Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Tiago Negrisoli de Oliveira

MONOGRAFIA DE PROJETO ORIENTADO EM COMPUTAÇÃO I

Coordenação multiagente em jogos de estratégia em tempo real

Belo Horizonte
2019/ 2º semestre
Universidade Federal de Minas Gerais

Instituto de Ciências Exatas
Departamento de Ciência da Computação
Curso de Bacharelado em Ciência da Computação

Coordenação multiagente em jogos de estratégia em tempo real

por

Tiago Negrisoli de Oliveira

Monografia Projeto Orientado em Computação I

Apresentado como requisito da disciplina de Projeto Orientado em Computação I do Curso de Bacharelado em Ciência da Computação da UFMG

Orientador Prof. Dr. Luiz Chaimowicz
Co-orientador Anderson Rocha Tavares

Belo Horizonte
2019 / 2º semestre

Aos familiares,
aos amigos,
aos professores,
aos colegas de trabalho,
dedico este trabalho.

AGRADECIMENTOS

Inicialmente quero agradeço aos meus familiares, pelo amor e suporte.

Agradeço aos amigos, pelo companheirismo.

E finalmente aos professores, pelos conhecimentos e oportunidades.

RESUMO

Os jogos RTS (Real-Time Strategy games) possuem um ambiente dinâmico, informações parciais do oponente e do mapa. Além disso, requerem dos jogadores gerenciamento da economia e recursos, criação e movimentação de unidades, construção de edifícios, aprimoramento de tecnologias e controle de embates contra oponentes. Todas essas características geram dificuldades nas aplicações de técnicas de inteligência artificial, tais como o grande número de estados, tempo de raciocínio limitado, recompensas em longo prazo e conjunto de ações combinatórias. Para isso, foi criado um modelo baseado em aprendizado por reforço, que primeiramente considera a escolha de comportamentos em alto nível em vez de ações básicas e que trata cada unidade como um agente independente. Dessa forma, as unidades escolherão scripts para definir seu próprio comportamento e aprendem as consequências de suas escolhas independentemente das outras. Além do mais, apresentamos uma função de aproximação linear para generalizar o aprendizado para estados similares e uma forma de agregação de conhecimento para acelerar o aprendizado. A coordenação ocorre de forma implícita e devido a recompensas baseadas no desempenho da equipe.

Palavras-chave: *Jogos de estratégia em tempo real, Inteligência artificial, Aprendizado por Reforço.*

ABSTRACT

RTS (Real-Time Strategy games) are dynamic environment with partial information of opponents and the map. Also, requires from the player resource and economic management, unit movement and creation, building, technologic upgrades and controls in combat scenarios. All of these characteristics create difficulties in applying artificial intelligence techniques, such as large number of states, limited thinking time, long-delayed rewards and combinatorial joint-action spaces. To deal with this environment, we create a reinforcement learning model, first by considering the assignment of high-level behavior instead of basic actions, and every unit is considered an independent learner. With that, units choose scripts to dictate its own behavior and learn the consequences of its choices independently from other units. In addition, we present a linear function approximation to generalize learned values across similar states and knowledge aggregation scheme to accelerate learning.

Keywords*:* *Real-Time Strategy games, Artificial Intelligence, Reinforcement learning.*

LISTA DE FIGURAS

Figura 1 Esquema do modelo criado Página 14
Figura 2 Equação de agregação de conhecimento Página 16
Figura 3 Gráfico de taxa de vitória Dragoon Zealot Página 18
Figura 4 Gráfico de taxa de vitória Zergling Marine Página 19

LISTA DE SIGLAS

RTS	Real-Time strategy game
RL	Reinforcement Learning
POE	Portfolio Online Evolution
PGS	Portfolio Greedy Search
SSS	Stratified Policy Search
GAB	Greedy Alpha-Beta Search
SAB	Stratified Alpha-Beta Search

