Universidade Federal do Rio Grande do Norte
Centro de Ciências Exatas e da Terra
Departamento de Informática e Matemática Aplicada
Bacharelado em Ciência da Computação


# Gerenciamento de Recursos em Starcraft: Experimentos com Aprendizagem por Reforço

**Diego Sabino Amorim de Araújo**

Natal-RN
junho de 2016

**Diego Sabino Amorim de Araújo**

# Gerenciamento de Recursos em Starcraft: Experimentos com Aprendizagem por Reforço

Monografia de Graduação apresentada ao Departamento de Informática e Matemática Aplicada do Centro de Ciências Exatas e da Terra da Universidade Federal do Rio Grande do Norte como requisito parcial para a obtenção do grau de bacharel em Ciência da Computação.

Orientador

Dr. Charles Andryê Galvão Madeira

Universidade Federal do Rio Grande do Norte – UFRN
Departamento de Informática e Matemática Aplicada – DIMAp

Natal-RN

junho de 2016

Monografia de Graduação sob o título *Gerenciamento de Recursos em Starcraft: Experimentos com Aprendizagem por Reforço* apresentada por Diego Sabino Amorim de Araújo e aceita pelo Departamento de Informática e Matemática Aplicada do Centro de Ciências Exatas e da Terra da Universidade Federal do Rio Grande do Norte, sendo aprovada por todos os membros da banca examinadora abaixo especificada:

---

Dr. Charles Andryê Galvão Madeira
Orientador
Instituto Metrópole Digital
Universidade Federal do Rio Grande do Norte

---

Dra. Anne Magaly De Paula Canuto
Departamento de Informática de Matemática Aplicada
Universidade Federal do Rio Grande do Norte

---

Dr. Adrião Duarte Dória Neto
Departamento de Engenharia de Computação e Automação
Universidade Federal do Rio Grande do Norte

Natal-RN, 13 de junho de 2016.

Dedico este trabalho aos meus pais Geraldo Sabino e Tereza Cristina, a meu irmão Daniel Sabino, às minhas irmãs Patrícia e Priscilla e a todos os meus amigos.

# Agradecimentos

Agradeço aos meus pais, Geraldo Sabino e Tereza Cristina, que ao decorrer da minha vida me ensinaram o valores da integridade e da educação.

Agradeço a meu irmão, Daniel Sabino, que me incentivou e ajudou em diversos momentos ao longo do meu curso, sempre acreditanto no meu potencial. Às minhas irmãs, Patrícia e Priscilla Sabino, por me proporcionarem tantos momentos de descontração.

Agradeço também a meu orientador, Charles Madeira, que me guiou neste trabalho desde o início, sempre disponível para me ajudar.

Finalmente agradeço a todos os meus amigos, que estão sempre do meu lado, torcendo por mim.

*Try not to become a man of success, but rather try to become a man of value.*

Albert Einstein

# Gerenciamento de Recursos em Starcraft: Experimentos com Aprendizagem por Reforço

Autor: Diego Sabino Amorim de Araújo
Orientador: Dr. Charles Andryê Galvão Madeira

## Resumo

Os jogos de estratégia em tempo real se tornaram ambientes de grandes desafios para pesquisadores da área de inteligência artificial devido à complexidade da tomada de decisão que esse gênero de jogos digitais apresenta. Dentre os diversos desafios existentes encontramos a tarefa do gerenciamento dos recursos que consiste no processo de escolha de ações referentes à coleta de minérios, ordem de construção de unidades, entre outras, a fim de gerar um exército de forma a otimizar o combate contra os adversários. Este trabalho propõe um modelo para o gerenciamento de recursos no jogo Starcraft através da noção de políticas de investimento, definindo classes de unidades e porcentagens que essas classes devem receber do total dos recursos disponíveis, se encarregando de gerenciar a linha de produção das unidades. Com o intuito de aprender políticas de investimento válidas, de forma automatizada, para o modelo proposto, algoritmos de aprendizagem por reforço foram utilizados. Os resultados obtidos através dos experimentos efetuados com um cenário simplificado do ambiente de Starcraft se mostraram bastante promissores.

*Palavras-chave*: Gerenciamento de Recursos, Aprendizagem por Reforço, Jogos de Estratégia em Tempo-real, Starcraft.

# Resource Management in Starcraft: Reinforcement Learning Experiments

Gerenciamento de Recursos em Starcraft: Experimentos com Aprendizagem por Reforço

Author: Diego Sabino Amorim de Araújo
Advisor: Dr. Charles Andryê Galvão Madeira

## Abstract

Real-time strategy games have become very challenging environments for researchers of Artificial Intelligence due to complexity of the decision-making that it presents. Among the various challenges we can find, the resource management task, which is a process for selecting actions concerning minerals gathering, building-order of units, and others, in order to compose armies able to optimize the combat strategy. This work proposes a model for resource management in Starcraft through the notion of investment policies by setting classes of units and tuning the interest of these classes, being responsible for managing the production line of units. In order to learn effective investment policies, in an automated way, for this model, reinforcement learning algorithms have been applied. The results obtained from experiments done in a simplified scenario of Starcraft was quite promising.

*Keywords*: Resource Management, Reinforcement Learning, Real-time Strategy Games, Starcraft.

# Lista de figuras

# Lista de tabelas

# Sumário

# 1 Introdução

Os jogos são atividades que na maioria das vezes têm fins recreativos e que cada vez mais se mostram como instrumentos educacionais em potencial. Eles são compostos por regras bem estruturadas que são aplicadas a algum ambiente que pode ser livre ou restrito (como um tabuleiro). Jogos de cartas, por exemplo, na maioria das vezes dependem de sorte para que o jogador vença a partida, mas em alguns casos é explorada a habilidade que o jogador tem de tomar decisões estratégicas.

Em jogos de estratégia, o jogador deve planejar quais ações devem ser escolhidas diante da situação em que ele vivencia a cada momento. Observar como o oponente está se comportando e quais são as possíveis decisões que serão tomadas por ele são exemplos de análise que o jogador deve fazer durante sua participação no jogo.

A Figura 1 apresenta um dos jogos de tabuleiro mais populares do mundo, o Xadrez, que é considerado pela comunidade científica como um jogo de estratégia clássico. Nesse jogo, cada jogador começa com dezesseis peças, cada uma com uma determinada característica. O objetivo é dar xeque-mate no adversário, ou seja, deixá-lo em uma situação que não seja possível realizar um movimento que escape da derrota. Apesar de ser um jogo relativamente simples, existem milhões de possibilidades de jogadas que podem ser efetuadas, fazendo com que haja uma necessidade de planejamentos de estratégias para que um jogador consiga a vitória.

No ano de 1997 a área de Inteligência Artificial (IA) obteve um marco devido à uma partida de Xadrez entre Garry Kasparov, considerado na época o melhor jogador do mundo deste jogo, e o Deep Blue, um sistema inteligente desenvolvido pela IBM (KENNEDY, 2015). Para o espanto de todos, o xeque-mate foi do sistema inteligente. Desde então, outros diversos jogos foram desenvolvidos e passaram a servir como excelentes ambientes de estudo para diversas técnicas computacionais, deixando de ser apenas ferramentas de entretenimento.

Outro jogo de estratégia clássico que se destaca em pesquisas da área de inteligência

Figura 1: Tabuleiro do Xadrez

artificial se chama Go. O ambiente desse jogo pode ser visto na Figura 2. Ele é considerado um dos jogos de estratégia clássicos mais difíceis que existem, que funciona da seguinte maneira: os jogadores devem, alternadamente, colocar pedras brancas e pretas em uma grade com 19 linhas verticais e 19 linhas horizontais, com o objetivo de cercar as peças do oponente.

Figura 2: Tabuleiro do Go

Dentro da categoria dos jogos digitais de estratégia, podemos destacar os jogos de estratégia em tempo real (Real-Time Strategy Games - RTS Games). Essa categoria possui um ambiente bem mais complexo do que os jogos de estratégia clássicos. O ambiente é altamente dinâmico, podendo haver mudança no estado do ambiente diversas vezes por segundo, diferente de jogos por turno, onde o ambiente muda apenas quando o jogador termina sua jogada. As informações não são completas, ou seja, apenas parte das informações do ambiente são disponíveis para o jogador, o que mais uma vez difere dos jogos de estratégia clássicos como o Xadrez, onde se tem a informação da posição de todas as peças do oponente. Starcraft é um famoso jogo de RTS criado pela Blizzard Entertainment, que se mostra como um excelente exemplo de jogo onde a complexidade do ambiente simulado é muito grande.

Um comparativo entre a complexidade dos jogos Go e Starcraft pode ser feito com o Xadrez. O espaço de situações possíveis (espaço de estados) no ambiente do Xadrez é da ordem de $10^{50}$. No Go, o espaço de estados é da ordem de $10^{170}$, o que já mostra

como uma diferença enorme da ordem complexidade desses jogos. Por essa razão, uma simples técnica de força bruta, como a que foi utilizada no Xadrez, não consegue trazer os mesmos resultados no Go. Em um jogo de estratégia em tempo real como Starcraft, que disponibiliza um mapa de 128 x 128, podendo haver até 400 unidades no mapa no qual cada uma delas se encontra em um determinado estado (minerando, andando, parado, atacando, etc.), a situação aplicada ao Xadrez passa a ser ainda mais irrelevante. Por exemplo, se for considerado apenas as posições de 400 unidades em um mapa de 128 x 128, isso resultará em um espaço de estados da ordem de $10^{1685}$. Se forem considerados outros fatores do jogo, resultará em ordens de grandeza ainda maiores.

Com o intuito de incentivar estudos e apresentar trabalhos realizados no contexto desses ambientes, diversas competições foram criadas nos últimos anos com o intuito de permitir aos pesquisadores em IA experimentarem e validarem as suas soluções (CHURCHILL, 2016). O principal objetivo delas é verificar o progresso da IA em jogos de estratégia em tempo real, visto que nessa categoria de jogos, os humanos ainda são muito melhores do que os sistemas artificialmente inteligentes. Como exemplo dessas competições podemos destacar a AIIDE Starcraft AI Competition[1] que é organizada pela Association for the Advancement of Artificial Intelligence (AAAI). Podemos destacar também a IEEE CIG Starcraft AI Competition[2] que faz parte de uma das maiores conferências na área, a Computational Intelligence in Games Conference, organizada pelo Institute of Electrical and Electronics Engineers (IEEE). Enfim, tem também a SSCAIT (Student StarCraft AI Tournament[3]) que é um evento educacional voltado para estudantes de IA e Ciência da Computação.

Devido à complexidade dessa categoria de jogos, estudos com diversas técnicas de IA estão sendo realizados com o objetivo de desenvolver programas capazes de simular o comportamento de jogadores especialistas. Mas para isso, um longo caminho deve ser percorrido.

## 1.1 Motivação

Quando a IBM conseguiu desenvolver um sistema inteligente que venceu o melhor jogador da época no Xadrez, motivou diversos pesquisadores a desenvolver técnicas que simulassem os comportamentos humanos. Porém, em diversos jogos de estratégia, os re-

[1] https://webdocs.cs.ualberta.ca/ cdavid/starcraftaicomp/
[2] http://cig2015.nctu.edu.tw/competitions
[3] http://sscaitournament.com

sultados das técnicas desenvolvidas não chegam nem perto dos resultados de um jogador humano devido à complexidade do ambiente. Para vencer a partida de Xadrez, a IBM desenvolveu um software bem primitivo que utilizava uma técnica de força bruta para calcular milhões de possibilidades de movimento e escolher aquela mais vantajosa. Isso serviu para o xadrez, mas para outras aplicações e jogos, a utilização de uma técnica tão primitiva não traz resultados ótimos.

Outro fato importante para o avanço das pesquisas na área foi o desenvolvimento do programa TD-Gammon (TESAURO, 1994), realizado por Gerald Tesauro, para jogar o Gamão. A Figura 3 mostra um exemplo do ambiente desse jogo.

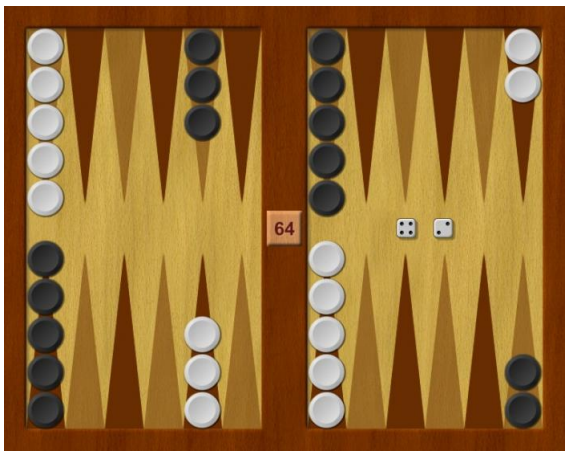


Figura 3: Estrutura do Gamão

O Gamão é um jogo para dois competidores no qual eles movem suas peças em sentidos contrários à medida em que jogam os dados e estes determinam quantas posições serão avançadas, conseguindo a vitória aquele conseguir levar todas as suas peças de um lado a outro do tabuleiro.

No gamão, o resultado desconhecido ao se jogar com dados, torna o uso de técnicas de força bruta inviável, sendo necessário o uso de técnicas mais avançadas. O TD-Gammon foi desenvolvido em 1992 e alcançou um nível muito próximo dos melhores jogadores da época. O programa consiste em uma rede neural que é treinada por uma técnica de aprendizagem por reforço chamada TD-Learning (TESAURO, 1994).

Esse fato mostra que técnicas aprendizagem por reforço podem conseguir bons resultados ao serem aplicadas em ambientes que possuem maior ordem de complexidade. No Xadrez foi possível utilizar uma técnica simples de força bruta para conseguir ganhar do melhor jogador da época, mas isso não foi possível para o Gamão, que só conseguiu tais resultados ao se desenvolver um sistema inteligente baseado em técnicas de aprendizagem por reforço.

Recentemente, a Deep Mind, que é uma empresa especializada em inteligência artificial que foi adquirida pela Google em 2014, deu um grande passo na direção da construção de uma inteligência artificial com aspectos de intuição humana. Ela desenvolveu um sistema inteligente capaz de aprender 49 tipos de jogos diferentes e usar o conhecimento adquirido de forma genérica (GIBNEY, 2016). Esse sistema conseguiu vencer tanto o campeão europeu quanto o campeão mundial de Go em diversas partidas.

Os criadores do AlphaGo (nome dado ao programa criado pela Deep Mind) dizem que as técnicas aplicadas são de propósito genérico e que pretendem futuramente usá-las para o desenvolvimento de melhores softwares de assistência pessoal, em que possam assimilar as preferências dos usuários a partir de seus comportamentos online e fazer mais recomendações intuitivas sobre produtos e eventos. A principal técnica utilizada no programa foi um método de aprendizado de máquina conhecido como "deep learning"(KNIGHT, 2016), que consiste em uma técnica que associa redes neurais e aprendizagem por reforço, tendo modelos livres de representação.

Depois que a Deep Mind conseguiu esse grande feito no contexto do jogo de Go, um dos responsáveis do projeto, chamado Jeff Dean, declarou em uma entrevista recente que StarCraft seria o próximo alvo (WEINBERGER, 2016). Essa declaração é de grande importância para pesquisadores da área, pois uma empresa do porte da Google, com tantos avanços tecnológicos, pode ajudar a trazer grandes resultados e, assim, impulsionar ainda mais a área.

Com todos esses fatos é possível perceber que a tarefa de desenvolver um sistema inteligente capaz de se comportar como um ser humano especialista nos jogos de estratégia em tempo real é bem complexo. Mas que, uma vez desenvolvido tal programa, as técnicas utilizadas nele serão capazes de solucionar diversos outros problemas em diversas áreas de atuação. Esse é o principal motivo pelo qual foi decidido desenvolver este trabalho nesta área.

## 1.2 Objetivos

Por serem de grande complexidade, os jogos de estratégia em tempo real se dividem em diversas áreas. Uma delas é o gerenciamento de recursos, que basicamente se preocupa em como investir os recursos coletados pelo jogador. Nesse gênero de jogos, os recursos coletados podem ser utilizados para construir unidades e construções de diferentes tipos com diferentes características.

Um jogador humano aprende com o tempo em quais unidades deve investir seus recursos e a ordem desse investimento, dependendo de como ele quer agir durante o jogo. Porém, como fazer para um sistema escolher automaticamente em quais unidades ele deve distribuir seus recursos? Além disso, um jogo como o Starcraft possui diversos tipos de unidades com características heterogêneas, onde estruturas de diversos tipos devem ser construídas para possibilitar a produção dessas unidades. Como fazer para um programa gerenciar tais construções e controlar suas produções?

