UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ARTES, CIÊNCIAS E HUMANIDADES
PROGRAMA DE PÓS-GRADUAÇÃO EM SISTEMAS DE INFORMAÇÃO

ERIC MUSZALSKA CLARO GOMES

Descoberta de *Options* Multi-tarefas: Um estudo em *StarCraft II*

São Paulo
2023

ERIC MUSZALSKA CLARO GOMES

Descoberta de *Options* Multi-tarefas: Um estudo em *StarCraft II*

Dissertação apresentada à Escola de Artes, Ciências e Humanidades da Universidade de São Paulo para obtenção do título de Mestre em Ciências pelo Programa de Pós-graduação em Sistemas de Informação.

Área de concentração: Metodologia e Técnicas da Computação

Versão corrigida contendo as alterações solicitadas pela comissão julgadora em 4 de Abril de 2023. A versão original encontra-se em acervo reservado na Biblioteca da EACH-USP e na Biblioteca Digital de Teses e Dissertações da USP (BDTD), de acordo com a Resolução CoPGr 6018, de 13 de outubro de 2011.

Orientador: Prof. Dr. Valdinei Freire da Silva

São Paulo

2023

Ficha catalográfica elaborada pela Biblioteca da Escola de Artes, Ciências e Humanidades, com os dados inseridos pelo(a) autor(a)
Brenda Fontes Malheiros de Castro CRB 8-7012; Sandra Tokarevicz CRB 8-4936

Muszalska Claro Gomes, Eric
Descoberta de options multi-tarefas: Um estudo em StarCraft II / Eric Muszalska Claro Gomes; orientador, Valdinei Freire da Silva. -- São Paulo, 2023.
74 p: il.

Dissertacao (Mestrado em Ciencias) - Programa de Pós-Graduação em Sistemas de Informação, Escola de Artes, Ciências e Humanidades, Universidade de São Paulo, 2023.
Versão corrigida

1. Aprendizado por Reforço. 2. Framework de Options. 3. Option Discovery. 4. Aprendizado por Imitação. 5. Starcraft II. I. Silva, Valdinei Freire da, orient. II. Título.

Dissertação de autoria de Eric Muszalska Claro Gomes, sob o título **"Descoberta de *Options* Multi-tarefas: Um estudo em *StarCraft II*"**, apresentada à Escola de Artes, Ciências e Humanidades da Universidade de São Paulo, para obtenção do título de Mestre em Ciências pelo Programa de Pós-graduação em Sistemas de Informação, na área de concentração Metodologia e Técnicas da Computação, aprovada em 4 de Abril de 2023 pela comissão julgadora constituída pelos doutores:

Prof. Dr. Valdinei Freire da Silva
Universidade de São Paulo
Presidente

Prof. Dr. Reinaldo Augusto da Costa Bianchi
Centro Universitário FEI

Prof. Dr. Denis Deratani Mauá
Universidade de São Paulo

Dedico este trabalho a minha esposa, Daniela, por todo o amor e apoio durante toda a jornada e ao meu orientador Valdinei, por seu tempo, paciência, dedicação e carinho. Obrigado por acreditarem em mim e por me guiarem nessa jornada.

Agradecimentos

Primeiramente, gostaria de agradecer minha esposa Daniela, por sua constante dedicação e apoio durante todo o processo de elaboração desta tese. Sua paciência e compreensão foram fundamentais para que eu pudesse me dedicar plenamente a esta jornada. Sem ela, este trabalho não seria possível.

Também gostaria de agradecer ao meu orientador, Valdinei, por sua orientação valiosa, disponibilidade constante e paciência imensurável. Seu conhecimento técnico e experiência foram fundamentais para o sucesso desta pesquisa. Além disso, agradeço a todos os professores e funcionários da EACH, por sua colaboração e por proporcionar um ambiente acadêmico estimulante.

Gostaria também de agradecer ao Itaú, não apenas como empresa, mas também como um conjunto de pessoas incríveis que me forneceram a chance seguir com todo esse trabalho.

Não posso esquecer de meus amigos, Patricia, Fabio, Ciro, Eduardo, Denis, Thiago, Gabriel, Erik e muitos outros, por serem fontes de inspiração e apoio durante todo este processo. Suas opiniões e conversas foram fundamentais para o desenvolvimento desta tese. A amizade e o apoio incondicional que recebi de todos eles foram fundamentais para o sucesso desta jornada.

Finalmente, gostaria de agradecer a todos aqueles que, de alguma forma, contribuíram para este trabalho, seja dando sugestões, discutindo ideias, revisando, ou mesmo me incentivando e dando forças para conseguir concluí-lo.

"As circunstâncias do nascimento de alguém são irrelevantes.
É o que você faz com o dom da vida que determina quem você é."
(POKEMON..., 1998)

Resumo

MUSZALSKA, Eric. **Descoberta de *options* multi-tarefas: Um estudo em *StarCraft II*** 2023. 74 f. Dissertação (Mestrado em Ciências) – Escola de Artes, Ciências e Humanidades, Universidade de São Paulo, São Paulo, 2023.

Este trabalho propõe e valida uma arquitetura para resolver problemas complexos em jogos de estratégia em tempo real, como o *Starcraft II*, utilizando o conceito hierárquico temporal de *option*. A arquitetura é baseada em uma abordagem de descoberta de *options* (*option discovery*) utilizando aprendizado por imitação para abstrair meta-políticas e políticas intra-options comuns a vários agentes. A validação foi realizada tanto em *minigames*, quanto em cenários criados especificamente para este estudo, que visam analisar o componente temporal do problema. Os resultados mostraram que a arquitetura proposta foi capaz de obter resultados próximos aos obtidos pelo agente padrão Reaver em alguns dos *minigames*, além de ser capaz de aprender uma única política genérica que se aplicaria a todos os *minigames*. Além disso, foi possível observar o comportamento das *options* para cada *minigame* no agente genérico, o que permitiu uma melhor compreensão da arquitetura proposta. Foi possível observar que a arquitetura *Multi-Level Discovery of Deep Options* (MLDDO) apresentou resultados significativos para a descoberta de *options* utilizando aprendizado por imitação. Esse trabalho também analisou o impacto da separabilidade de estados no MLDDO, comparando resultados do aprendizado com uma implementação específica para cada diferente cenário separável pelo espaço e com o aprendizado de uma única implementação generalista que busca aprender diferentes objetivos em cenários indistinguíveis pelo estado.

Palavras-chaves: Aprendizado por Reforço, *Framework* de *Options*, *Option Discovery*, Aprendizado por Imitação, Starcraft II

Abstract

MUSZALSKA, Eric. **Multi-task Option Discovery: A study in *StarCraft II*** 2023. 74 p. Dissertation (Master of Science) – School of Arts, Sciences and Humanities, University of São Paulo, São Paulo, 2023.

This work proposes and validates an architecture for solving complex problems in real time strategy games, such as Starcraft II, using the hierarchical temporal concept of options. The architecture is based on an approach of option discovery using imitation learning to abstract meta-policies and intra-options policies common to various agents. The validation was performed both on minigames, and on scenarios created specifically for this study, which aim to analyze the temporal component of the problem. The results showed that the proposed architecture was able to obtain results similar to those obtained by the standard Reaver agent in some of the minigames, and was also able to learn a single generic policy that would apply to all minigames. In addition, it was possible to observe the behavior of the options for each minigame in the generic agent, which allowed for a better understanding of the proposed architecture. It was observed that the MLDDO architecture presented significant results for the discovery of options using imitation learning. This work also analyzed the impact of state separability in the Multi-Level Discovery of Deep Options (MLDDO), comparing learning results with a specific implementation for each different scenario separable by space and with learning of a single generalist implementation that seeks to learn different goals in indistinguishable scenarios by state.

Keywords: Reinforcement Learning, Option Framework, Option Discovery, Imitation Learning, Starcraft II

Lista de figuras

Lista de algoritmos

Lista de tabelas

Lista de abreviaturas e siglas

A2C	Advanced Actor Critic
DDO	*Discovery of Deep Options*
DQN	*Deep Q Network*
KL	Kullback–Leibler
LSTM	memória de curto e longo prazo
MDPs	Processos de Decisão Markovianos
MLDDO	*Multi-Level Discovery of Deep Options*
PPO	Proximal Policy Optimization
SC2LE	*StarCraft II Learning Environment*
SMDP	Processo de Decisão Semi-Markoviano
StarCraft II	*StarCraft II*
TD	*Temporal Difference*
t-SNE	*T-distributed Stochastic Neighbor Embedding*
TPU	Unidade de processamento de tensor

Lista de símbolos

$\mathcal{S}$	Conjunto de estados
$\mathcal{A}$	Conjunto de ações
$\mathcal{R}$	Conjunto de recompensas
s	Estado atual
s'	Estado futuro
a	Ação
r	Recompensa
γ	Fator de desconto
π	Política
$\mathcal{V}$	Função valor de estado
$\mathcal{Q}$	Função valor Q de estado-ação
q	Valor do estado-ação para uma observação
ϵ	Fator de exploração
α	Taxa de aprendizado
θ	Parâmetros de um aproximador universal
ξ	Trajetória de um histórico
$\mathcal{I}$	Função distribuição de estado inicial
$\mathcal{H}$	Conjunto de *options*
h	*Option* atual
h'	*Option* do estado futuro
η	Função de meta-política (escolha de *option*)
ψ	Função terminal (troca de *option*)

Sumário

1 Introdução

O cenário de aprendizado por reforço descrito por sutton1999between tem ganhado notoriedade por propiciar a resolução de problemas considerados complexos. Esse atual reconhecimento pode ser evidenciado mais claramente por meio de implementações de modelos de aprendizado de máquina capazes de superar jogadores profissionais em jogos como xadrez e shogi (SILVER *et al.*, 2017a). O mesmo pode ser observado na implementação do *AlphaGo* (SILVER *et al.*, 2016), modelo de aprendizado de máquina capaz de vencer o campeão mundial de Go. Há ainda métodos de otimização, como o *AlphaGo* Zero (SILVER *et al.*, 2017b), que conseguiu superar o desempenho do *AlphaGo* com uma arquitetura mais simplificada, seja pelo número de redes neurais ou por remover a necessidade de observação de *replays* (HOLCOMB *et al.*, 2018).

Um exemplo de problema complexo que foi resolvido recentemente é o cenário apresentado em jogos de estratégia em tempo real, como *StarCraft II* (SC2). Para este jogo, a primeira abordagem capaz de superar jogadores profissionais foi o *AlphaStar* (VINYALS *et al.*, 2019b), implementada no final de 2018, pela Google Deepmind.

Embora a solução apresentada por Vinyals *et al.* (2019b) seja considerada o estado da arte em termos de ganho de partidas, ela apresenta um custo computacional extremamente elevado, o que sugere a existência de oportunidades de melhoria neste campo de pesquisa.

1.1 StarCraft II

Os *E-Sports*, ou esportes eletrônicos, vêm ganhando cada vez mais popularidade, chegando até mesmo, em alguns casos, a ultrapassar os jogos tradicionais em número de espectadores (SCHMITT; KOSTLER, 2017). Dentre esses jogos, os jogos de estratégia em tempo real (RTS – *Real Time Strategy*) são conhecidos por sua alta complexidade, exigindo habilidades de planejamento, tomada de decisão e raciocínio rápido. Um exemplo deste gênero é o jogo *StarCraft II*, no qual um jogador controla suas unidades para construir bases, coletar recursos e atacar seus oponentes. Por sua complexidade e variedade de possibilidades, o jogo *StarCraft II* é considerado um excelente ambiente para testar métodos de aprendizado por reforço.

O *StarCraft II* é um dos jogos de estratégia em tempo real mais populares atualmente, sendo reconhecido por sua quantidade elevada de diferentes estratégias. No jogo, cada jogador deve escolher uma raça, que pode ser selecionada entre Terrano, Protoss ou Zerg, cada qual com suas mecânicas específicas, estilos diferentes e vantagens e desvantagens próprias. Além disso, o jogo possui uma grande variedade de unidades e estruturas, o que aumenta ainda mais a complexidade das partidas.

No jogo há uma estrutura similar à "Pedra, Papel e Tesoura", em que algumas técnicas são mais adequadas contra diferentes estratégias (CHO *et al.*, 2017), sempre levando em conta a raça e estilo de jogo do seu oponente, não existindo, assim, uma única solução que possa ser adotada de forma universal para este problema. Uma das implementações utilizadas pelo *AlphaStar* é o conceito de ligas, que visa comportar e treinar diferentes tipos de estratégias em que agentes, modelos de aprendizado de máquina, interagem com outros agentes. Estes agentes são programados para tomar decisões baseadas em suas recompensas, ou seja, o impacto de suas ações no jogo, e aprender com essas interações para melhorar suas estratégias ao longo do tempo. Na figura 1 é possível ver uma base da raça Protoss controlada pelo *AlphaStar* em uma das partidas contra LiquidTLO, jogador profissional de *StarCraft II*.

Figura 1 – Imagem de partida realizada pelo AlphaStar.

Fonte – Vinyals *et al.* (2019a)

O jogo possui uma complexidade intrínseca devido à sua natureza multi-agente, uma vez que cada jogador pode interagir com até 200 unidades simultaneamente, além de ser necessário realizar uma gestão eficiente de recursos, como economia e unidades militares, simultaneamente.

Com isso, o *StarCraft II* apresenta uma quantidade de dimensões vastamente maior que os demais cenários já resolvidos de forma satisfatória com aprendizado por reforço, cerca de 10^{26} possíveis ações para cada estado (VINYALS *et al.*, 2019b).

Além do modo tradicional, o jogo permite a criação de diversos *minigames*, em que é possível definir objetivos secundários, como por exemplo, coletar recursos ou eliminar unidades inimigas. Como este universo é mais controlado e é possível obter recompensas de forma mais frequente quando comparado com uma partida convencional, é ideal para validar arquiteturas de aprendizado por reforço.

A maioria das unidades utilizadas no jogo possuem duas habilidades básicas: a de se mover e a de atacar. As unidades trabalhadoras de cada raça possuem a capacidade de coletar recursos. Além disso, muitas das unidades possuem habilidades especiais que podem ser utilizadas para alguma vantagem específica, como, por exemplo, se teleportar ou utilizar ataques especiais e mais eficientes. Desta forma, a estratégia utilizada para o controle eficaz de cada unidade traz uma complexidade muito grande ao jogo.

Existem também abordagens hierárquicas temporais, como o *framework* de *options* (SUTTON; PRECUP; SINGH, 1999), que permitem decompor o problema em objetivos macro e micro, auxiliando na resolução de problemas complexos. Esta abordagem pode ser muito útil especialmente em problemas com observações parciais, ou seja, ambientes em que o agente não tem conhecimento de todas as variáveis do problema, como ocorre em *StarCraft II*.

Cada um desses fatores aumenta a complexidade do modelo, tornando o ambiente perfeito para a validação de modelos de aprendizado por reforço em situações complexas. Esse ambiente é pouco explorado em pesquisas devido às suas características intrínsecas.

Apesar da grande complexidade do jogo e das dificuldades descritas anteriormente, em 2017, a Google DeepMind e a Blizzard criaram uma API chamada SC2LE (StarCraft II Learning Environment), que permite controlar as unidades por meio de códigos. Essa API foi desenvolvida com o objetivo de impulsionar o crescimento do campo de desenvolvimento de modelos de aprendizado de máquina usando o jogo como um ambiente de treino para o estudo de problemas complexos, como descrito em (VINYALS *et al.*, 2017).

1.2 Aprendizado por reforço

No campo de aprendizado de máquina, a abordagem de aprendizado por reforço é amplamente utilizada para resolver problemas de jogos. Essa abordagem busca otimizar o processo de tomada de decisão em um ambiente markoviano, em que o agente precisa determinar a melhor ação a ser tomada em cada situação (PUTERMAN, 1994). Um Processo de Decisão Markoviano, ou MDP, é descrito por um conjunto de estados $\mathcal{S}$, ações $\mathcal{A}$ e recompensas $\mathcal{R}$, onde cada estado $s \in \mathcal{S}$ leva, de forma determinística ou estatística, a um novo estado $s' \in \mathcal{S}$ gerando uma recompensa $r \in \mathcal{R}$. Além disso, um MDP pode incluir um fator de desconto γ para considerar a temporalidade das recompensas. Os algoritmos de aprendizado por reforço buscam encontrar a política ótima π, que escolhe a ação $a \in \mathcal{A}$ que resulta na maior recompensa acumulada esperada para o problema.

Para otimização dos MDPs, podem ser adotadas abordagens baseadas em valor (*value based*), na qual o valor de cada estado é calculado e a política resultante é dada pela escolha da ação que alcançará os estados com maior valor, ou por abordagens baseadas em busca de política (*policy search*), na qual é inicializada uma política arbitrária, que será atualizada em cada iteração de forma a escolher também as ações que resultarão na maximização da recompensa total esperada (SUTTON; BARTO, 1998).

Geralmente os problemas de aprendizado por reforço dependem de uma modelagem bem definida dos estados, principalmente em casos com estados contínuos, com muitas dimensões ou com ambientes multi-agente, em que os agentes podem ser adicionados ou removidos durante iterações. Para isso, uma abordagem *value based* geralmente utilizada em casos de maior complexidade é a utilização de uma rede neural, como por exemplo, um *Deep Q Network* (DQN) (MNIH *et al.*, 2013), na qual uma rede neural profunda é utilizada para interpretar o estado fornecido, retornando os possíveis valores para cada ação disponível.

Além disso, existem abordagens hierárquicas temporais, como o *framework* de *options* (SUTTON; PRECUP; SINGH, 1999) que permitem decompor o problema em macro objetivos e micro objetivos, auxiliando na resolução de problemas complexos. Esta abordagem pode ser muito útil especialmente em problemas com observações parciais, ou seja, ambientes que o agente não tem conhecimento de todas as variáveis do problema, como acontece Starcraft II.

Em problemas complexos, principalmente quando as recompensas são esparsas, uma alternativa bastante utilizada é o aprendizado por imitação (HAMIDI *et al.*, 2015). Essa técnica se baseia em tentar replicar uma política de outro agente ou de um especialista em determinado domínio. Uma vez que, em ambientes com recompensas esparsas, o tempo de exploração pode ser muito grande, este tipo de abordagem permite o direcionamento e a aceleração da exploração.

Na arquitetura de *options* (SUTTON; PRECUP; SINGH, 1999), um dos principais problemas é a definição de quais *options* são ideais para cada problema. Em uma partida de StarCraft II, por exemplo, é possível definir como *options* as decisões tomadas durante o jogo, como a de focar as unidades para coletar recursos ou a de atacar inimigos. Desta forma, a proposta seria aprender uma política que simule essas estratégias. No entanto, a definição prévia das *options* pode resultar em um viés indesejado no modelo.

Desta forma, uma possível otimização do conceito de *options*, que pode ou não utilizar o conceito de aprendizado por imitação, é o *option discovery* (LAKSHMINARAYANAN *et al.*, 2016). Essa estratégia busca identificar quais são as macro e micro ações realizadas por meio de observações de exemplos gerados por um especialista de forma a obter uma política o mais adequada possível para o processo em questão.