SUMÁRIO

RESUMO ..... **II**
ABSTRACT ..... **II**
LISTA DE FIGURAS ..... **II**
LISTA DE SIGLAS ..... **II**
1 INTRODUÇÃO **12**
2 CONTEXTUALIZAÇÃO E TRABALHOS RELACIONADOS **13**
3 DESENVOLVIMENTO DO TRABALHO **14**
4 RESULTADOS E DISCUSSÃO **17**
5 CONCLUSÕES **20**
6 REFERÊNCIAS **21**

1 INTRODUÇÃO

Jogos de estratégia em tempo real são ambientes dinâmicos que envolvem raciocínio em longo prazo sobre aspectos estratégicos, por exemplos economia, assim como em curto prazo em situações de combates entre unidades. Essas batalhas são pontos vitais para o sucesso nesse tipo de jogo e se tornou um tópico de estudo na área de inteligência artificial. Varias formas de se lidar com esse tipo de ambiente foram propostas. Muitas delas envolvem comportamentos fixos, scripts que atribuem a cada unidade ações seguindo uma regra pré-definida. Um problema dessa abordagem é que oponentes podem tirar vantagem da invariância do comportamento, e explorar alguma fraqueza. Assim, duas abordagens se destacaram para lidar com tais situações, busca e aprendizado. As formas baseada em busca, verificam vários estados do jogo e através de uma medida de desempenho dos estados, escolhem as ações que levam ao melhor estado possível. Embora essa abordagem consiga ter resultados muito bons em geral, dependem de uma modelagem do oponente, da modelagem do estado e podem levar muito tempo de processamento para conseguir levar a ações boas.

Dessa forma, esse trabalho propõe um modelo baseado em aprendizado por reforço. Nesse, foram utilizadas quatro ideias chaves para a sua implementação. (i) Nos simplificamos o espaço de estados identificando características que o descrevem e utilizamos uma função de aproximação linear para estimar valores de estados semelhantes. (ii) Cada unidade é tratada como um agente independente, assim, elas irão interagir com o ambiente e irão aprender com isso, considerando as outras unidades como parte do ambiente, sem um modelo explicito. (iii) As unidades irão escolher entre scripts para determinar seu comportamento, em vez de ações de baixo nível, para reduzir o espaço de ações. (iv) Um esquema de agregação de conhecimento é utilizado para acelerar o aprendizado das unidades. Com essas ideias, foi possível lidar com combates em jogos RTS com maior escala do que outras abordagens de aprendizado apresentadas.

A implementação da nossa abordagem foi feita em um simulador de combates para StarCraft, chamada de SparCraft. Isso permitiu a flexibilidade de escolher diferentes composições de exércitos, assim como posições das tropas e outras características para simular cenários reais de combate em jogos desse tipo. Além disso, permitiu que nosso modelo fosse testado contra oponentes baseados em busca, alguns estado da arte nessa categoria.

2 CONTEXTUALIZAÇÃO E TRABALHOS RELACIONADOS

Nossa abordagem é baseado em aprendizado por reforço, que, dado um ambiente definido por estados, é um modelo no qual um agente observa esse ambiente na forma de percepções e interage por meio de ações. Dessa forma, o agente em um determinado estado *s* recebe informações e escolhe uma ação *a* possível em *s*. Assim, essa ação muda o estado *s* para *s'* diferente e o agente recebe uma recompensa *r* correspondente a essa mudança.

Para a implementação do nosso modelo, foi utilizado uma variação do Q-Learning. Este é um método de aprendizado por reforço que procura aprender uma política ótima. Ele se baseia em na função *Q*, que mapeia estados e ações em um valor. Esse valor é utilizado na tomada de decisões. Esse valor Q é atualizado conforme a equação a seguir:

$$Q(s, a) \leftarrow (1 - \lambda)\, Q(s, a) + \lambda[\, R(s,a) + \gamma \max_a Q(s', a)]$$

Em que λ é a taxa de aprendizado, e γ é o fator de desconto, ambos são valores entre 0 e 1.