Baseado nessas perguntas, este trabalho será apresentado para propor um modelo que possibilite a um sistema automatizado perceber o ambiente do jogo e decidir o destino de seus recursos, através de definições de classes e porcentagens que as mesmas devem receber do total de recursos, ficando responsável por gerenciar toda a linha de produção.

Mas ainda assim, como o sistema vai saber as porcentagens de recursos que cada classe de unidade deve receber durante o jogo? Bem, a partir do modelo proposto, queremos mostrar que a aplicação de técnicas de aprendizagem por reforço pode gerar resultados satisfatórios através de ambientes simplificados com objetivos simples, para que em trabalhos futuros, tais técnicas possam ser aplicadas com representações mais completas do ambiente, a fim de atingir objetivos mais complexos.

## 1.3 Organização do trabalho

No capítulo 2 serão mostradas as características dos jogos de estratégia em tempo real e quais são os principais desafios que essa categoria de jogos traz para a IA, destacando o gerenciamento de recursos que será a problemática trabalhada neste documento. Além disso, será mostrado um pouco sobre Starcraft, que é um benchmark desse gênero de jogos digitais para a comunidade científica de IA..

O capítulo 3 trata de trabalhos relacionados. Mostraremos os trabalhos existentes na literatura que contribuíram para o estudo feito neste documento, destacando dois deles que serviram como base para o desenvolvimento do modelo aqui proposto.

O capítulo 4 introduz a Aprendizagem por Reforço e apresenta os dois algorítmos que foram aplicados no modelo aqui proposto, o Sarsa(0) e o Sarsa($\lambda$). Mostraremos os principais componentes que os constituem no propósito geral e uma pequena comparação entre eles.

No capítulo 5 será proposto o modelo de sistema de gerenciamento de recursos. Será

explicado como políticas de investimento funcionam para gerenciar a distribuição dos recursos utilizando ministérios para controle de suas funções. A partir do modelo implementado, será mostrado como as técnicas de aprendizagem por reforço foram adaptadas e utilizadas.

No capítulo 6 será apresentada a metodologia experimental, na qual dois cenários foram montados para a execução dos experimentos e como o ambiente foi representado para a aplicação dos algorítmos, assim como os resultados obtidos pela execução dos mesmos.

Por último serão apresentadas algumas considerações finais, mostrando o que foi feito no trabalho e qual é a nossa visão sobre os resultados obtidos, informando o que se pôde concluir e o que podemos esperar para trabalhos futuros.

# 2 Jogos RTS

Jogos de estratégia em tempo real (RTS) consistem em um subgênero de jogos de estratégia em que sua execução ocorre ao mesmo tempo para todos os jogadores, assim os participantes têm um espaço de tempo limitado para decidir o que fazer, ao contrário dos jogos que funcionam através de turno, onde cada jogador toma as suas decisões sem que os oponentes estejam agindo ao mesmo tempo a fim de escolher a melhor ação a se realizar. Nos jogos de estratégia em tempo real, os participantes devem coletar recursos para construir uma base e um exército com o objetivo de destruir o inimigo. Eles devem decidir suas ações dentro de um espaço de estados muito grande em um espaço de tempo muito curto. Além disso, na maioria dos jogos de estratégia em tempo real os jogadores não têm informação completa sobre o mundo simulado, proporcionando um grau de incerteza relevante.

Os jogos de estratégia em tempo real são ótimos laboratórios de pesquisa (ONTAÑÓN et al., 2013). Eles possuem uma boa representação do mundo real com diversas variáveis e incertezas, possuem diversos agentes que permitem que através deles seja possível perceber o mundo e realizar diversas ações, além de que são ótimos ambientes de simulação, onde diversos testes podem ser realizados de forma iterativa e sem prejuízos causados por falhas.

Devido à complexidade dessa categoria de jogos, diversos desafios para a IA podem ser identificados e separados em seis áreas (ONTAÑÓN et al., 2013):

1. Planejamento
2. Aprendizagem
3. Incerteza
4. Raciocínio Temporal e Espacial
5. Conhecimento do Domínio
6. Decomposição de Tarefas

A área de **Planejamento** diz respeito aos desafios encontrados para planejamento das ações que serão tomadas durante a partida do jogo. Como foi mencionado anteriormente, o espaço de estados nesse gênero de jogos é muito elevado e simples técnicas como uma busca em árvore não é aplicável. Nessa categoria, o planejamento possui diversos níveis de abstração, onde no nível mais alto são tratados os problemas a longo prazo com o intuito de desenvolver uma forte economia e exército. No nível mais baixo, devem ser planejadas as decisões de forma individual, como por exemplo, o caminho que uma unidade deve tomar até chegar em determinada posição ou a forma com que uma unidade deve atacar um personagem do oponente.

Com relação à **Aprendizagem** podemos destacar três diferentes formas de desafios: a primeira diz respeito a como podemos explorar dados previamente disponíveis como replays de partidas ou mapas já conhecidos. A segunda trata da aprendizagem online, ou seja, como os programas podem aprender durante a partida e melhorar suas ações ao longo do jogo. Por último, os chamados *Inter-game learning*, que são técnicas que devem ser usadas para aprender com uma partida e usar esse conhecimento para melhorar os resultados da partida seguinte.

Durante uma partida, o jogador enfrenta diversos processos de decisões sobre as **incertezas** oferecidas pelo jogo. Por exemplo, o mapa é coberto por uma neblina de guerra, o leva a três estados possíveis: ativo, visto ou desconhecido. O estado ativo é a região onde não possui a neblina, ou seja, é o campo de visão atual do jogador e é onde ele tem a informação em tempo real do que está acontecendo. O estado visto é a região que o jogador já visitou, mas não se encontra nela no momento. Nessa região, ele tem a informação de como as estruturas estavam da última vez que ele visitou. Por último, no estado desconhecido, o jogador não tem informação alguma sobre aquele local. Isso já oferece um grau de incerteza muito grande sobre o ambiente, diferente dos jogos nos quais o participante tem a informação total sobre o mundo simulado.

Tendo isso em vista, o jogador deve descobrir onde o inimigo está e enviar constantemente unidades para colher informações sobre o estado do inimigo. Deve decidir qual estratégia de produção utilizará, ou seja, decidir quais unidades vão ser construídas e qual a ordem de construção delas de acordo com a raça do oponente que está enfrentando. Essas e diversas outras variáveis nos mostra o grau de incerteza dessa categoria de jogos.

**Raciocínio Temporal e Espacial** trata de problemas com relação ao uso do espaço e do tempo no jogo. Um exemplo de desafio ligado à exploração do terreno é o fato de que em alguns RTS é sempre vantagem posicionar unidades militares em terrenos de mais

Figura 4: Imagem que representa o fog of war no RTS Age of Empires

alto relevo, pois os que estão nos terrenos de mais baixo relevo não têm boa visão. Com relação à exploração do tempo, podemos ter como exemplo a decisão do momento certo em que deve ser realizado um ataque. Decidir se é melhor realizar ataques rápidos com uma quantidade menor de unidades, ou ataque a longo prazo, porém com muitas unidades militares.

A área de **Exploração do Conhecimento de Domínio** tenta implantar as formas de conhecimentos existentes nos programas criados. Por exemplo, existem diversos guias de estratégias de jogo para uma partida de Starcraft, mas ainda não se sabe ao certo como implantar essas estratégias em um sistema inteligente para este jogo, de forma que na hora que o programa estivesse em uma partida, ele apenas escolhesse uma delas. Além disso, existem bancos de dados com diversos replays armazenados. Estudos estão sendo realizados com o intuito de implementar uma aprendizagem automática desses replays para os programas criados.

A maioria dos sistemas inteligentes implementados para os jogos de estratégia em tempo real trabalham com **Decomposição de Tarefas**, ou seja, dividem problemas de mais alto nível em problemas menores para serem tratados individualmente. A decomposição mais comum possui um hierarquia de problemas. O nível mais alto de hierarquia trata do problema de estratégias gerais. Decide, por exemplo, se vai realizar ataques rápidos ou vai fortalecer sua base para realizar ataques mais fortes a longo prazo. No nível intermediário de hierarquia é realizada a implementação da estratégia. Por exemplo, decide quais grupos de unidades serão criados ou a formação tática que os exércitos irão tomar. No nível mais baixo de hierarquia, é feita a implementação do nível tático, ou seja, trata-se cada unidade individualmente. Por exemplo, decide como cada unidade vai realizar um ataque, ou como será feita a coleta de recursos.

O presente trabalho, especificamente, trata sobre o gerenciamento de recursos, que consiste no processo de escolha de ações referentes à coleta de minérios, ordem de construção de unidades, entre outras, a fim de gerar um exército de forma a otimizar o combate contra os adversários. Quando se fala em gerenciamento, é possível associar essa ideia ao planejamento, pois para qualquer tipo de gerenciamento é necessário um planejamento prévio. É preciso planejar, por exemplo, se será feito um investimento rápido dos recursos ou se será feito um investimento a longo prazo, o que seria também, nesse caso, um problema de Raciocínio Temporal. Porém, quando uma estratégia for decidida, ela não poderá assumir a partida até o final, pois as variáveis do jogo mudam constantemente. Com isso, é preciso trabalhar na parte de Aprendizagem, utilizando técnicas que se adaptam ao contexto. Contudo, para realizar tal aprendizagem é preciso saber tratar as incertezas presentes no jogo, pois como vimos, não temos todas as informações do mapa. Com isso, é possível perceber que o problema de gerenciamento de recursos faz parte de várias áreas de desafios para a IA e é nessa problemática que este trabalho será focado.

## 2.1 Starcraft

Starcraft é um jogo de estratégia em tempo real desenvolvido pela Blizzard Entertainment no ano de 1998. Sua história se concentra em uma guerra entre três raças distintas: Zerg, Protoss e Terranos.

O jogo é baseado na aquisição de dois recursos, minério e gás, que são necessários para construir unidades militares e estruturas. O mineral é o recurso mais básico, necessário para construir qualquer unidade do jogo e pode ser coletado desde o início da partida. O gás é necessário para unidades mais avançadas e melhorias. Uma estrutura particular (Refinaria) é necessária para possibilitar a coleta desse recurso. A Figura 5 mostra um exemplo de uma base com os minerais e a refinaria construída.

Para colher os recursos são necessários trabalhadores, que são responsáveis por coletar os recursos e levar para a base mais próxima. Cada jogador começa com uma base e cinco trabalhadores.

Além dos recursos principais, existem os suprimentos que são necessários para construir as unidades. Quando se esgotam, é preciso construir mais da sua fonte para possibilitar a construção de mais unidades.

Ao se coletar minério e gás e produzir fontes de suprimentos, é possível construir um exército que pode ser usado para atacar a base do inimigo ou se defender dele. Equipes


Figura 5: Exemplo de base dos Terranos e seus recursos disponíveis. O item destacado em vermelho é uma refinaria, o item destacado em verde é um minério e o item destacado em azul é a base principal.

podem ser formadas para competições, onde os jogadores que destruírem as bases dos oponentes são os vencedores.

### 2.1.1 Raças

Cada raça possui unidades e construções distintas, porém bem equilibradas, assim, os jogadores devem utilizar estratégias distintas para cada uma das raças.

#### 2.1.1.1 Protoss

Os protoss possuem unidades de grandes poderes militares, porém necessitam de mais recursos e tempo para suas produções. Os jogadores que escolhem essa raça geralmente realizam ataques a longo prazo. Sua unidade básica, os Zealots, é a mais forte entre as raças, que apesar de demorar um pouco mais para ser construída, consegue confrontar um número maior de unidades básicas de outra raça. Eles possuem a vantagem de poder construir diversas estruturas ao mesmo tempo com um mesmo trabalhador, garantindo uma necessidade de um número menor de trabalhadores. Porém suas construções são mais demoradas e mais caras, onde as mesmas precisam ser construídas perto de uma estrutura chamada Pylos.

A Figura 6 mostra uma base dos Protoss com algumas unidades básicas. A unidade que está destacada em vermelho é um trabalhador, ele pode coletar recursos e mandar construir diversas estruturas ao mesmo tempo. A unidade destacada em verde é a unidade básica de combate e a estrutura que está destacada em azul é o Pylo, necessário para fornecer suprimentos para construir mais unidades e permitir que estruturas possam ser construídas ao seu redor. Essas são apenas as unidades básicas da raça, mas com elas já


Figura 6: Algumas estruturas e unidades das Protoss. A unidade destacada em vermelho é um trabalhador, a unidade destacada em azul é o Pylo e a unidade destaca em verde é o Zealot

é possível realizar um ataque ao inimigo, ou se defender dele.

#### 2.1.1.2 Zergs

Os Zergs são unidades com poderes menores, mas que conseguem se multiplicar rapidamente, possibilitando ataques rápidos e com grande quantidade de unidades. Todas as suas unidades são construídas através de uma mesma estrutura, as Larvas. A contrário das outras raças, que fabricam uma unidade após a outra em uma espécie de fila, os Zergs podem produzir até 3 unidades em paralelo. A maioria de suas estruturas e unidades são baratas e de rápida fabricação. Sua unidade básica é o *Zergling*, que apesar de ser a unidade básica mais fraca dentre as outras, são fabricadas de 2 em 2, podendo então ser construídas até 6 em paralelo. Eles possuem a desvantagem de só poder construir suas estruturas sobre "creep", que consiste no terreno dos Zergs.

A Figura 7 mostra um exemplo de uma base dos Zergs também com algumas unidades básicas. A unidade que está destacada em vermelho é um trabalhador e as unidades que estão destacadas em verde são os *Zerglings*, que são construídos de 2 em 2.

Jogadores que escolhem esta raça, geralmente utilizam estratégias de ataques rápidos e em massa, surpreendendo os inimigos logo no início da partida. Jogadores com outras raças, quando sabem que estão jogando contra Zergs, devem construir defesas no início do jogo, sabendo que há uma grande probabilidade de haver um ataque no início do jogo.


Figura 7: Algumas estruturas e unidades dos Zergs. A unidade destacada em vermelho é um trabalhador e as unidades destacadas em verde são os *Zerglings*

#### 2.1.1.3 Terranos

Entre os Zergs e os Protoss estão os Terranos, que possuem características equilibradas. As produções de suas unidades e estruturas não demoram tanto quando as dos Protoss nem são tão rápidas quanto as dos Zergs, assim como os custos para suas produções são medianos. Eles possuem a vantagem de poder construir suas estruturas em qualquer lugar do mapa, diferente dos Protoss que precisam construir ao redor dos Pylos, ou dos Zergs que precisam construir sobre seu próprio terreno. Além disso, algumas de suas construções podem voar, possibilitando a locomoção das mesmas. Suas unidades básicas são os *Marines*, que possuem a vantagem de atacar de longa distância, podendo atacar também unidades aéreas.


Figura 8: Algumas estruturas e unidades dos Terranos. A unidade destacada em vermelho é um trabalhador e a unidade destacada em verde é um Marine.

A Figura 8 mostra uma base com algumas unidades dos Terranos. A unidade destacada em preto é um trabalhador, responsável por coletar os recursos e construir estruturas. A unidade destacada em verde é um *Marine*, que é a unidade básica de combate dos Terranos.

# 3 Trabalhos Relacionados

Na literatura existem diversos trabalhos que tratam dos desafios apresentados pelos jogos de estratégia em tempo real, tratando de diversas problemáticas que podem ser exploradas e tratadas separadamente para fins de estudos. Ontañón et al. (2013) classifica os diversos desafios em grupos distintos de forma clara e objetiva e explica como eles são separados em diferentes níveis de abstração, além de mostrar as estruturas dos modelos gerais que são implementados por especialistas na área. É um ótimo artigo para quem quer entender, de forma geral, essa complexa área de conhecimento.

Além dele, podemos destacar também o trabalho de Robertson e Watson (2014) que mostra os diversos algorítmos de inteligência artificial que são aplicados em jogos de estratégia em tempo real. Esse trabalho apresenta que as principais áreas de pesquisa acadêmica atuais focam em tomada de decisões táticas e estratégicas, além de aprendizagem e reconhecimento de plano, e descreve algumas contribuições existentes para cada uma dessas áreas.

No que diz respeito ao gerenciamento de recursos, especificamente, podemos destacar alguns trabalhos importantes. Kovarsky e Buro (2006) propuseram um modelo com foco na otimização da coleta de recursos e na otimização da criação de unidades e construção na fase inicial do jogo. O modelo se concentrava em dois objetivos principais: 1) minimizar o tempo para atingir determinado objetivo, como por exemplo, construir duas torres de defesa aérea e quatro tanques de guerra; e 2) otimizar a coleta de recursos em um determinado tempo, por exemplo, maximizar a coleta de minerais em um intervalo de dez minutos. O problema dessa abordagem era que não levava em conta o fator dinâmico do jogo, ou seja, uma vez otimizada no inicio da partida, a estratégia escolhida não mudaria.