Todavia, dada a complexidade do StarCraft II, acredita-se que não exista uma política ótima para todos os problemas, mas sim várias políticas funcionais, que muitas vezes são dependentes do estilo de jogo de cada pessoa. O AlphaStar, quando realizou sua implementação, utilizou-se de um conceito de uma meta-política parametrizada pela sequência de ações de uma observação, que chamou de estatística z (VINYALS *et al.*, 2019b). Desta forma, foi possível determinar quais ações seriam as mais adequadas para cada partida específica seguindo um estilo de jogo previamente definido.

1.3 Objetivo

Este trabalho tem como objetivo principal propôr e validar uma arquitetura que utiliza o conceito de *options*, por meio de *option discovery* com aprendizado por imitação para abstrair políticas *intra-options* (micro ações) comuns a vários agentes, enquanto os agentes aprendem meta-políticas (macro atividades) específicas para resolver o problema em questão.

Com a arquitetura proposta, é esperado que o agente consiga aprender uma política que se assemelhe à obtida através de observações, utilizando do *framework* temporal de *options*. Não faz parte do objetivo deste trabalho, entretanto, obter uma política que supere em performance a política utilizada para treino originalmente.

A arquitetura proposta para o *StarCraft II* será avaliada com base na utilização do conceito de *option discovery* de *Multi-Level Discovery of Deep Options* (MLDDO) (FOX *et al.*, 2017) tanto em *minigames* propostos por Vinyals *et al.* (2017), cada um com objetivos e estados distintos, quanto em cenários criados especificamente para este trabalho, denominados como "cenários separáveis pelo espaço", onde o agente precisa de conhecimento temporal para distinguir qual tarefa está realizando, já que todos os estados podem ser aplicados a qualquer tarefa.

Outro objetivo deste trabalho é analisar o impacto da separabilidade de estados no MLDDO, comparando resultados do aprendizado com uma única rede neural para diferentes cenários separáveis pelo espaço e com o aprendizado de uma rede generalista que busca aprender diferentes objetivos em cenários indistinguíveis pelo estado. Isso permitirá avaliar a contribuição do framework de options e sua natureza temporal e hierárquica para o aprendizado.

1.4 Contribuições

Este trabalho apresenta as seguintes principais contribuições:

- A proposta de uma arquitetura que combina o conceito de *options* e meta-políticas para abstrair políticas comuns e aprender políticas específicas para resolver problemas complexos, como o cenário de *Starcraft II*;
- A realização de uma análise sobre o impacto da separabilidade de estados no MLDDO, utilizando e comparando resultados do aprendizado em *minigames* e em cenários separáveis pelo espaço, permitindo avaliar a contribuição do framework de options e sua natureza temporal e hierárquica para o aprendizado;
- A validação da eficiência da arquitetura proposta através de experimentos em *Starcraft II*;

- A contribuição para o campo de aprendizado por reforço em ambientes complexos, fornecendo uma abordagem eficiente para lidar com problemas que envolvem múltiplas tarefas e objetivos.

Essas contribuições têm o potencial de ajudar no desenvolvimento de agentes de aprendizado por reforço mais eficientes e generalistas para jogos e outros ambientes complexos. Além disso, a validação em um jogo real como *Starcraft II* fornece uma base sólida para a aplicabilidade da arquitetura proposta em outros cenários similares.

1.5 Organização dos capítulos

O capítulo 2 introduz os conceitos teóricos principais de Processos de Decisão Markovianos (MDPs), aprendizado por imitação e *framework* de *options*. O capítulo 3 mostra as principais implementações utilizadas para o ambiente *StarCraft II*. A metodologia deste trabalho é apresentada no capítulo 4, os resultados estão presentes no capítulo 5 e, finalmente, no capítulo 6 são apresentadas as conclusões e considerações finais.

2 Fundamentação teórica

Neste capítulo, serão detalhados os fundamentos teóricos abordados neste trabalho. A seção 2.1 apresenta os principais conceitos e algoritmos relacionados ao aprendizado por reforço. A seção 2.2 discute os conceitos e a teoria do aprendizado por imitação. Por fim, na seção 2.3, será explicada a teoria principal de abstração temporal e como ela se aplica ao *framework* de *options*.

2.1 Aprendizado por reforço

Nos problemas de aprendizado por reforço, há um processo em que um agente interage com um ambiente em determinado tempo t, com estado atual s_t, exercendo ações a_t, recebendo recompensa r_t e transitando para um novo estado s_{t+1} com probabilidade definida por uma função de transição $\mathcal{T}$.

Por sua vez, esta modelagem pode ser descrita como um processo de decisão markoviano (MDP). Um MDP com horizonte finito pode ser descrito pela tupla

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, \gamma, R, \mathcal{T}), \tag{1}$$

em que $\mathcal{S}$ e $\mathcal{A}$ representam o conjunto de estados e ações, respectivamente, $R : \mathcal{S} \times \mathcal{A} \to R$ indica a função recompensa dado conjunto de estado e ação, ou seja, r_t é a recompensa no tempo t em que $r_t = R(s_t, a_t)$, $\gamma \in [0, 1)$ representa o fator de desconto e $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ é a função de transição de estados, que indica as probabilidades de transições, isto é, $\mathcal{T}(s, a, s') = \Pr(s_{t+1} = s' | s_t = s, a_t = a)$.

Em um MDP, um agente interage com o ambiente em determinado tempo t em determinado estado $s_t \in \mathcal{S}$ utilizando a ação $a_t \in \mathcal{A}$, resultando em um novo estado s_{t+1} e uma recompensa r_t. Como o MDP aborda problemas probabilísticos, a chance de um estado futuro $s_{t+1} = s'$ é dada por $\mathcal{T}(s_t, a_t, s')$.

A função de escolha de qual será a ação para cada estado é chamada de política, que pode ser determinística, ou seja, $\pi : \mathcal{S} \to \mathcal{A}$ em que para o tempo t, $a_t = \pi(s_t)$, obtendo a recompensa $r_t = R(s_t, a_t)$, ou probabilística, em que $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$, que pode ser descrita por $\Pr(a_t = a | s_t = s) = \pi(s, a)$, ou seja, a ação a pode ser obtida de forma probabilística representada por $a_t \underset{a}{\sim} \pi(s_t, a)$.

Para avaliação de uma determinada política, é possível calcular a recompensa esperada $\mathcal{V}^{\pi}$, definida por

$$\mathcal{V}^{\pi} = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^{t} r_{t} | \pi\right]. \tag{2}$$

Adicionalmente, é possível calcular o valor de um estado para determinada política $\mathcal{V}^{\pi}(s)$, em que

$$\mathcal{V}^{\pi}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^{t} r_{t} | \pi, s_{0} = s\right]. \tag{3}$$

No aprendizado por reforço, é comum a busca pela política ótima π^{*} tal que $V^{\pi^{*}} \geq V^{\pi}\ \forall\ \pi \in \Pi$, sendo Π o conjunto de possíveis políticas para o MDP em questão (SUTTON; PRECUP; SINGH, 1999).

A obtenção do valor ótimo de uma política para determinado estado $\mathcal{V}^{\pi}(s)$ pode ser obtido pela equação 4, equação de Bellman:

$$\mathcal{V}^{\pi}(s) = \max_{a \in \mathcal{A}} \left\{ R(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{T}(s,a,s') \mathcal{V}^{\pi}(s') \right\}. \tag{4}$$

De forma simples, pode-se considerar que os algoritmos de aprendizado por reforço buscam identificar a política ótima π^{*}, que escolherá a melhor ação $a \in \mathcal{A}$, que resultará no estado $s \in \mathcal{S}$ de maiores valores $\mathcal{V}^{\pi}$. A política π^{*} é dada como ótima se

$$\mathcal{V}^{\pi^{*}}(s) = \max_{\pi \in \Pi} (\mathcal{V}^{\pi}(s)) \qquad \forall \quad s \in \mathcal{S}. \tag{5}$$

2.1.1 Aprendizado por reforço baseado em função valor

Quando a função de transição é conhecida, pode-se utilizar o algoritmo *Value Iteration* para calcular as funções valores, em que os valores $\mathcal{V}(s)$ são inicializados arbitrariamente e atualizados pela equação de Bellman, da forma que

$$\mathcal{V}(s) \leftarrow \max_{a \in \mathcal{A}} \left\{ R(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{T}(s,a,s') \mathcal{V}(s') \right\}, \tag{6}$$

enquanto de forma análoga, é possível calcular o valor de cada política $\mathcal{V}^{\pi}(s)$ por

$$\mathcal{V}^{\pi}(s) = \sum_{a \in \mathcal{A}} \pi(s,a) \left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{T}(s,a,s') \mathcal{V}^{\pi}(s') \right]. \tag{7}$$

Para cenários em que as funções de transição $\mathcal{T}(s,a,s')$ e funções de recompensa $R(s,a)$ não são conhecidas, é possível utilizar simulações de Monte Carlo (BONATE, 2001)

para extrair exemplos práticos da função de transição. Nestes casos, pode-se calcular uma função $\hat{\mathcal{V}}^{\pi}(s)$ aproximada obtida com observações de interações de um agente no ambiente utilizando-se da equação 3.

Em um horizonte T, com observação $n \in N$ e recompensa r_T^n da observação no tempo $t = T$, o valor de uma política $\hat{\mathcal{V}}^{\pi}(s)$ pode ser calculado por

$$\hat{\mathcal{V}}^{\pi}(s) = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=0}^{T} \gamma^t r_t^n. \tag{8}$$

Para casos em que é desejado o cálculo observando apenas uma tupla $\langle s, a, r, s' \rangle$, é possível realizar o cálculo do valor da política com base na recompensa de cada observação n em que s^n é o estado na observação, s'^n é o estado futuro e $r^n = R(s^n, a)$ é a recompensa, pela equação

$$\hat{\mathcal{V}}^{\pi}(s) = \frac{1}{N} \sum_{n=1}^{N} r^n + \gamma \mathcal{V}(s'^n). \tag{9}$$

O algoritmo *Temporal Difference* (TD) (SUTTON; BARTO, 1998) utiliza-se deste princípio para calcular o valor da política. Para isso, é utilizado o conceito de média incremental, em que

$$\overline{x}_N = \overline{x}_{N-1} + \frac{1}{N}(x_N - \overline{x}_{N-1}) = \overline{x}_{N-1} + \alpha\delta, \tag{10}$$

sendo α uma taxa de aprendizado e δ a medida de diferença entre a observação atual e a média anterior.

No algoritmo de *Temporal Difference*, para cada tupla $\langle s, a, r, s' \rangle$ é possível calcular o valor do estado para uma política dadas as observações até o momento $\mathcal{V}_n^{\pi}(s)$ pela equação

$$\mathcal{V}_n^{\pi}(s) \leftarrow \mathcal{V}_{n-1}^{\pi}(s) + \alpha\delta, \tag{11}$$

sendo

$$\delta = r + \gamma \mathcal{V}_{n-1}(s') - \mathcal{V}_{n-1}(s). \tag{12}$$

Uma abordagem relativamente comum em aprendizado por reforço é a simplificação da equação de Bellman em função de $(\mathcal{S}, \mathcal{A})$ por um valor $\mathcal{Q}(s, a)$. Nesta forma, não seria mais necessário calcular o valor de cada estado, mas sim o valor do conjunto de estado e ação de forma simultânea.

Este valor pode ser obtido pela equação

$$\mathcal{Q}(s, a) = R(s, a) + \sum_{s' \in S} \mathcal{T}(s, a, s') \mathcal{V}(s'). \tag{13}$$

Desta forma, substituindo a equação 13 na equação 4, é possível determinar a política ótima π^* em que

$$\pi^*(s,a) = \arg\max_{a \in \mathcal{A}} \mathcal{Q}(s,a). \tag{14}$$

De forma similar ao algoritmo TD, é possível calcular então a função $\mathcal{Q}(s,a)$ na n-ésima observação pela equação

$$\mathcal{Q}_n(s,a) \leftarrow \mathcal{Q}_{n-1}(s,a) + \alpha\delta, \tag{15}$$

sendo

$$\delta = r + \gamma \max_{a' \in \mathcal{A}} \mathcal{Q}_{n-1}(s',a') - \mathcal{Q}_{n-1}(s,a). \tag{16}$$

A versão tabular do *Q-Learning*, descrita no algoritmo 1 representa o processo de otimização da função $\mathcal{Q}(s,a)$ no *Q-Learning* utilizando uma taxa de aprendizado α (MITIć; MILJKOVIć; BABIć, 2011).

Algoritmo 1 $\mathcal{Q}$-Learning

Função $\mathcal{Q}(s,a)$**:**

```
t ← 0;
Inicializa Q_t(s,a) arbitrariamente;
Inicializa s_t com estado inicial;
enquanto condição de parada não cumprida faça
    a_t ~_a π(s_t, a);
    s_{t+1} ~_{s_{t+1}} T(s_t, a, s_{t+1})
    Q_t(s,a) ← Q_{t-1}(s,a) + α[r_t + γ max_{a∈A} Q_{t-1}(s_{t+1}, a_{t+1}) − Q_t(s_t, a_t)]
    s_t ← s_{t+1}
    t ← t + 1
```

O algoritmo segue em execução enquanto não encontrar um critério de parada, que pode ser por tempo limite, determinado número de iterações ou convergência, o que significa não possuir mais variações significativas.

Outra abordagem mais comum é a utilização de um aproximador universal para criação de uma função que descreve a função valor $\mathcal{Q}$. Este é o caso do *Deep Q-Learning* (MNIH *et al.*, 2013), que utiliza uma rede neural profunda, com uma saída referente a cada ação e seu valor depende do estado de entrada. Esta abordagem permite trabalhar com problemas de estados contínuos e criar aproximações para estados não visitados.

A implementação de *Deep Q-Learning* utiliza-se do mesmo princípio do algoritmo acima, mas a função $\mathcal{Q}_\theta(s,a)$, aproximação do modelo da função real $\mathcal{Q}(s,a)$ com parâmetros θ, é otimizada utilizando-se do gradiente descendente de forma que os

parâmetros θ da rede neural são atualizados seguindo a equação $\theta_n \leftarrow \theta_{n-1} + \alpha \nabla_\theta l(s, a, r, s')$, sendo $l(s, a, r, s')$ uma função de custo.

Uma função de custo pode ser qualquer função de medida de erro, mas como exemplo, pode-se adotar o erro médio quadrático, em que

$$l(s, a, r, s') = (r + \gamma \max_{a' \in \mathcal{A}} \mathcal{Q}_\theta(s', a') - \mathcal{Q}_\theta(s, a))^2. \tag{17}$$

Um dos problemas dos algoritmos de Q-Learning é que, como ele apresenta a necessidade de uma saída por possível ação, estas acabam se tornando inviáveis em cenários de ações contínuas, ou com grande quantidade de ações possíveis. Para este tipo de problema, uma abordagem que pode auxiliar são os algoritmos de *Actor-Critic*.

2.1.2 *Actor-Critic*

Enquanto os algoritmos de Q-Learning buscam otimizar a função $\mathcal{Q}(s, a)$, obtendo um valor esperado para cada possível saída, o *Actor-Critic* (CRITES; BARTO, 1995) propõe uma arquitetura que segue o mesmo princípio, decompondo o problema em duas etapas, utilizando-se de dois aproximadores, um *Actor* e um *Critic*.

O *Actor* é responsável por gerar uma função aproximadora para determinação da ação π_θ em função de θ.

Com esta função definida, o *Critic* consegue calcular o valor da política $\mathcal{V}_w^\pi(s)$ em função dos parâmetros w de um segundo aproximador, para o estado resultante da ação a escolhida pelo *Actor* e consegue realizar o cálculo do valor da política. Com este valor definido, é responsável por realizar um *policy-search*, ou seja, otimizar a política através da variação de valores utilizando o gradiente descendente.

Um pseudo-código da lógica do *Actor-Critic*, aproximado pelos parâmetros θ e w, está descrito no algoritmo 2.

Algoritmo 2 Actor-Critic

Função `ActorCritic(`s`):`

$t \leftarrow 0$;
Inicializa θ e w arbitrariamente;
Inicializa $\mathcal{V}_w^\pi(s)$ arbitrariamente;
$I \leftarrow 1$;
enquanto *condição de parada não cumprida* **faça**
$s_{t+1} \underset{s_{t+1}}{\sim} \mathcal{T}(s_t, a, s_{t+1})$
$a_t \underset{a}{\sim} \pi_\theta(s_t, a)$;
$\delta = r + \gamma \mathcal{V}_w^\pi(s_{t+1}) - \mathcal{V}_w^\pi(s_t)$
$w \leftarrow w + \alpha_w \delta \nabla_w \mathcal{V}_w^\pi(s_t)$;
$\theta \leftarrow \theta + \alpha_\theta I \delta \nabla_\theta \ln \pi_\theta(s, a)$;
$I \leftarrow \gamma I$;
$s_t \leftarrow s_{t+1}$
$t \leftarrow t + 1$

2.1.3 Exploração e Explotação

Um grande dilema na área de aprendizado por reforço é como equilibrar a exploração (*exploration*) e a explotação (*exploitation*). Exploração refere-se a investigar novos estados ou ações que podem levar a recompensas maiores, enquanto explotação se refere a seguir os caminhos já conhecidos que levam a recompensas maiores.

Em problemas do mundo real, muitas vezes não é possível navegar por todos os possíveis estados, então geralmente são utilizadas algumas abordagens para otimizar a forma de interação do agente com o ambiente.

Independente do algoritmo utilizado, por conta deste dilema, deve ser ponderado se um agente deve focar em investigar outros estados ou se deve aprofundar-se nos caminhos mais promissores já encontrados.

Uma abordagem consistindo apenas em explotação, um agente sempre escolherá as rotas com maiores valores dadas as observações anteriores. Essa ação permite calcular com maior precisão os valores de cada estado da ramificação aprofundada, bem como atualizar as rotas mais promissoras no entorno do caminho escolhido. Este procedimento, entretanto, pode induzir o algoritmo para um máximo local, uma vez que não foram investigados a fundo outros caminhos.

Uma forma de combater essa especificidade é a exploração, na qual o agente tem como objetivo interagir com os ramos que não possuem os menores valores, com a finalidade

de tentar identificar outros máximos locais que podem mostrar-se como máximos globais. Entretanto, uma política consistindo apenas de exploração, em contrapartida, pode resultar em investigação errática, resultando em gasto de tempo desnecessário.

Uma possível técnica para resolver este dilema é a estratégia ε-greedy, em que um parâmetro $\varepsilon \in [0, 1]$ é determinado inicialmente, representando a probabilidade de ser definida uma política exploratória. Isso permite que o agente equilibre entre explorar novos estados e aproveitar os estados já conhecidos.

Além disso, outras técnicas também podem ser utilizadas, como aprendizado por imitação, buscando mimetizar comportamentos observados.

2.2 Aprendizado por imitação

O aprendizado por imitação é uma abordagem de aprendizado de máquina que visa aprender um comportamento que pode ser observado por algum meio (HAMIDI *et al.*, 2015). É amplamente utilizado para replicar comportamentos de especialistas em determinados domínios e pode ser usado como uma forma de auxiliar a convergência de algum processo de aprendizado por reforço.

Existem diferentes técnicas de aprendizado por imitação, incluindo a captura de dados de comportamento humano, como a observação de partidas de jogadores, e a utilização de algoritmos de aprendizado supervisionado para prever qual ação seria tomada em um determinado estado.