A variação utilizada na implementação é o Sarsa, que em vez de considerar o atual estado s^i, e o próximo estado s^{i+1}, ao executar *a* no tempo *i*, ele vai considerar o estado s^i e o s^{i-1}. O Sarsa foi utilizado por apresentar melhor desempenho quando aplicado a função de aproximação linear proposta nesse trabalho.

Outro mecânismo utilizado no aprendizado é a ideia do ξ-greedy. Esse é uma forma de conciliar entre executar a ação que é considerada a melhor até o momento ou tentar novas ações ainda não exploradas, com o objetivo de tentar descobrir novas melhores ações. Assim, para cada ação que a unidade for tomar, ela escolhera a melhor ação com a probabilidade (1-ξ) e com uma probabilidade ξ escolherá uma ação aleatória. Sendo que ξ é um valor entre 0 e 1.

Sistemas multiagentes são ambientes em que existe mais de um agente interagindo, e que podem cooperar com aliados e competir com oponentes. Para se lidar com esses sistemas na abordagem de aprendizado, são destacados dois tipos: Agentes independentes e Coordenação. Agentes independentes é a aplicação direta dos algorítmos de aprendizado para ambientes com um único agente. Nessa forma, o agente considera que outros são parte do ambiente, sem uma modelagem explícita deles. A cooperação é dada por recompensas e objetivos em comum. É provado que mesmo sem a modelagem de outros, a coordenação ocorre, porém pode levar um tempo longo até o conjunto de ações convergir para o ótimo. Assim, coordenação é a modelagem de outros agentes, levando em consideração suas possíveis ações durante o processo de aprendizado. Algumas abordagens modelam isso na tomada de decisão, enquanto outras lidam com esse aspecto na atualização do conhecimento.

A coordenação entre agentes não foi implementada nesse trabalho, pois nessa primeira parte o foco foi experimentos no modelo com agentes independentes. Assim, ter uma base de comparação quando for feita a implementação da coordenação explícita. Outros trabalhos foram propostos para coordenação de agentes em sistemas multiagentes tais como Friend-or-Foe Q-Learning [Littman – 2001], Joint-Action Learning [Claus and Boutilier – 1998] e Correlated Q-Learning [Greenwald and Hall – 2003].

3 DESENVOLVIMENTO DO TRABALHO

O desenvolvimento do trabalho foi dividido em três partes. Essas são: implementação do modelo sem coordenação, teste e experimentos, implementação de métodos de coordenação. Neste vão ser discutidos os dois primeiros passos, e todas as decisões tomadas. O modelo pode ser visualizado pela seguinte figura:

