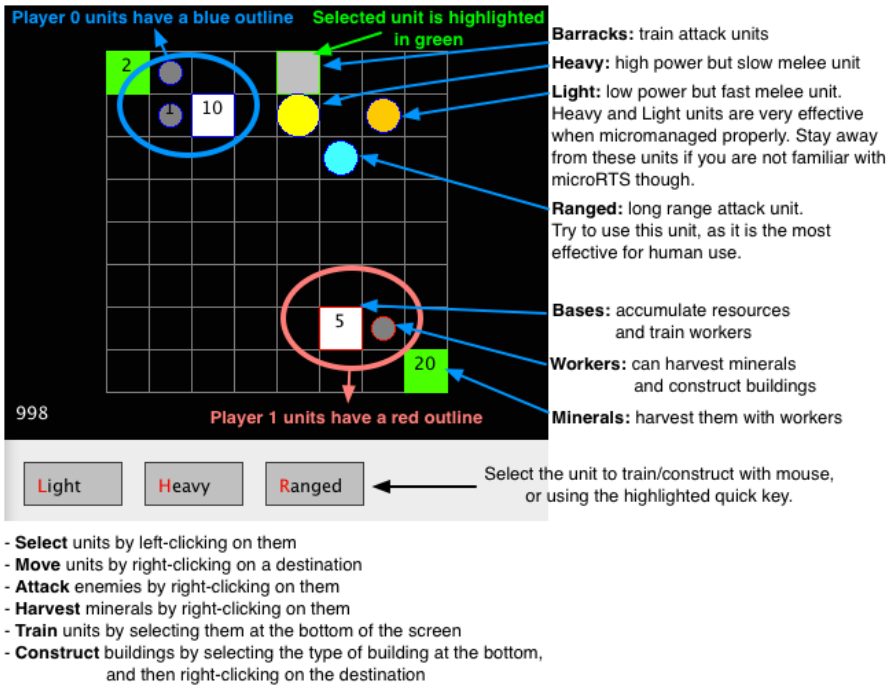

Figura 1. Partida de microRTS. Fonte: https://github.com/santiontanon/microrts

3.1.1. Cenários

Dois cenários foram utilizados durante a realização deste trabalho. O primeiro deles é um mapa 3×3, onde não existe nenhum recurso e ambos os jogadores começam com uma única unidade com 1 de vida cada. Nesta situação, o primeiro jogador a atacar o inimigo vence a partida. Apesar de ser muito simples para a realização de experimentos relevantes, este cenário foi muito utilizado para verificarmos o correto funcionamento do agente inteligente e foi essencial para detectarmos diversos problemas existentes. O estado inicial de uma partida neste mapa pode ser visto na Figura 2.

O outro cenário que foi utilizado é um mapa 24×24, onde cada jogador inicia a partida em pontos opostos do mapa, com uma base e uma unidade básica. Este cenário é bem mais complexo que o anterior, dando diversas possibilidades de ações para os jogadores e abrindo espaço para inúmeras estratégias. Este é o cenário que foi utilizado em todos os experimentos que serão descritos nas seções a seguir. A representação deste mapa pode ser vista na Figura 3.

3.2. MetaBot

O agente utilizado neste trabalho é chamado de *MetaBot*[2], e foi desenvolvido inicialmente pelo grupo de pesquisa responsável por [Tavares et al. 2018]. Nesta seção iremos descrever seu funcionamento.

[2]https://github.com/andertavares/micrortsMetaBot

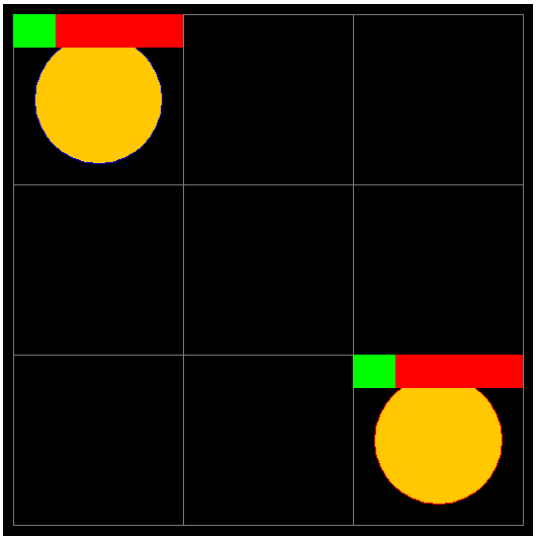

Figura 2. Mapa de microRTS utilizado para testes internos

3.2.1. Aprendizado

Para realizar seu aprendizado, o MetaBot utiliza o algoritmo SARSA (*State–action–reward–state–action*), que é um aprendizado por reforço *on-policy*, onde o agente busca otimizar suas ações levando em consideração a política utilizada para tomar suas decisões.