Alguns trabalhos se concentram particularmente na definição da ordem de construção das unidades e estruturas, a chamada *build-order*. Porém, o trabalho de Churchill e Buro (2011) se concentrou na otimização do plano de investimento, considerando que a ordem de construção já estava definida. Eles usaram técnicas de busca para achar um conjunto

de ações que minimizassem o tempo para realizar o plano de construções definidas, ou seja, o principal objetivo não era decidir a ordem de construção das unidades e estruturas, mas diminuir o tempo de execução do plano de construção.

O trabalho de Souza (2013) teve como proposta um modelo para gerenciamento de recursos baseado em uma abstração para realizar tomadas de decisões utilizando políticas de investimento, onde através destas, possibilite a distribuição de recursos para diferentes classes de unidades.

Oliveira (2014) propôs um modelo para gerenciamento de recursos baseado em ministérios, cada um possuindo responsabilidades distintas, de forma a interagir entre si para realizar diversas tarefas como o controle de produção de unidades, coleta de recursos, combate de unidades, entre outras.

O trabalho de Oliveira e Madeira (2015) apresentou um modelo baseado na técnica de campos potenciais para tratar o problema do posicionamento das estruturas, de forma a formar obstáculos para dificultaro deslocamento dos adversários no mapa. têm sido desenvolvidos neste tema. Os resultados obtidos por Oliveira e Madeira (2015) aproximam-se da forma em que jogadores profissionais se comportam.

Muitos trabalhos relacionados ao gerenciamento de recursos se preocupam com o problema da ordem de construção. Porém, isso é apenas uma parte do problema no que diz respeito ao gerenciamento de recursos. Especificar a ordem com que as unidades e estruturas serão construídas é de grande importância, mas esse não é o único aspecto que deve ser levado em consideração. Gerenciar e otimizar a coleta de recursos, especificar prioridades nas construções, verificar o contexto e a adaptabilidade ao meio são outros exemplos de problemáticas que essa área deve se preocupar.

O trabalho de Souza (2013) utilizou algumas dessas ideias para implementar uma abordagem para o gerenciamento de recursos baseado em Políticas de Investimento. Esse, juntamente com o trabalho de Oliveira (2014), formam a base do nosso modelo. Por essa razão, explicaremos cada um deles com um pouco mais de detalhe.

## 3.1 PICFlex: uma abordagem de gerenciamento de recursos baseada em Política de Investimento Contextual e Flexível

O foco principal deste trabalho se concentra no conceito de Políticas de Investimento mostrado no trabalho de Souza (2013). É preciso entender como essa abordagem funciona para compreender o modelo que será proposto no capítulo 5.

### 3.1.1 Definição de Classes e Políticas de Investimento

Em jogos de estratégia em tempo real, como Starcraft, há uma grande variedade de características das unidades. Algumas delas, por exemplo, podem ser unidades aéreas, enquanto que outras terrestes. Também podem haver unidades que possuem baixo poder de ataque, mas que são construídas rapidamente, enquanto outras que possuem ataques mais fortes, porém demoram a ser construídas. Essas e diversas outras características das unidades de Starcraft levaram Souza (2013) a dividí-las em classes, que nada mais é do que um agrupamento de unidades que possuem características semelhantes. Assim, permite-se ter um melhor controle de como investir os recursos que são obtidos, além de que dá um melhor direcionamento para a estratégia a ser adotar durante as partidas.

Uma vez definido o conceito de classes de unidades, pode-se definir o conceito de Políticas de Investimento. Souza (2013) define formalmente uma Política de Investimento **P** como sendo:

$$P = [c_1, t_1], [c_2, t_2], ...[c_n, t_n], onde \sum_{i=1}^{n} t_i = 100\%$$

onde $c_1, ..., c_n$ são as classes definidas para as unidades e $t_1, ..., t_n$ são as metas de gastos para cada uma dessas classes.

Suponha, por exemplo, que tenham sido definidas 3 classes de unidades: c1 = Unidades Terrestres, c2 = Unidades Aéreas e c3 = Construtores. Uma possível política de investimento para essas classes poderia ser:

$$P = [UnidadesTerrestres, 20], [UnidadesAerias, 30], [Construtores, 50]$$

com esta configuração, 20% dos gastos totais serão direcionados para as unidades terrestres, 30% para as unidades aéreas e 50% para os trabalhadores.

### 3.1.2 Adaptabilidade das Políticas de Investimento

A partir do momento que é definida uma política de investimento, ela não deve ser estática. Ao analisar, por exemplo, os investimentos efetuados por um jogador humano durante um certo momento mais tranquilo, ou seja, quando nenhum dos adversários estão atacando, pode-se perceber que eles são bastante diferentes dos investimentos efetuados nos momentos em que está acontecendo algum ataque.

Tendo isso em vista, surge a necessidade de realizar uma mudança na política de investimento de acordo com o contexto do jogo. Observar como o inimigo está se comportando, situações de ataques ou defesas, exército atual do inimigo, são exemplos de contextos que podem ser levados em consideração para a decisão da mudança da política de investimento e isto pode ser definido formalmente da seguinte maneira:

$$F(P, C) \rightarrow P'$$

onde P é a política de investimento atual, C é o contexto em um determinado momento e P' é a política de investimento atualizada. Em outras palavras, isso significa que dada a situação do jogo em um determinado momento, deve ser feito um ajuste nos valores da política, se necessário.

Uma grande incógnita para o uso efetivo dessa estratégia é a forma com que é definido o contexto. Quais são os fatores que devem ser levados em consideração e o quanto que esses fatores influenciam numa tomada de decisão? Essas questões serão tratadas no capítulo 5 onde será introduzido o modelo proposto no presente trabalho.

## 3.2 Cerebrate: um modelo para personagens inteligentes de jogos de estratégia em tempo real

Para completar a base do modelo que será introduzido no capítulo 5, é preciso também apresentar a proposta desenvolvida por Oliveira (2014), em seu trabalho de conclusão de curso do Bacharelado em Ciência da Computação da Universidade Federal do Rio Grande do Norte.

Essa proposta, chamada Cerebrate, se baseia na forma em que os governos atuais distribuem suas tarefas através de ministérios, estes sendo responsáveis por gerenciar tipos distintos de tarefas. Na Tabela 1, é possível visualizar quais ministérios foram definidos e suas respectivas responsabilidades.

Tabela 1: Ministérios e suas responsabilidades

| **Ministério** | **Responsabilidade** |
|---|---|
| Ministério de Minas e do Trabalho | Administração dos trabalhadores |
| Ministério da Economia | Monitoramento de recursos |
| Ministério de Infraestrutura | Construção de estruturas |
| Ministério da Indústria | Administração da ordem de construção |
| Ministério da Defesa | Administração do exército |
| Ministério de Tecnologia | Pesquisa de novas tecnologias |
| Agência de Inteligência Militar | Coletar dados do jogador inimigo |

O modelo de Cerebrate é composto de seis ministérios e uma Agência de Inteligência que serve como uma entidade de auxílio, fornecendo as informações necessárias para o funcionamento dos mesmos.

O **Ministério de Minas e do Trabalho** é responsável por manter os trabalhadores minerando através de uma máquina de estados. Nela é feita a transição dos estados que fazem com que o trabalhador vá até a mina de melhor posição, colete o recurso e retorne para a base para entregar o que colheu. Este ministério também é responsável por manter um equilíbrio da quantidade de trabalhadores entre as bases.

O **Ministério da Economia** tem como principal função fornecer as informações de economia do jogador. Ele mantem o controle da lista de gastos reservados. Assim, quando o jogador quer saber quanto de minério ele possui, ele pergunta ao Ministério da Economia a quantidade de recursos disponíveis, ou seja, a quantidade que não está reservada.

O papel do **Ministério de Infraestrutura** é selecionar um trabalhador entre os mineradores e mandar construir a estrutura solicitada em uma posição específica. De maneira similar ao Ministério de Minas, suas ações são controladas através de uma máquina de estados que move o trabalhador selecionado até a posição escolhida para a estrutura. Quando o trabalhador chega na posição, seu estado muda e a construção começa a ser feita. Além disso, também há uma mudança de estado para que o trabalhador comece a fugir quando for atacado.

O **Ministério da Indústria** mantém uma fila única de produção com os itens que o jogador deseja construir. Ele solicita ao Ministério da Economia a quantidade de recursos disponíveis e dá ordem ao Ministério de Infraestrutura e de Tecnologia para executar a construção quando há recursos suficientes para construir o primeiro elemento da fila, assim como dá ordens às estruturas de produção, as quais fazem parte de sua responsabilidade. De forma geral, ele olha o primeiro elemento da lista, vê se tem recurso suficiente para

construir e em caso afirmativo, solicita a construção que pode ser feita pelo Ministério de Infraestrutura, Ministério de Tecnologia ou por ele mesmo, dependendo do item em questão.

O **Ministério de Defesa** tem como objetivo controlar as unidades militares usando alguma estratégia já pesquisada ou não. Oliveira (2014) cita alguns exemplos de estratégias que foram implementadas por outros pesquisadores, como, por exemplo, o trabalho de Uriarte e Ontañón (2012) que mostraram como criar um comportamento de ataque e fuga usando mapas de influência.

A função do **Ministério de Tecnologia** consiste em pesquisar novas tecnologias, utilizando, caso necessário, a Agência de Inteligência para obter informações do oponente que ajude a identificar qual tecnologia é necessária em um dado momento.

Como dito anteriormente, existe uma entidade que não é classificada como Ministério, denominada **Agência de Inteligência**, que serve de auxílio. Ela é responsável por analisar o mapa, verificar as movimentações do exército inimigo e predizer ataques iminentes, além de que ela também verifica e guarda as informações sobre o desenvolvimento estratégico inimigo.

As entidades mencionadas interagem umas com as outras de acordo com suas necessidades. Associando as funções de todos os ministérios com as da Agência de Inteligência, é possível realizar a coleta de recursos, a construção de unidades, estruturas e tecnologias, obter informações sobre o inimigo, manter uma ordem de construção desejada e um controle dos gastos.

## 3.3 Outros Trabalhos Relevantes

Na literatura existem alguns trabalhos que utilizam aprendizagem por reforço para atingir objetivos específicos em jogos de estratégia em tempo real. Como por exemplo, Wender e Watson (2012) que apresentam uma avaliação da aplicação de algoritmos de aprendizagem por reforço em um ambiente de combate simplificado. Eles simularam as técnicas em um cenário que consiste em uma unidade de combate controlada por um sistema inteligente lutando contra um grupo de unidades posicionadas ao seu redor. A unidade de combate deveria aprender a destruir todos os inimigos. Para avaliar os resultados obtidos, foi feita uma comparação entre os algoritmos aplicados.

Além disso, Siebra e Neto (2014) propuseram um modelo de aplicação do algoritmo

Sarsa de aprendizagem por reforço em situações de combate. O objetivo do trabalho deles era colocar agentes controlados por um sistema inteligente sem nenhum conhecimento inicial a fim de aprender a vencer diferentes situações de combate.

Ao se pesquisar outros trabalhos na literatura, é possível observar que há uma tendência para o uso de algoritmos de aprendizagem por reforço nos jogos de estratégia em tempo real. Apesar desses algoritmos serem mais utilizados no contexto do combate, o modelo que será proposto neste documento utiliza técnicas de aprendizagem por reforço como base para o gerenciamento de recursos. Por esse motivo, o próximo capítulo relata um pouco mais sobre esta área.

# 4 Aprendizagem por Reforço

Aprendizagem por reforço é uma área da Inteligência Artificial que se preocupa em como um agente deve realizar suas ações em um ambiente, de tal forma que acumule o máximo possível de recompensas (Sutton e Barto (1998)). O agente deve aprender como se comportar em um ambiente dinâmico através da sua interação com o ambiente, ou seja, a cada ação escolhida o agente recebe uma recompensa ou uma punição, de acordo com o seu objetivo final.

A Figura 9 apresenta uma abstração simplificada do modelo de aprendizado por reforço. A cada passo o agente recebe o estado atual do ambiente, escolhe uma determinada ação a ser tomada, realiza tal ação e observa o retorno através de um valor de sinal de reforço, indicando se a ação tomada foi uma boa ou má decisão. O processo se repete para o próximo estado do ambiente que pode, inclusive, ser o mesmo.


Figura 9: Abstração do Modelo de Aprendizagem por Reforço

Um modelo padrão de aprendizagem por reforço é constituído por:

- Um conjunto de estados S
- Um conjunto de possíveis ações A que o agente pode tomar
- Uma função de reforço R
- Uma função de probabilidade de transição P

onde a função de probabilidade de transição informa a probabilidade do agente ir de um estado para outro, ou permanecer no mesmo, a partir de uma ação executada.

Existe ainda a Política do Agente, que é a função que mapeia os estados do ambiente nas ações que o agente deve tomar. O objetivo da técnica é achar uma política $\pi$ que maximize o somatório de todos os reforços das ações escolhidas do início até o final do processo. Esta não é uma técnica gulosa, ou seja, para um determinado estado específico, a política encontrada pode não escolher a ação que retorna a melhor recompensa, mas a longo prazo, o somatório de todas as recompensas será maximizado.

Uma das maneiras de modelar problemas de aprendizagem por reforço é utilizando o Processo de Decisão de Markov (MDP- Markov Decision Process). Nesse modelo, um agente, em cada instante de tempo está em algum estado $s$ e deve escolher uma ação $a$ que é válida para o estado $s$. O processo responde no próximo instante de tempo indo para algum estado $s'$ e recebendo uma recompensa $R_a(s, s')$.

A probabilidade de um agente mover para um novo estado $s'$ é influenciada pela ação escolhida ou, mais especificamente, é dada pela função de transição de estado $P_a(s, s')$. Assim o próximo estado $s'$ depende apenas do estado $s$ e da escolha ação escolhida pelo agente.

De uma maneira mais formal, podemos definir o Processo de Decisão de Markov em 5 tuplas:

- **S**: conjunto de estados
- **A**: conjunto de ações
- $\boldsymbol{P_a(s, s') = Pr(s_{(t+1)} = s' | s_t = s, a_t = a)}$: probabilidade que uma ação $a$ no estado $s$ em um instante de tempo $t$ leva ao estado $s'$ no instante de tempo $t + 1$
- $\boldsymbol{R_a(s, s')}$: recompensa recebida ou recompensa esperada por sair do estado $s$ e ir para o estado $s'$;
- $\boldsymbol{\gamma \in [0, 1]}$: fator de desconto que informa o quanto recompensas futuras influenciam no presente.

Ao longo do processo de aprendizagem o agente deve escolher entre explorar novas possibilidades ou usar a melhor política já conhecida. Deve haver um equilíbrio entre exploração e uso de conhecimento. Um agente que escolhe explorar muito novas possibilidades pode deixar de aproveitar o conhecimento adquirido. Por outro lado, um agente que

explora pouco pode usar soluções que estejam muito longe da solução ótima. Uma possibilidade interessante para esse problema é a utilização de técnicas híbridas, que começam com uma taxa de exploração alta e com o passar do tempo essa taxa vai diminuindo de forma que, a longo prazo, o sistema explore o mínimo possível e use a melhor solução conhecida.

### 4.0.1 Algoritmo Sarsa(0)

Existem diversos algoritmos de aprendizagem por reforço existentes na literatura, mas será mostrado nesse trabalho apenas o Sarsa(0) e a sua generalização, o Sarsa($\lambda$), que são as técnicas escolhidas para serem aplicadas no modelo proposto no próximo capítulo.

Como mostramos anteriormente, para cada estado existente no modelo, existe um número possível de ações que podem ser escolhidas pelo agente. A base do funcionamento do Sarsa está no cálculo do que chamamos de Função Ação-Valor Q(s,a) que é o valor do reforço futuro esperado, dado a realização de uma determinada ação em um determinado estado específico.

A Tabela 2 mostra um exemplo de uma tabela Q para um sistema com dois estados e duas ações possíveis. Nela, os estados estão representados pela letra S, as ações pela letra A e os valores de reforço acumulativo referente à associação de cada para estado-ação pela letra R. A medida que o algoritmo vai sendo executado, os valores de Q vão sendo atualizados a fim de convergir para uma política ótima. Depois de um tempo T, a política ótima é dada pela escolha dos pares estado-ação de maiores valores da tabela.

Tabela 2: Q(s,a)

| **Estado** | **Ação** | **Valor** |
|---|---|---|
| S1 | A1 | R1 |
| S1 | A2 | R2 |
| S2 | A1 | R3 |
| S2 | A2 | R4 |

O algoritmo em linguagem simplificada é demonstrado na Figura 10. Todos os valores da tabela Q são inicializados com um valor arbitrário, normalmente 0. O algoritmo então executa um número determinado de episódios para atualizar os valores de Q.