No caso do StarCraft II, por exemplo, é possível observar comportamentos comuns e constantes de jogadores profissionais para criar uma política inicial. Ao observar um replay de uma partida, é possível obter o estado atual, a ação tomada, o estado resultante e a recompensa associada a cada ação realizada pelo jogador.

Esses dados podem ser usados para treinar um modelo, como uma rede neural, para prever qual ação seria tomada em um determinado estado. A otimização dos parâmetros do modelo é feita de acordo com o algoritmo de aprendizado por imitação utilizado, com o objetivo de maximizar a verossimilhança ou minimizar a divergência de Kullback–Leibler.

É importante notar que a escolha do algoritmo de aprendizado por imitação e do aproximador universal (como redes neurais) deve ser cuidadosamente considerada, pois

cada um tem seus próprios critérios e peculiaridades. Podemos considerar esse aproximador como uma função dependente dos parâmetros θ.

Utilizando-se a log-verossimilhança como medida de desempenho, é possível descrever a qualidade de uma política $\pi_\theta(s_t, a_t)$ dependente de θ para prever a trajetória $\xi = \langle s_0, a_0, s_1, a_1, ..., s_{T-1}, a_{T-1}, s_T \rangle$, ou seja, a sequência de ações a_t e estados s_t observadas em um determinado histórico para cada tempo t, em que $\mathcal{T}(\varnothing, \varnothing, s_0)$ representa a probabilidade de $s_t = s$ para $t = 0$, através da equação

$$L[\theta; \xi] = \log \mathcal{T}(\varnothing, \varnothing, s_0) + \sum_{t=0}^{T-1} \log(\pi_\theta(s_t, a_t)\mathcal{T}(s_t, a_t, s_{t+1})). \tag{18}$$

2.3 Framework de Options

O *framework* de *options* é uma abordagem utilizada para lidar com problemas de aprendizado por reforço que possuem uma estrutura temporal complexa. Ele foi introduzido pela primeira vez por Sutton, Precup e Singh (1999) e consiste em uma hierarquia de políticas, onde há uma política externa, conhecida como meta-política, que toma decisões sobre qual caminho seguir, e políticas internas, conhecidas como *intra-options*, que determinam como alcançar esse objetivo.

A ideia principal por trás do *framework* de *options* é a de quebrar um problema de aprendizado por reforço em subproblemas menores e mais gerenciáveis. Isso é feito através da definição de *options*, que são ações abstratas que representam diferentes objetivos parciais. Cada *option* tem uma política interna associada que especifica como alcançar esse objetivo e uma função terminal que determina a probabilidade de mudar da *option* atual para uma nova *option*, baseada no estado atual.

O *framework* de *options* é útil em cenários com ações temporais, pois permite ao agente planejar ações a longo prazo, enquanto também considera ações curtas. Isso é possível graças à estrutura hierárquica de políticas, onde as *options* externas (meta-políticas) determinam a melhor estratégia a seguir, enquanto as políticas internas (intra-options) determinam as ações a serem tomadas para alcançar o objetivo.

Por exemplo, em um jogo como StarCraft II, o framework de *options* poderia ser utilizado para permitir que o agente planeje estratégias a longo prazo, como construir uma

determinada unidade ou capturar um determinado objeto, enquanto também considera ações curtas, como lidar com ameaças imediatas ou coletar recursos.

É importante notar, no entanto, que existem problemas associados à definição de *options* diretamente. Em muitos casos, pode ser difícil ou impossível prever todas as *options* que serão necessárias para resolver um problema dado. Além disso, o número de *options* pode ser muito grande, tornando difícil a representação e o treinamento de políticas para cada uma delas.

Para lidar com esses problemas, é possível utilizar técnicas de descoberta de *options*, que buscam aprender as *options* a partir dos dados, ao invés de defini-las diretamente. Essas técnicas serão discutidas em detalhes na próxima seção.

Além disso, o framework de *options* tem uma aplicação importante em jogos de estratégia, como StarCraft II, onde há uma série de objetivos parciais e ações temporais a serem consideradas. Por exemplo, no StarCraft II, um jogador pode precisar tomar decisões sobre quais unidades produzir, como expandir seu território e como atacar o adversário. O framework de *options* permite que o agente aprenda a selecionar a melhor estratégia a seguir e as ações a serem tomadas para alcançar esses objetivos parciais.

Este *framework* pode ser representado matematicamente pela tripla $\langle \mathcal{I}_h, \pi_h, \psi_h \rangle$ para uma *option* $h \in \mathcal{H}$, em que $\mathcal{I}_h \subseteq \mathcal{S}$ é a definição dos possíveis estados iniciais para determinada *option*, $\pi_h : \mathcal{S} \times \mathcal{A} \to [0, 1]$ em que $Pr(a_t = a | s_t = s, h_t = h) = \pi_h(s, a)$ é a política interna da *option* h e $\psi_h : \mathcal{S} \to [0, 1]$ determina uma função terminal, ou seja, a probabilidade de mudança de *option*.

Em qualquer estado é possível calcular a função $\psi_h(s)$, que determina qual a probabilidade de mudança da *option* atual h para uma nova *option* h'. Caso seja realizada a mudança, a função $\eta : \mathcal{S} \times \mathcal{H} \to [0, 1]$ define $Pr(h_t = h | s_t = s) = \eta(s, h)$. Uma descrição mais detalhada da execução pode ser encontrada no algoritmo 3.

Algoritmo 3 Option Framework

Função `Escolhe ação com` *`option`*`(`s`):`
- **se** h_t *não está inicializado* **então**
  - $h_t \underset{h}{\sim} \eta(s, h)$;
- $mudaDeOption \sim \psi_{h_t}(s)$;
- **se** $mudaDeOption$ **então**
  - $h_t \underset{h}{\sim} \eta(s, h)$;
- $a_t \underset{a}{\sim} \pi_{h_t}(s, a)$;
- **Resultado:** a_t

É importante notar que, ao utilizar o framework de *options*, o problema é transformado em um Processo de Decisão Semi-Markoviano (SMDP),que possuem funções valor $\mathcal{V}_h(s)$ e função $\mathcal{Q}_h(s,h)$ correspondentes às funções de um MDP convencional. (SUTTON; PRECUP; SINGH, 1999)

No *framework*, é possível definir a descrição das funções de meta-política $\eta(s,h)$, políticas $\pi_h(s,a)$ e terminais $\psi_h(s)$ da forma que for mais conveniente. As políticas $\pi_h(s,a)$ podem ser definidas por regras arbitrárias predefinidas, enquanto a meta-política e a função terminal são ambas treinadas em algum algoritmo de aprendizado por reforço.

O problema deste tipo de abordagem é que é necessário ter um conhecimento prévio do problema, assim como as ações mais comuns. Este conhecimento, por sua vez, depende de uma descrição muito bem detalhada da solução específica que está sendo desejada no momento, o que pode não ser tão fácil de obter. Para contornar esse tipo de problema, existem abordagens de *option discovery*, implementações que permitem descobrir e otimizar simultaneamente as funções $\eta(s,h), \pi_h(s,a)$ e $\psi_h(s)$. Além disso, para problemas complexos, como no caso de StarCraft II, onde existem diversas ações possíveis e estados, é esperado que essas abordagens se mostrem ainda mais eficazes, pois permitem uma melhor compreensão do problema e a otimização das ações e estratégias a serem tomadas.

2.3.1 Option-Discovery

Enquanto o desenho original do *framework* de *options* não especifica nenhuma forma de obter cada política interna, alguns estudos recentes trazem alternativas para suas obtenções. Dentre os principais artigos sobre *option discovery*, destacam-se o option-critic framework (BACON; HARB; PRECUP, 2017), *framework* laplaciano para descoberta de *options* (MACHADO; BELLEMARE; BOWLING, 2017), *framework* de *Sucessor Options* (RAMESH; TOMAR; RAVINDRAN, 2019) e o *Multi-Level Discovery of Deep Options* Fox *et al.* (2017).

As três primeiras implementações utilizam-se de ferramentas para descoberta de *options* dentro do treino do aprendizado por reforço, enquanto a última utiliza-se de aprendizado por imitação como fonte de descoberta de *options*. Como o foco desta pesquisa consiste na descoberta de *options* entre diferentes estilos de jogo, neste trabalho será

aprofundada apenas a implementação *Multi-Level Discovery of Deep Options* (MLDDO) de Fox *et al.* (2017), que permite a extração deste estilo de jogo através de *replays*.

O MLDDO utiliza-se da execução do *Discovery of Deep Options* (DDO) em diversos níveis. Para simplificar, somente o DDO será focado para a futura generalização. No DDO, todas as definições criadas pelo *framework* de *options* permanecem válidas, mas adicionalmente é calculada uma probabilidade de determinada trajetória ξ ocorrer seguindo os mesmos princípios utilizados em aprendizado por imitação. Essa probabilidade é utilizada para otimizar as funções $\eta(s,h)$, $\pi_h(s,a)$ e $\psi_h(s)$ de forma a maximizar a verossimilhança da trajetória observada. A ideia principal é que, através da otimização dessas funções, seja possível identificar padrões de comportamento que caracterizam diferentes estilos de jogo.

É importante notar que, ao utilizar essas técnicas de *option discovery*, é possível obter políticas internas de forma automatizada, sem a necessidade de um conhecimento prévio do problema. Isso é particularmente útil em cenários onde o espaço de estados é muito grande ou onde as ações mais comuns não são conhecidas a priori.

Neste *framework*, o conceito de verossimilhança é otimizado para N observações de forma que

$$\theta = \theta + \alpha \sum_{n \in N} \nabla_\theta L[\theta; \xi_n], \tag{19}$$

em que

$$L[\theta; \xi_n] = \log Pr_\theta(\xi). \tag{20}$$

No artigo original, é demonstrado que é possível calcular a probabilidade de uma trajetória ocorrer dados os parâmetros θ da arquitetura, simplificada por $Pr_\theta(\xi)$, pelas componentes progressiva $\phi(h)$ e regressiva $\omega(h)$ para tempo t com horizonte T, da forma

$$Pr_\theta(\xi) = \sum_{h \in \mathcal{H}} Pr_\theta(\xi, h_t = h) = \sum_{h \in \mathcal{H}} \phi_t(h)\omega_t(h), \tag{21}$$

em que

$$\phi_t(h) = Pr_\theta(s_0, a_0, ..., s_t | h_t = h) \tag{22}$$

e

$$\omega_t(h) = Pr_\theta(a_t, s_{t+1}, ..., s_T | s_t, h_t = h). \tag{23}$$

Para isso, tem-se de forma progressiva que

$$\phi_0(h) = Pr(s = s_0 | t = 0)\eta(s_0, h) \tag{24}$$

e para $0 \leq t < T - 1$

$$\phi_{t+1}(h') = \sum_{h\in\mathcal{H}} \left(\phi_t(h)\pi_h(s_t,a_t)\mathcal{T}(s_t,a_t,s_{t+1})\psi_h(s_{t+1})\right)\eta(s_{t+1},h') \\ +\phi_t(h')\pi(s_t,a_t)\mathcal{T}(s_t,a_t,s_{t+1})(1-\psi_h(s_{t+1})). \tag{25}$$

Da mesma forma, tem-se de forma regressiva que

$$\omega_{T-1}(h) = \pi_h(s_{T-1},a_{T-1})\mathcal{T}(s_{T-1},a_{T-1},s_T) \tag{26}$$

e para $0 \leq t < T-1$

$$\omega_t(h) = \pi(s_t,at)\mathcal{T}(s_t,a_t,s_{t+1})\left(\psi_h(s_{t+1})\sum_{h'\in\mathcal{H}}\eta(s_{t+1},h')\omega_{t+1}(h') + (1-\psi_h(s_{t+1}))\omega_{t+1}(h)\right). \tag{27}$$

No mesmo artigo, é realizada a demonstração do gradiente descendente do logaritmo da verossimilhança, objeto de otimização para definição de um modelo que maximiza a probabilidade de replicar trajetórias observadas. De forma alternativa, é possível calcular o valor do gradiente descendente da função de custo utilizando bibliotecas como o pyTorch ou Tensorflow.

No artigo apresentado, entretanto, a arquitetura não é apresentada como função final, mas sim como objeto de definição de *options*, em que a função $\eta(s,h)$ ainda poderia ser otimizada por um algoritmo de aprendizado por reforço arbitrário para o resultado final.

3 Implementações em *StarCraft II*

Este capítulo apresenta trabalhos encontrados na literatura que têm abordagens semelhantes às propostas neste trabalho e apresentam resultados promissores. Esses estudos servem como motivação para o desenvolvimento da pesquisa envolvendo o MLDDO e o StarCraft II. É importante mencionar que, durante a revisão bibliográfica, não foi encontrado nenhum artigo que aplique o MLDDO no StarCraft II.

Com a API desenvolvida por Vinyals *et al.* (2017), surgiram muitas implementações de inimigos cada vez mais fortes. A maioria das abordagens consiste em implementações de regras de ações condicionais, como na implementação de Takino e Hoki (2015), ou algoritmos genéticos, como nos trabalhos de Schmitt e Kostler (2017) e Gajurel e Louis (2019).

Outras abordagens se concentram na otimização de estratégias macro, como é o caso da proposta de Tang *et al.* (2018), em que o algoritmo decide quais ações devem ser tomadas, enquanto as execuções são realizadas por sequências de comandos pré-programadas.

As estratégias supracitadas são relativamente simples, mas há algumas abordagens que criaram arquiteturas bastante complexas, como a Google DeepMind, que criou um modelo de aprendizado de máquina capaz de vencer um jogador profissional de StarCraft II, usando técnicas de aprendizado por transferência e reforço (ARULKUMARAN; CULLY; TOGELIUS, 2019). Outra abordagem de sucesso foi o SCC (WANG *et al.*, 2021), baseada no AlphaStar, desenvolvida por Wang *et al.* (2020). Ambos os casos apresentam uma complexidade da arquitetura, envolvendo várias redes neurais diferentes, cada qual com suas peculiaridades e técnicas específicas, gerando uma impossibilidade de treinamento na maioria das máquinas atuais.

Para essa pesquisa, é importante destacar algumas implementações que se concentram em minigames do StarCraft II. A DeepMind SC2LE, por exemplo, possui implementações em uma coleção de exemplos de minigames (VINYALS *et al.*, 2017). O Reaver, por sua vez, é uma implementação que utiliza agentes específicos para cada minigame, utilizando o Advanced Actor Critic (A2C) ou Proximal Policy Optimization (PPO) (RING, 2018). A implementação de ByeongUk (2019) também é uma opção relevante. Essas implementações em minigames são úteis como referência para cenários de teste mais controlados,

em comparação com as abordagens mais complexas, como o AlphaStar ou o SCC, que são capazes de vencer partidas completas.

Para continuar, a seção seguinte apresenta uma revisão detalhada da implementação da DeepMind SC2LE, seguida por uma descrição da implementação do Reaver e finalizando com uma revisão da implementação AlphaStar.

3.1 DeepMind SC2LE

A DeepMind SC2LE é uma implementação de um ambiente de desenvolvimento, que possui exemplos de *minigames* desenvolvida pela Google DeepMind para o jogo StarCraft II. *Esses* minigames foram criados com o objetivo de fornecer um ambiente de teste controlado para a pesquisa em aprendizado de máquina, permitindo que os pesquisadores testem e desenvolvam agentes de aprendizado por reforço.

Os minigames incluem desafios como combate em espaço fechado, coletar recursos e construção de unidades. Eles foram desenvolvidos para serem desafiadores, mas ao mesmo tempo simples o suficiente para serem resolvidos por agentes de aprendizado por reforço. São eles:

- *MoveToBeacon*: o jogador controla um soldado, unidade militar da raça Terrano, que deve chegar a maior quantidade de vezes possível em pilares brilhantes. Há 1 pilar aleatoriamente espalhado pelo mapa por vez;
- *CollectMineralShards*: o jogador controla dois soldados e deve chegar a maior quantidade de vezes possível em cristais espalhados pelo mapa. Há 20 cristais aleatoriamente espalhados por vez;
- *DefeatRoaches*: o jogador controla nove soldados, que devem eliminar quatro baratas, unidade militar da raça Zerg, do lado oposto do mapa, sendo que todas as unidades estão dentro do campo de visão inicial;
- *FindAndDefeatZerglings*: o jogador controla três soldados, que devem eliminar 26 zergnídeos, unidade militar da raça Zerg, espalhados pelo mapa, sendo que 23 destes estão fora do campo de visão inicial;
- *DefeatZerglingsAndBanelings*: o jogador controla nove soldados, que devem eliminar seis zergnídeos e quatro tatus-bomba, unidade militar da raça Zerg do lado oposto do mapa, sendo que todas as unidades estão dentro do campo de visão inicial;

- *CollectMineralsAndGas*: o jogador controla 12 VCEs, unidade coletora, e um Centro de Comando, estrutura, todos da raça Terrano. No mapa há 16 campos de mineral e um gêiser de vespeno, ambos recursos naturais sem dono. O jogador tem por objetivo pegar a maior quantidade possível de minérios e gás dos recursos do mapa em um tempo limite;
- *BuildMarines*: o jogador controla 12 VCEs e um Centro de Comando. No mapa há 8 campos de mineral. O jogador tem por objetivo juntar recursos, construir um depósito de suprimentos, depois construir um quartel e então construir um soldado;

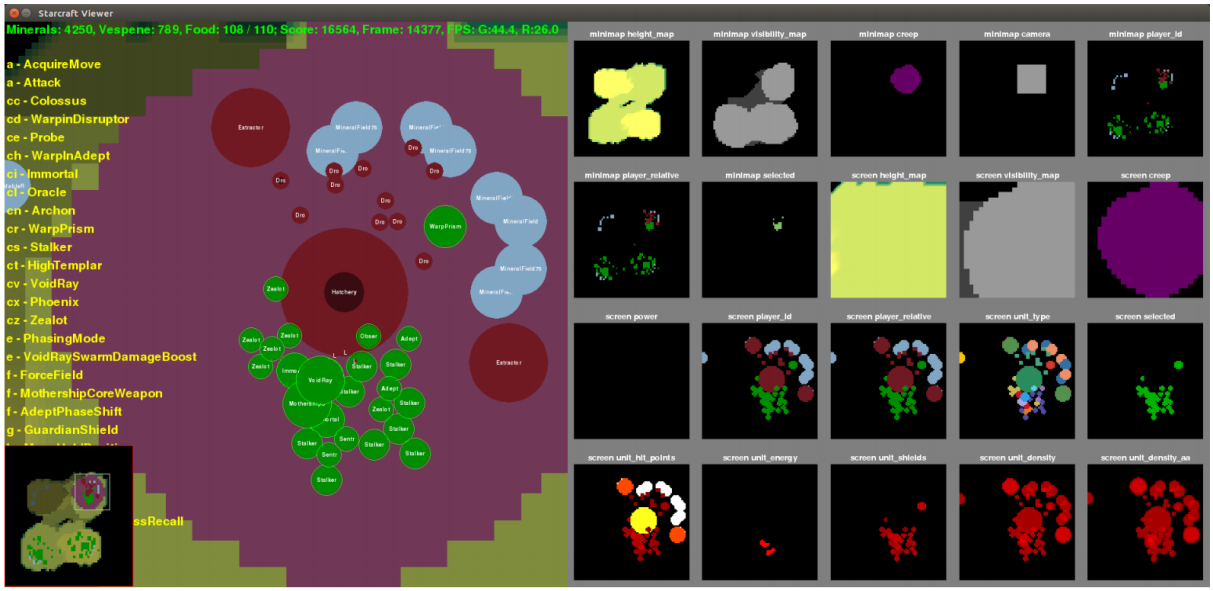
No *StarCraft II*, um jogador possui duas visões principais, a tela, que contém detalhes de todas as unidades que se encontram na área em destaque, e um minimapa, que trás informações mais genéricas sobre o mapa como um todo.