A grande diferença do funcionamento do aprendizado neste bot, em relação ao RL tradicional, está no conjunto de ações, que, ao invés de ser um conjunto com todas as possíveis ações que podem ser realizadas, é um conjunto com apenas 6 estratégias (scripts) predefinidas que são responsáveis por definir qual ação será executada. Estas estratégias são as seguintes:

- **Worker Rush:** Cria novas unidades *Worker* (unidade básica do *microRTS*) sempre que possível. Um deles coleta recursos enquanto o resto é enviado para combate contra os inimigos.
- **Ranged Rush:** Envia uma unidade *Worker* inicial para coletar recursos. Após acumular recursos suficientes, constrói *Barracks* (edifício que permite a criação de unidades militares), e cria unidades *Ranged* (soldados de longo alcance) sempre que possível, as enviando para combate assim que estiverem disponíveis.
- **Light Rush:** Semelhante à Ranged Rush, mas cria unidades *Light* (soldados fracos e rápidos) ao invés de *Ranged*.
- **Heavy Rush:** Semelhante à Ranged Rush, mas cria unidades *Heavy* (soldados fortes e lentos) ao invés de *Ranged*.
- **Build Barracks:** Constrói novas *Barracks*, permitindo uma produção mais rápida de unidades militares.
- **Expand:** Constrói novas Bases, permitindo uma produção mais rápida de unidades *Worker* e acelerando a coleta de recursos.

Todas estas estratégias são simples mas bastante eficazes para este problema. Um exemplo disto é a Light Rush, que é uma estratégia extremamente eficaz para o cenário 24×24,

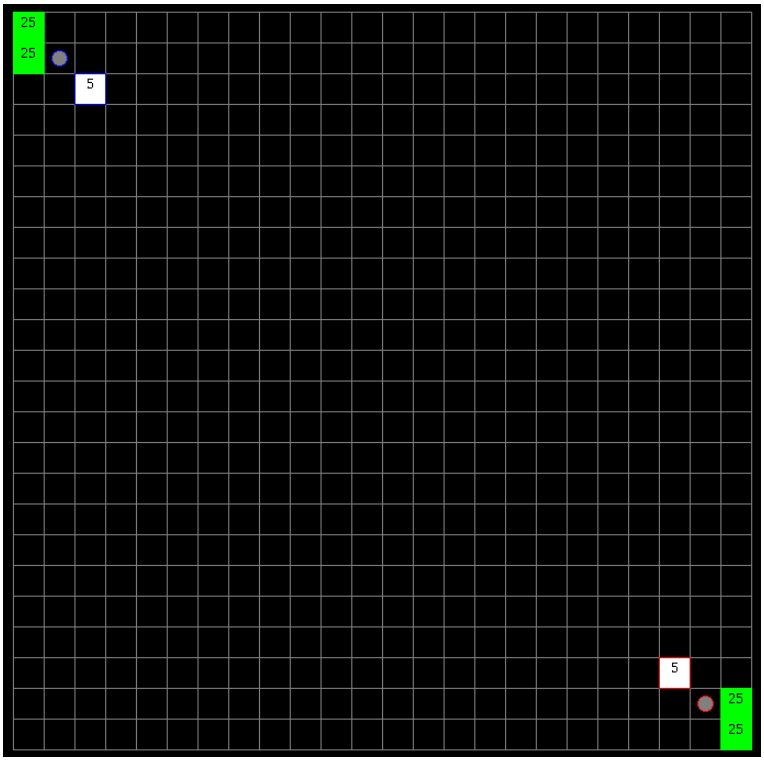

Figura 3. Mapa de microRTS utilizado na execução dos experimentos deste trabalho

sendo capaz de super diversas IAs.

Em relação às recompensas recebidas durante o aprendizado, para todos os estados não terminais o MetaBot recebe uma recompensa neutra, de valor zero. Apenas ao final da partida ele recebe uma recompensa que pode ser positiva ou negativa, dependendo de seu resultado. Desta forma, os valores dos estados terminais são propagados lentamente ao estados anteriores ao longo das partidas de treino, até chegarem aos estados iniciais.

3.2.2. Function Approximation

O MetaBot utiliza *Linear Funcion Approximation* para aproximar a representação de estados semelhantes, nos permitindo generalizar aprendizados realizados em um estado à outros. Para isto, o mapa do jogo é dividido em quadrantes e em seguida características de cada um destes quadrantes (como quantidade de unidades aliadas e inimigas no quadrante, vida total das unidades aliadas e inimigas no quadrante, etc) são utilizadas para gerar um vetor de features, que irá determinar o estado do jogo.

3.2.3. Estratégia de Exploração

Existe um dilema fundamental no Aprendizado por Reforço chamado *Exploration vs. Exploitation* onde o agente inteligente deve utilizar alguma estratégia para decidir se deve focar sua exploração nos estados que são conhecidamente bons (*Exploit*) ou se deve expandir seu conhecimento explorando outros estados, que inicialmente podem parecer pio-

res mas que podem acabar levando para soluções superiores às conhecidas até o momento (*Explore*).