Em cada passo do episódio que está sendo executado, o algoritmo verifica o estado atual, escolhe uma ação para ser executada, verifica a recompensa dada pela mudança do ambiente devido a execução de tal ação, analisa o estado atual do ambiente depois da

realização da ação e calcula o valor de Q(s,a) por:

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

onde $\alpha$ é a taxa de aprendizagem que indica o quanto que o agente aprende com a realização de suas ações a cada passo, $r$ é o reforço (recompensa ou punição) resultado da realização da ação, $\gamma$ é um parâmetro $0 \leq \gamma \leq 1$ chamado de taxa de desconto que desvaloriza recompensas em instantes de tempo anteriores ao instante atual. No Sarsa(0) consideramos apenas um passo a frente, então normalmente o valor de $\gamma$ é igual a 1.

Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
  Initialize $s$
  Choose $a$ from $s$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
  Repeat (for each step of episode):
    Take action $a$, observe $r$, $s'$
    Choose $a'$ from $s'$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    $Q(s,a) \leftarrow Q(s,a) + \alpha\big[r + \gamma Q(s',a') - Q(s,a)\big]$
    $s \leftarrow s'$; $a \leftarrow a'$;
  until $s$ is terminal

Figura 10: Algorítmo Sarsa

As escolhas das ações $a$ e $a'$ são feitas seguindo uma política $\epsilon$-greedy, que possui um parâmetro $\epsilon$ que indica a chance de escolha de uma ação aleatória. Por exemplo, suponha que o valor de $\epsilon$ é 0.8, então o algorítmo tem 80% de chance de escolher uma ação aleatória e 20% de chance de escolher a melhor ação já conhecida para aquele estado.

### 4.0.2 Algoritmo Sarsa($\lambda$)

O Sarsa(0) atualiza o valor de Q(s,a) a cada passo e segue trocando o estado atual pelo estado seguinte. O Sarsa($\lambda$) funciona de maneira semelhante, a única diferença sendo que ao invés de atualizar apenas o valor de Q para o estado-ação atual, ele deve atualizar os valores de Q para todos os pares estado-ação visitados anteriormente no episódio.

O algoritmo Sarsa($\lambda$) (Sutton e Barto (1998)) pode ser visto na Figura 11.

Basicamente o que esse algoritmo faz é calcular o valor de Q para o par estado-ação atual e propagar esse valor para todos os pares estado-ação visitados anteriormente aplicando um decremento determinado por $\lambda$. O cálculo do valor que será propagado é dado por:

Initialize $Q(s,a)$ arbitrarily and $e(s,a) = 0$, for all $s, a$
Repeat (for each episode):
  Initialize $s$, $a$
  Repeat (for each step of episode):
    Take action $a$, observe $r$, $s'$
    Choose $a'$ from $s'$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    $\delta \leftarrow r + \gamma Q(s', a') - Q(s, a)$
    $e(s, a) \leftarrow e(s, a) + 1$
    For all $s, a$:
      $Q(s, a) \leftarrow Q(s, a) + \alpha \delta e(s, a)$
      $e(s, a) \leftarrow \gamma \lambda e(s, a)$
    $s \leftarrow s'$; $a \leftarrow a'$
  until $s$ is terminal

Figura 11: Algoritmo Sarsa($\lambda$)

$$\delta \leftarrow r + \gamma Q(s', a') - Q(s, a)$$

e a atualização dos valores de Q é feita da seguinte maneira:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \delta e(s, a)$$

onde e(s,a) tem seu valor decrementado por $\lambda$ toda vez que é atualizado, assim, se um par estado-ação foi visitado há muitos instantes de tempo, ele receberá uma parcela pequena do valor atual que foi calculado, mas se um par estado-ação ocorreu em um instante de tempo anterior ao instante atual, ele receberá uma parcela grande. É como se todo par estado-ação que foi visitado fosse colocado em uma pilha de execução e toda vez que $\delta$ fosse calculado, o valor se propagasse para todos os elementos da pilha, sendo que o valor de propagação sempre é decrementado por $\lambda$ para os elementos mais fundo da pilha.

A Figura 12 mostra um comparativo entre a execução dos algorítmos Sarsa(0) e Sarsa ($\lambda$) em um cenário onde um robô deve se movimentar realizando quatro tipos de ações (CIMA, BAIXO, DIREITA OU ESQUERDA), com o objetivo de chegar na posição objetivo destacada por *.

Quando o robô acha a posição objetivo, ele recebe uma recompensa por isso. Perceba que no algoritmo Sarsa(0) (ou one-step Sarsa), quando o agente chega no objetivo, a recompensa é dada apenas para o último estado-ação visitado. No Sarsa($\lambda$) o valor é propagado para todos os pares estado-ação visitados anteriormente com seu valor cada

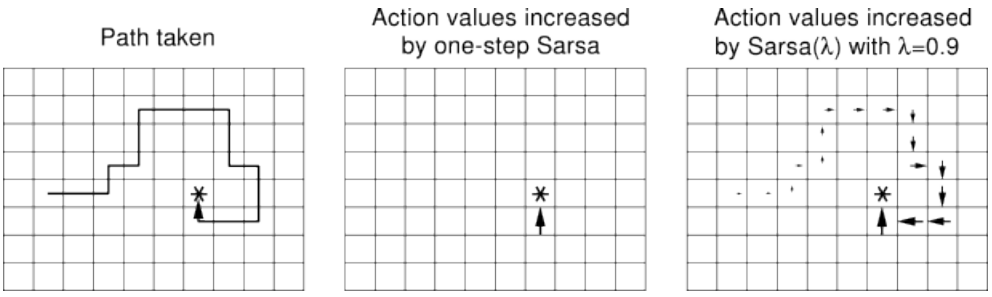


Figura 12: Comparação entre o Sarsa(0) e o Sarsa($\lambda$)
em um cenário onde um robô deve encontrar uma determinada posição objetivo

vez menor a medida que vai se distanciando da posição objetivo.

É fácil perceber que com o Sarsa($\lambda$) o aprendizado acontece de forma mais rápida. Assim, para ambientes que possuem muitos estados e ações possíveis, o método mais recomendado é o Sarsa($\lambda$).

# 5 Modelo de Gerenciamento de Recursos

O modelo que será proposto neste capítulo utiliza como base algumas das ideias de trabalhos apresentados anteriormente, como por exemplo, as políticas de investimento do trabalho de Souza (2013) e o modelo de divisão de tarefas proposto por Oliveira (2014). A figura 13 mostra a relação do modelo proposto com os trabalhos que serviram como base para a sua implementação.

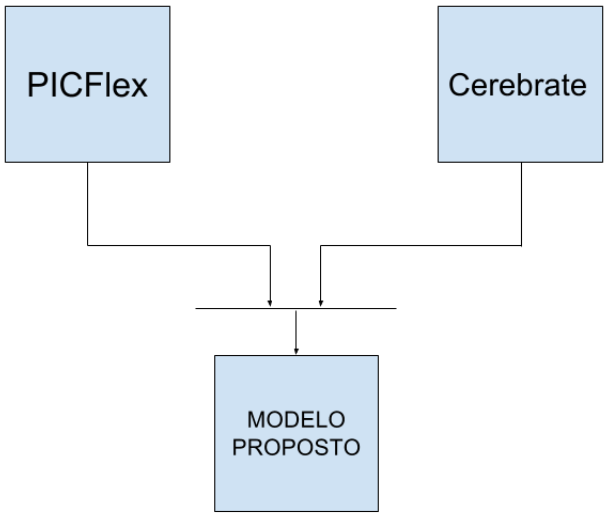


Figura 13: Visão geral da relação entre modelo proposto e os trabalhos que serviram como base para sua implementação.

O trabalho de Oliveira (2014) utilizou a ideia de divisão das responsabilidades em ministérios, onde cada um deles realizava uma rotina de atividades específica. Porém, o gerenciamento da linha de produção era feito de forma não automatizada, de forma

que era necessário especificar cada item que seria construído. O modelo que será proposto neste capítulo realiza uma melhoria no ministério da indústria, de forma que as produções passem a ter um sentido paralelo e automatizado. Para isso, é necessário utilizar a ideia das políticas de investimento, para que o ministério da economia realize a divisão dos recursos para cada classe definida.

A figura 14 mostra uma visão geral dos relacionamentos entre os ministérios definidos no trabalho de Oliveira (2014). Estão destacados em verde os ministérios que foram alterados e que contribuíram para a implementação do modelo proposto. O ministério da economia é de grande importância para o funcionamento do sistema, pois a partir dele, os outros ministérios fazem consultas sobre os recursos disponíveis para que possam realizar suas devidas rotinas. É possível retirar, por exemplo, o ministério de tecnologia e continuar executando as rotinas dos outros ministérios, mas é impossível retirar o ministério da economia e continuar executando as rotinas do ministério da indústria e do ministério de infraestrutura.

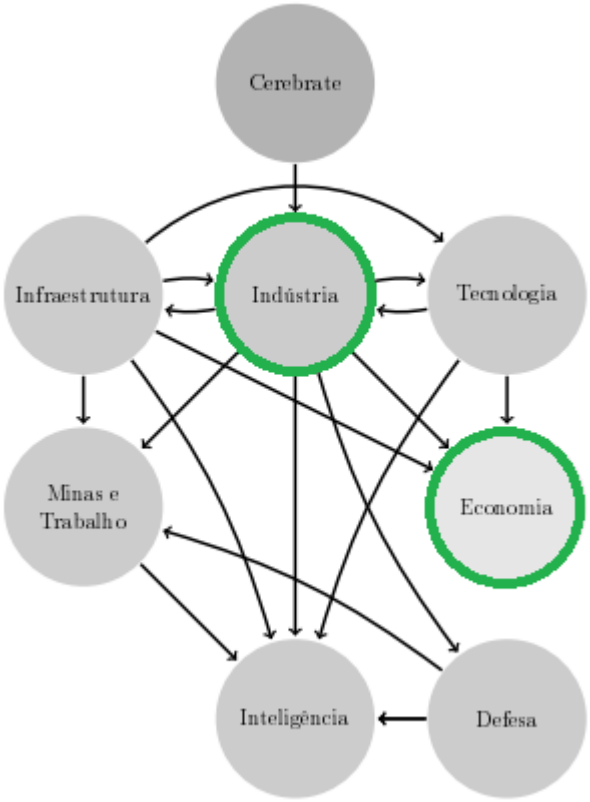


Figura 14: Relação entre os ministérios definidos no Cerebrate, destacando os dois ministérios que foram modificados para a implementação do modelo proposto.

Nas próximas seções serão mostradas as melhorias feitas em cada ministério, bem como o funcionamento do modelo proposto.

## 5.1 Aplicação das Políticas de Investimento

Oliveira (2014) dividiu as atividades do agente em diferentes ministérios, sendo possível explorá-los separadamente. Dessa forma, é possível trabalhar, por exemplo, no gerenciamento de recursos sem se preocupar com estratégias de combate, se preocupando apenas com as responsabilidades do gerenciamento de recursos.

Para criar um modelo para o gerenciamento de recursos utilizando algoritmos de aprendizagem por reforço através de políticas de investimentos apresentadas no trabalho de Souza (2013), será utilizada a divisão das responsabilidades em ministérios oferecidas no trabalho de Oliveira (2014).

### 5.1.1 Rotina do Ministério da Economia

Como mostrado anteriormente, o ministério da economia é responsável por gerenciar a economia do jogador. Quando é necessário saber quanto de recurso está disponível, é a ele que os outros ministérios devem consultar. Então fica claro que é nesse ministério que deve ser implantada a ideia das Políticas de Investimento, já que estas serão as responsáveis por direcionar os gastos dos recursos do jogador.

As políticas definem métricas que direcionam os gastos do jogador de acordo com as classes previamente definidas. A Figura 15 ilustra através de um exemplo, como essa ideia foi modelada.

Para realizar a divisão dos gastos dos recursos, foi projetado um sistema de acúmulo e distribuição para cada classe. Assim, toda vez que for coletado um novo valor de recurso, ele será dividido entre as classes de acordo com a porcentagem definida na política de investimento.

O exemplo da Figura 15 considera que foram definidas três classes de unidades para o modelo: Unidades Aéreas , Unidades Terrestres e Trabalhadores. Quando um trabalhador coletar uma certa quantidade de minério, o Ministério da Economia irá consultar os valores da política de investimento que está em vigor e dividir essa quantidade entre as classes. No mesmo exemplo, consta que 25% dos recursos coletados devem ser gastos com as unidades aéreas, 0% com as unidades terrestres e 75% com trabalhadores. Então, caso 8 unidades de minério sejam coletadas, 2 irão para as unidades aéreas, nenhuma irá para as unidades terrestres e 6 irão para os trabalhadores. Assim, ao longo do tempo, serão acumuladas quantidades distintas de recursos para cada classe. Feito isso, os outros

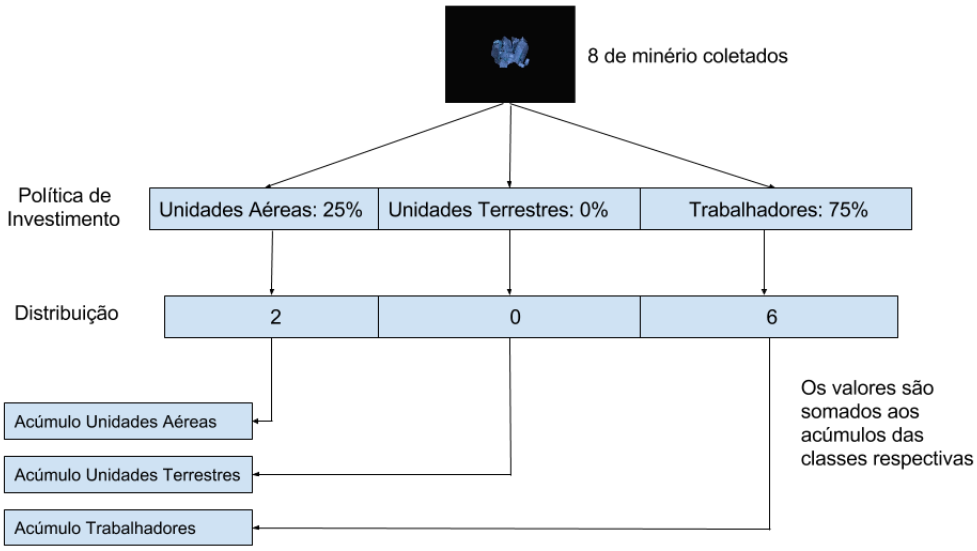


Figura 15: Exemplo de funcionamento da Política de Investimento no Ministério da Economia

ministérios passam a consultar essas quantidades, ao invés a quantidade total que era informada anteriormente.

Com esta abordagem, é possível realizar uma divisão, de modo que os gastos se direcionem para um determinado conjunto de unidades. No exemplo dado, 75% dos recursos coletados serão direcionados para unidades Trabalhadoras, o que significa que essa política de investimento está preocupada em produzir muito mais unidades trabalhadoras do que unidades aéreas, por exemplo.

### 5.1.2 Rotina do Ministério da Indústria

Foi explicado anteriormente que os recursos coletados são divididos por classes de unidades. Em adição a isso, será mostrado que a produção das unidades também será organizada por essas mesmas classes, de acordo com o acúmulo de recursos que cada uma delas tem. Mas antes, será introduzida a estrutura que agrupa as unidades em classes distintas.

#### 5.1.2.1 Estrutura da Classe de Unidade

No Starcraft há um grande número de unidades com características heterogêneas. Com intuito de realizar um melhor gerenciamento na produções das mesmas, foi criado um sistema de classes que divide as unidades em grupos com características semelhantes. Desta forma, é possível destinar os recursos para um grupo de unidades, ao invés de tratar cada uma separadamente. A definição dessas classes é feita de acordo com a estratégia adotada pelo usuário do sistema.

Para poder realizar essa classificação, foi criada uma estrutura para associar cada unidade ao seu grupo. A Figura 16 representa em forma de diagrama de classe a entidade UnitClass que é responsável por essa associação.

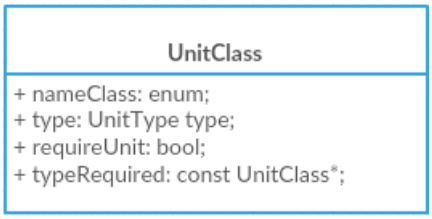


Figura 16: Estrutura que agrupa as unidades

A propriedade "nameClass", definida na classe, informa a qual grupo a unidade pertence. Por exemplo, caso uma unidade tenha em sua propriedade ”nameClass” o valor ”UnidadeTerrestre” significa que ela pertence ao grupo de unidades terrestres.

Há mais duas propriedades na estrutura. Elas serão utilizadas na rotina do Ministério de Produção que será explicada na próxima seção. Em Starcraft, para construir determinada unidade, às vezes é exigido que alguma estrutura seja antecipadamente construída. A propriedade ”requireUnit” é um booleano que informa se a unidade em questão necessita ou não de alguma estrutura. Em caso afirmativo, a propriedade "typeRequired", que é um ponteiro para a própria estrutura UnitClass, será atribuída com a unidade que é exigida.