Estas informações podem ser observadas e sintetizadas pela visão da API na maior imagem da figura 2. Em todos os cenários, o jogador só consegue visualizar unidades que possui ou que estão dentro de um raio de visão de suas unidades, enquanto os inimigos não revelados não aparecem nos dados ou imagens.

Para a implantação da API, e a implementação dos agentes no SC2LE, a descrição de estado é composta primariamente por quatro tipos de informações:

- Informações escalares: dados de quanto tempo se decorreu desde o início da partida, quantidade de minerais e gás coletados, entre outras informações que descrevem o estado momentâneo do jogador;
- Entidades: dados de cada unidade no jogo que o jogador tem ciência, como qual o tipo de unidade, quanto de vida esta possui, posição, entre outros;
- Minimapa: imagens disponibilizadas pelo jogo contendo visões como o relevo do mapa em questão, posição das unidades aliadas e inimigas pelo mapa, disposição de recursos, etc.
- Ações: um vetor de possíveis identificador de ações disponíveis dado estado atual;

Figura 2 – Visão gerada pela API sobre os dados disponíveis do jogo.



Fonte – Vinyals *et al.* (2019b)

No caso do StarCraft II, em especial, há uma certa discrepância entre o que um jogador comum consegue ver e as informações disponíveis na API. Na figura 2 é possível ver 21 diferentes visões das informações disponíveis na API, sendo alguns componentes dimensões do minimapa e outros, dimensões da tela. Adicionalmente, em verde e na parte superior da imagem maior, há algumas das informações gerais de estado, e em amarelo na parte esquerda da figura, é possível observar os identificadores disponíveis para o estado atual.

Adicionalmente, a ação do jogador é considerada como um vetor de 12 dimensões, todas categóricas, sendo elas:

- *function_id*: Lista de indicadores de ações disponíveis pelos agentes. Pode ser até 573 valores;
- *screen*: variável categórica de duas dimensões com resolução utilizada na execução, como por exemplo 16x16, 64x64 ou 128x128;
- *minimap*: variável categórica de duas dimensões com resolução utilizada na execução, como por exemplo 16x16, 64x64 ou 128x128;
- *screen2*: variável categórica de duas dimensões com resolução utilizada na execução, como por exemplo 16x16, 64x64 ou 128x128;
- *queued*: Indicador se a ação deve ser executada instantaneamento (zero), ou se deve ser colocada em uma fila para execução (um);

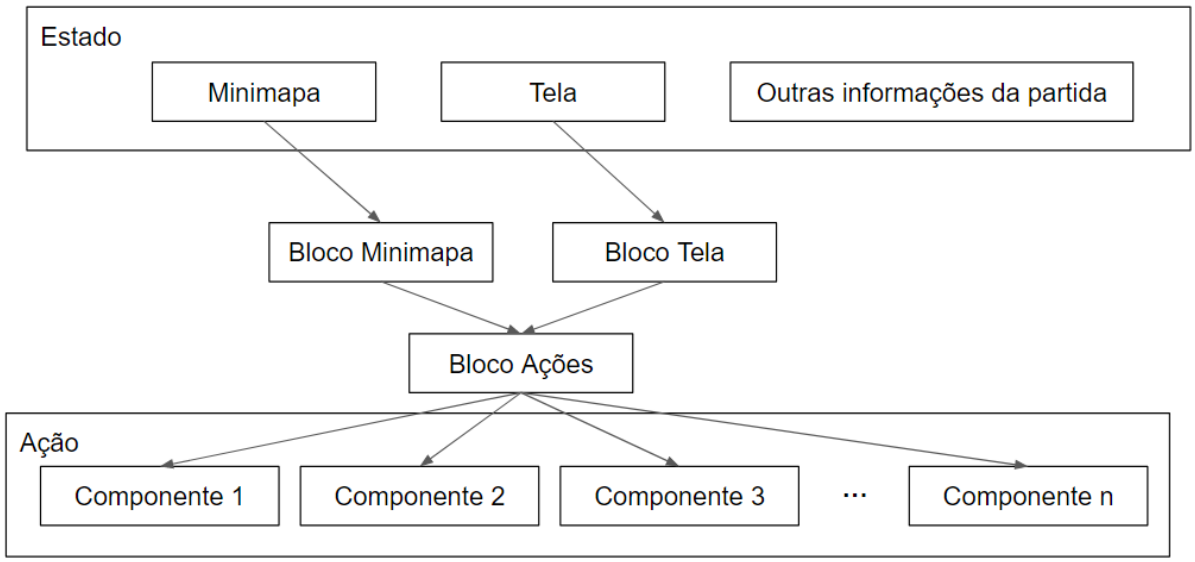
- *control_group_act*: Variável com 5 possíveis valores;
- *control_group_id*: Variável com 10 possíveis valores;
- *select_add*: Variável com 2 possíveis valores;
- *select_point_add*: Variável com 4 possíveis valores;
- *select_unit_act*: Variável com 4 possíveis valores;
- *select_worker*: Variável com 4 possíveis valores;
- *build_queue_id*: Variável com 10 possíveis valores;

3.2 Reaver

O Reaver é uma implementação de aprendizado por reforço para o jogo StarCraft II que foi desenvolvida para proporcionar um ambiente de teste controlado para a pesquisa em aprendizado de máquina. Ele inclui três agentes programados: um agente randômico, usado como baseline; um agente Advanced Actor Critic (A2C) e um agente Proximal Policy Optimization (PPO).

A arquitetura do Reaver é relativamente simples, comparada com outras abordagens mais complexas, como o AlphaStar ou o SCC. Ele utiliza informações do estado da tela e do minimapa, processadas separadamente em blocos convolucionais, e o resultado é enviado para um bloco de ações, que retorna um vetor de dez componentes de ação. Isso significa que ele desconsidera duas dimensões originalmente disponíveis no ambiente DeepMind SC2LE.

Figura 3 – Desenho simplificado da arquitetura do A2C implementada no Reaver.



Fonte – De autoria própria.

Na figura 3 há uma visão geral da arquitetura utilizada pelo Reaver, enquanto nas figuras 4, 5 e 6, há detalhes das camadas da rede neural utilizadas para minimapa, tela e ações, respectivamente.

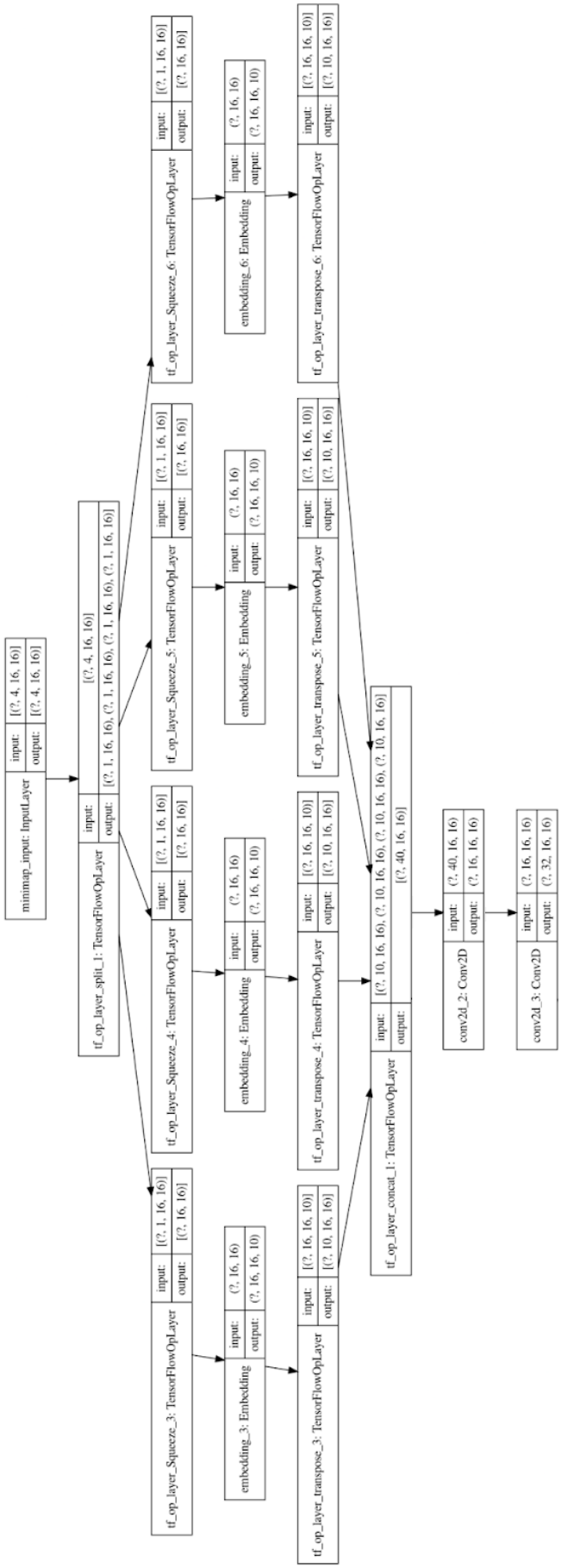
Os resultados do Reaver são comparados com os baselines do agente A2C aprendendo 6 minigames individualmente, comparando-se com os resultados do DeepMind SC2LE e do DeepMind ReDRL. Em geral, o Reaver é uma opção interessante para estudiosos de aprendizado por reforço que desejam trabalhar com StarCraft II, devido à sua simplicidade e facilidade de uso, além de ter uma arquitetura simplificada.

3.3 AlphaStar

A solução desenhada no AlphaStar pode ser dividida em dois principais pontos: treinamento supervisionado e aprendizado por reforço. Ambos compartilham uma definição de estado definidos pela mesma arquitetura base, mas com peculiaridades no treinamento e aplicação.

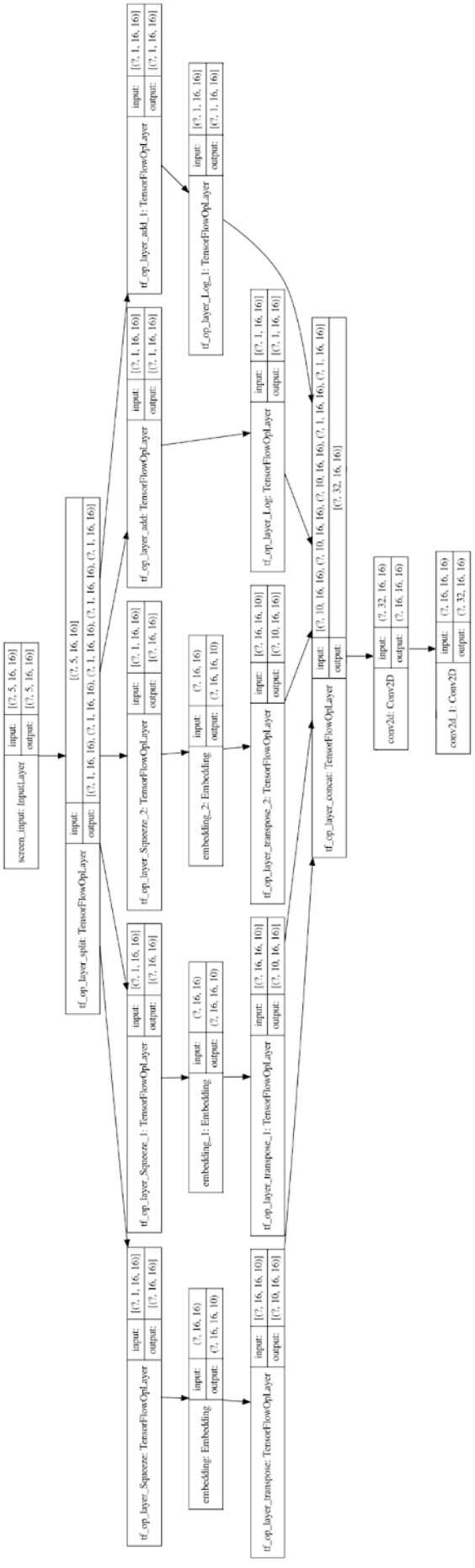
Como descrito anteriormente, uma das maiores dificuldades encontradas em um problema de aprendizado por reforço é a definição de estado. Cada um desses tipos de informações é consolidado por um encoder específico, e seus resultados são utilizados como input para uma rede neural central em profundidade com uma memória de curto e longo

Figura 4 – Detalhes do bloco de minimapa utilizado na arquitetura do A2C implementada no Reaver.



Fonte – De autoria própria.

Figura 5 – Detalhes do bloco de tela utilizado na arquitetura do A2C implementada no Reaver.



Fonte – De autoria própria.

Figura 6 – Detalhes do bloco de ações utilizado na arquitetura do A2C implementada no Reaver.

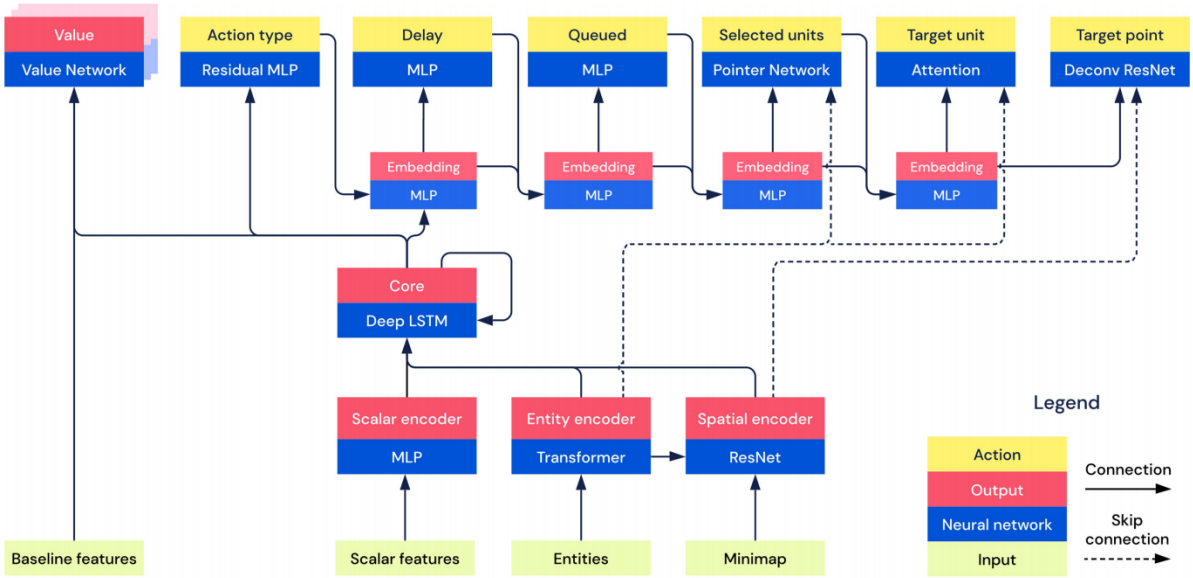
Fonte – De autoria própria.

prazo (LSTM). O resultado desta última rede é um vetor capaz de descrever todo o estado da partida, considerando as entradas anteriormente descritas e os eventos anteriores.

Além das entradas supracitadas, são utilizadas informações consideradas como *baseline features*. Estas são compostas primariamente pela estatística z coletada de outras partidas observadas de jogadores. Durante extração de informações de *replays*, a sequência das 20 primeiras construções e unidades realizadas pelo jogador é coletada, bem como suas posições e informações de unidades, construções, efeitos e atualizações que são realizados durante a partida.

Com os dados da rede neural central e das *baseline features*, uma sequência de redes neurais é utilizada para determinar qual ação será realizada, quando será realizada, por quais unidades, e quais serão os alvos. Na figura 7, há um desenho da arquitetura no qual é possível identificar com mais detalhes cada técnica utilizada, bem como cada rede da solução.

Figura 7 – Desenho da arquitetura implementada no AlphaStar.



Fonte – Vinyals *et al.* (2017)

Como explicado anteriormente, uma das formas de extração de dados de jogadores abordados pelo AlphaStar é o uso da estatística z. Essa abordagem permite abstrair estratégias diferentes utilizadas por jogadores em diversas partidas, a fim de montar um plano que será executado na partida.

A estatística z consiste na coleta de dados da construção do jogador ao longo da partida. Na coleta, são obtidos dados das 20 primeiras construções ou unidades feitas pelo

jogador, bem como outros dados, como momento de atualizações ou construções. Nesta coleta, para reduzir o viés, em 10% das observações, os dados são zerados, não ponderando o treinamento.

Tanto a implementação de aprendizado supervisionado quanto o aprendizado por reforço buscam determinar a política $\pi_\theta(a|s,z)$, porém possuem formas diferentes de otimizar a política. No primeiro, a arquitetura anterior é treinada para otimizar a divergência de Kullback–Leibler (KL) entre os dados amostrados do jogador e os realizados pela implementação.

No aprendizado por reforço, a modelagem das recompensas leva em consideração, além da recompensa original dada pela função do aprendizado por diferença temporal (TD(λ)) (TESAURO, 1995), uma pseudo-recompensa que pondera a distância de Hamming da ação executada com a estatística z.

A implementação do AlphaStar trouxe também o conceito de treinamento em liga. Com a abordagem anterior, foram criados mais de 900 agentes, que jogaram entre si, e que foram divididos em três categorias: agentes principais, exploradores da liga e exploradores principais, em que as partidas eram realizadas em self-play e que a abordagem permitia explorar falhas de agentes.

Embora esta abordagem tenha gerado uma evolução muito grande em desempenho, o custo para tal implementação é muito grande. Durante o experimento, foram utilizados 32 TPUs de terceira geração (unidade de processamento de tensor), quantidade que não está disponível para aquisição de usuários finais durante o período de desenvolvimento do trabalho, por um período de treinamento de 44 dias.

Dados os recursos disponíveis para a pesquisa e o tempo hábil para a execução da mesma, este tipo de treinamento não será abordado aqui e terá uma abordagem simplificada.

4 Metodologia

Este capítulo tem como objetivo descrever a metodologia utilizada neste trabalho. Inicialmente, é apresentado o escopo deste trabalho e as premissas assumidas. Em seguida, são descritas as técnicas utilizadas, incluindo o escolhido como base para o trabalho, o MLDDO.

A seção 4.1 detalha as peculiaridades da implementação da arquitetura, incluindo as principais equações utilizadas e a representação da arquitetura na figura. É destacado que foram utilizadas redes neurais convolucionais para aproximar as funções de meta-política, probabilidade de troca de opções e política intra-opção. Além disso, é apresentado o conceito de diferenciação entre estado macro e estado específico, utilizado para criar visões distintas entre meta-política e política intra-opção.