Para solucionar este dilema, o MetaBot aplica uma estratégia $\varepsilon - Greedy$ com decaimento, na qual ele escolhe *explore* com probabilidade ε e *exploit* com probabilidade $1 - \varepsilon$ sempre que for tomar alguma decisão, com o valor de ε decaindo ao longo dos episódios. Como o algoritmo selecionado para a realização do aprendizado é *on-policy*, o decaimento deve ser suficiente para que o valor de ε fique próximo de zero ao final do treino, garantindo assim que o agente aprenda a se comportar em casos onde deverá escolher sempre a melhor opção disponível.

3.2.4. Adaptações implementadas

O código inicial do MetaBot utilizado para este trabalho possuia alguns problemas na parte de aprendizado, além de vários bugs. Por isto, em um primeiro momento foi necessário realizar diversos ajustes e correções para que o MetaBot apresentasse o comportamento desejado. Veremos na seção de resultados como estas mudanças foram essenciais e tiveram grande impacto na performance do agente.

Após esta fase inicial, foram implementadas adaptações que permitissem ao MetaBot utilizar uma estratégia chamada *Sticky Actions* durante seu aprendizado. Esta estratégia consiste em fixar os algoritmos selecionado pelo agente por períodos predefinidos durante a partida. Por exemplo, se a duração das sticky actions é definida como 50 quadros do jogo, cada vez que o MetaBot escolher um algoritmo para selecionar ações, este algoritmo ficará responsável pela seleção de ações pelos próximos 50 quadros e só então um novo algoritmo poderá ser escolhido pelo MetaBot. O objetivo desta estratégia é facilitar o processo de aprendizagem por meio de uma redução na quantidade de pontos de decisão. Em casos extremos, isto pode reduzir um episódio de treino a um *one-shot game*, onde o agente toma uma única decisão que precisa ser a decisão correta para que ele vença a partida. Esta versão adaptada do MetaBot pode ser encontrada em: https://github.com/Marcelo-Lemos/micrortsMetaBot

3.3. Adversários

Dois tipos de adversários foram utilizados para a avaliação do desempenho do MetaBot durante este trabalho, bots do estado da arte de microRTS, e scripts que apresentam bons desempenhos em cenários específicos.

3.3.1. Bots

Uma forma interessante de avaliar o desempenho do MetaBot é utilizando outras IAs do estado da arte de microRTS para ver como ele se sai contra elas. Para isto, selcionamos os seguintes bots:

- NaiveMCTS [Ontanón 2017]
- PuppetAB [Barriga et al. 2015]
- PuppetMCTS [Barriga et al. 2017b]
- StrategyTactics [Barriga et al. 2017a]

Vários destes oponentes alcançaram posições de destaque em competições, como o StrategyTactics que ganhou o campeonato de microRTS em 2017 e o NaiveMCTS que ficou entre os 5 melhores desta mesma disputa.

3.3.2. Scripts

Apenas dois scripts foram utilizados para avaliar o desempenho do MetaBot. O primeiro deles é a estratégia Light Rush que foi citada previamente na seção de aprendizado. Este script é a estratégia dominante para o mapa 24×24 e possui um desempenho superior ao dos Bots mencionados. O Light Rush é um oponente impiedoso que consegue punir até mesmo os menores erros do adversário, o que faz com que ele seja um ótimo oponente para avaliarmos o desempenho do MetaBot.

O segundo script, chamado $\varepsilon - LightRush$, foi desenvolvido a partir do Light Rush aplicando-se conceitos de *curriculum learning* em um contexto adversarial. A ideia por trás deste agente foi criar um oponente que não punisse demais os erros de decisão do MetaBot na fase inicial do treinamento, mas que com o passar do tempo se tornasse tão desafiador quanto o Light Rush. Este agente funciona da seguinte forma: a cada quadro do jogo ele escolhe uma ação aleatória com probabilidade ε ou toma a mesma ação que o Light Rush selecionaria com probabilidade $1 - \varepsilon$, com o valor de ε decaindo ao final de cada episódio. Deste modo, MetaBot não precisaria dicidir corretamente em todos os momentos do jogo durante os primeiros episódios de treino para ser capaz de vencer alguma partida, mas após o valor de ε se aproximar o suficente de zero, ele estaria essencialmente treinando contra o Light Rush.

3.4. Experimentos

Os experimentos realizados neste trabalho podem ser divididos em duas partes. A primeira delas, também chamada de *versus algoritmos de busca*, é uma tentativa de reproduzir parte dos experimentos feitos em [Tavares et al. 2018], enquanto a segunda parte, chamada de *aprendizado com currículo*, foi executada para avaliar as novas estratégias propostas com o objetivo de melhor o desempenho do aprendizado sobre algoritmos.

