Uma Arquitetura de Uso Geral baseada em Planejamento Probabilístico para Agentes Completos em Jogos de Estratégia em Tempo Real

Thiago França Naves

Universidade Federal de Uberlândia
Faculdade de Computação
Programa de Pós-Graduação em Ciência da Computação

Uberlândia
2017

Thiago França Naves

Uma Arquitetura de Uso Geral baseada em Planejamento Probabilístico para Agentes Completos em Jogos de Estratégia em Tempo Real

Tese de doutorado apresentada ao Programa de Pós-graduação da Faculdade de Computação da Universidade Federal de Uberlândia como parte dos requisitos para a obtenção do título de Doutor em Ciência da Computação.

Área de concentração: Ciência da Computação

Orientador: Carlos Roberto Lopes

Uberlândia
2017

Dados Internacionais de Catalogação na Publicação (CIP)
Sistema de Bibliotecas da UFU, MG, Brasil.

N323a 2017

Naves, Thiago França, 1988-
Uma arquitetura de uso geral baseada em planejamento probabilístico para agentes completos em jogos de estratégia em tempo real / Thiago França Naves. - 2017.
183 f. : il.

Orientador: Carlos Roberto Lopes.
Tese (doutorado) -- Universidade Federal de Uberlândia, Programa de Pós-Graduação em Ciência da Computação.
Disponível em: http://dx.doi.org/10.14393/ufu.te.2017.13
Inclui bibliografia.

1. Computação - Teses. 2. Inteligência artificial - Teses. 3. Jogos por computador - Teses. I. Lopes, Carlos Roberto, 1962-. II. Universidade Federal de Uberlândia. Programa de Pós-Graduação em Ciência da Computação. III. Título.

CDU: 681.3

SERVIÇO PÚBLICO FEDERAL
MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Ata da defesa de TESE DE DOUTORADO junto ao Programa de Pós-graduação em Ciência da Computação da Faculdade de Computação da Universidade Federal de Uberlândia.

Defesa de Tese de Doutorado: PPGCO-10/2017

Data: 27 de outubro de 2017 Hora de início: 14:00

Discente: Thiago França Naves Matrícula: 11223CCP005

Título do Trabalho: Uma Arquitetura de Uso Geral Baseada em Planejamento Probabilístico para Agentes Completos em Jogos de Estratégia em Tempo Real

Área de concentração: Ciência da Computação

Linha de pesquisa: Inteligência Artificial

Reuniu-se na sala 1B132, Bloco 1B, Campus Santa Mônica da Universidade Federal de Uberlândia, a Banca Examinadora, designada pelo Colegiado do Programa de Pós-Graduação em Ciência da Computação assim composta: Professores doutores: Márcia Aparecida Fernandes – FACOM/UFU, Rita Maria da Silva Julia – FACOM/UFU, Flávio Soares Corrêa da Silva – IME/USP, Luiz Chaimowicz – DCC/UFMG e Carlos Roberto Lopes FACOM/UFU, orientador do candidato.

Ressalta-se que o Prof. Dr. Flávio Soares Corrêa da Silva participou da defesa por meio de vídeo conferência desde a cidade de São Paulo - SP. Os outros membros da banca e o aluno participaram *in loco*.

Iniciando os trabalhos o presidente da mesa Prof. Dr. Carlos Roberto Lopes apresentou a Banca Examinadora e o candidato, agradeceu a presença do público, e concedeu ao Discente a palavra para a exposição do seu trabalho. A duração da apresentação do Discente e o tempo de arguição e resposta foram conforme as normas do Programa.

A seguir o senhor presidente concedeu a palavra, pela ordem sucessivamente, aos examinadores, que passaram a arguir a candidata. Ultimada a arguição, que se desenvolveu dentro dos termos regimentais, a Banca, em sessão secreta, atribuiu os conceitos finais.

Em face do resultado obtido, a Banca Examinadora considerou o candidato **aprovado**.

Esta defesa de Tese de Doutorado é parte dos requisitos necessários à obtenção do título de Doutor. O competente diploma será expedido após cumprimento dos demais requisitos, conforme as normas do Programa, legislação e regulamentação interna da Universidade Federal de Uberlândia.