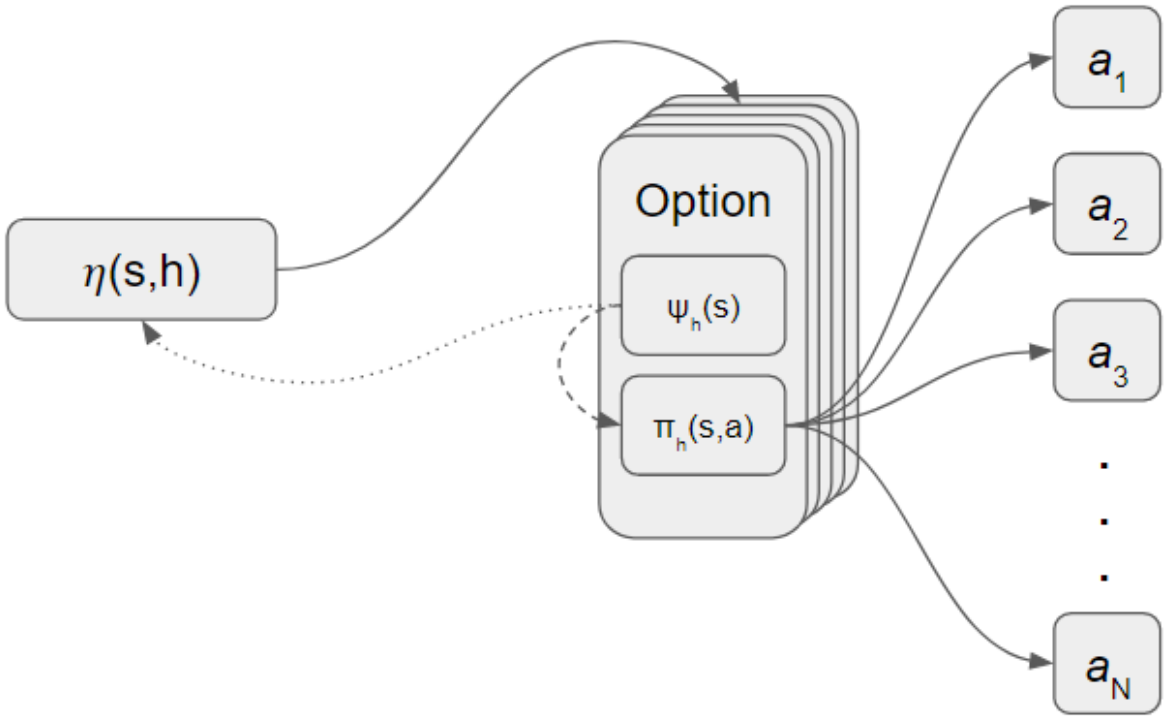
Na seção 4.2 são descritas as técnicas utilizadas para treinar, avaliar e comparar os resultados e a seção 4.3 apresenta as detalhes necessários para a implantação da arquitetura proposta pelo trabalho no *StarCraft II*.

Por fim, na seção 4.4 é apresentada uma crítica dos resultados obtidos e uma discussão dos impactos da arquitetura nos cenários anteriormente mencionados, bem como sugestões para direcionar estratégias desejadas durante a implementação proposta.

4.1 Arquitetura proposta

Foi escolhido o MLDDO como o *framework* base, e para isso, a arquitetura precisa refletir quatro equações principais: a função de meta-política, representada por $\eta(s, h)$; a função que determina a probabilidade de troca de opção, representada por $\psi_h(s)$; a política intra-opção, representada por $\pi_h(s, a)$; e a função de custo, determinada pela verossimilhança, representada por $L[\theta; \xi_n] = \log(\sum_{h \in \mathcal{H}} \phi_t(h)\omega_t(h))$, em que $\phi_t(h)$ e $\omega_t(h)$ são definidos nas equações 22 e 23, respectivamente. A figura 8 ilustra como essas iterações são executadas na arquitetura proposta.

Figura 8 – Desenho da arquitetura proposta. A função $\psi_h(s)$ determina se deve haver uma troca de *option*, direcionando para a função $\eta(s,h)$ representada pela linha pontilhada, ou se direciona para a política $\pi_h(s,a)$ pela linha tracejada, em que é calculada a probabilidade de cada ação $a \in \mathcal{A}$.



Fonte – De autoria própria.

Para as funções $\eta(s,h)$, $\psi_h(s)$ e $\pi_h(s,a)$ foram utilizadas redes neurais convolucionais profundas como aproximadores. Devido à natureza hierárquica do *framework* de opções, foi utilizada uma visão geral do ambiente para a função $\eta(s,h)$, enquanto as funções $\psi_h(s)$ e $\pi_h(s,a)$ refletem uma visão detalhada, mas parcial do estado.

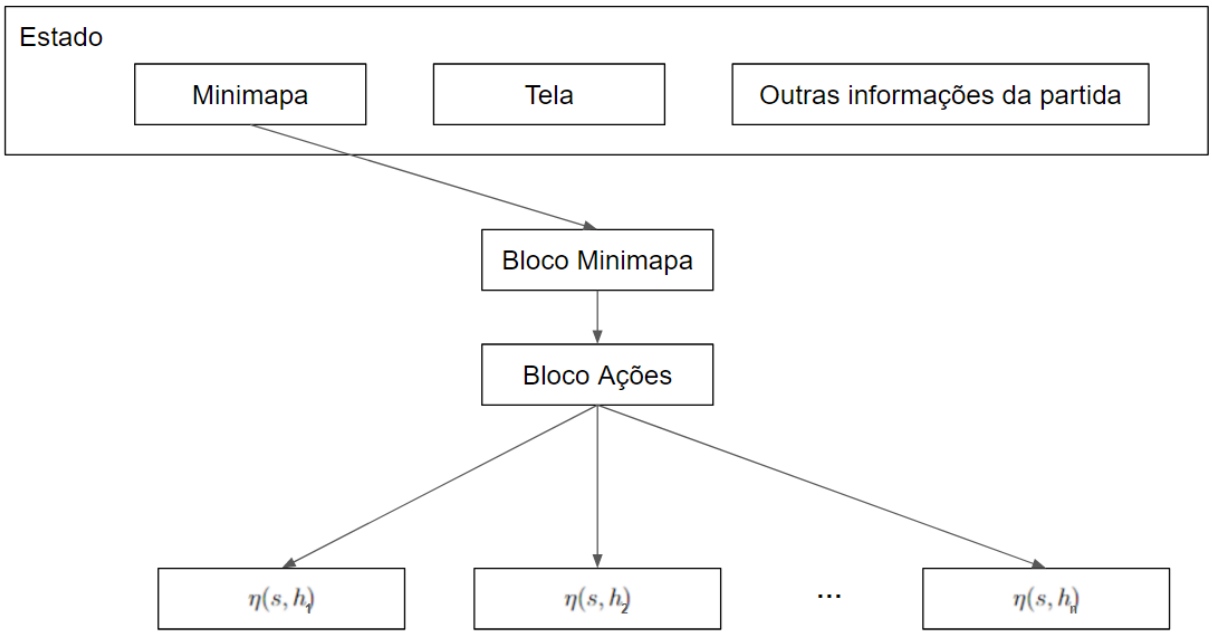
Foi considerada a diferenciação entre estado macro e estado específico para criar visões distintas entre meta-política e política intra-opção. Para o cenário do jogo *StarCraft II*, o estado macro é representado pela parte do estado do minimapa, pois permite uma visão geral do jogo, mas sem detalhes, como vida das unidades aliadas e inimigas. Já o estado específico é representado pela tela do jogador, que contém mais informações, mas é recortada para apenas um segmento do todo.

A arquitetura do Reaver foi escolhida como base para a construção dos aproximadores da arquitetura proposta, pois é suficiente para descrever o problema do jogo e é relativamente rápida para treinar. A entrada para a função $\eta(s,h)$ foi limitada apenas às

dimensões relacionadas ao minimapa, passando pelo bloco de minimapa, representado na figura 4.

A construção do aproximador da função $\eta(s, h)$ foi baseada em entradas relacionadas ao minimapa, passando pelo bloco de minimapa, representado na figura 4. Este bloco foi seguido por um consolidador, que foi reaproveitado da arquitetura do bloco de ações, detalhado na figura 6. A arquitetura desse aproximador é ilustrada na figura 9.

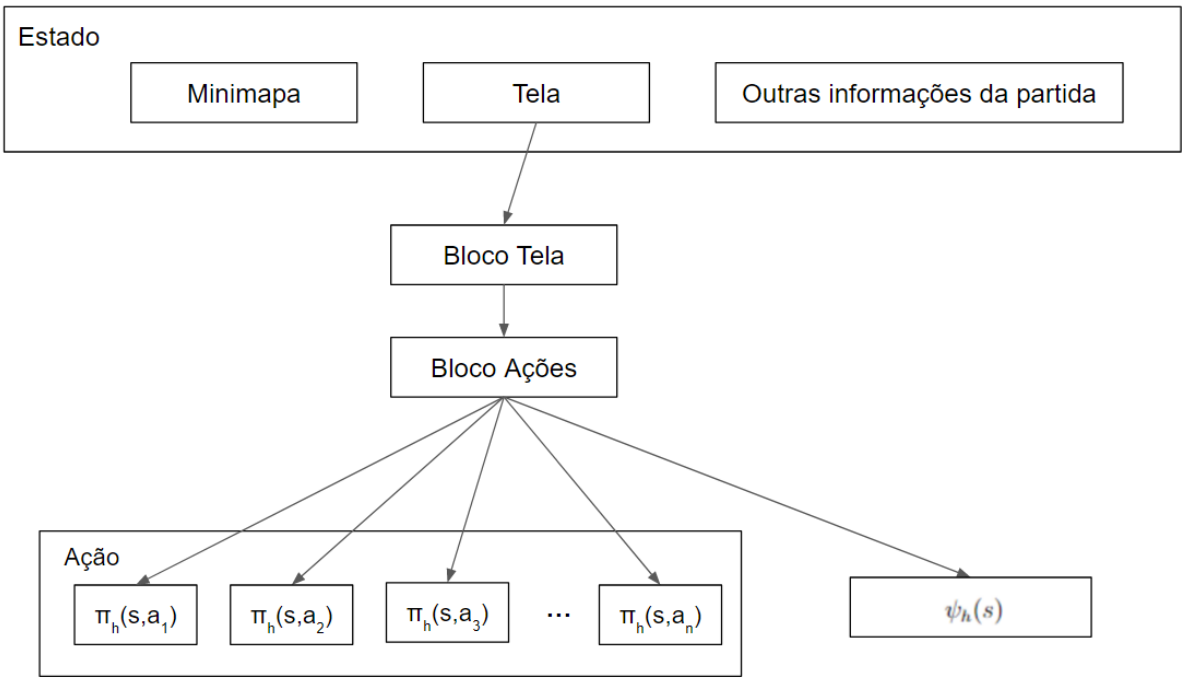
Figura 9 – Desenho da parte da arquitetura para cálculo de $\eta(s, h)$. Cada componente de saída da rede resulta na probabilidade de cada *option* h_n para N *options*.



Fonte – De autoria própria.

Para as funções $\psi_h(s)$ e $\pi_h(s, a)$, que dependem das opções, foi utilizado como entrada componentes do estado baseadas na tela do jogador, através do bloco de tela, ilustrado na figura 5. Além disso, foi utilizada uma única rede neural para cada opção para representar ambas as funções a partir do bloco de ações. A arquitetura utilizada para cada opção é ilustrada na figura 10.

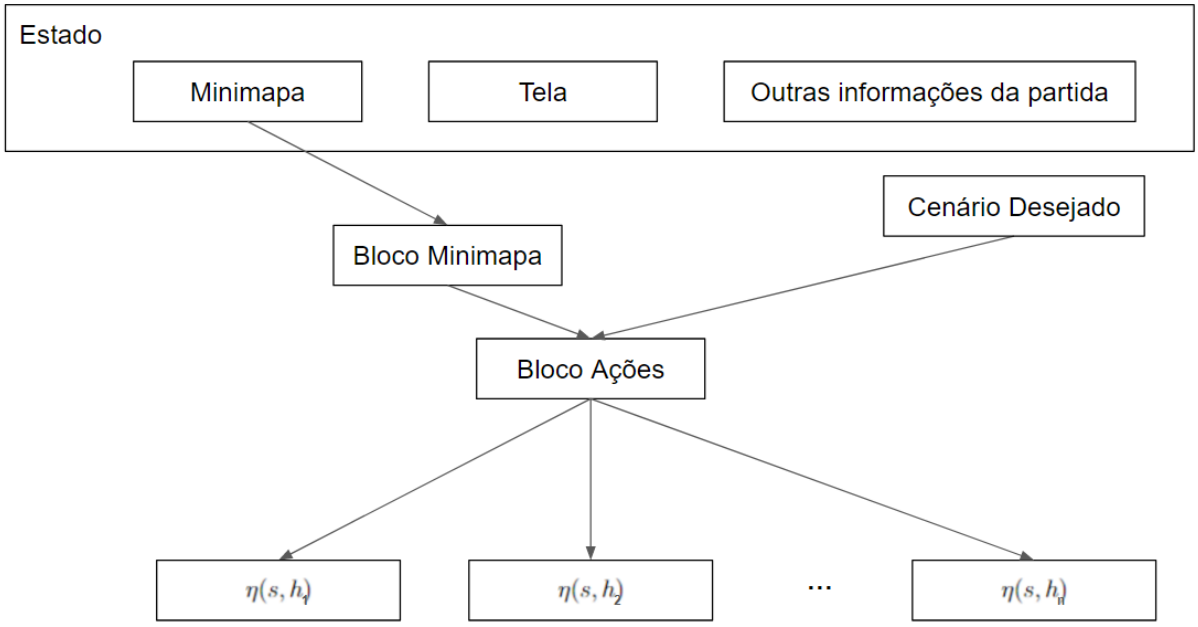
Figura 10 – Desenho da parte da arquitetura para cálculo de $\psi_h(s)$ e $\pi_h(s,a)$ para uma *option*. Como a definição de ação no *Starcraft II* é composta por várias componentes, o modelo calcula as probabilidades para cada componente.



Fonte – De autoria própria.

A arquitetura proposta foi projetada para abstrair os principais aspectos da modelagem do MLDDO, adequados ao cenário do jogo *Starcraft II*. No entanto, ela não é capaz de abstrair diferentes estratégias, como o AlphaStar faz utilizando a estatística z. Uma possível simplificação para incluir diferentes estratégias seria adicionar um vetor que indica qual estratégia deve ser utilizada, o que será discutido na sessão 4.4, sobre a separabilidade dos estados. A figura 11 ilustra a adição desse vetor à arquitetura da meta-política proposta anteriormente.

Figura 11 – Desenho da parte da arquitetura para cálculo de $\eta(s, h)$ considerando um vetor de entrada para diferentes estratégias.



Fonte – De autoria própria.

4.2 Treino e avaliação

Inicialmente, foi realizada a implementação do algoritmo de aprendizado por imitação baseado no MLDDO, descrito no artigo original (FOX *et al.*, 2017), com o objetivo de replicar os resultados independentemente do problema. A função de custo utilizada foi o negativo do logaritmo da verossimilhança, e o gradiente foi calculado utilizando a biblioteca PyTorch.

Para simplificar os cálculos, considerou-se que a probabilidade de ocorrer o mesmo conjunto de estados e ações $\langle s_t, a_t \rangle$ mais de uma vez é desprezível e, portanto, o processo foi assumido como determinístico, simplificando a função de transição para $\mathcal{T}(s_t, a_t, s_{t+1}) = 1$ $\forall$ $(s \in \mathcal{S}, a \in \mathcal{A}, t \in T)$.

4.3 Preparação de Dados para StarCraft II

Inicialmente, foram selecionados os cenários de teste baseado na implementação apresentada por (VINYALS *et al.*, 2017) e descritos na subseção 3.1. Esses cenários, conhecidos como *minigames*, foram utilizados para definir o escopo do estudo.

Para consolidar as informações do jogo em um modelo de processo de decisão Markoviano (MDP), foram utilizados os frameworks descritos anteriormente na metodologia. Dessa forma, foi possível definir os espaços, ações e recompensas a partir das aplicações.

Para facilitar o desenvolvimento, o Reaver ofereceu tanto agentes prontos para aprender os cenários quanto um framework para criação de modelos personalizados. Isso permitiu obter replays personalizados de forma indefinida, que foram utilizados como fonte para o aprendizado por imitação.

Entretanto, foram encontrados alguns problemas durante a obtenção dos resultados. O primeiro foi que o Reaver não estava mais recebendo atualizações, o que causou dificuldades na configuração do pacote. O segundo problema foi encontrar uma forma de salvar os replays para treinamentos futuros. Isso foi resolvido realizando alterações diretamente no pacote baseado nas orientações do *Pull Request* 29 no repositório do framework.

Com as adaptações realizadas, foi possível executar o treinamento para cada um dos cenários selecionados, coletando replays prontos para os treinamentos. Durante esses treinamentos, o minigame *BuildMarines* não foi capaz de aprender uma política adequada, o que era esperado com base nos resultados originais do Reaver. Portanto, esse minigame foi descartado dos cenários de teste. Para cada cenário, foi treinado um agente Reaver A2C diferente.

Para a leitura dos replays, foi utilizada a implementação de (NARHEN, 2018), que foi capaz de ler as informações dos replays de forma similar à apresentada pelo Reaver e pela API de desenvolvimento. Entretanto, foram necessários alguns ajustes para padronização das entradas e saídas.

Por fim, a arquitetura do MLDDO foi adaptada para a arquitetura descrita na Figura 9. Testes foram realizados para validar se essa arquitetura proposta seria capaz de servir como uma implementação generalista, capaz de aprender todos os cenários ao mesmo tempo, enquanto aproveita as estratégias de cada minigame em cenários diferentes.

Um problema encontrado durante essa implementação foi a limitação de memória das máquinas de processamento, o que foi contornado limitando o histórico para segmentos de horizonte com dez iterações contendo uma tupla $\langle s_t, a_t, s_{t+1} \rangle$.

Com as adaptações e ajustes realizados, foi possível preparar os dados para a utilização do modelo MLDDO no jogo StarCraft II e iniciar as etapas de treinamento e avaliação do desempenho do modelo.

4.4 Cenários e Separação dos Estados

Os minigames são cenários simplificados criados para testar o aprendizado de tarefas específicas em jogos. Embora eles possuam recompensas de ganho claras e demonstrem o aprendizado de diversas tarefas, cada um deles possui um único objetivo e uma única política ótima. Isso pode ser um problema ao utilizar o *framework* de *options*, pois ele tem como objetivo separar temporalmente diferentes estratégias. A arquitetura pode definir uma única política para cada espaço, o que não seria desejável para este tipo de estudo.

Para contornar essa situação, foram idealizados cenários não separáveis por espaço, como os quatro cenários descritos na seção anterior. Esses cenários foram desenvolvidos a partir do *minigame FindAndDefeatZergling*, pois ele é o único que apresenta observação parcial, gerenciamento de posição de câmera e, portanto, informações diferentes entre o estado de minimapa e tela do jogador, e possuem objetivos e recompensas diferentes. Dessa forma, a arquitetura não pode definir uma única política para cada espaço, o que permite testar a capacidade do modelo MLDDO de separar temporalmente diferentes estratégias. Além disso a observação parcial propicia gerenciamento de posição de câmera, o que aumenta a complexidade e a similaridade com cenários reais de jogos. Isso os torna mais adequados para avaliar a capacidade do modelo de lidar com estratégias múltiplas e situações de incerteza.