O sistema então associa todas as unidades do jogo às suas classes previamente definidas e as guarda em um vetor estático. Assim, toda vez que o sistema precisar consultar a classe de alguma unidade, será feita uma consulta nesse vetor.

Um ponto importante a ressaltar é que as classes utilizadas na política de investimento são as mesmas classes que são definidas na propriedade ”nameClass”. Ou seja, cada unidade do jogo, será associada à uma classe e esta terá uma porcentagem dos recursos coletados para o jogador. Se, por exemplo, foram definidas três classes, consequentemente todas as

unidades serão classificadas com uma dessas três. A política de investimento então dividirá os recursos para as três classes, de acordo com suas porcentagens.

#### 5.1.2.2 Linhas de Produção

Foi explicado anteriormente que cada unidade do jogo é agrupada em alguma classe. Assim sendo, foi criada no Ministério da Indústria uma fila de produção para cada uma delas.

No modelo Cerebrate, a produção é única e sequencial. Portanto, é preciso definir quais itens serão produzidos e qual será a sequência de suas produções item por item. No modelo proposto neste documento, a produção passa a ter um sentido paralelo, de forma que itens de diferentes classes não precisam ficar em espera por itens que não fazem parte de seu gênero. Imagine, por exemplo, que se queira construir uma torre de defesa aérea, mas sua fila de produção esteja com vários itens diferentes que já foram inseridos anteriormente. A torre deverá esperar para que todos os itens sejam construídos até que chegue a sua vez. Neste modelo, ao solicitar a construção de uma torre de defesa, ela será colocada na fila de produção da sua classe, onde haverá apenas itens com características semelhantes.

A Figura 17 apresenta uma visão geral de como funciona a produção dos itens no Ministério da Indústria.

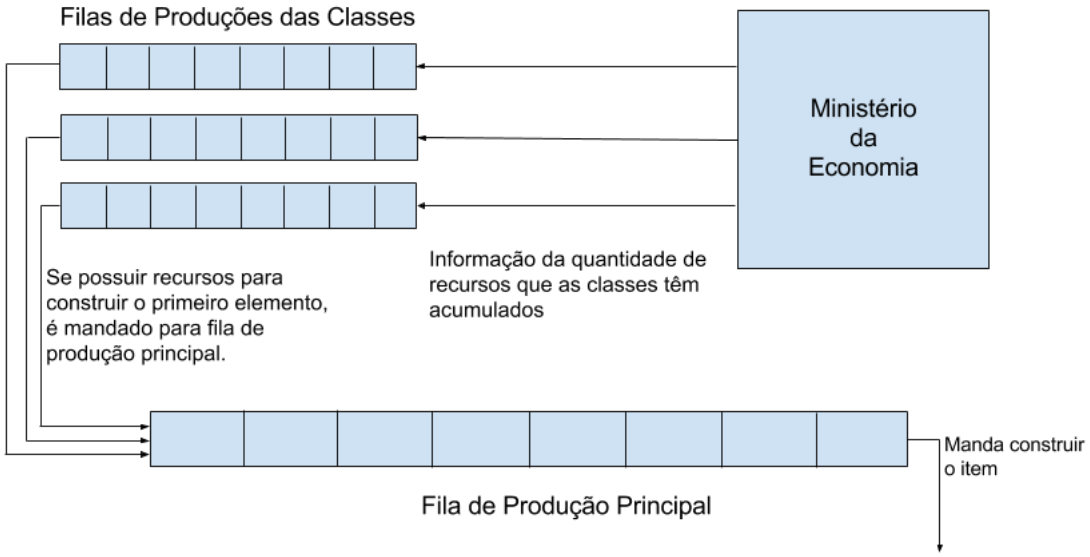


Figura 17: Visão geral do funcionamento das produções no Ministério da Indústria

O Ministério da Economia divide os recursos de forma diferente para cada classe. As-

sim sendo, as filas de produções solicitam informação constantemente sobre as economias de suas classes. No momento em que uma das filas de produções possuir um montante suficiente de recursos exigidos para construir o primeiro elemento, esse ministério solicitará o item para a fila de produção principal, onde lá será enviado para construção. A fila de produção modelada nesta proposta funciona da mesma forma que no Cerebrate, ou seja, o primeiro elemento da lista é enviado para construção e assim segue sucessivamente para os próximos itens. A diferença é que quando o item chega na fila de produção principal, ele não ficará preso esperando acumular recurso para construir. A espera é feita na própria fila de produção da classe daquele elemento. Então quando o item chega na fila principal, ele já vai ser mandado para a construção imediatamente.

Sabendo disso, é possível perceber que a população de unidades do jogador será proporcional à porcentagem de sua classe na política de investimento. Se for considerado, por exemplo, que a classe de unidades terrestres deve receber 80% dos recursos coletados, isso significa que a fila de produção das unidades terrestres enviará seus itens para construção com muito mais frequência do que as outras e, consequentemente, haverá muito mais unidades terrestres do que qualquer outra.

Desta forma, fica claro que é de grande importância saber definir a porcentagem de recursos que deve ser aplicada para cada classe na política de investimento. Além disso, também fica claro que essas porcentagens provavelmente devam variar durante a duração da partida, pois em um determinado momento podemos precisar, por exemplo, de mais unidades terrestres do que unidades aéreas e em outro momento, o contrário.

#### 5.1.2.3 Adicionando Itens nas Filas de Produções

Após explicar como funcionam as filas de produções de forma geral, é possível explicar como cada uma delas são populadas com suas respectivas produções.

Cada classe definida deve possuir um conjunto de unidades com características comuns. Portanto, o sistema verifica quando uma fila de produção está vazia e então sorteia uma dentre todas as unidades daquela classe para ser produzida. No momento em que a unidade é enviada para construção, a lista fica novamente vazia e então é sorteada outra unidade da classe para ser produzida, podendo, inclusive, ser a mesma.

Quando uma unidade é sorteada para produção, ela pode não ser produzida diretamente, pois pode existir alguns "pré-requisitos" para a sua construção. A Figura 18 ilustra um exemplo desse comportamento.

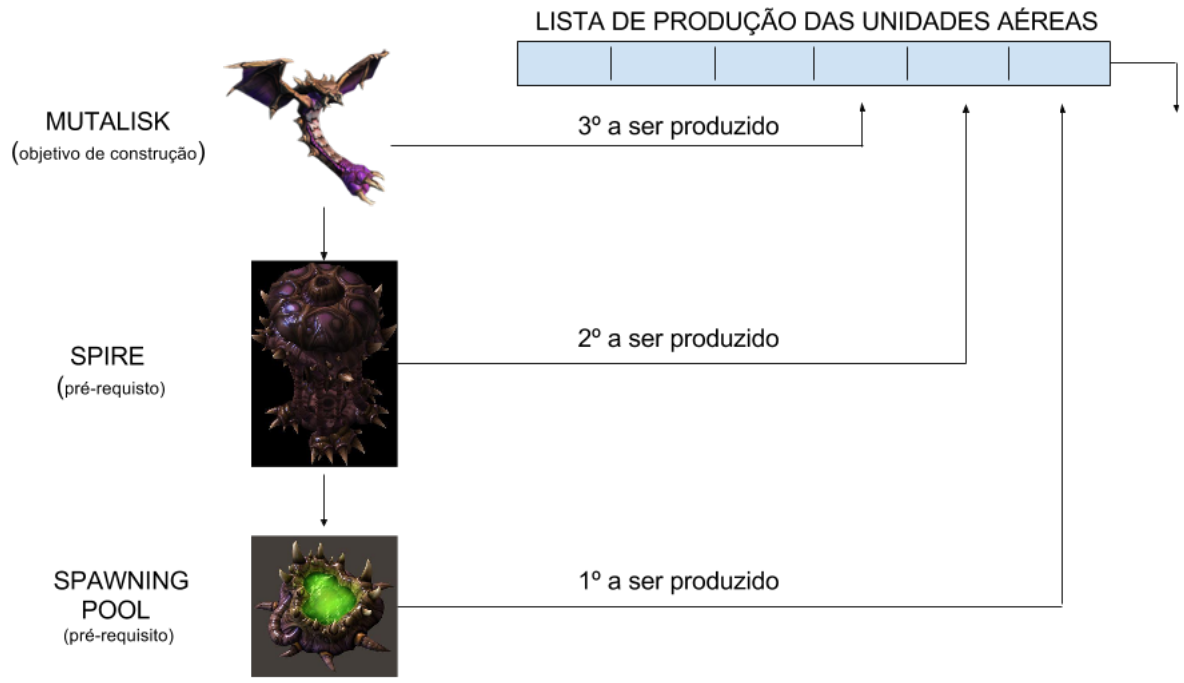


Figura 18: Ordem de produção das unidades de uma classe

Ao explicar como ocorre o funcionamento da estrutura das classes na Figura 16, duas propriedades que se referiam aos ”pré-requisitos” das unidades foram comentadas. A propriedade ”requireUnit” informa se uma unidade tem um pré-requisito para construção, enquanto que a propriedade ”typeRequired” informa qual é esse pré-requisito.

Através dessas propriedades, quando o Ministério da Indústria precisa adicionar uma unidade na sua respectiva lista de produção, essa adição é feita de forma bottom-up e recursiva para todos os pré-requisitos que são exigidos para a construção dessa unidade.

A Figura 18 ilustra um exemplo em que um Mutalisk, unidade da raça Zerg em Starcraft, foi classificado como unidade aérea e foi sorteado para ser produzido. Assim, o sistema verifica que antes de construir o Mutalisk, é preciso construir um Spire por ser um pré-requisito. Quando o Spire é adicionado na lista, o sistema verifica que é necessário um Spawning Pool devido a também ser um pré-requisito. Finalmente, é verificado que não existe pré-requisito para a adição do Spawning Pool na lista de produção e assim sendo, o sistema o adiciona. Feito isso, o processo de retorno da recursão adiciona o Spire e depois o Mutalisk.

Quando todos da lista forem construídos, outra unidade daquela classe é sorteada para construção e o algoritmo irá trabalhar da mesma maneira. Porém, o sistema verifica se os pré-requisitos dessas unidades já foram construídos e adiciona apenas aqueles que ainda não foram. Imagine que depois de construir o Mutalisk, por coincidência, ele seja

sorteado para produção novamente. Assim, o algoritmo verifica que seus pré-requisitos já foram construídos e ele é adicionado na lista diretamente.

## 5.2 Produções em Comum Entre Classes

Ao se classificar as unidades em diferentes grupos, criou-se a possibilidade de unidades de grupos diferentes possuírem pré-requisitos em comum. Isso pode ser considerado um problema se observarmos o fato de que estaremos dividindo os recursos para diferentes classes, mas que em um determinado momento elas estarão acumulando recursos para fabricar um mesmo item em comum. Isso poderá causar um atraso, além de redundância.

Para tratar esse problema, foi desenvolvido um esquema em que quando há uma detecção de produção em comum entre diferentes classes, o Ministério da Economia passa a doar os recursos delas para um mesmo fim.

A Figura 19 mostra um exemplo de produção em comum entre duas classes, sendo elas unidades terrestres e unidades aéreas.

Se, por exemplo, um Mutalisk for sorteado para ser produzido na lista de unidades aéreas e um Zergling for sorteado para ser produzido na lista de unidades terrestres (na figura eles são representados pelos itens mais a esquerda nas listas de produções), o sistema adiciona inicialmente os pré-requisitos de construção na fila, para depois adicionar a unidade em questão.

Quando o Ministério da Indústria percebe que há um produto em comum em duas filas de produções diferentes, ele passa a tratá-lo de forma conjunta. Para isso, ele avisa ao Ministério da Economia quais são as classes que possuem tal produção. Quando o Ministério da Economia recebe as informações, ele passa a doar os recursos das classes que estão envolvidas para um mesmo fim, até que haja recursos suficientes para enviar o item em comum para a construção. Quando isso ocorre, o item é removido das listas de produções envolvidas e o Ministério da Economia volta a distribuir seus recursos de forma padrão.

## 5.3 Aplicação de Aprendizagem por Reforço no Modelo

Foi mostrado até agora que o gerenciamento de recursos pode ser modelado pelas políticas de investimento e que apenas com a definição das classes e com a definição da

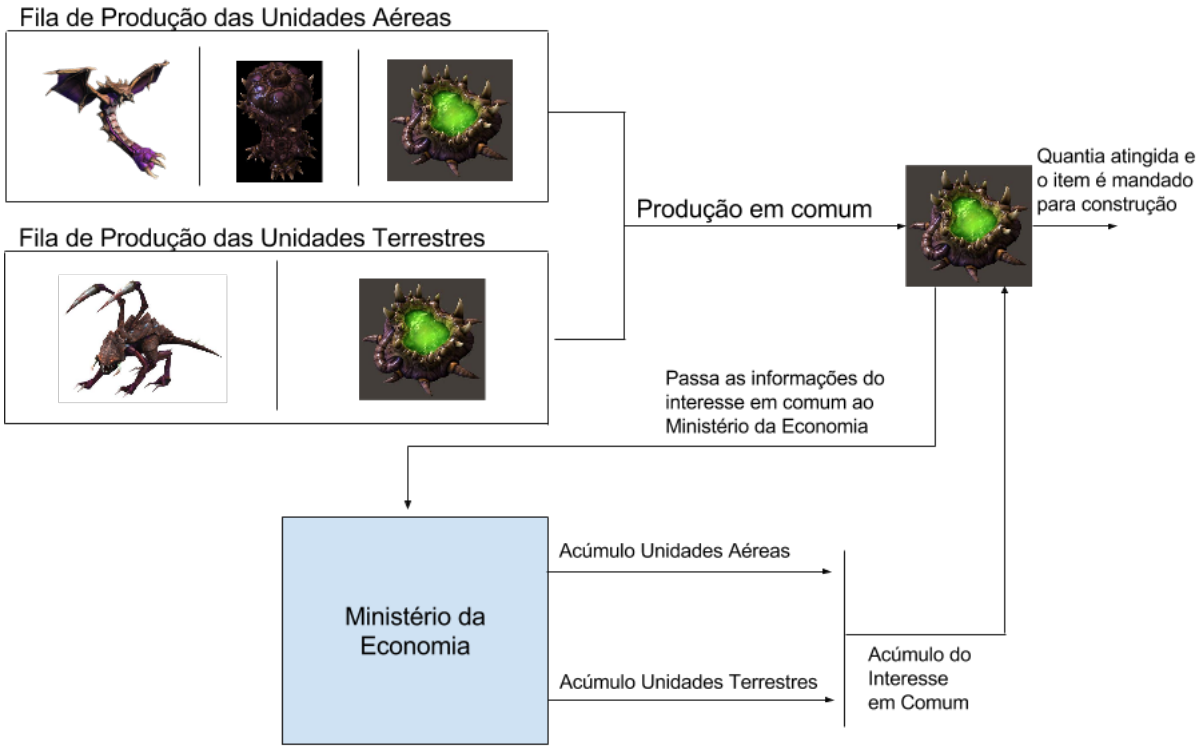


Figura 19: Exemplo de uma produção em comum entre duas classes diferentes

política é possível distribuir os recursos para criar uma população para um determinado fim. Por exemplo, se for criada apenas uma classe de unidades aéreas e uma classe de unidades terrestres e definido que os recursos coletados serão 100% para as unidades terrestres, é criada uma população apenas com as unidades terrestres. De mesma forma, se for definido que 50% dos recursos serão para unidades aéreas e 50% para unidades terrestres, é criada uma população com metade das unidades terrestres e metades das unidades aéreas.

Ao definir esse modelo, deve haver uma preocupação com alguns tipos de problemas. O primeiro é com relação à definição da política ideal para atingir um determinado objetivo. Por exemplo, qual é a política que deve ser empregada para conseguir destruir um inimigo em uma partida de 10 minutos? Qual é a política que deve ser empregada para conseguir realizar uma defesa de ataques em massa? O segundo problema diz respeito à mudança da política durante a partida. Não basta definir apenas uma e aplicá-la durante toda a partida, mas deve ser feita uma transição entre elas até o final. Por exemplo, se o objetivo fpr criar unidades fortes e em massa para realizar um ataque depois de um período mais longo, será preciso primeiro investir em unidades construtoras e coletoras (drones) para coletar recursos e construir o necessário, para depois investir todos os seus recursos em unidades de combate. Além disso, se por um determinado tempo a estratégia tomada for

de realizar construções defensivas e, a partir de um certo momento, for decidido que deve haver um fortalecimento do exército para realizar um ataque, os investimentos devem mudar. Assim, durante uma partida deve haver uma transição entre diversas políticas de investimento para atingir um determinado objetivo.