Nada mais havendo a tratar foram encerrados os trabalhos às 14 horas e 30 minutos. Foi lavrada a presente ata que após lida e achada conforme foi assinada pela Banca Examinadora.

Participou por meio de vídeo conferência

Prof. Dr. Flávio Soares Corrêa da Silva
IME/USP

Prof. Dr. Luiz Chaimowicz
DCC/UFMG

Prof.ª Dr.ª Márcia Aparecida Fernandes
FACOM/UFU

Prof.ª Dr.ª Rita Maria da Silva Julia
FACOM/UFU

Prof. Dr. Carlos Roberto Lopes
FACOM/UFU(Orientador)

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
Coordenação do Programa de Pós-Graduação
CONFERE COM O ORIGINAL
27/11/17 1431675
Data Assinatura/ SIAPE

Dedico esse trabalho a toda minha família e em especial ao meu pai José Batista Naves e minha mãe Marli França Naves.

Agradecimentos

Agradeço inicialmente aos meus pais e toda a minha família, que durante todo o meu doutorado sempre se preocuparam com o andamento desse e com meu bem estar. Vocês foram o meu principal pilar durante todo esse processo.

Agradeço a minha mãe Marli França Naves, por estar sempre disposta a motivar os meus estudos e carreira profissional na docência e pesquisa. Desde que iniciei os estudos, ir para casa em alguns fins de semana sempre foi uma maneira de renovar as energias e isso acontecia devido a sua alegria ao me receber e demonstrar todo o seu carinho. Saiba que sempre irei para casa não importa a distância.

Agradeço ao meu pai José Batista Naves, esse que foi sempre o meu conselheiro oficial em quaisquer dilemas que passei durante todo esse tempo, sua sabedoria sempre foi muito importante e sempre irei consultá-la. Além disso, sempre foi o meu parceiro nas empreitadas para criar alguma coisa ou "instalar" alguma nova tecnologia que fosse interessante, sempre me diverti nessas aventuras com você.

Agradeço a minha irmã Thays França Naves, que no seu próprio estilo me apoiou em todo o meu processo de estudo acadêmico, além de estar seguindo nessa área hoje o que me deixa muito orgulhoso. Saiba que onde eu chegar você tem capacidade de ir além.

Agradeço a minha namorada Bianca Souza Malvaccini que me conheceu durante o doutorado e foi minha companheira inseparável nesse processo. Obrigado por sempre me apoiar e proporcionar momentos de diversão e carinho juntos, todos eles foram muito importantes. E um obrigado especial por entender sempre que eu tive que abrir mão de estar perto de você por motivos profissionais e acadêmicos.

Agradeço os meus sogros Pedro Henrique Malvaccini e Alcione Souza Malvaccini, que sempre foram muito amáveis e me receberam em sua família como um filho. Sempre serei grato a vocês e a todos os momentos que passei na casa de vocês.

Agradeço ao meu orientador Dr. Carlos Roberto Lopes, são sete anos de trabalho juntos e agradeço imensamente a todo o apoio que você sempre me deu, principalmente nos momentos desafiadores da pesquisa onde sus palavras e paciência como pessoa e profissional foram vitais para o término desse doutorado.

acadêmica desde o mestrado e sempre foi muito atenciosa, me ajudando na fase final do doutorado para que esse fosse finalizado. Sempre serei grato a você.

Agradeço ao secretário do PPGCC Erisvaldo Araújo Fialho, sempre muito prestativo e disposto a ajudar os alunos, seu trabalho foi importante e sempre me ajudou mesmo quando era preciso despender esforço adicional de sua parte.

Agradeço a CAPES por ter ajudado no trabalho financiando ele durante todo o processo.