3.4.1. Versus Algoritmos de Busca

Esta primeira parte é constituída por dois conjuntos de exprimentos. No primeiro, chamado de Específico, o agente é treinado por 500 jogos contra AHTN, PuppetAB e PuppetMCTS; e por 100 jogos contra NaiveMCTS e StrategyTactics. A política resultante de cada um destes treinos é então testada, ao longo de 100 jogos, contra o mesmo adversário na qual foi treinada.

O segundo, chamado de *Nemesis*, consiste em três etapas:

1. MetaBot é treinado contra PuppetMCTS por 500 jogos e a política resultante é fixada.
2. Uma nova instância do MetaBot é treinada contra a política resultante da etapa anterior durante 500 jogos.
3. A política resultante do treinamento em 2 é testada contra cada um dos bots em 100 jogos.

3.4.2. Aprendizado com Currículo

A segunda parte dos experimentos foi executada no mesmo formato dos experimentos específicos da seção anterior, mas foram realizados com 1000 jogos e treino e apenas contra o Light Rush e o $\varepsilon - LightRush$, que são os adversários mais desafiadores para o MetaBot. Nesta parte procuramos verificar a eficácia da estratégia Sticky Actions tentando encontrar os melhores valores para sua duração.

4. Resultados

4.1. Versus Algoritmos de Busca

O desempenho do MetaBot em sua versão inicial foi muito inferior ao esperado neste experimento, o que nos fez suspeitar que existissem erros em seu código. Após investigação, percebemos que o MetaBot não estava realizando seu aprendizado da forma planejada devido a alguns *bugs*. Após a correção deste problema, refizemos este experimento e obtivemos resultados mais próximos ao esperado. Os resultados de ambas as versões serão apresentados nesta seção.

Cada um dos experimentos desta parte foram executados 5 vezes e os resultados a seguir são baseados na média obtida.

4.1.1. Específico

A taxa de vitórias do MetaBot (tanto na versão inicial quanto após os ajustes) contra cada um dos adversários selecionados pode ser vista na Figura 4.

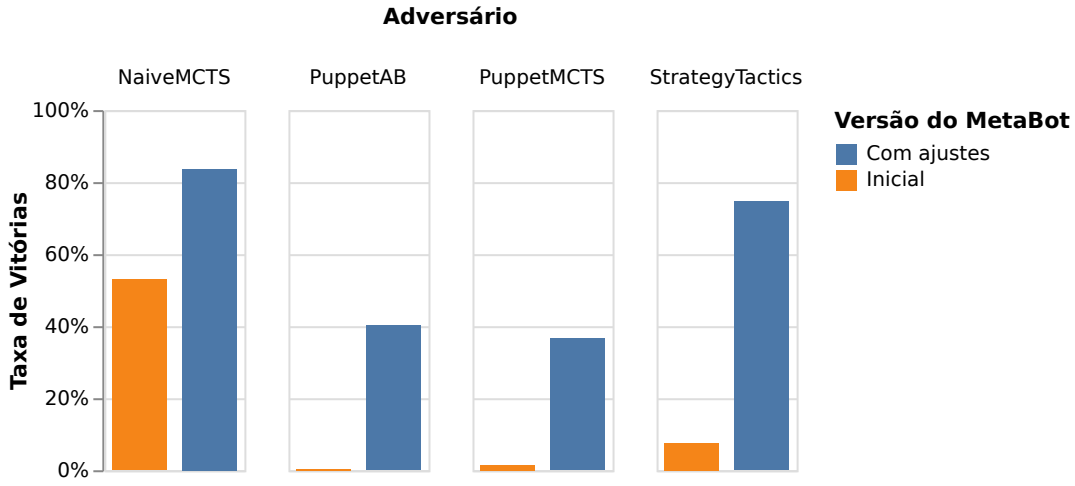

Figura 4. Taxa de vitórias do Metabot antes e após os ajustes nos experimentos Específicos

Vemos que na versão inicial o MetaBot teve dificuldade em alcançar uma taxa de vitórias de 10% contra 3 dos 4 adversários testados. No entanto, após os ajustes necessários serem implementados, ele obteve resultados muito bons, superando uma taxa de 75% de vitórias contra dois de seus oponentes, o NaiveMCTS e o StrategyTactics, que

são oponentes mais fáceis. Contudo, ele ainda apresenta dificuldades em obter resultados satisfatórios (taxa de vitórias acima de 50%) contra adversários mais desafiadores como PuppetAB e PuppetMCTS.

4.1.2. Nemesis

Os resultados nos experimentos Nemesis foram bastante semelhantes aos específicos, tanto para a versão inicial do MetaBot quanto para a versão após os ajustes. A taxa de vitórias do MetaBot contra cada um dos adversários nos experimentos Nemesis pode ser vista na Figura 5.