Neste trabalho foi aplicado o algoritmo Sarsa de aprendizagem por reforço explicado anteriormente, mas também poderia ter sido algum outro algoritmo. Foi escolhida a área de aprendizagem por reforço pelo fato de não necessitar uma especificação de como as tarefas devem ser feitas. Basta definir os estados do ambiente, as ações que o agente pode realizar e uma função de reforço para que a aprendizagem possa ocorrer por meio dos acertos e erros.

Para adaptar o formato da aprendizagem por reforço ao modelo de gerenciamento de recursos proposto, é preciso primeiramente definir os estados do ambiente e as ações que podem ser tomadas pelo agente em cada estado. Foi considerado como agente do sistema o agente gerenciador de recursos, que é responsável por definir a política de investimento atual do sistema.

Diferentes representações do ambiente podem ser formuladas e fica a critério do usuário do modelo escolher quais variáveis do ambiente usar para realizar essa representação.

As ações do agente são possíveis configurações da política de investimento. Um conjunto de políticas diferentes são armazenadas em memória para que a cada estado visitado pelo agente, ele realize a escolha de uma. Com isso definimos o básico da aprendizagem por reforço, onde a cada instante de tempo, o agente se encontra em um determinado estado e pode realizar uma ação, que nada mais é do que a escolha da configuração da política de investimento.

A Figura 20 mostra de modo geral como um agente verifica o estado do ambiente e escolhe uma dentre as ações possíveis existentes na memória. No exemplo da figura existem 4 ações possíveis para o agente, cada uma representada por uma linha da tabela. O agente então verifica em qual estado está e escolhe uma delas para colocar em ação.

Outro aspecto importante para a aprendizagem por reforço é a determinação da função de reforço. Isso é de grande importância para o resultado final, pois para diferentes objetivos devem ser considerados diferentes recompensas. Por exemplo, se quisermos maximizar o número de unidades criadas em um determinado tempo, podemos recompensar positivamente toda vez que uma unidade for criada. Assim, as políticas que criarem mais unidades vão sendo consideradas como sendo as melhores. Se o objetivo for destruir uni-

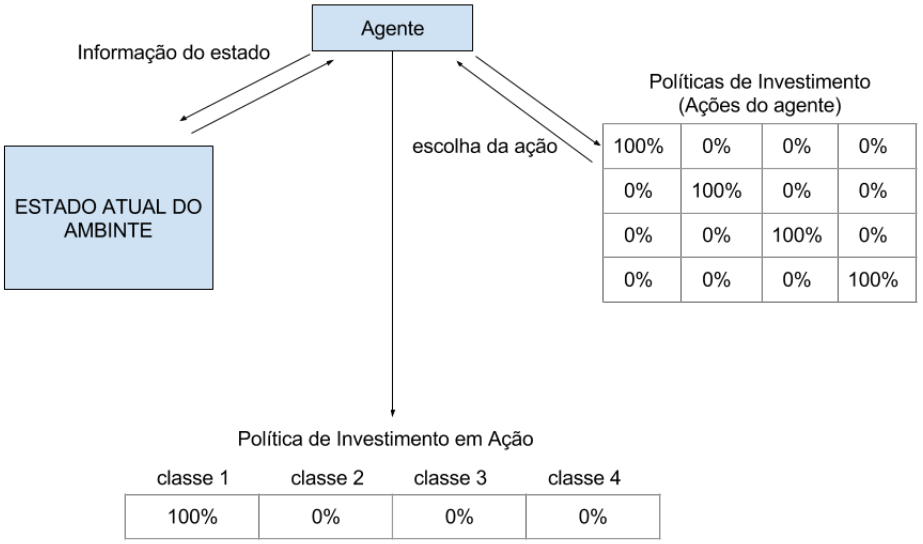


Figura 20: Modelo geral de percepção do ambiente e escolha da ação do agente no modelo proposto

dades inimigas, podemos considerar como recompensa positiva toda vez que uma unidade inimiga for destruída. Assim, as políticas que destruírem mais unidades serão consideradas como sendo as melhores.

Depois de definidos os estados, as ações e a função de reforço, é possível empregar o algorítmo Sarsa(0) ou Sarsa($\lambda$) no sistema, que seguirá os seguintes passos em cada episódio executado:

- O agente verifica o estado atual do ambiente.
- Para o estado atual, ele escolhe uma ação;
- A função de reforço calcula a recompensa ou a punição gerada após a transição de um estado para outro. O reforço depende do objetivo que deve ser atingido, por exemplo, o número de unidades destruídas.
- Quando a transição do estado do ambiente é efetuada, o sistema atualiza a função valor Q para o estado-ação que estava em vigor utilizando o cálculo da função de reforço efetuado no passo anterior. A atualização é feita de acordo com o algorítmo Sarsa(0) ou Sarsa($\lambda$) empregado;

- O algorítmo repete o procedimento para o novo estado em vigor até atingir o estado final.

No próximo capítulo serão apresentados os experimentos realizados com o intuito de validar o modelo de gerenciamento de recursos proposto, assim como os resultados por ele obtidos.

# 6 Experimentos e Resultados Obtidos

O objetivo deste capítulo é apresentar a metodologia empregada para modelagem dos experimentos, bem como os resultados obtidos pela aplicação dos algoritmos Sarsa(0) e Sarsa($\lambda$) no modelo proposto.

## 6.1 Metodologia de Experimentação

Todos os experimentos feitos neste trabalho foram focados na raça Zerg. Criamos ambientes em que o agente deve desenvolver uma população distribuindo seus recursos para as classes definidas com o objetivo de destruir uma determinada população do inimigo. A Figura 21 mostra a população inicial que o agente possui em todas as partidas jogadas. Ele começa apenas com drones e a base principal e deve distribuir seus recursos de forma que crie uma população suficientemente boa para destruir as unidades do inimigo.

### 6.1.1 Definição das Classes

Para o modelo proposto é preciso definir as classes de unidades que serão trabalhadas no ambiente. Para isso, consideramos 4 tipos de unidades dos zergs: Drone, Sunken Colony, Zergling e Hydralisk. Drone é a unidade que coleta os recursos e realizam construções. Sunken Colony é a torre de defesa que ataca unidades terrestres do inimigo. Zergling é a unidade de combate mais simples dos Zergs, que é barata e de rápida produção e ataca corpo a corpo unidades terrestres do inimigo. Hydralisk é uma unidade terrestre que ataca inimigos à distância, podendo inclusive atacar unidades aéreas.

Figura 21: População inicial de unidades que o agente começa as partidas nos experimentos feitos. 4 drones e a base principal

### 6.1.2 Cenários

Criamos dois cenários para a realização dos experimentos, ambos utilizando as unidades da figura 21, variando apenas as unidades que serão enfrentadas.

#### 6.1.2.1 Cenário 1: IA Contra Marines

O primeiro cenário que criamos para testar o modelo de gerenciamnto de recursos auxiliado por algoritmos de aprendizagem por reforço consta com 20 marines que devem ser destruídos pelo agente. A Figura 22 mostra as unidades que ficam posicionadas no mapa. Os marines são unidades terrestres que atacam à distância. Toda partida o agente começa com a configuração mostrada na Figura 21 e deve desenvolver uma população para que quando alcance o estado final, ataque e destrua todos os marines. Essas unidades ficam paradas no outro lado do mapa. No estado final, o agente leva todos as suas unidades de combate para o ataque.

As partidas duram, em média, 8 minutos. Como o inimigo possui uma população de 20 marines, o agente deve aprender a construir uma população forte em pouco tempo. Nessas condições fica claro que a produção de torres de defesa fica inviável, já que elas não vão contribuir para um ataque massivo no outro lado do mapa. Assim, políticas de investimento que possuam valores altos de investimento para torres de defesa vão sendo consideradas como políticas ruins, enquanto que políticas que produzem zerglings e

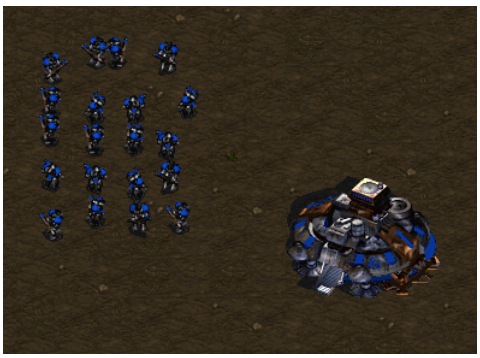

Figura 22: Unidades do primeiro cenário de teste(20 marines)

hydralisk vão sendo consideradas como boas. Além disso, a produção de drones é essencial no início da partida, mas políticas que continuam investindo em drones da metade para o final, gastam recursos que poderiam ser utilizados para produção de unidades de combate e também não terão muito sucesso. Então, o ideal é que haja uma produção de drones no início da partida para que haja uma boa taxa de coleta de recursos, mas que com pouco tempo o investimento seja totalmente guiado para unidades de combate.

#### 6.1.2.2 Cenário 2: IA Contra Wraiths

O segundo cenário é feito sobre o mesmo mapa, porém utilizamos 15 Wraiths que devem ser destruídos. Wraith é uma unidade de combate aéreo dos Terranos que também pode atacar unidades terrestres. A Figura 23 mostra as unidades posicionadas no mapa.

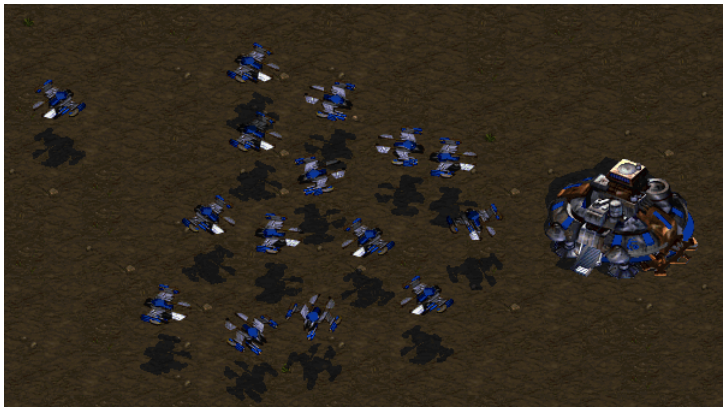

Figura 23: Unidades do segundo cenário(15 Wraiths)

O agente começa com a mesma configuração da situação anterior e deve desenvolver uma população que consiga destruir todas as unidades adversárias. Para este caso, só os Hydralisks têm capacidade de destruir os Wraiths, pois são os únicos que apresentam maior probabilidade de atacar unidades aéreas. Portanto o número de possibilidades de políticas que conseguem atingir o resultado ótimo é bem menor, tendo em vista que

na situação anterior diferentes tipos de unidades conseguiam atacar os inimigos e nesta situação apenas um tipo de unidade consegue. Assim, espera-se que o agente desenvolva rapidamente uma população relativamente grande de Hydralisks.

### 6.1.3 Modelagem da Aprendizagem por Reforço

Com o intuito de inicialmente validar o modelo proposto com uma representação simples do ambiente, foi decidido utilizar apenas o tempo como variável para representação do ambiente. Dessa forma, cada instante de tempo de uma partida pertence a um determinado estado. A Figura 24 mostra como foi adaptado o modelo que está representado na Figura 20 para ser representado com o tempo.

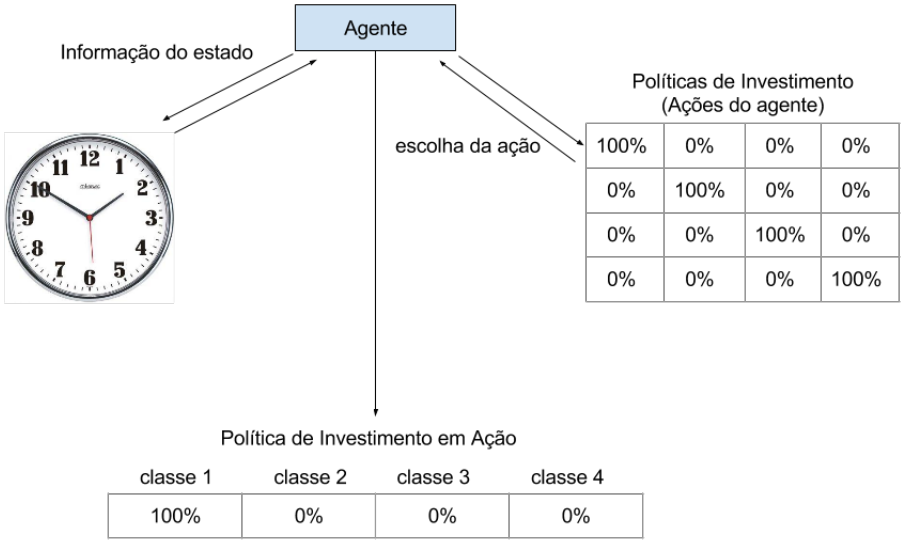


Figura 24: Modelo de representação do ambiente utilizando o tempo como variável

Foram definidos quatro estados possíveis do ambiente, onde cada estado possui um intervalo de dois minutos. Os primeiros dois minutos da partida pertencem ao estado inicial. De dois em dois minutos o estado muda para o próximo até atingir o estado final, totalizando oito minutos de partida. Quando o estado final é finalizado, todas as unidades de combate são enviadas para atacar em massa as unidades do adversário. A quantidade de unidades destruídas é o que representará a recompensa dada no processo de aprendizagem por reforço.

Definimos 55 configurações de políticas de investimento que representam as ações do agente.

Tabela 3: Ações do agente (Políticas de Investimento)

| Drone | Sunken C. | Zergling | Hydralisk | Drone | Sunken C. | Zergling | Hydralisk |
|---|---|---|---|---|---|---|---|
| 100% | 0% | 0% | 0% | 20% | 20% | 0% | 60% |
| 80% | 20% | 0% | 0% | 20% | 20% | 40% | 20% |
| 80% | 0% | 20% | 0% | 20% | 20% | 20% | 40% |
| 80% | 0% | 0% | 20% | 20% | 0% | 80% | 0% |
| 60% | 40% | 0% | 0% | 20% | 0% | 0% | 80% |
| 60% | 0% | 40% | 0% | 20% | 0% | 60% | 20% |
| 60% | 0% | 0% | 40% | 20% | 0% | 20% | 60% |
| 60% | 20% | 20% | 0% | 20% | 0% | 40% | 40% |
| 60% | 20% | 0% | 20% | 0% | 100% | 0% | 0% |
| 60% | 0% | 20% | 20% | 0% | 80% | 20% | 0% |
| 40% | 60% | 0% | 0% | 0% | 80% | 0% | 20% |
| 40% | 0% | 60% | 0% | 0% | 60% | 40% | 0% |
| 40% | 0% | 0% | 60% | 0% | 60% | 0% | 40% |
| 40% | 40% | 20% | 0% | 0% | 60% | 20% | 20% |
| 40% | 40% | 0% | 20% | 0% | 40% | 60% | 0% |
| 40% | 20% | 40% | 0% | 0% | 40% | 0% | 60% |
| 40% | 20% | 0% | 40% | 0% | 40% | 40% | 20% |
| 40% | 0% | 40% | 20% | 0% | 40% | 20% | 40% |
| 40% | 0% | 20% | 40% | 0% | 20% | 80% | 0% |
| 20% | 80% | 0% | 0% | 0% | 20% | 0% | 80% |
| 20% | 0% | 80% | 0% | 0% | 20% | 40% | 40% |
| 20% | 0% | 0% | 80% | 0% | 0% | 80% | 20% |
| 20% | 60% | 20% | 0% | 0% | 0% | 20% | 80% |
| 20% | 60% | 0% | 20% | 0% | 0% | 60% | 40% |
| 20% | 40% | 40% | 0% | 0% | 0% | 40% | 60% |
| 20% | 40% | 0% | 40% | 0% | 0% | 100% | 0% |
| 20% | 40% | 20% | 20% | 0% | 0% | 0% | 100% |
| 20% | 20% | 60% | 0% | | | | |

Cada linha da Tabela 3 representa uma ação possível que o agente pode tomar em um determinado estado, ou seja, cada linha representa uma configuração diferente da política de investimento. Como tratamos as políticas como porcentagem, o número total de possibilidades é infinito, então foi preciso gerar esse subconjunto pré-definido de políticas possíveis para conseguir resultados satisfatórios.

Pois bem, o agente então a cada dois minutos da partida escolhe uma dentre todas essas políticas possíveis para investir seus recursos. Por exemplo, nos dois minutos iniciais ele pode escolher a ação 1 da tabela e investir todos os seus recursos em drones. Quando alcançar dois minutos de partida, ele pode escolher, por exemplo, a ação 2 e diminuir seu

investimento nos drones para 80% e investir 20% em Sunken Colony. De mesma forma acontece para os próximos estados até chegar no estado final no qual será realizado um ataque massivo a fim de obter a recompensa pelo número de unidades destruídas. O objetivo então é encontrar a distribuição dos recursos ideal para construir uma população de unidades que possa destruir um conjunto de unidades do inimigo no cenário previsto. Nas próximas seções mostraremos os cenários que foram criados e os resultados obtidos.