Os quatro cenários criados foram:

- *FindAndDefeatZergling*: o *minigame* original. A recompensa é dada como +1 cada vez que o jogador derrota um inimigo e -1 cada vez que uma unidade aliada é perdida;
- *Small*: o objetivo é minimizar a distância entre as unidades do jogador e as unidades inimigas. Para cada conjunto de unidade aliada e inimiga que estejam próximas, é dada uma recompensa de +1. A distância foi configurada para que as unidades do jogador possam ser atacadas e, portanto, é esperado que este cenário tenha duração mais curta;
- *Medium*: o objetivo é manter uma distância intermediária das unidades inimigas. Para cada conjunto de unidade aliada e inimiga que estejam dentro dos limites de distância, é dada uma recompensa de +1. A distância foi configurada de forma que as unidades inimigas possam ver as unidades aliadas, comecem a persegui-las, mas não consigam atacá-las;

- *Big*: o objetivo é manter uma distância suficientemente grande das unidades inimigas para que elas não possam vê-lo e, portanto, não tentem atacá-lo. Para cada conjunto de unidade aliada e inimiga que estejam fora dos limites de distância, é dada uma recompensa de +1.

Em todos os cenários, o jogador controla três soldados, que podem atacar a uma distância média e os inimigos espalhados pelo mapa são Zergnídeos, que só possuem ataque de curto alcance. Estes cenários foram criados para garantir que a arquitetura do modelo MLDDO não definisse uma única política para cada espaço, mas sim fosse capaz de aprender diferentes estratégias para cada cenário.

Os cenários foram selecionados e configurados de forma que fosse difícil para o modelo distinguir qual estratégia estaria sendo executada apenas com base na definição do estado. Dessa forma, o objetivo destes cenários adicionais foi testar a capacidade do modelo de aprender diferentes estratégias e separá-las temporalmente, o que é uma das principais vantagens do uso do *framework* de *options*.

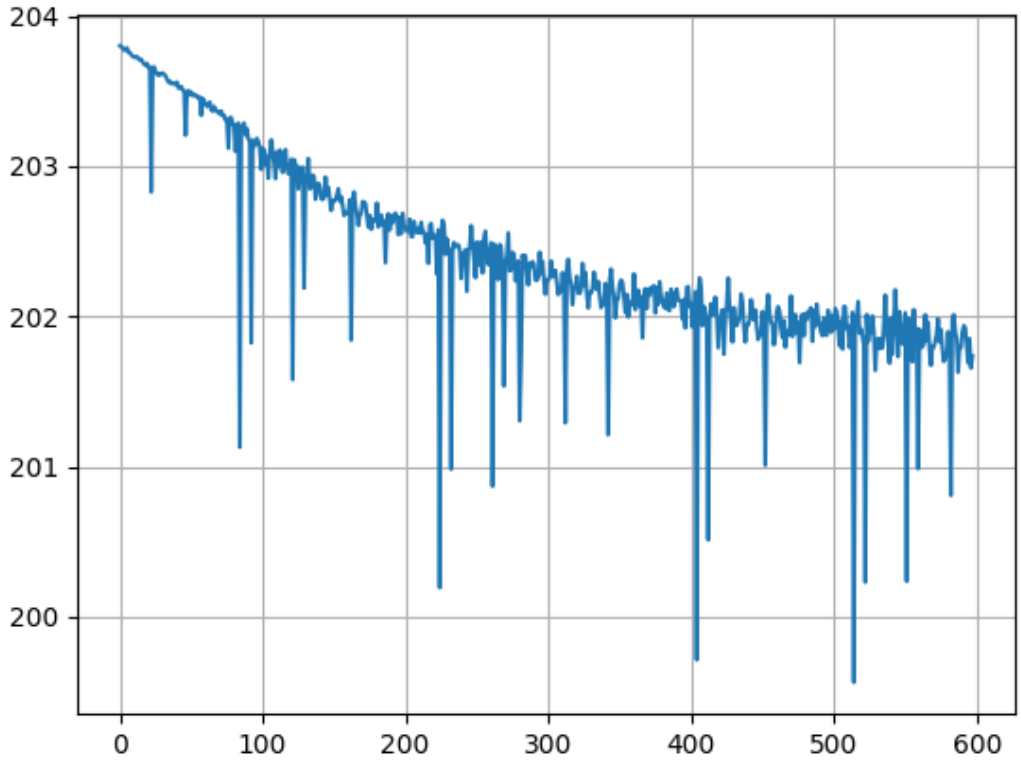
5 Resultados

Este capítulo apresenta os principais resultados obtidos a partir da pesquisa realizada, tendo em vista as arquiteturas propostas no capítulo anterior. A seção 5.1 apresenta os resultados obtidos ao aplicar a arquitetura proposta aos *minigames* propostos pelo SC2LE (StarCraft II Learning Environment), enquanto na seção 5.2, são apresentados os resultados obtidos para ambientes não separáveis pelo espaço, conforme descritos na seção 4.4.

5.1 Aplicação em minigames

Após treinamento dos agentes Reaver até convergência, foram utilizados os *replays* do agente Reaver para cada *minigame*, como insumo para treinamento de agentes MLDDO utilizando a arquitetura proposta. Diferente da implementação padrão do Reaver, foram habilitadas todas as ações disponíveis no jogo, o que pode gerar uma pequena diferença nos números de recompensas em comparação com o artigo original do Reaver. Em uma primeira implementação, foi gerado um agente MLDDO treinado apenas com cenário específico do *minigame FindAndDefeatZerglings*. A figura 12 demonstra a curva de aprendizado utilizando o negativo da verossimilhança como critério de otimização, seguindo as equações descritas no capítulo de metodologia.

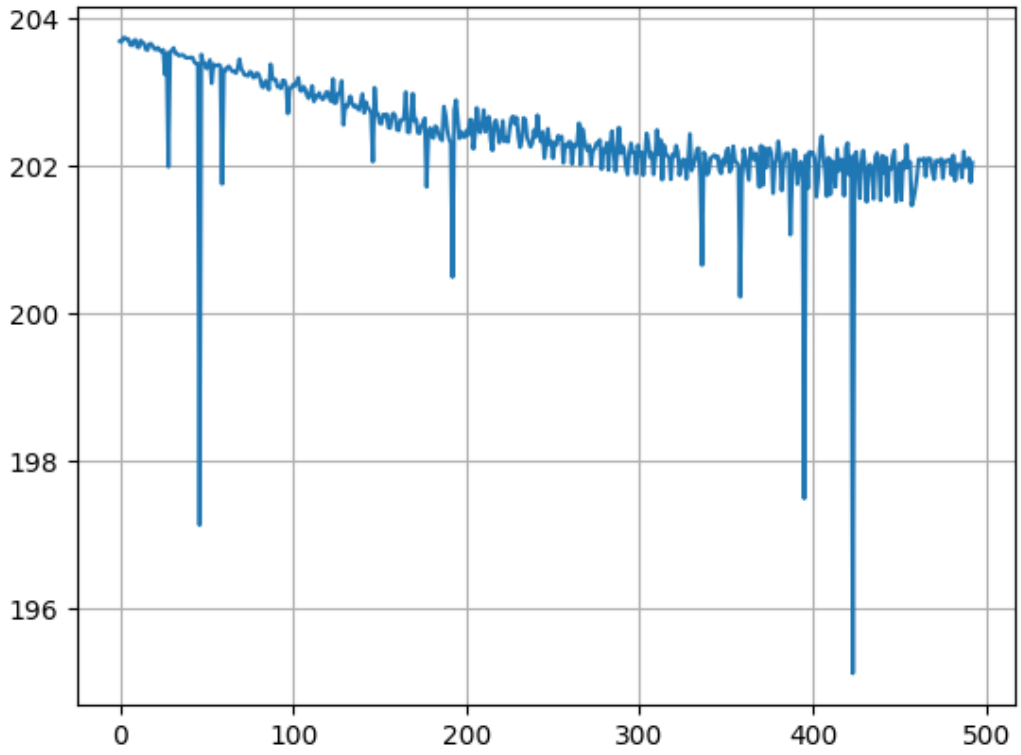
Figura 12 – Otimização de log-verossimilhança negativa como critério de otimização de arquitetura para implementação específica para o *FindAndDefeatZerglings* ao longo de iterações de treinamento.



Fonte – De autoria própria.

Em seguida, foi treinada uma implementação genérica, que buscava aprender uma única política que servisse para todos os *minigames*, utilizando os *replays* gerados pelo agente Reaver. A proposta aqui foi avaliar se seria possível criar uma arquitetura capaz de obter resultados satisfatórios em todos os *minigames*, além de avaliar o comportamento da utilização de *options* para cada *minigame*. A figura 13 apresenta a curva de aprendizado utilizando a verossimilhança como critério de otimização, considerando todos os *minigames* ao mesmo tempo.

Figura 13 – Otimização de log-verossimilhança negativa como critério de otimização de arquitetura para implementação generalista ao longo de iterações de treinamento.



Fonte – De autoria própria.

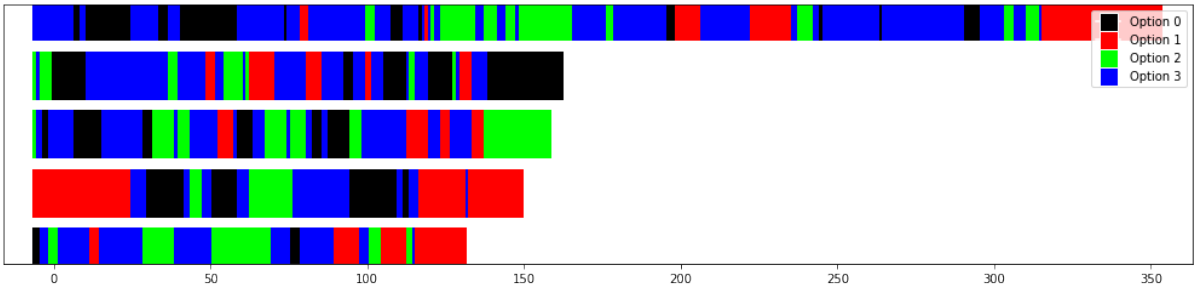
Foram então comparadas os desempenhos dos agentes Reaver utilizados no treinamento, com o agente MLDDO específico para o *FindAndDefeatZerglings* e com o MLDDO para generalista. A tabela 1 apresenta os resultados dos desempenhos de cada modelo utilizado nos experimentos. Adicionalmente, as figuras 14 a 19 demonstra alguns exemplos do comportamento das *options* para cada *minigame* no agente genérico.

Tabela 1 – Recompensas médias, desvios padrões entre parenteses e valores máximos e mínimos entre colchetes, obtidos pelas implementações treinadas pelo agente Reaver A2C em contraste com o desempenho do MLDDO generalista

Minigame	Agente Reaver específico	MLDDO generalista
FindAndDefeatZerglings	16,86 (5,19) [2; 25]	3,40 (3,30) [-3; 15]
MoveToBeacon	22,8 (1,85) [19; 28]	0,78 (0,89) [0; 4]
CollectMineralShards	39,86 (8,52) [19; 62]	16,12 (3,33) [8; 36]
DefeatRoaches	41,95 (27,54)[1; 220]	2,24 (10,39) [-9; 36]
DefeatZerglingsAndBanelings	37,54 (19,35) [11; 118]	30,95 (18,21) [1; 103]
CollectMineralsAndGas	1569,61 (608,25) [0; 2320]	22,77 (52,2) [0; 390]

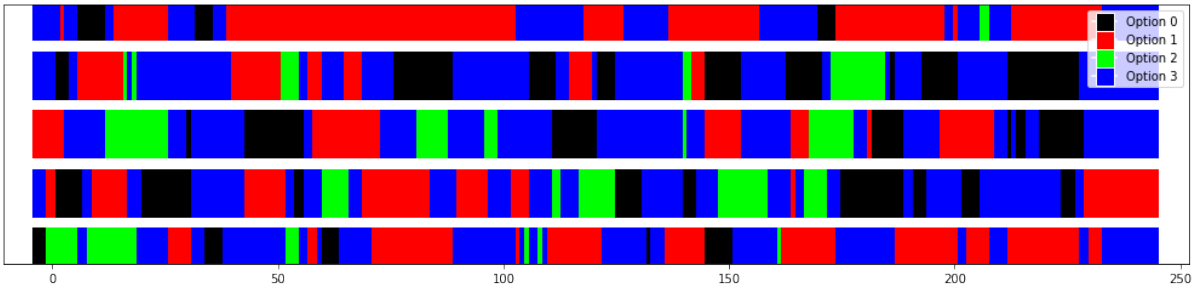
Fonte – De autoria própria.

Figura 14 – Comportamento das *options* selecionadas ao longo das iterações da implementação generalista no *minigame* *FindAndDefeatZerglings*. Cada linha representa uma partida diferente



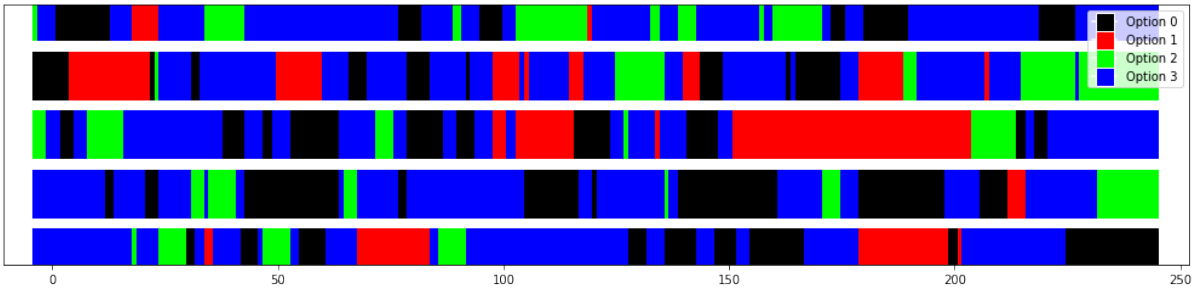
Fonte – De autoria própria.

Figura 15 – Comportamento das *options* selecionadas ao longo das iterações da implementação generalista no *minigame* *MoveToBeacon*. Cada linha representa uma partida diferente



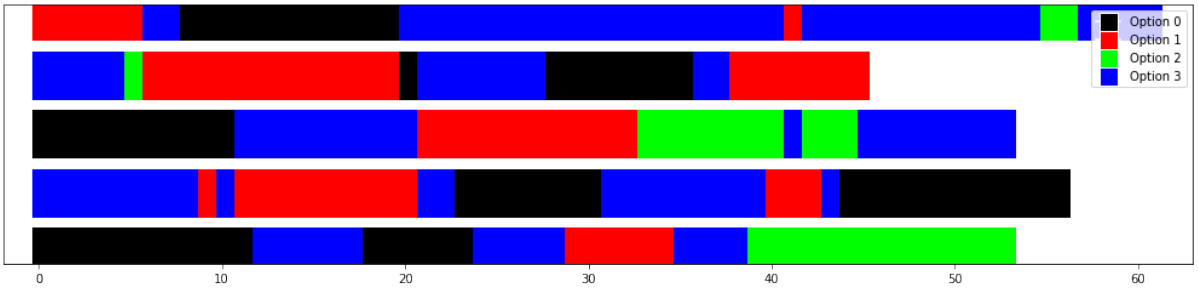
Fonte – De autoria própria.

Figura 16 – Comportamento das *options* selecionadas ao longo das iterações da implementação generalista noo *minigame CollectMineralShards*. Cada linha representa uma partida diferente



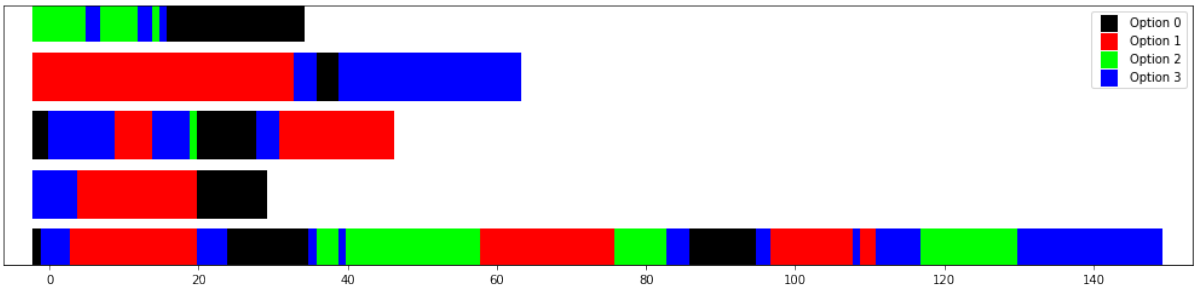
Fonte – De autoria própria.

Figura 17 – Comportamento das *options* selecionadas ao longo das iterações da implementação generalista no *minigame DefeatRoaches*. Cada linha representa uma partida diferente



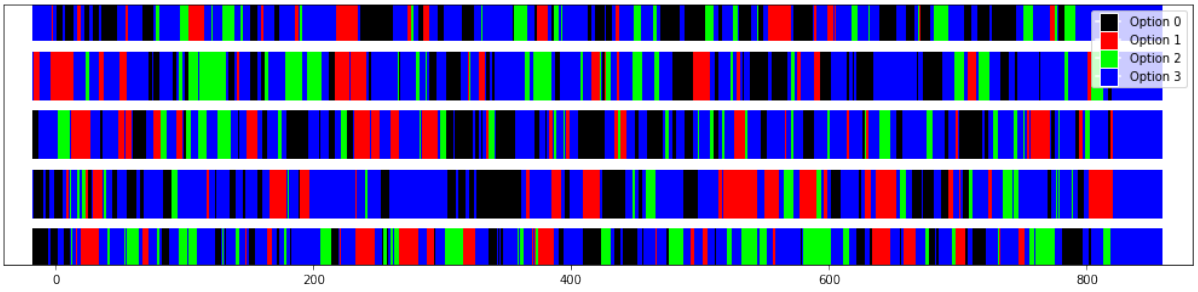
Fonte – De autoria própria.

Figura 18 – Comportamento das *options* selecionadas ao longo das iterações da implementação generalista no *minigame DefeatZerglingsAndBanelings*. Cada linha representa uma partida diferente



Fonte – De autoria própria.

Figura 19 – Comportamento das *options* selecionadas ao longo das iterações da implementação generalista no *minigame* *CollectMineralsAndGas*. Cada linha representa uma partida diferente



Fonte – De autoria própria.

É possível observar que a arquitetura MLDDO apresentou resultados próximos para os *minigames* *CollectMineralShards*e *DefeatZerglingsAndBanelings*, enquanto não foi capaz de abstrair os comportamentos para os *minigames* *FindAndDefeatZerglings*, *MoveToBeacon*, *DefeatRoaches* e *CollectMineralsAndGas*. Nenhum dos cenários parece ter preferência por utilizar uma *option* específica.