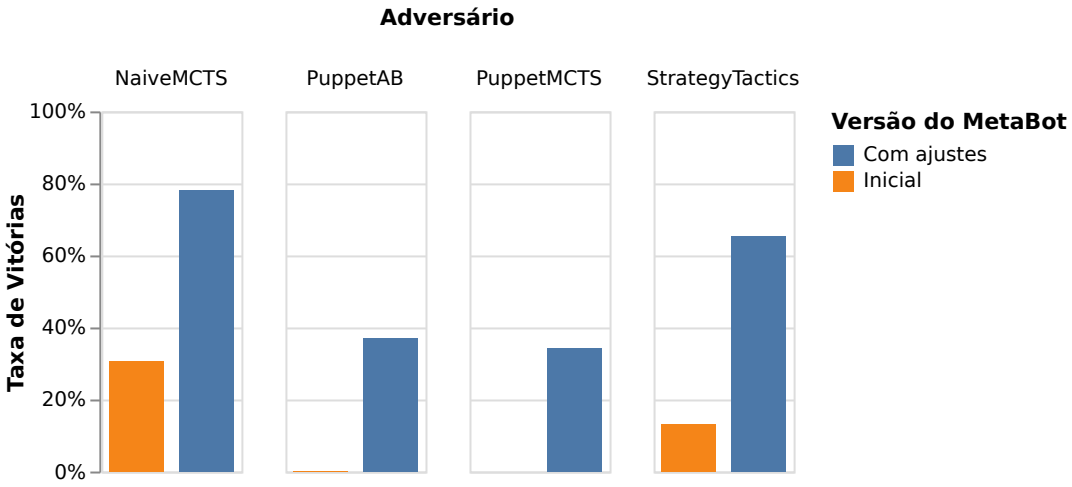

Figura 5. Taxa de vitórias do Metabot antes e após os ajustes nos experimentos Nemesis

4.1.3. Representação dos Estados

Durante a investigação dos problemas existentes com o MetaBot durante esta etapa, notamos algumas coisas importantes quanto ao funcionamento do agente que poderiam causar impactos negativos. A primeira delas é quanto ao vetor de features utilizadas para caracterizar os estados. No mapa 24×24 são geradas 130 features, mas apenas pouco mais de 20 features sofrem alguma alteração ao longo das partidas, como podemos ver na Figura 6. A grande maioria das features permanecem com o valor zero durante toda a duração dos jogos. Desta forma, todos os estados são percebidos pelo MetaBot como se tivessem inúmeras similaridades, fazendo com que ele generalize demais seus aprendizados, tornando-os ineficazes em alguns casos. Isto é um forte indício que esta representação não está muito adequada para este problema e poderia ser reprojetada a fim de melhorar o desempenho do agente.

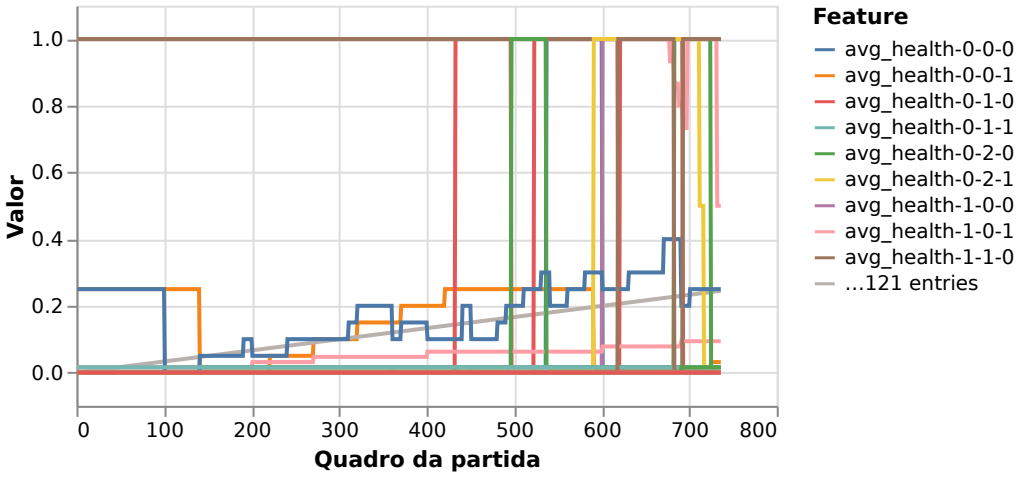

Figura 6. Evolução das features ao longo de uma partida de microRTS

4.1.4. Fator de Desconto γ

Outra questão importante que notamos foi quanto ao fator γ, que inicialmente estava sendo utilizado com o valor 0.9, indicando que recompensas recebidas no futuro possuem uma valorização de 90% quando comparadas à recompensa atual. No entanto, em alguns casos, isto estava causando ciclos durante o aprendizado do MetaBot, uma vez que atualizações que deveriam ser negativas para alguns *Q-values* eram reduzidas devido a este fator e acabavam gerando tendências positivas, fazendo com que o MetaBot ficasse preso em decisões ruins. Para exemplificar como isto pode ocorrer, vamos partir da função de atualização do SARSA, dada por:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