Esse procedimento permite que a cada episódio executado o sistema teste diferentes políticas para cada estado e aprenda ao longo do tempo quais são as melhores para cada um deles. Perceba que a cada execução de um episódio, ou seja, a cada execução de uma partida, depois que o agente passa de um estado para o outro, ele não pode retornar, já que representamos os estados do ambiente apenas com o tempo. Mostraremos que essa abordagem permite que o sistema aprenda a distribuir seus recursos para atingir o objetivo de destruir uma população de unidades estáticas no mapa. O sistema aprende para cada instante de tempo qual deve ser a política que deve empregar para atingir tal objetivo. A ideia principal é mostrar que é possível aplicar a técnica de aprendizagem por reforço no modelo através de uma representação simplificada do ambiente, para que em trabalhos futuros possam ser feitos novos estudos da aplicação em ambientes mais complexos, com representações mais completas.

### 6.1.4 BWAPI

Para a implementação do modelo, utilizamos a BWAPI (Brood War Application Programming Interface), que é um framework gratuito para a criação de módulos de IA. Essa ferramenta possibilita que os programadores obtenham informações do ambiente do jogo e realizem ações sobre as unidades. A partir desta ferramenta, o Starcraft virou um ótimo laboratório para técnicas de Inteligência Artificial, pois além de seu ambiente complexo, é possível realizar diversos testes e observar os resultados sem qualquer preocupação com falhas, pois o ambiente é virtual e não causa nenhum tipo de prejuízo caso a técnica simulada falhe.

## 6.2 Resultados

Para cada cenário foram executados os algorítmos Sarsa(0) e Sarsa($\lambda$). As próximas seções mostram os resultados obtidos.

### 6.2.1 Resultados Obtidos no Cenário 1

Primeiramente foram executadas cerca de 8 mil partidas com o algoritmo Sarsa(0), com uma taxa de aprendizagem $\alpha = 0.1$ , a taxa de desconto $\gamma = 1$ e o grau de aleatoriedade $\epsilon$ começando com 0.9 e diminuindo 10% a cada 200 partidas, para que assim tenha uma taxa de exploração alta no início, mas que vai diminuindo rapidamente com o tempo.

A Figura 25 apresenta um gráfico referente à evolução das médias do reforço a acumulativo para cada 10 partidas executadas. Cada marine destruído gera uma recompensa de 100 pontos, totalizando 2000 como pontuação máxima. Ou seja, se forem destruídos, por exemplo, 5 marines em uma partida, haverá uma recompensa de 500 pontos, e se forem destruídos os 20 marines, haverá uma recompensa de 2000 pontos. Essa pontuação é gerada pela pela função getKillScore pertencente à BWAPI. Diferentes tipos de unidades geram diferentes tipos de pontuação.

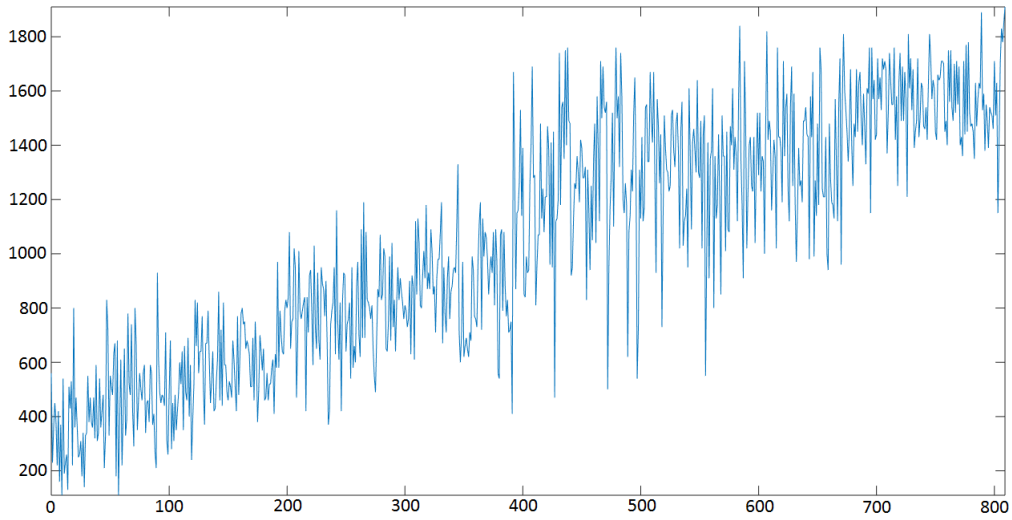


Figura 25: Média das recompensas a cada 10 partidas executas pelo Sarsa(0) no cenário da IA contra Marines. O eixo vertical indica o valor da recompensa e o eixo horizontal indica o índice da partida executada

Percebe-se que nas primeiras partidas executadas as médias das recompensas possuem valores baixos e com o tempo esses valores vão aumentando. Isso mostra que o agente foi aprendendo com o tempo quais eram as políticas que geravam mais recompensas. Um dos motivos que isso acontece é por causa do grau de aleatoriedade que vai diminuindo com o tempo. No início, o grau de aleatoriedade é alto, pois o agente não tem conhecimento sobre quais são suas melhores ações. A medida que ele vai explorando e recebendo as recompensas, o grau da aleatoriedade vai diminuindo pelo fato das melhores ações passarem

a ser conhecidas e escolhidas.

Bem, isso nos mostra que conseguimos aplicar o algoritmo Sarsa(0) no modelo para este primeiro cenário e ele conseguiu aprender ao longo do tempo políticas que retornavam recompensas melhores do que as já conhecidas, porém a exploração não foi o bastante para aprender a política ótima. Então decidimos aplicar também o Sarsa($\lambda$) e avaliar os resultados obtidos. Os testes foram feitos utilizando a mesma configuração de antes. O $\alpha = 0.1$, o $\gamma = 1$ e o $\epsilon$ começou em 0.9 e foi diminuindo 10% a cada 200 partidas executadas. Além disso, para este novo experimento também foi definido o $\lambda = 0.9$ que é a poncentagem de propagação do valor para os pares estado-ação executados. Como mostramos antes neste documento, o Sarsa ($\lambda$) tende a aprender mais rapidamente, pois as recompensas recebidas ao atingir um objetivo são propagadas para todas aquelas ações que contribuíram.

A Figura 26 mostra o resultado da execução do Sarsa($\lambda$) da mesma forma que foi mostrado o Sarsa(0). Para este caso foram executadas um pouco mais que cinco mil partidas. Perceba que este algorítmo também faz com que o agente aprenda com o tempo quais são as melhores ações do agente na medida em que as partidas vão sendo jogadas. Nas primeiras partidas, as médias das recompensas são mais baixas e, em seguida, vão aumentando com o tempo. Neste caso o agente aprendeu suas melhores ações por volta de 2500 partidas jogadas.

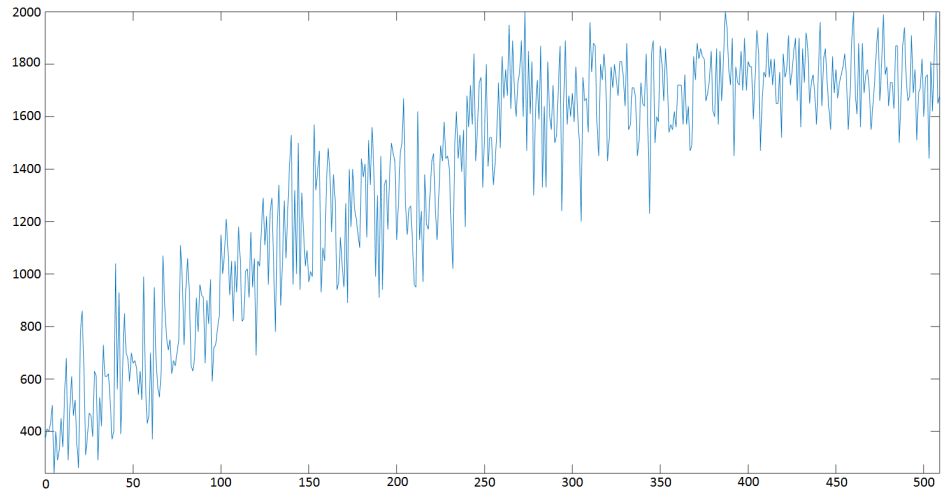


Figura 26: Média das recompensas a cada 10 partidas executadas pelo Sarsa($\lambda$) no cenário da IA contra Marines. O eixo vertical indica o valor da recompensa e o eixo horizontal indica o índice da partida executada

A Figura 27 mostra uma comparação do resultado da execução dos dois algoritmos,

através do cálculo da média das recompensas a cada 100 execuções. Lembrando que os dois algoritmos executaram números de partidas diferentes devido à diferença em velocidade de convergência. No Sarsa(0) foram jogadas 8000 partidas enquanto que no Sarsa($\lambda$) foram jogadas 5000. Pelo gráfico podemos ver que o Sarsa($\lambda$) consegue aprender mais rápido. Com ele, o agente começa a ter seus rendimentos mais altos por volta de 2500 partidas jogadas, enquanto que no Sarsa(0) seria preciso executar mais partidas com uma taxa maior de aleatoriedade para o algoritmo poder conhecer a política ótima. Além disso, podemos perceber que a evolução das recompensas ao longo das primeiras partidas é mais rápida no Sarsa($\lambda$) do que no Sarsa(0).

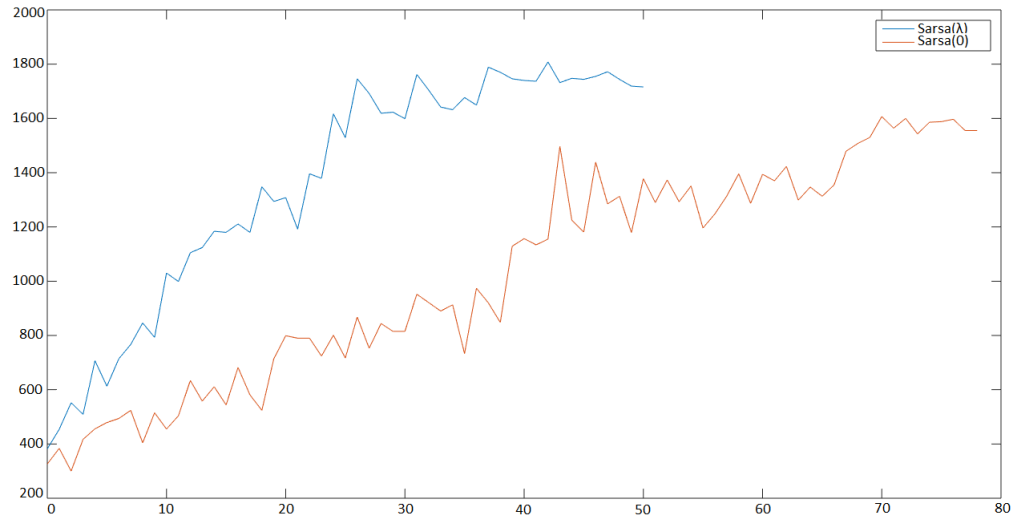


Figura 27: Comparação entre a execução do Sarsa (0) que está representado pela linha vermelha e o Sarsa($\lambda$) que está represetado pela linha azul, no cenário da IA contra Marines
. O eixo vertical indica o valor da recompensa e o eixo horizontal indica o índice da partida executada

Esses resultados mostram que para o primeiro cenário testado, os algoritmos de aprendizagem por reforço funcionaram bem, mesmo dispondo apenas de uma representação bem simples do ambiente, de forma que depois de um certo número de execuções os algoritmos tendem a encontrar suas melhores ações para atingir seus objetivos. Contudo, o Sarsa(0) precisa executar um número bem maior de episódios com uma taxa maior de exploração para poder chegar nos seus resultados ótimos. Assim, podemos ver que o Sarsa($\lambda$) tem um rendimento bem melhor do que o Sarsa(0). Porém, mesmo utilizando a mesma taxa de exploração para os dois algoritmos, o Sarsa(0), depois de executar por volta de 7000 partidas, conseguiu chegar em resultados bem próximos aos do Sarsa($\lambda$).

### 6.2.2 Resultados Obtidos no Cenário 2

Seguindo o mesmo raciocínio do experimento do cenário anterior, primeiro realizamos os testes com o algorítmo Sarsa(0). Utilizamos os mesmo parâmetros, ou seja, o $\alpha = 0.1$, o $\gamma = 1$ e o $\epsilon$ decrescendo da mesma maneira. A Figura 28 apresenta o gráfico de evolução das médias das recompensas a cada 10 partidas executadas pelo algoritmo Sarsa(0). Cada Wraith destruído gera uma recompensa de 800 pontos, totalizando em uma pontuação máxima de 12000.

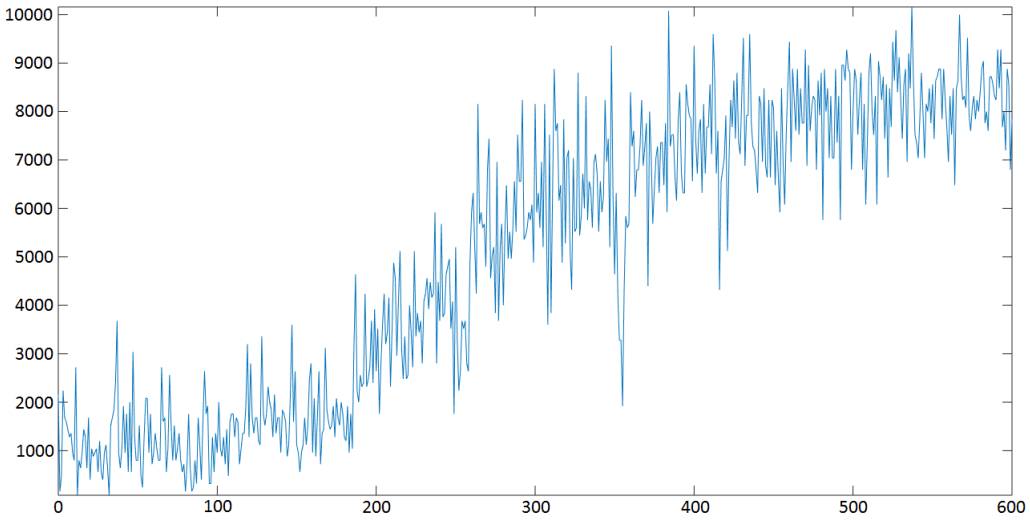


Figura 28: Média das recompensas a cada 10 partidas executadas pelo Sarsa(0) no cenário da IA contra Wraiths
. O eixo vertical indica o valor da recompensa e o eixo horizontal indica o índice da partida executada

Foram jogadas 6000 partidas e, como pode-se perceber, o Sarsa(0) conseguiu aprender políticas que aumentaram suas recompensas ao longo do tempo. Porém, a melhor política encontrada por ele resulta em uma média de recompensas de 10000 pontos, o que nos leva a concluir que não foi possível aprender a política ótima dentro do total de partidas jogadas.

De forma semelhante ao cenário anterior, o resultado mostra que o algoritmo tende a descobrir a melhor configuração, porém precisaria executar um número maior de partidas com uma taxa de exploração maior.

Executamos o Sarsa($\lambda$) utilizando os mesmo parâmetros do Sarsa(0) e com o $\lambda = 0.9$. A Figura 29 apresenta a média das recompensas a cada 10 partidas executadas. Neste caso, foram lançadas 6000 partidas e os resultados demonstram uma evolução bem mais

rápida das médias de reforço acumulativo. Por volta de 4000 partidas executadas o algoritmo demonstrou conseguir aprender uma política ótima, obtendo resultados próximos do máximo. Pode-se perceber que a escala das recompensas neste gráfico alcança os 12000 pontos, que é a pontuação máxima. Como explicamos antes, o Sarsa($\lambda$) atualiza os valores da tabela Q para todos os pares estado-ação que constribuíram para o resultado obtido durante toda a duração de uma partida.