A tabela 2 mostra os resultados do modelo MLDDO específico para o *FindAndDefeatZerglings*, assim como resultados individuais de cada *option* e a figura 20 demonstra alguns exemplos do comportamento das *options* para o mesmo modelo.

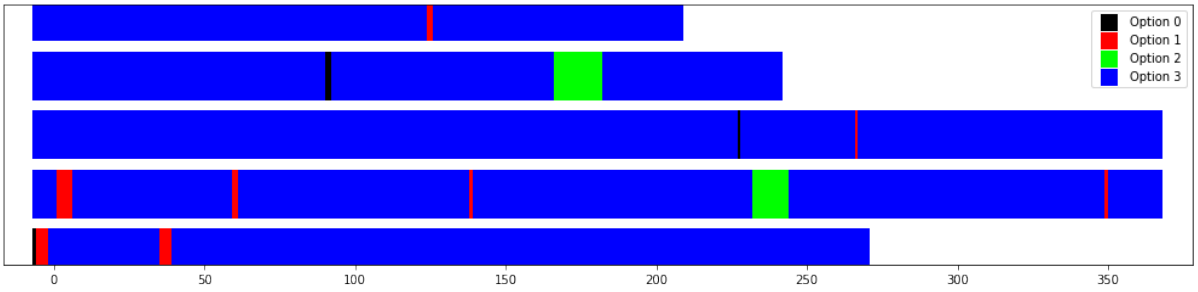
Nela é possível observar que as recompensas estão muito mais próximas que o modelo generalista, mas ainda não obtiveram a mesma recompensa que o modelo original. Outro fator que chama atenção é que a arquitetura completa possui uma recompensa média pior que a maioria das options sozinhas, o que indica que uma melhoria na meta-política seria necessária para melhorar o desempenho do modelo como um todo.

Tabela 2 – Recompensas médias, desvios padrões entre parenteses e valores máximos e mínimos entre colchetes, obtidos pelas implementações treinadas pelo agente Reaver A2C em contraste com o desempenho do MLDDO generalista

Política	Recompensa
Arquitetura completa	6,29 (4,55) [-3; 21]
Option 0	10,47 (6,25) [-3; 26]
Option 1	8,41 (3,85) [-2; 20]
Option 2	4,97 (4,58) [-3; 20]
Option 3	6,31 (4,56) [-3; 19]

Fonte – De autoria própria.

Figura 20 – Comportamento das *options* selecionadas ao longo das iterações para a implementação MLDDO criada específica para o *minigame FindAndDefeatZerglings*. Cada linha representa uma partida diferente



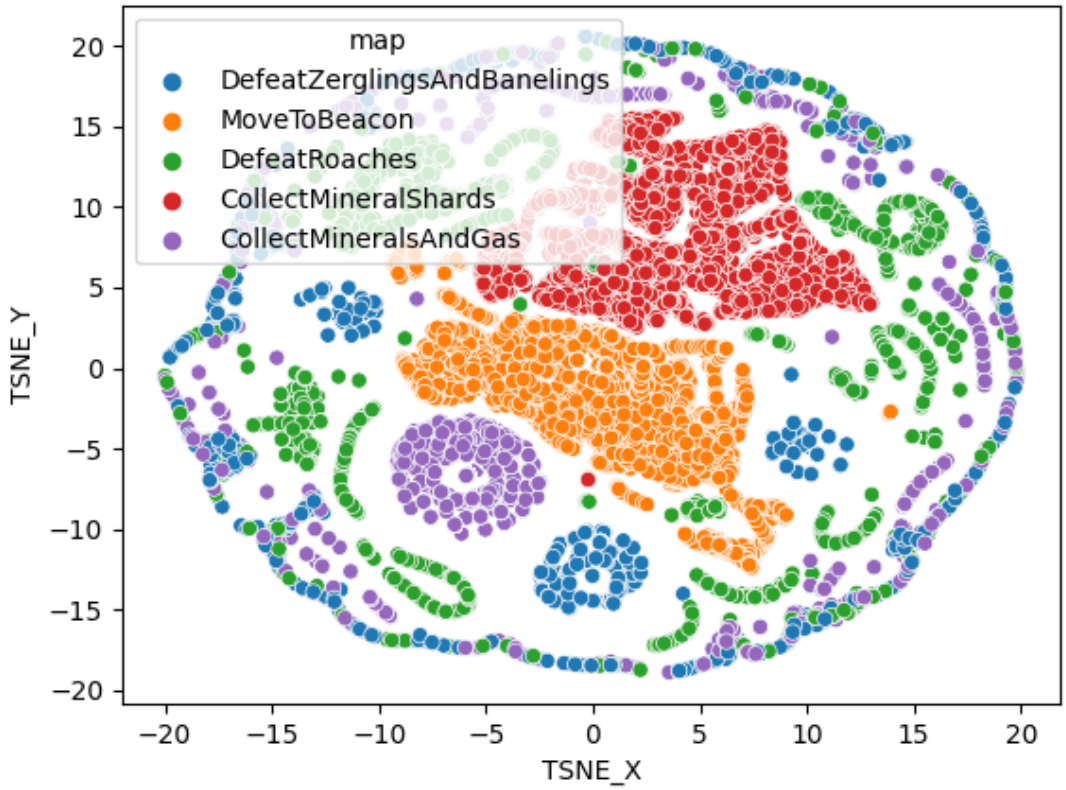
Fonte – De autoria própria.

Na figura 20 fica evidente uma preferência pela *option* 3. o que demonstra que, ao treinar o MLDDO especificamente para o *minigame FindAndDefeatZerglings*, o fator hierárquico temporal acabou não trazendo benefícios.

Essas informações demonstram que a arquitetura foi capaz de aprender políticas que possuem certa capacidade de aprender comportamentos utilizando observações de *minigames*, mas o desempenho possui oportunidades para melhoria. Um questionamento levantado foi o quanto esses resultados poderiam ter sido influenciados pela definição de estados. Cada *minigame* possui peculiaridades, como unidades diferentes, ou elementos na tela ou minimapa que o distinguem dos demais, de forma que apenas ao observar um estado, é possível distinguir qual a estratégia a ser seguida. Um possível impacto disso, seria uma possibilidade da arquitetura sozinha ser capaz de criar uma política única que resolveria todos os problemas.

Em uma primeira análise, foi realizado um levantamento da distribuições de ações realizadas por cada agente Reaver, a fim de observar quão distintas elas seriam por ambientes. Para analisar a similaridade das ações, foi utilizado um *T-distributed Stochastic Neighbor Embedding* (t-SNE), utilizando PCA para decompor os vetores de entrada. Essa abstração permitiu realizar uma visualização da separabilidade das ações, que eram compostas por vetores com várias componentes.

Figura 21 – Análise de separabilidade das ações por estado nos agentes reaver.



Fonte – De autoria própria.

A figura 21 demonstra a separabilidade das ações por ambientes. Como pode ser observado nela, há agrupamentos, que separam a distribuição das ações. As ações que possuem comportamento mais similares são as de combate, com *DefeatZerlingsAndBanelings* e *DefeatRoaches*, em que as ações concentram-se em avançar para os inimigos e atacar. As similaridades das distribuições das ações podem ser interpretadas no gráfico por agrupamentos com sobreposições.

Embora tanto *MoveToBeacon* quanto *CollectMineralShards* tenham comportamentos de mover as unidades para ponto de destino, as distribuições das ações foram bastante separadas. Acredita-se que o motivo seja devido à diferença no caso de comportamento de agente simples para o comportamento multi-agente.

Ainda sobre essa figura, uma dúvida que surgiu seria do motivo da sobreposição de muitos pontos do *CollectMineralsAndGas* com *DefeatZerglingsAndBanelings* e *DefeatRoaches*. Como é o único agente que realmente efetiva a ação de minerar, era esperado

que sua distribuição fosse separada dos demais. É possível que o agrupamento de cor roxa na imagem traga esta representação, enquanto os demais pontos com sobreposição representem observações onde ações de movimento foram majoritária.

5.2 Aplicação em cenários não separáveis por espaço

Para avaliar os impactos do fato dos estados conseguirem distinguir o *minigame*, foram então feitos novos treinos nos cenários descritos na seção 4.4 deste trabalho. Como primeira etapa, foram treinados agentes Reaver A2C específicos para cada um deste cenários. A tabela 3 demonstra as recompensas obtidas por cada agente por cada agente em cada um dos quatro cenários. AgentBig representa o agente treinado para o cenário Big, enquanto o AgentFindAndDefeatZerglings é o agente treinado apenas para o cenário *FindAndDefeatZerglings* e assim por diante.

Tabela 3 – Recompensas médias obtidas pelas implementações treinadas pelo agente Reaver A2C específicos para cada ambiente.

Agente Reaver	Cenário *Big*	Cenário *Medium*	Cenário *Small*	Cenário *FindAnd-Defeat-Zerglings*
AgentBig	187,05	4,93	11,22	5,20
AgentMedium	84,45	14,83	51,09	12,00
AgentSmall	54,16	6,23	32,24	4,21
AgentFindAndDefeatZerlings	75,11	19,24	14,64	16,48

Fonte – De autoria própria.

Todos estes agentes foram treinados utilizado o A2C do reaver para maximizar as respectivas recompensas. Alguns dados da tabela 3 se destacam, como por exemplo, o AgentBig e o AgentFindAndDefeatZerglings foram os agentes com maior recompensa nos respectivos cenários, *Big* e *FindAndDefeatZerglings* respectivamente, o que era esperado.

Entretanto, os AgentSmall e AgentMedium não foram os agentes que conseguiram maximizar os respectivos cenários. Uma possível justificativa para o AgentSmall é que, como o objetivo dele é sempre se manter a uma distância que os inimigos estariam possíveis de ataque, a política aprendida foi a de se aproximar do oponente, e com resultado disso, ele acaba morrendo mais rápido.

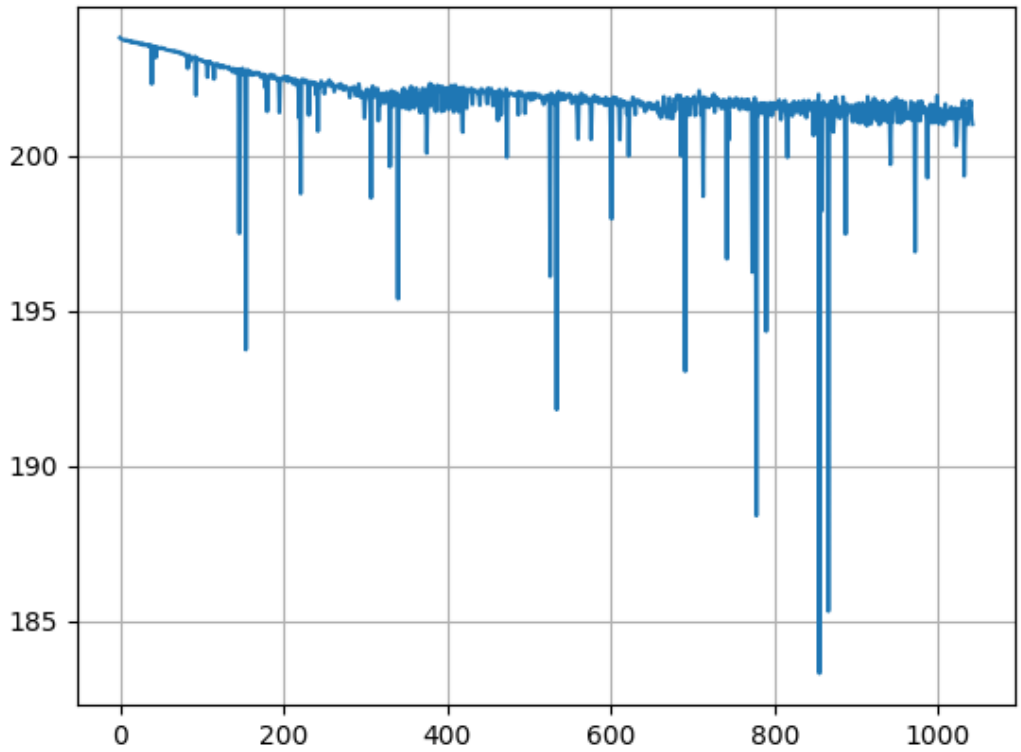
Para o AgentMedium, acredita-se que o fato de ter como objetivo manter-se sempre a uma distância que seria perseguido por inimigos, seria mais fácil para oponentes

encurralar os agentes, resultando em uma partida que sobreviveria por menos tempo que o AgentFindAndDefeatZerglings.

Independente destes pontos que destoaram-se do esperado, a proposta da criação dos cenários seria para garantir que haveriam quatro políticas diferentes, mas com espaços não diferenciáveis. O espaço original dos quatro cenários é exatamente o mesmo para todos os cenários e os resultados da tabela 3 demonstram que a proposta da elaboração dos cenários se mantém válida.

Um novo agente parametrizado com quatro *options* foi treinado utilizando a arquitetura proposta com observações dos agentes anteriores utilizando um parâmetro que definiria qual cenário do treinamento. A figura 22 demonstra a curva de aprendizado para o agente e a tabela 4 demonstra os resultados obtidos para cada parâmetro utilizado.

Figura 22 – Otimização de log-verossimilhança negativa como critério de otimização de arquitetura para implementação em cenários não separáveis por espaço ao longo de iterações de treinamento.



Fonte – De autoria própria.

Tabela 4 – Recompensas médias obtidas pelo agente generalista treinado para os cenários não separáveis por espaço. As quatro primeiras linhas demonstram o comportamento quando uma option é fixada, enquanto as quatro últimas linhas demonstram o comportamento quando o vetor de entrada define a estratégia desejada.

Implementação	Cenário *Big*	Cenário *Medium*	Cenário *Small*	Cenário *FindAndDefeatZerglings*
Apenas Option 0	48,03 (37,67) [0; 250]	7,26 (8,16) [0; 61]	5,71 (6,12) [0; 69]	3,56 (3,91) [-3; 20]
Apenas Option 1	80,38 (50,93) [0; 293]	9.8 (8,58) [0; 50]	6,74 (5.8) [0; 29]	5,92 (4,13) [-3; 19]
Apenas Option 2	64,56 (43,48) [0; 235]	8,95 (8,23) [0; 42]	7,04 (6,49) [0; 39]	5,14 (4,01) [-3; 18]
Apenas Option 3	112,29 (61,29) [9; 409]	14,16 (10.8) [0; 80]	8,67 (6.2) [0; 35]	8,54 (3.9) [-2; 19]
AgentBigMLDDO	73,71 (48,66) [0; 328]	10,83 (9,74) [0; 66]	7,55 (6,48) [0; 38]	6,45 (4,48) [-3; 18]
AgentMediumMLDDO	73,25 (45,23) [1; 259]	10,81 (9,73) [0; 53]	7,65 (6,22) [0; 33]	6.4 (4,17) [-3; 19]
AgentSmallMLDDO	72,75 (44.4) [0; 312]	10,14 (9.5) [0; 84]	7,58 (6,49) [0; 41]	6,17 (4.2) [-3; 18]
AgentFindAndDefeatZerglingsMLDDO	75,86 (47.4) [0; 252]	10,79 (9,53) [0; 62]	7,82 (6,84) [0; 36]	6,47 (4,34) [-3; 18]

Fonte – De autoria própria.

Observando os dados da tabela 4, a primeira análise realizada foi que todos os agentes MLDDO tiveram as recompensas muito parecidas, obtendo valores medianos para todos os cenário quando comparados com os dados dos agentes Reaver, descritos na tabela 3. Nenhum agente MLDDO obteve recompensa no cenário *Big* equivalente ao AgentBig, mas para este cenário, todos os outros números foram condizentes com os obtidos pelos demais agentes reaver.

Ao analisar apenas as *options*, as recompensas obtidas possuem uma variação maior, refletindo uma diferença de estratégias. As *options* 1 e 3 foram as que obteveram maior recompensas médias no cenário *Big*, superando inclusive todos os agentes MLDDO.

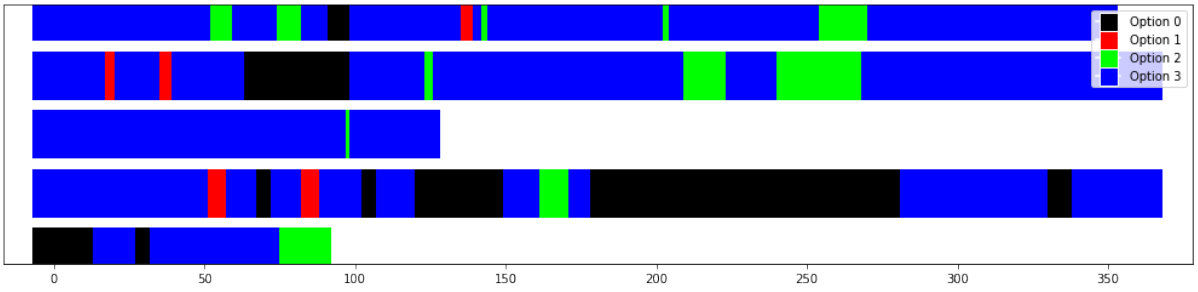
Embora a *option* 0 acabe tendo um comportamento que se destaca por possuir melhor recompensa média em nenhum cenário, não é possível afirmar que a *option* como um todo é ruim, pois pode ser que resulte em um comportamento momentâneo, como aproximar-se de oponentes, por exemplo, que pode não possuir recompensa sozinho, mas pode auxiliar o processo como um todo. Qualitativamente, pode-se observar que para o

cenário *Big* representa a ação de desviar ou andar até os inimigos. Recompensas muito altas neste cenário, como valores acima de 150, dependem do agente sobreviver por muito tempo nesta situação, o que foi observado pelo AgentBig, mas não por nenhum agente com aprendizado por imitação ou por nenhuma *option* específica.

Para o cenário *Medium*, todas as recompensas parecem ter ficado dentro da faixa de valores dos agentes Reaver de maior valor no mesmo cenário. No cenário *Small*, por sua vez, todos os agentes treinados nesta implementação tiveram resultados similares aos piores resultados *Small* dos agentes Reaver. Acredita-se que o principal motivo do baixo desempenho no cenário *Small* se dê pela dificuldade de manter-se vivo por bastante tempo em uma distância que os inimigos podem atacar.

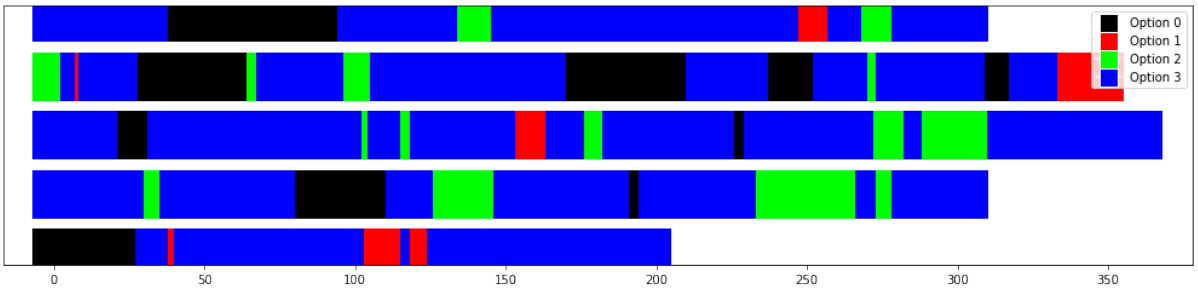
As figuras 23 a 26 demonstram a mudança de *options* ao longo de iterações exemplo em cenários específicos.