Se estivermos em uma situação onde $Q(s_t, a_t) = -0.9$ e $Q(s_{t+1}, a_{t+1}) = -1$, o valor de $Q(s_t, a_t)$ deveria ser atualizado em direção a -1, mas caso α seja 0.1, r_{t+1} seja 0 e γ seja 0.8, o valor de $Q(s_t, a_t)$ será atualizado no sentido contrário, como podemos ver a seguir.

$$Q(s_t, a_t) \leftarrow -0.9 + 0.1[0 + (0.8)(-1) + 0.9]$$

$$Q(s_t, a_t) \leftarrow -0.89$$

Este problema não ocorre quando $\gamma = 1$, portanto solucionamos este problema fixando o valor de γ em 1.

4.2. Aprendizado com Currículo

Esta segunda parte dos experimentos foi motivada devido aos resultados inferiores aos desejados na parte anterior. Aqui, nás verificamos o impacto da estratégia Sticky Actions no aprendizado do MetaBot contra o Light Rush e contra o $\varepsilon - LightRush$.

A duração destas partidas poder variar de 500 a 3000 quadros, mas a maior parte dos jogos possuem uma duração entre 1000 e 2000 quadros. A partir da duração das partidas e da duração das sticky actions podemos analisar melhor os resultados, levando em consideração a quantidade de decisões que são tomadas pelo nosso agente.

Cada um dos experimentos desta parte foram executados 10 vezes e os resultados abaixo são baseados na média obtida.

4.2.1. Light Rush

A taxa de vitórias do MetaBot contra o Light Rush com diferentes durações de sticky actions pode ser vista na Figura 7. Os resultados do MetaBot contra o Light Rush foram muito bons quando a duração das sticky actions é de 1000 quadros ou superior, mas para valores abaixo disto há uma queda drástica na taxa de vitorias, que provavelmente se dá por conta do aumento na quantidade de pontos de decisão durante uma partida.

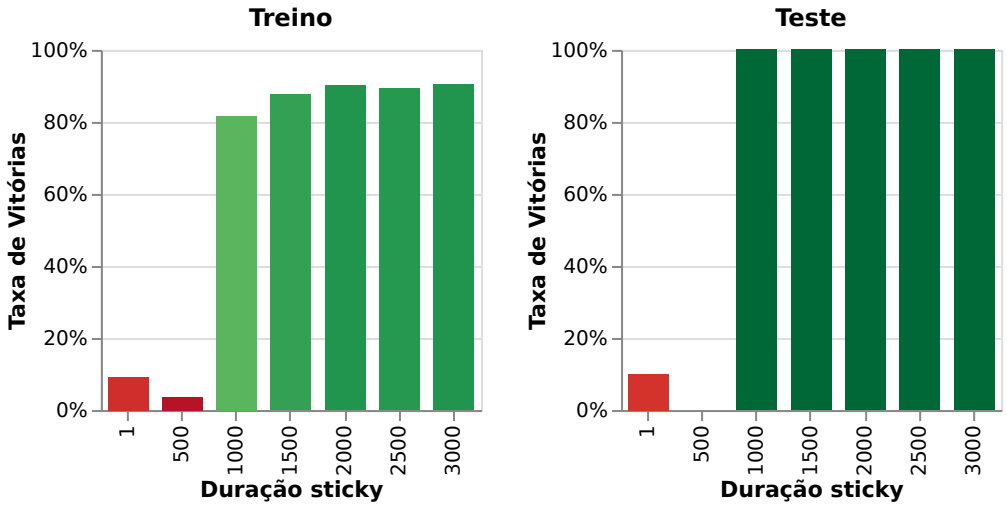

Figura 7. Taxa de vitórias do Metabot durante o treino e nos testes contra o Light Rush com diferentes valores de sticky actions

Olhando para a taxa de aprendizado do nosso agente na Figura 8, é possível perceber que a partir de 1000 quadros de duração o aprendizado com sticky actions funciona exatamente como esperado, alcançando taxas de vitória muito próximas de 100% após cerca de 700 jogos. Enquanto isso, os testes com durações menores apresentam aprendizados que parecem estar estagnados, com baixas taxas de vitórias.