3.1

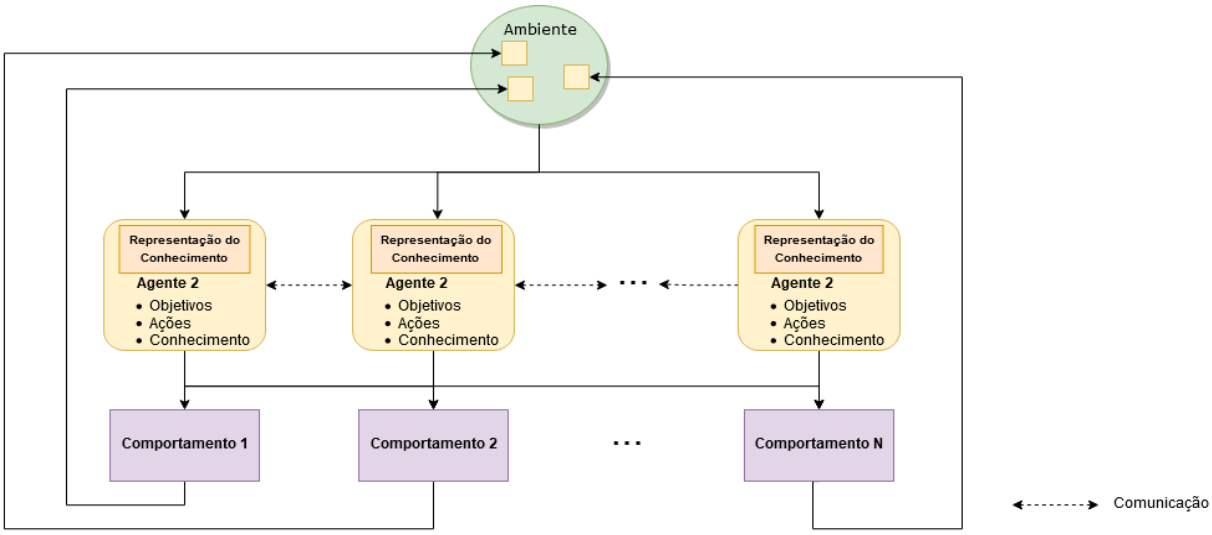

Figura 1: Esquema do modelo criado

SIMPLIFICANDO O ESPAÇO DE ESTADOS

Jogos RTS possuem grande espaço de estados e ações. Estimativas indicam próximos de 10^{1685} estados e 10^{50} ações por estado. Dessa forma, é inviável armazenar os valores Q(s,a) na sua forma tabular. Para lidar com tal complexidade, foi utilizado uma função de aproximação, que tem como objetivo generalizar valores aprendidos de um conjunto estado-ação em outros similares. O estado foi representado por *n* características que o descreve, organizada em um vetor *f*. Essas características variam de acordo com o estado atual. Assim, foram armazenados um vetor W com *n* pesos para cara ação $a \in A$. A função de estado-ação Q(s, a) é então aproximada da forma $Q^a(s, a) = f(s) \cdot w^a = \Sigma^n f_i(s) \cdot w_i{}^a$. A generalização se da pelo fato de que estados similares tendem a ter os valores das características de f similares, assim essa aproximação de Q(s, a) tende a ter valores similares para a mesma ação em diferentes estados. Assim, o aprendizado do agente se dá pela atualização dos pesos W a cada iteração com o ambiente. Quando um agente executa a ação *a* no estado *s* , recebendo como recompensa *r* e chegando ao estado *s'*, a atualização se da pela seguinte equação:

$$w_i^a \leftarrow w_i^a + \lambda \, [\, r + \gamma \, max_a \, Q^a(s', a') - Q^a(s, a)] \, f_i(s)$$

As características escolhidas foram:

- Porcentagem de vida restante;
- Tempo até o próximo ataque;
- Número de unidades inimigas no alcance;
- Número de unidades inimigas que podem atacar a unidade;

- Número de unidades aliadas no alcance;
- Dano causado pela unidade;
- Alcance total da unidade;
- Porcentagem de vida da unidade inimiga mais próxima;

3.2 SIMPLIFICANDO O ESPAÇO DE ESTADOS

Descrevendo o estado por um vetor de características simplifica muito o espaço de estados, pois geralmente $n < |S|$. Entretanto, o número de ações ainda é muito grande, e nós armazenamos um vetor de pesos para cada ação. O número de ações é muito grande, pois cada jogador controla muitas unidades, e cada uma delas escolherá uma ação. Assim, o número de combinações de possíveis ações é alto. Dessa forma, primeiro consideramos cada unidade como um agente independente. Com isso, cada agente terá que aprender apenas os seus próprios valores Q(s, a).

Mesmo com essa consideração, o número de ações continua alto, pelo fato de que cada unidade tem um número de ações, e algumas delas recebem como parâmetro posição, po exemplo. Dessa forma, o número possíveis de ações para uma única unidade é grande. Assim, cada unidade em vez de considerar todas as possíveis ações e seus parâmetros, ela terá apenas um conjunto limitado de scripts. Esses scripts irão definir o comportamento da unidade, e eles lidam com as ações de níveis mais baixos.