Figura 23 – Comportamento das *options* selecionadas ao longo das iterações para a implementação MLDDO criada para cenários não separáveis por espaço quando o indicado para tentar simular o comportamento do *AgentFindAndDefeatZerglings*. Cada linha representa uma partida diferente



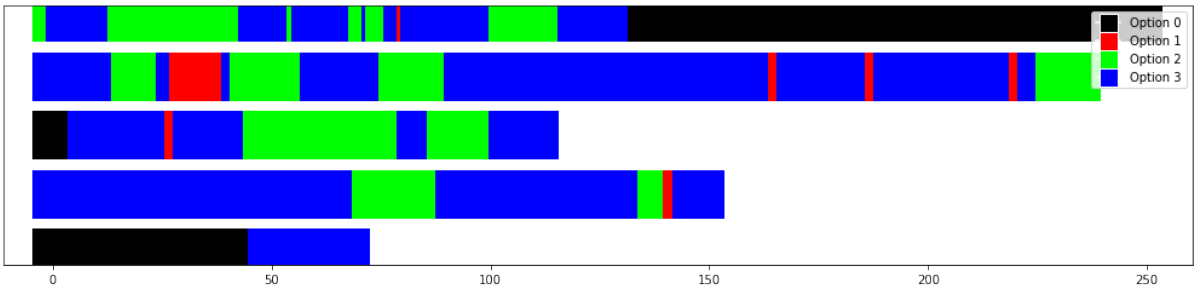
Fonte – De autoria própria.

Figura 24 – Comportamento das *options* selecionadas ao longo das iterações para a implementação MLDDO criada para cenários não separáveis por espaço quando o indicado para tentar simular o comportamento do *AgentBig*. Cada linha representa uma partida diferente



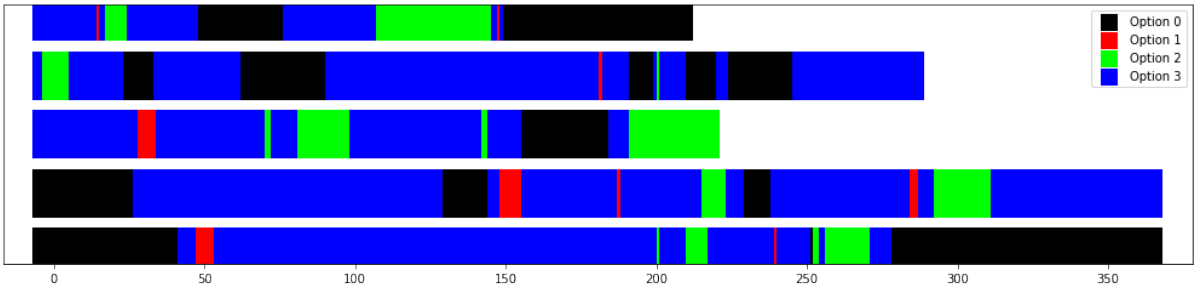
Fonte – De autoria própria.

Figura 25 – Comportamento das *options* selecionadas ao longo das iterações para a implementação MLDDO criada para cenários não separáveis por espaço quando o indicado para tentar simular o comportamento do *AgentMedium*. Cada linha representa uma partida diferente



Fonte – De autoria própria.

Figura 26 – Comportamento das *options* selecionadas ao longo das iterações para a implementação MLDDO criada para cenários não separáveis por espaço quando o indicado para tentar simular o comportamento do *AgentSmall*. Cada linha representa uma partida diferente



Fonte – De autoria própria.

Pode-se notar uma sutil diferença na prioridade das *options*, como por exemplo o cenário tentando replicar o *AgentMedium* demonstra uma prioridade maior pela *option* 2 que os demais. Entretanto, acredita-se que de maneira geral, a meta-política acaba direcionando para cenários muito parecidos, o que pode ser comprovado pela proximidade das recompensa dos agentes da tabela 4.

De maneira geral, é possível afirmar que a arquitetura implementada foi capaz de obter comportamentos diferentes para cada *option*, mas o comportamento da meta-política acabou direcionando todos os agentes MLDDO para um comportamento que se assemelha em todas as aplicações.

No artigo original de Fox *et al.* (2017), é proposta após a descoberta de *options* a utilização de técnicas de aprendizado por reforço para otimizar a meta-política, acredita-se que essa estratégia poderia ser muito benéfica para os experimentos aqui realizados.

6 Conclusão e considerações finais

Este trabalho propôs e validou uma arquitetura que utiliza o conceito de *options*, por meio de *option discovery* utilizando o aprendizado por imitação para abstrair políticas *intra-options* comuns a vários agentes, enquanto os agentes aprendem meta-políticas específicas para resolver o problema em questão. A validação foi realizada tanto em *minigames* propostos por (VINYALS *et al.*, 2017), quanto em cenários criados especificamente para este trabalho.

Conforme o objetivo inicial deste trabalho, que almejava uma performance próxima, mas não superior ao agente a ser imitado, os resultados mostraram que a arquitetura proposta foi capaz de obter recompensas próximas aos obtidos pelo agente padrão Reaver em alguns os *minigames*. Além disso, foi possível observar o comportamento das *options* para cada *minigame* no agente genérico, o que permitiu uma melhor compreensão da arquitetura proposta.

Outro objetivo deste trabalho foi analisar o impacto da separabilidade de estados no MLDDO, comparando resultados do aprendizado para diferentes cenários separáveis pelo espaço e com o aprendizado de uma rede generalista que busca aprender diferentes objetivos em cenários indistinguíveis pelo estado. Os resultados mostraram que a arquitetura proposta foi capaz de obter resultados satisfatórios mesmo em cenários com estados não separáveis pelo espaço.

Em geral, os resultados obtidos neste trabalho sugerem que a arquitetura proposta é uma abordagem eficaz para auxiliar a resolver problemas complexos, como os encontrados em *Starcraft II*. A utilização de *option discovery* permitiu a abstração de políticas comuns e o aprendizado de políticas específicas, o que permitiu ao agente se adaptar a diferentes tarefas e ambientes. Além disso, a análise da separabilidade de estados auxiliou a contribuição para usos do *framework* de *options* e sua natureza temporal e hierárquica para o aprendizado.

É importante notar que, embora os resultados obtidos sejam promissores, ainda há espaço para melhorias e extensões do trabalho. Por exemplo, seria interessante avaliar o desempenho da arquitetura proposta em outros jogos além de *Starcraft II* e também testar a generalização da arquitetura para outros cenários e tarefas além dos *minigames* utilizados neste trabalho. Além disso, futuras pesquisas poderiam explorar outras abordagens de

option discovery, como o aprendizado de *options* de forma autônoma, sem a necessidade de insumos humanos.

Outra possível extensão seria o uso de aprendizado por reforço para otimizar as meta-políticas, permitindo ao agente aprender ainda mais rapidamente e de forma mais eficiente. Isso poderia ajudar a melhorar ainda mais o desempenho do agente e sua capacidade de adaptação a diferentes tarefas e ambientes. Além disso, poderia ser interessante explorar a possibilidade de usar a arquitetura proposta em conjunto com outros algoritmos de aprendizado por reforço, para melhorar ainda mais o desempenho do agente.

Em resumo, este trabalho propôs e validou uma arquitetura que utiliza o conceito de *options* abstrair políticas comuns e aprender políticas específicas para resolver problemas complexos. Os resultados obtidos mostraram que a arquitetura é capaz de obter resultados satisfatórios em *Starcraft II* e apresentou um bom desempenho mesmo em cenários com estados não separáveis pelo espaço. Este trabalho contribui para o campo da aprendizado de máquina e do aprendizado por reforço ao propor uma abordagem eficaz para resolver problemas complexos.

Os códigos implementados e utilizados neste trabalho estão disponíveis no link GitHub-SC2-MLDDO.

Referências

ARULKUMARAN, K.; CULLY, A.; TOGELIUS, J. Alphastar: An evolutionary computation perspective. In: . [s.n.], 2019. p. 314–315. Cited By 0. Disponível em: ⟨https://www.scopus.com/inward/record.uri?eid=2-s2.0-85070622576&doi=10.1145%2f3319619.3321894&partnerID=40&md5=57c2668992e08da1e7d6a459631cf4aa⟩. Citado na página 37.

BACON, P.-L.; HARB, J.; PRECUP, D. The option-critic architecture. In: *Thirty-First AAAI Conference on Artificial Intelligence*. [S.l.: s.n.], 2017. Citado na página 34.

BONATE, P. L. A brief introduction to monte carlo simulation. *Clinical pharmacokinetics*, Springer, v. 40, n. 1, p. 15–22, 2001. Citado na página 26.

BYEONGUK, M. *MinervaSc2*. [S.l.]: GitHub, 2019. ⟨https://github.com/phraust1612/MinervaSc2⟩. Citado na página 37.

CHO, H.; PARK, H.; KIM, C.-Y.; KIM, K.-J. Investigation of the effect of "fog of war"in the prediction of starcraft strategy using machine learning. *Computers in Entertainment*, v. 14, n. 1, 2017. Cited By 1. Disponível em: ⟨https://www.scopus.com/inward/record.uri?eid=2-s2.0-85012181567&doi=10.1145%2f2735384&partnerID=40&md5=252e2b8651cd297953e8d52983f1333c⟩. Citado na página 19.

CRITES, R. H.; BARTO, A. G. An actor/critic algorithm that is equivalent to q-learning. In: *Advances in Neural Information Processing Systems*. [S.l.: s.n.], 1995. p. 401–408. Citado na página 29.

FOX, R.; KRISHNAN, S.; STOICA, I.; GOLDBERG, K. *Multi-Level Discovery of Deep Options*. 2017. Citado 5 vezes nas páginas 23, 34, 35, 52 e 69.

GAJUREL, A.; LOUIS, S. Multiobjective multi unit-type neuroevolution for micro in rts games. In: . [s.n.], 2019. p. 169–170. Cited By 0. Disponível em: ⟨https://www.scopus.com/inward/record.uri?eid=2-s2.0-85070585629&doi=10.1145%2f3319619.3322064&partnerID=40&md5=a7cb7cd0f70162fe10b63c08dfe018df⟩. Citado na página 37.

HAMIDI, M.; TADEPALLI, P.; GOETSCHALCKX, R.; FERN, A. Active imitation learning of hierarchical policies. In: *IJCAI*. [S.l.: s.n.], 2015. p. 3554–3560. Citado 2 vezes nas páginas 22 e 31.

HOLCOMB, S. D.; PORTER, W. K.; AULT, S. V.; MAO, G.; WANG, J. Overview on deepmind and its alphago zero ai. In: *Proceedings of the 2018 International Conference on Big Data and Education*. New York, NY, USA: ACM, 2018. (ICBDE '18), p. 67–71. ISBN 978-1-4503-6358-7. Disponível em: ⟨http://doi.acm.org/10.1145/3206157.3206174⟩. Citado na página 18.

LAKSHMINARAYANAN, A. S.; KRISHNAMURTHY, R.; KUMAR, P.; RAVINDRAN, B. Option discovery in hierarchical reinforcement learning using spatio-temporal clustering. *arXiv preprint arXiv:1605.05359*, 2016. Citado na página 22.

MACHADO, M. C.; BELLEMARE, M. G.; BOWLING, M. H. A laplacian framework for option discovery in reinforcement learning. *CoRR*, abs/1703.00956, 2017. Disponível em: ⟨http://arxiv.org/abs/1703.00956⟩. Citado na página 34.

MITIć, M.; MILJKOVIć, Z.; BABIć, B. Empirical control system development for intelligent mobile robot based on the elements of the reinforcement machine learning and axiomatic design theory. *FME Transactions*, v. 39, p. 1–8, 03 2011. Citado na página 28.

MNIH, V.; KAVUKCUOGLU, K.; SILVER, D.; GRAVES, A.; ANTONOGLOU, I.; WIERSTRA, D.; RIEDMILLER, M. A. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. Disponível em: ⟨http://arxiv.org/abs/1312.5602⟩. Citado 2 vezes nas páginas 21 e 28.

NARHEN. *pysc2-replay*. [S.l.]: GitHub, 2018. ⟨https://github.com/narhen/pysc2-replay⟩. Citado na página 53.

POKEMON O Filme: Mewtwo contra-ataca. [S.l.]: United Kingdom: Universal Studios., 1998. Citado na página 6.

PUTERMAN, M. L. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. 1st. ed. USA: John Wiley Sons, Inc., 1994. ISBN 0471619779. Citado na página 21.

RAMESH, R.; TOMAR, M.; RAVINDRAN, B. Successor options: An option discovery framework for reinforcement learning. *CoRR*, abs/1905.05731, 2019. Disponível em: ⟨http://arxiv.org/abs/1905.05731⟩. Citado na página 34.

RING, R. *Reaver: Modular Deep Reinforcement Learning Framework*. [S.l.]: GitHub, 2018. ⟨https://github.com/inoryy/reaver⟩. Citado na página 37.

SCHMITT, J.; KOSTLER, H. A multi-objective genetic algorithm for simulating optimal fights in starcraft ii. In: . [s.n.], 2017. Cited By 0. Disponível em: ⟨https://www.scopus.com/inward/record.uri?eid=2-s2.0-85015385667&doi=10.1109%2fCIG.2016.7860422&partnerID=40&md5=af77c1c2bffcee93ff23cc5a579c58e3⟩. Citado 2 vezes nas páginas 18 e 37.

SILVER, D.; HUANG, A.; MADDISON, C. J.; GUEZ, A.; SIFRE, L.; DRIESSCHE, G. V. D.; SCHRITTWIESER, J.; ANTONOGLOU, I.; PANNEERSHELVAM, V.; LANCTOT, M. *et al.* Mastering the game of go with deep neural networks and tree search. *nature*, Nature Publishing Group, v. 529, n. 7587, p. 484–489, 2016. Citado na página 18.

SILVER, D.; HUBERT, T.; SCHRITTWIESER, J.; ANTONOGLOU, I.; LAI, M.; GUEZ, A.; LANCTOT, M.; SIFRE, L.; KUMARAN, D.; GRAEPEL, T.; LILLICRAP, T.; SIMONYAN, K.; HASSABIS, D. *Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm*. 2017. Citado na página 18.

SILVER, D.; SCHRITTWIESER, J.; SIMONYAN, K.; ANTONOGLOU, I.; HUANG, A.; GUEZ, A.; HUBERT, T.; BAKER, L.; LAI, M.; BOLTON, A. *et al.* Mastering the game of go without human knowledge. *nature*, Nature Publishing Group, v. 550, n. 7676, p. 354–359, 2017. Citado na página 18.

SUTTON, R. S.; BARTO, A. G. *Reinforcement Learning: An Introduction*. MIT Press, 1998. Disponível em: ⟨http://www.cs.ualberta.ca/~sutton/book/the-book.html⟩. Citado 2 vezes nas páginas 21 e 27.

SUTTON, R. S.; PRECUP, D.; SINGH, S. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, Elsevier, v. 112, n. 1-2, p. 181–211, 1999. Citado 6 vezes nas páginas 20, 21, 22, 26, 32 e 34.

TAKINO, H.; HOKI, K. Human-like build-order management in starcraft to win against specific opponent's strategies. In: . [s.n.], 2015. p. 97–102. Cited By 0. Disponível em: ⟨https://www.scopus.com/inward/record.uri?eid=2-s2.0-84962732307&doi=10.1109%2fACIT-CSI.2015.25&partnerID=40&md5=1c2e13eb1cc3f1dddd4a7cfc776001d8⟩. Citado na página 37.

TANG, Z.; ZHAO, D.; ZHU, Y.; GUO, P. Reinforcement learning for build-order production in starcraft ii. In: . [s.n.], 2018. p. 153–158. Cited By 1. Disponível em: ⟨https://www.scopus.com/inward/record.uri?eid=2-s2.0-85052369007&doi=10.1109%2fICIST.2018.8426160&partnerID=40&md5=7df22ea798ec2977f741492665f2ea9f⟩. Citado na página 37.

TESAURO, G. Temporal difference learning and td-gammon. *Communications of the ACM*, v. 38, n. 3, p. 58–68, 1995. Citado na página 47.

VINYALS, O.; BABUSCHKIN, I.; CHUNG, J.; MATHIEU, M.; JADERBERG, M.; CZARNECKI, W.; DUDZIK, A.; HUANG, A.; GEORGIEV, P.; POWELL, R.; EWALDS, T.; HORGAN, D.; KROISS, M.; DANIHELKA, I.; AGAPIOU, J.; OH, J.; DALIBARD, V.; CHOI, D.; SIFRE, L.; SULSKY, Y.; VEZHNEVETS, S.; MOLLOY, J.; CAI, T.; BUDDEN, D.; PAINE, T.; GULCEHRE, C.; WANG, Z.; PFAFF, T.; POHLEN, T.; YOGATAMA, D.; COHEN, J.; MCKINNEY, K.; SMITH, O.; SCHAUL, T.; LILLICRAP, T.; APPS, C.; KAVUKCUOGLU, K.; HASSABIS, D.; SILVER, D. *AlphaStar: Mastering the Real-Time Strategy Game StarCraft II*. 2019. ⟨https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/⟩. Citado na página 19.

VINYALS, O.; BABUSCHKIN, I.; CZARNECKI, W. M.; MATHIEU, M.; DUDZIK, A.; CHUNG, J.; CHOI, D. H.; POWELL, R.; EWALDS, T.; GEORGIEV, P. *et al.* Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, Nature Publishing Group, v. 575, n. 7782, p. 350–354, 2019. Citado 4 vezes nas páginas 18, 20, 22 e 40.

VINYALS, O.; EWALDS, T.; BARTUNOV, S.; GEORGIEV, P.; VEZHNEVETS, A. S.; YEO, M.; MAKHZANI, A.; KÜTTLER, H.; AGAPIOU, J.; SCHRITTWIESER, J.; QUAN, J.; GAFFNEY, S.; PETERSEN, S.; SIMONYAN, K.; SCHAUL, T.; HASSELT, H. van; SILVER, D.; LILLICRAP, T. P.; CALDERONE, K.; KEET, P.; BRUNASSO, A.; LAWRENCE, D.; EKERMO, A.; REPP, J.; TSING, R. Starcraft II: A new challenge for reinforcement learning. *CoRR*, abs/1708.04782, 2017. Disponível em: ⟨http://arxiv.org/abs/1708.04782⟩. Citado 6 vezes nas páginas 20, 23, 37, 46, 52 e 70.

WANG, X.; SONG, J.; QI, P.; PENG, P.; TANG, Z.; ZHANG, W.; LI, W.; PI, X.; HE, J.; GAO, C.; LONG, H.; YUAN, Q. *SCC: an efficient deep reinforcement learning agent mastering the game of StarCraft II*. 2020. Citado na página 37.

WANG, X.; SONG, J.; QI, P.; PENG, P.; TANG, Z.; ZHANG, W.; LI, W.; PI, X.; HE, J.; GAO, C. *et al.* Scc: An efficient deep reinforcement learning agent mastering the game of starcraft ii. In: PMLR. *International conference on machine learning*. [S.l.], 2021. p. 10905–10915. Citado na página 37.