"Nunca puxamos o saco de ninguém para chegarmos onde chegamos."
(Billie Joe Armstrong)

Resumo

Jogos de Estratégia em Tempo Real, também conhecidos como jogos RTS, são caracterizados por atuar em um ambiente dinâmico, com incertezas e vários recursos a ser gerenciados. Esse gênero de jogos torna-se um ótimo domínio de testes para algoritmos de inteligência artificial (IA), em especial utilizando abordagens de planejamento e tomada de decisão, que são tópicos ativos de pesquisa em IA. Este trabalho tem por objetivo propor o desenvolvimento de um agente jogador completo para jogos RTS. Para que o agente seja considerado completo, existem várias tarefas que ele deve executar, como: modelagem de dados entre partidas disputadas; tomada de decisão sob incerteza; gerenciamento de recursos; planejamento contra o adversário em tempo real; escalonamento de ações. Desse modo, para a implementação completa de um agente jogador que obtenha sucesso, é necessária uma abordagem integradora, que gerencie tais tarefas em diferentes níveis de abstração. Dentre os principais trabalhos no domínio de jogos RTS, são poucas as referências que propõem uma abordagem integradora, pois a grande maioria utiliza apenas técnicas apoiadas em scripts predefinidos ou regras condicionais. Assim, esta tese propõe uma nova abordagem, baseada em planejamento probabilístico, para controle completo de agentes jogadores em RTS. Essa abordagem é proposta sob uma arquitetura que opera com algoritmos de mineração de dados sequenciais, árvores de predição; processo de decisão de Markov parcialmente observável (POMDP), planejamento reativo e escalonamento de ações. A abordagem consegue gerenciar todas as tarefas do jogo com respostas compatíveis, considerando as restrições de tempo real desses jogos. Para validar a proposta, experimentos contra outros agentes, jogadores humanos, com testes de performance e qualidade são executados, e seus resultados discutidos.

Palavras-chave: POMDP. Tomada de Decisão. Planejamento. Mineração de Padrões Sequenciais. Ações. Jogos RTS. Inteligência Artificial.

Abstract

Real-time Strategy games, also known as RTS games, are characterized by acting in a dynamic environment, with uncertainties and various resources to be managed. This genre of games becomes a great testbed domain for artificial intelligence (AI) algorithms, in particular using planning and decision-making approaches, which are active AI research topics. This work aims to propose the development of a complete player agent for RTS games. In order for the agent to be considered complete, there are several tasks that it must perform, such as: data modeling between disputed matches; decision-making under uncertainty; resource management; planning against the opponent in real time; scheduling of actions. Thus, for the complete implementation of a successful player agent, an integrative approach is needed, which manages such tasks at different levels of abstraction. Among the main works in the field of RTS games, there are few references that propose an integrative approach, since the vast majority use only techniques based on predefined scripts or conditional rules. Thus, this thesis proposes a new approach, based on probabilistic planning, for complete control of players agents in RTS. This approach is proposed under an architecture that operates with sequential data mining algorithms, prediction trees; partially observable Markov decision process (POMDP), reactive planning and scheduling of actions. The approach manages all the tasks of the game with compatible answers, considering the real-time restrictions of these games. To validate the proposal, experiments against other agents, human players, with performance and quality tests are performed, and their results discussed.

Keywords: POMDP. Decision Making. Planning. Sequential Pattern Mining. Actions. RTS games. Artificial Intelligence.