Os scripts escolhidos são:

- AlphaBeta: realiza uma busca alpha-beta com profundidade baixa e retorna a melhor ação a ser realizada;
- AttackClosest: ataca a unidade mais próxima;
- AttackValue: ataca a unidade com maior valor (medido por dano-por-segundo);
- AttackWeakest: ataca a unidade com menor porcentagem de vida;
- No-Overkill AttackValue (NOKAV): mesmo que o AttackValue, mas sem causar dano extra.
- Kiter: ataca a unidade mais próxima, recuando das unidades inimigas entre ataques.
- Kiter AttackValue: kiter, mas ataca a unidade de maior valor;
- Kiter NOKAV: NOKAV com a realização do Kiter.

3.3 ACELERANDO O APRENDIZADO

O último aspecto do modelo é a agregação de conhecimento. A ideia é que cada unidade está explorando diferentes partes do conjunto de estado-ações possíveis. Assim, com o compartilhamento do conhecimento adquirido, as unidades aprenderão mais rápido, pois elas irão se beneficiar da experiência das outras. No nosso modelo, ao final de cada episódio (partida), os pesos agregados (ou globais) $\hat{w}_i^a$, da característica *i* e do script *a* é atualizada da seguinte forma:

$$\bar{w}_i^a \leftarrow \frac{\sum_{u \in U} {}^u w_i^a \cdot t_u(a)}{\sum_{u \in U} t_u(a)}$$

Figura 2: Equação de agregação de conhecimento

O esquema de agregação é ponderado pela quantidade de vezes que a unidade escolher o script *a*, e normalizado pela soma total de vezes que o script *a* foi selecionado. Ao final da partida, a agregação é realizada e distribuída entre as unidades.

4 RESULTADOS E DISCUSSÃO

Foram realizados inúmeros experimentos nessa primeira etapa do trabalho. Focando entender melhor o funcionamento do modelo sem coordenação.

O primeiro teste para validação da nossa abordagem foi o teste contra oponentes de comportamento fixo. Por serem oponentes com invariância, o nosso modelo deveria conseguir aprender as ações para ganhar dele. Vários composições de exércitos e números de unidades foram testados. Os números foram {8, 32, 96}, e as composições para *n* unidades foram:

- *n/2 Protoss Dragoon e n/2 Protoss Zealot: Ambas são unidades fortes. Dragoons são unidades de ataque a distância, e os Zealots são unidades de ataques pertos;*
- *n/2 Dragoon Protoss e n/2 Terran Marine: Ambas são unidades de ataques a distância. Protoss são unidades fortes e Marines são unidades fracas;*
- *n/2 Zerg Zergling e n/2 Terran Marine: Ambas são unidades fracas. Zergling são unidades com ataques pertos, e Marines são unidades de ataque a distância.*

Os experimentos contra oponente de comportamento fixo levou em consideração o script AttackClosest, que para todas as unidades, para todos os momentos do jogo, elas irão atacar a unidade inimiga mais próxima.

Partidas de Treino	Unidades	Número de Unidades	Taxa de vitória no Teste
1,00E+06	Zergling Marine	8	95,40%
1,00E+06	Zergling Marine	32	75,05%
1,00E+06	Zergling Marine	96	69,70%
1,00E+06	Dragoon Marine	8	99,40%
1,00E+06	Dragoon Marine	32	97,70%
1,00E+06	Dragoon Marine	96	94,85%
1,00E+06	Dragoon Zealot	8	99,80%
1,00E+06	Dragoon Zealot	32	99,20%
1,00E+06	Dragoon Zealot	96	99,35%

Dessa forma, os experimentos mostraram que nossa abordagem consegue ganhar desse tipo de oponente de forma concisa e em todos os cenários apresentados. Uma observação sobre o cenário Zergling e Marine, que não apenas nesse experimento como em vários outros se mostrou o pior caso. Em que o nosso modelo apresenta dificuldade em aprender.