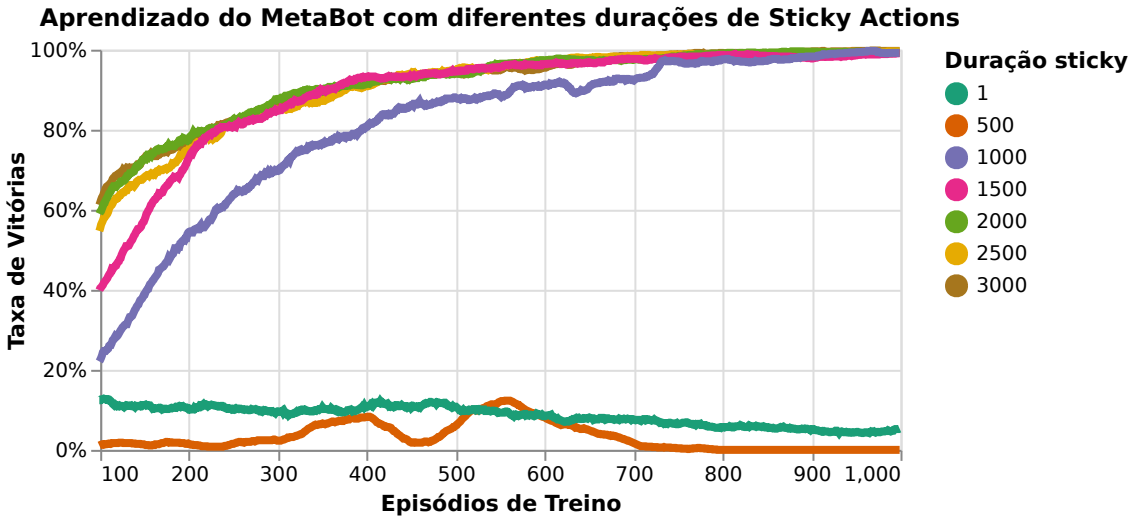

Figura 8. Curva de aprendizado do Metabot contra o Light Rush para diferentes valores de sticky actions

4.2.2. Epsilon Light Rush

Para os experimentos contra o $\varepsilon - LightRush$, esperávamos obter uma melhoria de desempenho para sticky actions com durações menores, mas os resultados alcançados foram bem diferentes disto, como podemos ver na Figura 9. As taxas de vitórias foram reduzidas tanto com duração 1 quanto com duração 1000 nos testes.

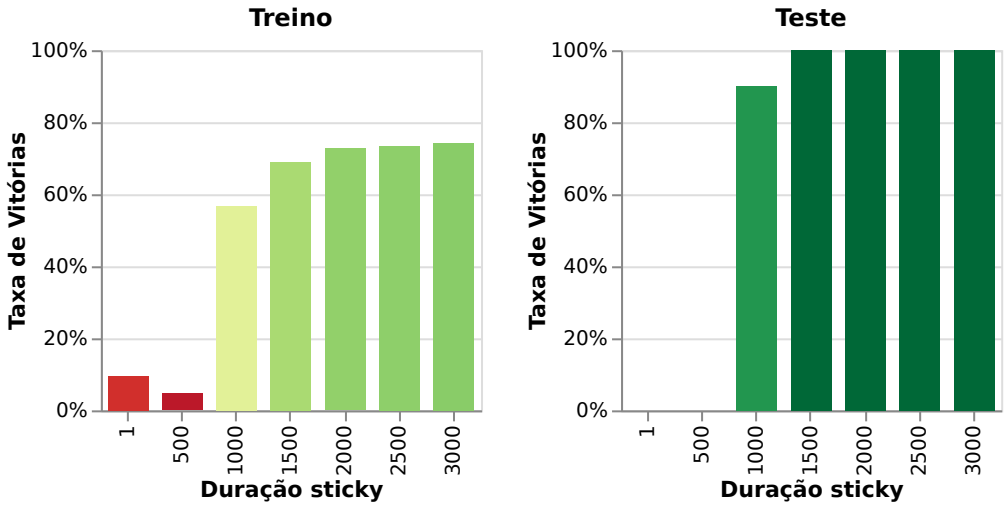

Figura 9. Taxa de vitórias do Metabot durante o treino e nos testes contra o Epsilon Light Rush com diferentes valores de sticky actions

Além disso, o MetaBot parece ter tido maior dificuldade durante o aprendizado, como podemos perceber pela redução nas taxas de vitórias durante o treino para valores maiores de sticky actions e pela curva de aprendizado mostrada na Figura 10. Em todos

estes testes a taxa de vitórias do MetaBot apresentou um queda significativa nos primeiros 300 episódios de treino. Isto é um indício que o $\varepsilon - LightRush$ está se tornando desafiador em um ritmo rápido demais para nosso agente, que esta tendo dificuldades em acompanhá-lo. Pode ser necessário um decaimento mais lento de ε e uma quantidade muito maior de episódios de treino para que o MetaBot tenha mais tempo para se adaptar às mudanças graduais do $\varepsilon - LightRush$.