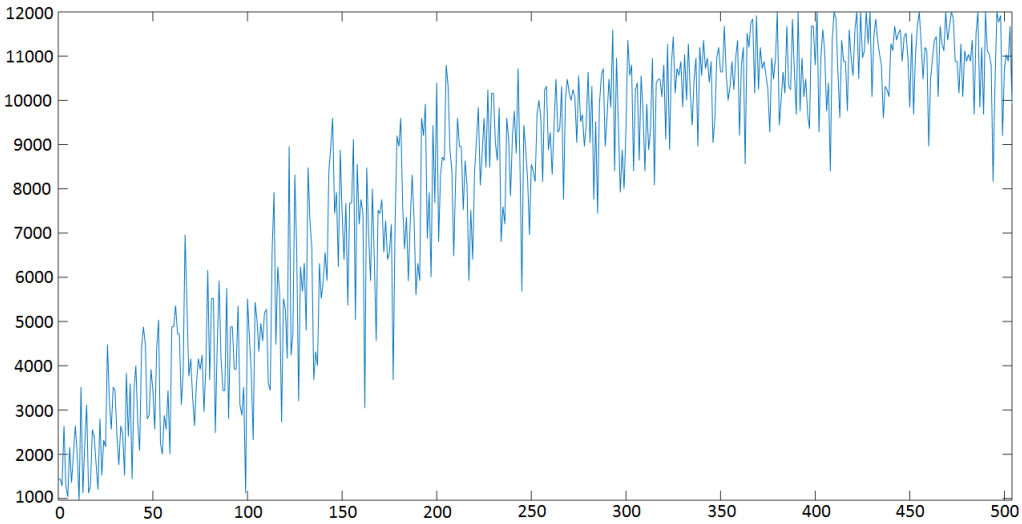


Figura 29: Média das recompensas a cada 10 partidas executadas pelo Sarsa($\lambda$) no cenário da IA contra Wraiths
. O eixo vertical indica o valor da recompensa e o eixo horizontal indica o índice da partida executada

Se imaginarmos que na primeira partida executada, o algoritmo tenha decidido, de forma aleatória, por selecionar uma sequência de políticas de investimento que atinja todas as unidades adversárias e, assim, conseguiu atingir seu objetivo. Quando usamos o Sarsa($\lambda$), todos os pares estado-ação que constribuíram para isso receberão uma parcela da recompensa. Se na segunda partida executada, selecionarmos a melhor política $\pi$ para ser executada, todas aquelas políticas de investimento que fizeram parte da primeira execução serão escolhidas. Por outro lado, se o algoritmo executado for o Sarsa(0), ao final da partida apenas o último par estado-ação receberá a recompensa e esse valor será propagado para os outros pares à medida em que outras partidas forem sendo executadas. Por esse motivo, o Sarsa ($\lambda$) consegue aprender de forma mais rápida quais são as melhores políticas.

A Figura 30 apresenta um gráfico correspondente à evolução das médias das recompensas a cada 100 partidas executadas para os dois algoritmos. O Sarsa($\lambda$) está repre-

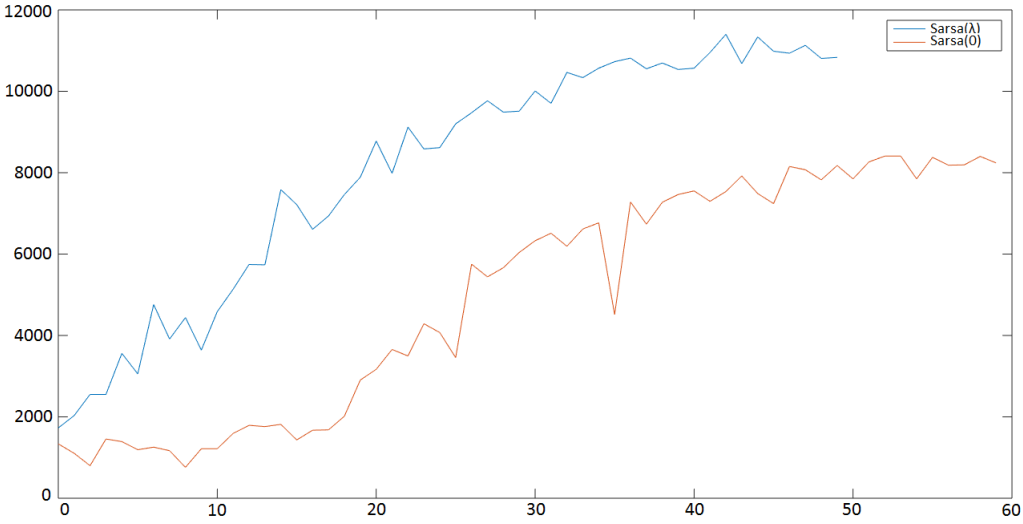


Figura 30: Comparação entre a execução do Sarsa (0) que está representado pela linha vermelha e o Sarsa($\lambda$) que está represetado pela linha azul, no cenário da IA contra Wraiths.
. O eixo vertical indica o valor da recompensa e o eixo horizontal indica o índice da partida executada

sentado pela linha azul e o Sarsa(0) está representado pela linha vermelha. No gráfico é possível perceber com clareza a diferença da evolução da curva de aprendizagem dos dois algoritmos e como o Sarsa($\lambda$) consegue aprender de forma bem mais rápida. Diferente do primeiro cenário, onde o Sarsa (0) aplicado conseguiu obter resultados próximos do ótimo, nesta situação ele não obteve o mesmo sucesso, ainda que executando um número maior de partidas. Isso acontece porque o número de possibilidades de políticas que conseguem desenvolver uma população de unidades capazes de destruir todos os Wraiths é bem menor do que na situação do primeiro cenário, onde é possível destruir os Marines usando Hydralisk ou Zergling. Neste cenário só é possível destruir os Wraiths com os Hydralisks, sendo assim, políticas de investimento que possuem grandes valores de investimento para torres e zerglings são consideradas ruins. Mas para saber disso, o agente deve utilizá-las. Portanto, utilizando o Sarsa(0) seria preciso uma taxa de exploração maior e um número maior de execuções para conseguir resultados ótimos.

Para demonstrar com um pouco mais de clareza, consideremos a Figura 31 que apresenta a população de unidades criadas pelo agente no início de seu processo de aprendizagem para este segundo cenário. A unidade destacada em vermelho é o Zergling, que é a unidade terrestre que ataca apenas unidades terrestres. A unidade destacada em azul é a torre de defesa terrestre que também só ataca unidades terrestres e não consegue se locomover para atacar uma unidade do outro lado do mapa. Perceba que a população criada

pelo agente fabricou muitos Zerglings e 5 torres. Essas unidades criadas não conseguem atacar os Wraiths que estão localizados do outro lado do mapa, então a política que gera essa população pode ser considerada como uma política ruim.


Figura 31: População de unidades desenvolvida no início da aprendizagem. A unidade destacada em vermelho é o Zergling, que não consegue atacar unidades aéreas. A unidade destacada em azul é a torre de defesa terrestre, que também ataca unidades terrestres.

No início da aprendizagem, o agente escolhe uma política de investimento aleatória para cada estado do ambiente. A medida que as partidas vão sendo executadas, a tabela Q vai sendo atualizada e o agente vai conhecendo quais são as melhores políticas de investimento para cada período de tempo definido como estado. Ao longo das primeiras partidas o grau de aleatoriedade é bem alto, portanto, diferentes populações vão sendo desenvolvidas, de forma que depois de um certo período, o agente passa a explorar novas políticas com menos frequência e usar a melhor que ele conhece.

A Figura 32 apresenta a população criada pelo agente depois de uma certa quantidade de partidas executadas. A unidade destacada em verde é o Hydralisk, que é a única unidade entre as definidas no modelo, que consegue atacar unidades aéreas. Como pode-se perceber, a quantidade de hydralisks criados é bem maior, enquanto que a quantidade de Zergling e torres diminuíram bastante, provando que o agente aprendeu que a produção dessas unidades não traz benefícios para o objetivo definido.

Essa população já é suficiente para destruir os 15 Wraiths que se localizam do outro lado do mapa. Por esse motivo ainda existem Zerglings e torres na população. Se houvesse mais Wraiths para serem destruídos, provavelmente o agente aprenderia a não gastar seus


Figura 32: População de unidades depois de um certo período de aprendizagem. A unidade destacada em verde é o Hydralisk, que consegue atacar unidades aéreas.

recursos em torres e zerglings e investiria todos os seus recursos apenas em Drones (que servem para coletar os recursos) e em Hydralisks.

Com todos os experimentos que foram realizados e ao analisar os seus resultados, podemos concluir que é possível aplicar a aprendizagem por reforço no modelo de gerenciamento de recursos proposto e que a partir de uma representação simples, onde consideramos apenas o tempo como estado do ambiente, conseguimos atingir alguns objetivos simples, como por exemplo, destruir unidades localizadas no mapa. Também podemos concluir, de acordo com os resultados obtidos, que para este modelo é mais indicado o uso do algoritmo Sarsa($\lambda$), pois com tantas possíveis combinações de políticas de investimento durante uma partida, ele consegue aprender mais rapidamente. Portanto, depois de um certo número de partidas executadas, o agente consegue gerar a população de unidades ideal para atingir o objetivo previsto.

# 7 Considerações finais

O presente trabalho mostrou que, ao longo do tempo, pesquisadores vêm obtendo sucesso ao aplicar técnicas de inteligência artificial em jogos de estratégia clássicos, onde programas desenvolvidos conseguiram se comportar como jogadores especialistas ou até vencê-los. Por outro lado, foi mostrado que para jogos de estratégia em tempo real é preciso percorrer ainda um longo caminho, devido à grande complexidade encontrada nessa categoria de jogos digitais.

Dentre os diversos trabalhos relacionados aos jogos de estratégia em tempo real encontrados na literatura, foram mostrados os que contribuíram para o desenvolvimento deste documento, destacando aqueles que trataram do assunto de gerenciamento de recursos. Dois desses trabalhos serviram como base para o desenvolvimento do modelo proposto e, por esse motivo, foi feita uma explicação mais detalhada sobre eles.

Foi visto que nos trabalhos existentes na literatura há uma tendência para o uso de técnicas de aprendizagem por reforço para área de combates em jogos de estratégia em tempo real. Devido aos resultados satisfatórios obtidos por eles, este trabalho propôs a aplicação de aprendizagem por reforço para o gerenciamento de recurso. Para isso, foi feita uma breve explicação sobre o modelo geral dessa técnica, destacando os dois algoritmos que foram utilizados.

O principal objetivo deste trabalho foi desenvolver um modelo para o gerenciamento de recursos no contexto do jogo Starcraft, através da noção de políticas de investimento. Mostramos que ao se definir classes e porcentagens que essas classes devem receber do total de recursos disponíveis, o modelo se encarrega de gerenciar a linha de produção das unidades, fazendo desnecessária a definição da ordem de construção das unidades. A partir do modelo implementado, aplicamos técnicas de aprendizagem por reforço com intuito de aprender políticas de investimento válidas em ambientes simplificados para objetivos simples.

Mostramos que é possível aplicar os algoritmos de aprendizagem por reforço Sarsa(0)

e Sarsa($\lambda$) para atingir os objetivos definidos. Para a realização dos experimentos criamos dois cenários, cada um com diferentes tipos de unidades, para que o agente aprendesse através dos algorítmos como aplicar as políticas de investimento em cada uma das situações. Através dos resultados obtidos, demonstramos que os algoritmos conseguem aprender a aplicar as políticas para gerar uma população de unidades capazes de destruir os inimigos no cenário. Ao se comparar os resultados dos dois algoritmos experimentados, constatamos claramente que o Sarsa ($\lambda$) realiza a aprendizagem de forma mais rápida.

A partir deste modelo e com a comprovação de que a aplicação de técnicas de aprendizagem por reforço funcionam para ambientes simplificados, trabalhos futuros podem ser desenvolvidos para que se possa realizar novos estudos experimentando representações mais detalhadas do ambiente visando atingir objetivos mais complexos. Neste modelo, utilizamos apenas o tempo como variável para representação dos estados e, mesmo assim, conseguimos obter resultados promissores, mesmo que sendo para ambientes muito simples. Representações mais detalhadas, com a utilização de outras variáveis do ambiente, podem vir a trazer resultados ainda melhores.

Além disso, o modelo permite que trabalhos futuros possam ser desenvolvidos para realizar experimentos com outras técnicas de aprendizagem por reforço, outros parâmetros de aprendizagem ($\alpha$, $\epsilon$, $\gamma$, $\lambda$), outros cenários utilizando unidades de diferentes tipos, ou até cenários dinâmicos, onde o agente deve aprender a se comportar para situações distintas através de um único processo de aprendizagem.

# Referências

CHURCHILL, D. A history of starcraft ai competitions. *AIIDE Starcraft AI Competitions*, 2016. Acessado em 07 de junho de 2016. Disponível em: <http://webdocs.cs.ualberta.ca/ cdavid/starcraftaicomp/history.shtml>.

CHURCHILL, D.; BURO, M. Build order optimization in starcraft. In: *Proccedings of the Seventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. The AAAI Press, 2011. p. 14 – 19. Disponível em: <https://www.aaai.org/ocs/index.php/AIIDE/AIIDE11/paper/viewFile/4078/4407>.

GIBNEY, E. Google ai algorithm masters ancient game of go. *Nature News and Comment*, 2016. Acessado em 03 de junho de 2016. Disponível em: <http://www.nature.com/news/google-ai-algorithm-masters-ancient-game-of-go-1.19234>.

KENNEDY, M. Signals: The man vs. the machine. *ESPN Video*, 2015. Acessado em 07 de junho de 2016. Disponível em: <http://espn.go.com/video/clip?id=11694550>.

KNIGHT, W. Google's ai masters the game of go a decade earlier than expected. *MIT Technology Review*, 2016. Acessado em 07 de junho de 2016. Disponível em: <https://www.technologyreview.com/s/546066/googles-ai-masters-the-game-of-go-a-decade-earlier-than-expected/>.

KOVARSKY, A.; BURO, M. A first look at build-order optimization in real-time strategy games. In: *Proceedings of the GameOn Conference*. [s.n.], 2006. p. 18–22. Disponível em: <https://skatgame.net/mburo/ps/planprop.pdf>.

OLIVEIRA, C. F. *Cerebrate: um modelo para personagens inteligentes de jogos de estratégia em tempo real*. 2014. Monografia (Bacharel em Ciência da Computação), UFRN (Universidade Federal do Rio Grande do Norte), Natal, Brazil.

OLIVEIRA, C. F. de; MADEIRA, C. A. G. Creating efficient walls using potential fields in real-time strategy games. In: *2015 IEEE Conference on Computational Intelligence and Games, CIG 2015, Tainan, Taiwan, August 31 - September 2, 2015*. [s.n.], 2015. p. 138–145. Disponível em: <http://dx.doi.org/10.1109/CIG.2015.7317921>.

ONTAÑÓN, S. et al. A survey of real-time strategy game AI research and competition in starcraft. *IEEE Trans. Comput. Intellig. and AI in Games*, v. 5, n. 4, p. 293–311, 2013. Disponível em: <http://dx.doi.org/10.1109/TCIAIG.2013.2286295>.

ROBERTSON, G.; WATSON, I. A Review of Real-Time Strategy Game AI. In: *Association for the Advancement of Artificial Intelligence*. [s.n.], 2014. Disponível em: <http://www.aaai.org/ojs/index.php/aimagazine/article/view/2478>.

SIEBRA, C. d. A.; NETO, G. P. B. Evolving the behavior of autonomous agents in strategic combat scenarios via sarsa reinforcement learning. In: *2014 Brazilian Symposium on Computer Games and Digital Entertainment*. [S.l.: s.n.], 2014. p. 115–122. ISSN 2159-6654.

SOUZA, T. A. *Gerenciador de Recursos em Ambientes Complexos: uma aplicação aos Jogos de Estratégia em Tempo Real*. 2013. Dissertação(Pós-Graduação em Ciência da Computação), UFPE (Universidade Federal de Pernambuco), Recife, Brazil.

SUTTON, R. S.; BARTO, A. G. *Reinforcement Learning: An Introduction*. MIT Press, 1998. ISBN 0262193981. Disponível em: <http://www.cs.ualberta.ca/%7Esutton/book/ebook/the-book.html>.

TESAURO, G. Td-gammon, a self-teaching backgammon program, achieves master-level play. *Neural Comput.*, MIT Press, Cambridge, MA, USA, v. 6, n. 2, p. 215–219, mar. 1994. ISSN 0899-7667. Disponível em: <http://dx.doi.org/10.1162/neco.1994.6.2.215>.

URIARTE, A.; ONTAñóN, S. Kiting in rts games using influence maps. In: *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. [s.n.], 2012. Disponível em: <https://www.aaai.org/ocs/index.php/AIIDE/AIIDE12/paper/view/5497>.

WEINBERGER, M. Now that google's artificial brain is conquering go, this classic computer game from 1998 could be next. *Business insider*, 2016. Acessado em 29 de maio de 2016. Disponível em: <http://www.businessinsider.com/google-deepmind-could-play-starcraft-2016-3>.

WENDER, S.; WATSON, I. D. Applying reinforcement learning to small scale combat in the real-time strategy game starcraft: Broodwar. In: *Conference on Computational Intelligence and Games*. [S.l.]: IEEE, 2012. p. 402–408.