O próximo experimento foi em relação a taxa de aprendizado. Em muitos outras aplicações que utilizam a função de aproximação, mostram que a taxa de aprendizado menor melhoram o aprendizado. Assim, testamos contra o mesmo oponente de comportamento fixo, no nosso pior caso, e variando a taxa de aprendizado.

Partidas de Treino	Unidades	Número de Unidades	Taxa de Aprendizado	Taxa de Vitória no Teste
1,00E+06	Zergling Marine	32	0,1	32,30%
1,00E+06	Zergling Marine	32	0,01	62,30%
1,00E+06	Zergling Marine	32	0,001	58,10%
1,00E+06	Zergling Marine	32	0,0001	69,60%
1,00E+06	Zergling Marine	32	0,00001	19,80%

Dessa forma, os experimentos mostraram que de fato taxas de aprendizado menores têm um certo benefício. Em todos os casos, a curva de aprendizado foi crescente, porém, para taxas de aprendizado grandes, a convergência foi muito rápida, mas não para o ótimo. Conforme a taxa foi diminiuindo, a convergência demorou mais, porém teve um ganho grande em desempenho. Até um certo limite, em que embora tivesse convergindo, contra um oponente de comportamento fixo, um milhão de partidas não foram suficientes para a convergência. Logo, as taxas foram fixadas entre 0,01 e 0,0001.

Por último, foram feitos testes contra oponentes de busca. Eles são oponentes mais robustos, em que pensam sobre suas ações, e as possíveis ações do oponente. Alguns deles, como o GAB e SAB são estado da arte para essa aplicação. Por se basearem em busca, alguns testes demoraram muito tempo para serem realizados, assim, um número menor de unidades foi testado, e por menos partidas. Os cenários apresentados são Dragoon Zealot, que é o nosso melhor caso, e o Zergling Marine, o pior caso.