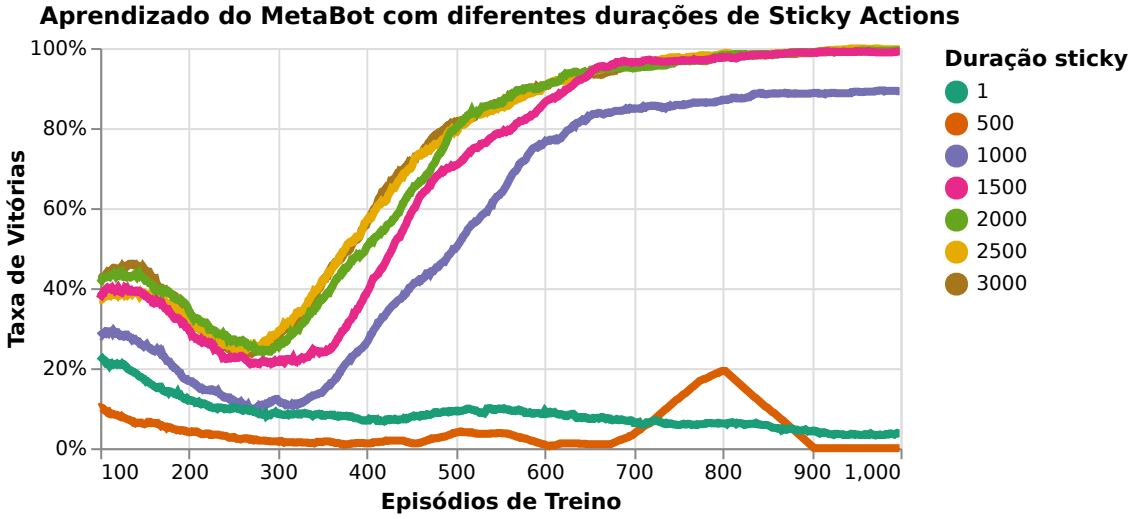

Figura 10. Curva de aprendizado do Metabot contra o Epsilon Light Rush para diferentes valores de sticky actions

5. Conslusão

O aprendizado por reforço sobre algoritmos se mostrou uma possibilidade interessante no contexto de jogos RTS. Conseguimos alcançar taxas de vitórias excepcionalmente boas com o uso de sticky actions, mas ele ainda apresenta limitações em alguns pontos.

A representação dos estados não está muito adequada ao problema. Poderia ser utilizada uma *Deep Q Network* (DQN) para se melhorara representação e, consequentemente, o desempenho do agente.

Outra coisa interessante a ser feita é executar estes mesmos experimentos com o aprendizado sobre ações, o que nos permitirá realizar uma comparação direta do desempenho das duas abordagens. Também poderiam ser realizados experimentos onde o aprendizado sobre algoritmos é colocado para jogar contra o aprendizado sobre ações para verificarmos se algum deles possui vantagens sobre o outro neste cenário.

Referências

[Barriga et al. 2015] Barriga, N. A., Stanescu, M., and Buro, M. (2015). Puppet search: Enhancing scripted behavior by look-ahead search with applications to real-time strategy games. In *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*.

[Barriga et al. 2017a] Barriga, N. A., Stanescu, M., and Buro, M. (2017a). Combining strategic learning with tactical search in real-time strategy games. In *Thirteenth Artificial Intelligence and Interactive Digital Entertainment Conference*.

[Barriga et al. 2017b] Barriga, N. A., Stanescu, M., and Buro, M. (2017b). Game tree search based on nondeterministic action scripts in real-time strategy games. *IEEE Transactions on Games*, 10(1):69–77.

[Bellemare et al. 2013] Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279.

[Bengio et al. 2009] Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48.

[Machado et al. 2018] Machado, M. C., Bellemare, M. G., Talvitie, E., Veness, J., Hausknecht, M., and Bowling, M. (2018). Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562.

[Ontanón 2017] Ontanón, S. (2017). Combinatorial multi-armed bandits for real-time strategy games. *Journal of Artificial Intelligence Research*, 58:665–702.

[Ontanón and Buro 2015] Ontanón, S. and Buro, M. (2015). Adversarial hierarchical-task network planning for complex real-time games. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

[Ontanón et al. 2013] Ontanón, S., Synnaeve, G., Uriarte, A., Richoux, F., Churchill, D., and Preuss, M. (2013). A survey of real-time strategy game ai research and competition in starcraft. *IEEE Transactions on Computational Intelligence and AI in games*, 5(4):293–311.

[Rice 1976] Rice, J. R. (1976). The algorithm selection problem. In *Advances in computers*, volume 15, pages 65–118. Elsevier.

[Sutton and Barto 2018] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

[Tavares et al. 2018] Tavares, A. R., Anbalagan, S., Marcolino, L. S., and Chaimowicz, L. (2018). Algorithms or actions? a study in large-scale reinforcement learning. In *IJCAI*, pages 2717–2723.