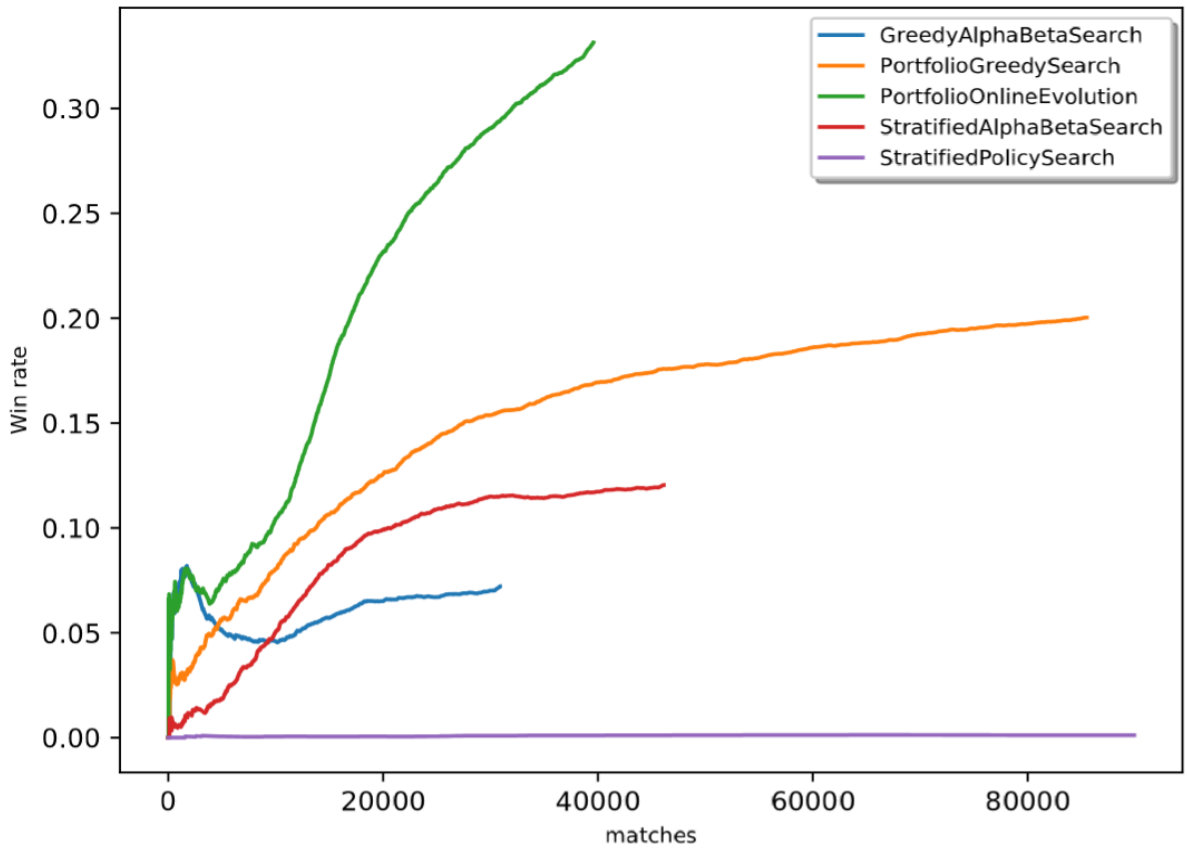

Figura 2: Gráfico de taxa de vitória contra oponentes de busca Dragoon Zealot

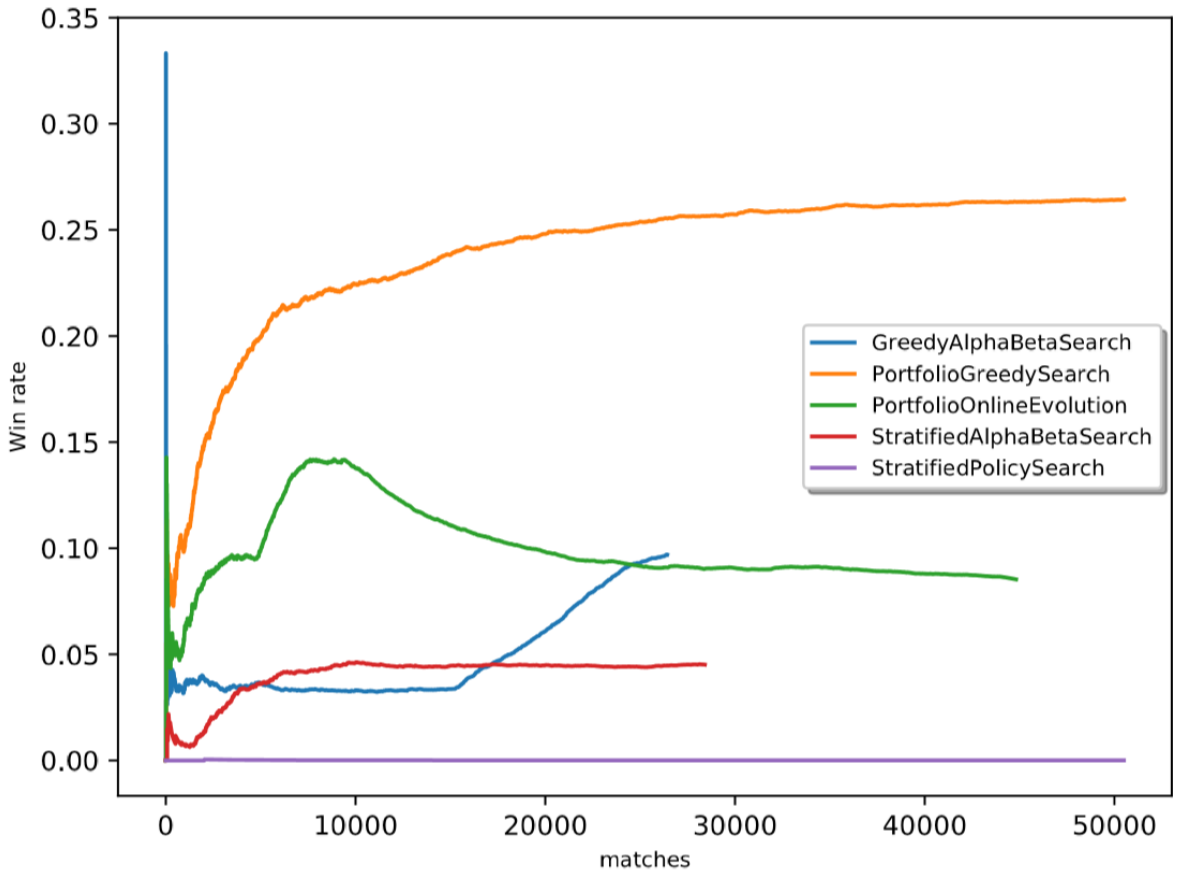

Figura 3: Gráfico de taxa de vitória contra oponentes de busca Zergling Marine

Pelos resultados, pode-se perceber que de fato o Zergling Marine é o pior caso do nosso modelo. Além disso, o Stratified Policy Search (SSS) em ambos consegue ganhar de forma categórica, embora não seja o oponente mais forte. Além disso, podemos ver que o ganho comparado a esses algorítmos não é ainda satisfatório. Entretando as curvas mostram um comportamento de aprendizado, que talvez necessite de mais tempo de treino.

5 CONCLUSÕES

Com os experimentos feitos, foi possível concluir alguns pontos importantes para serem tratados nos trabalhos futuros. (i) Diminuir a taxa de aprendizado é benéfico para o treinamento, até certo ponto. (ii) Desempenho contra oponentes mais sofisticados ainda é insatisfatório. (iii) O modelo consegue aprender em diferentes cenários, mas possui dificuldade no Zergling Marine.

Dessa forma, com a realização do trabalho, foi possível entender melhor seu comportamento e irá servir de base de comparação ao implementar os algorítmos de coordenação.

Para os próximos passos, serão implementados e testados algorítmos de coordenação explícita, com o intuito de que a convergência ocorra mais rápido, e que tenha um ganho de desempenho contra oponentes mais sofisticados. Além disso, será testado um modelo de deep reinforcement learning, para verificar se alguns dos problemas apresentados pelo atual estado do modelo é devido as features, pois são escolhidas de forma manual, ou pela linearidade da função de aproximação, que pode não conseguir descrever bem as aproximações.

6 REFERÊNCIAS

Lowe, R. & Wu, Y. (2017). Multi-agent actor-critic for mixed cooperative-competitive environments. In Aviv Tamar et al. Advances in Neural Information Processing Systems

Claus, C., & Boutilier, C. (1998). The Dynamics of Reinforcement Learning in Cooperative Multiagent Systems. In Proceedings of the Fifteenth National Conference on Artificial Intelligence (pp. 746–752).

Littman, M. L. (2001). Friend-or-Foe Q-learning in General-Sum Games. In Proceedings of the Eighteenth International Conference on Machine Learning (ICML01) (pp. 322–328). San Francisco, CA, USA: Morgan Kaufmann.

Peng, P., Yuan, Q., Wen, Y., Yang, Y., Tang, Z., Long, H., … Group, A. (2017). Multiagent Bidirectionally-Coordinated Nets for Learning to Play StarCraft Combat Games.

Panait, L. & Luke, S. (2005). Cooperative Multi-Agent Learning: The State of the Art. In Journal Autonomous Agents and Multi-Agent Systems (pp. 387- 434).