Lista de ilustrações

Lista de tabelas

Lista de siglas

ABT Árvore de Comportamentos Ativos

ABCD *Alpha-Beta* Considerando Durações

AUG Arquitetura de Uso Geral

GDA Autonomia Guiada por Objetivos

CBP Planejamento em Tempo Real Baseado em Casos

CBR Raciocínio Baseado em Casos

CE Computação Evolucionária

CHAID Chi-squared Automatic Interaction Detector

CL Modelo Classe Latente

DBD Descoberta de Conhecimentos em Bancos de Dados

DT Árvore de Decisão

EM Maximização da Expectativa

FIB *Fast Informed Bound*

FPG Crescimento de Padrões Frequentes

FPGrowth Frequent Pattern Growth

FSM Máquina de Estados Finitos

GDL *Game Description Language*

GI Ganho de Informação

GSP Algoritmo de Classificações Sequenciais

GTC Geração e Teste de Candidatos

HOAI Heurística para Alvos Inteligentes

HMM Modelos Ocultos de Markov

HTN Redes de Tarefas Hierárquicas

IA Inteligência Artificial

IMG Informação Mútua Global

KMN *k-Means*

MCTS Árvore de Busca de Monte Carlo

MPS Mineração de Padrões Sequenciais

NNGE *Non-nested Generalized Exemplars*

PDDL *Planning Domain Definition Language*

PG Proporção do Ganho

POMDP Processo de Decisão de Markov Parcialmente Observável

PrefixSpan Prefix-projected Sequential Pattern Mining

PR Planejamento Reativo

RAC Redundância de Atributo Classe

RL Aprendizado por Reforço

RTS *Real Time Strategy Game*

SEJ Sequência Entre Jogadores

TGS Árvore de Busca em Jogos

UCB *The Upper Confidence Bound*

Sumário

Capítulo 1

Introdução

1.1 Motivação

O campo de pesquisa de Inteligência Artificial (IA) para jogos de estratégia em tempo real (*Real Time Strategy Game* (RTS)) tem avançado e crescido significativamente em termos de contribuições científicas, desde a chamada para trabalhos feita por (BURO, 2004) (BURO; FURTAK, 2003). Além dos relevantes desafios para a área de IA presentes no ambiente de jogos RTS, as competições organizadas por associações e conferências vêm incentivando a exploração de diversos novos tipos de abordagens. Competições como "ORTS RTS Game AI Competition" (organizada de 2006 a 2009), "AIIDE StarCraft AI Competition" (organizada desde 2010) e "CIG StarCraft RTS AI Competition" (organizada desde 2011) estabelecem disputas em diversas disciplinas de IA presentes nessa categoria de jogos. A modalidade mais disputada é a batalha entre agentes jogadores autônomos (*bot*), os quais são capazes de controlar, gerenciar e executar ações em diversos níveis de abstração dentro de um jogo RTS.

Os jogos RTS provêm um ambiente rico e dinâmico em termos de possibilidades de ações e gerenciamento de recursos, apresentando diversos desafios que podem ser explorados. Alguns deles são: ambientes dinâmicos, complexos e incertos, onde não existe informação completa sobre os estados ou mesmo a dinâmica do ambiente (ONTANON et al., 2013); tomada de decisão em tempo real com grande espaço de decisões (AHA; MATTHEW; PONSEN, 2005); raciocínio concorrente entre metas em diversos níveis de controle e abstração (WEBER et al., 2010); múltiplos movimentos e controle simultâneo de unidades; e escalonamento de ações. Encontrar técnicas eficientes para obter bons resultados nesses desafios é importante para diversas subáreas da IA. O principal jogo usado como ambiente de testes nas competições citadas e também nesta pesquisa é o StarCraft[1], considerado o jogo de RTS com maior quantidade de recursos, restrições entre ações e complexidade espacial (CABRERA; COTTA; LEIVA, 2013). Nele o jogador consegue

[1] *StarCraft*, produzido pela *Blizzard Entertainment*: <http://www.blizzard.com/>

executar diversas ações sobre um mapa de jogo para reunir recursos, coordenar unidades, planejar ataques e defesas e construir diferentes tipos de tecnologia.

Para a arquitetura proposta nesta tese, foram definidos três níveis de abstração essenciais, dentre aqueles existentes em um jogo RTS: estratégia, tática e controle reativo. Estratégia é o mais alto nível de abstração. As decisões nesse nível afetam todos os demais e são caracterizadas por definir as ações de um jogador a longo prazo (2 a 3 minutos), como, por exemplo, adotar uma estratégia em que recursos defensivos sejam criados para formar uma base que suporte ataques do adversário. Tática é o nível logo abaixo de estratégia, em que decisões de tempo médio (1 minuto), envolvendo recursos produzidos pela abstração anterior, são tomadas. Um exemplo de tática seria reunir os recursos ofensivos e executar uma formação de combate específica. Controle reativo é o nível mais baixo de abstração dentre os três, e corresponde a ações executadas rapidamente (2 segundos ou menos). Tais ações normalmente envolvem o controle de recursos de forma individual, que é feito quando eventos não previstos no jogo acontecem, como, por exemplo, um ataque inesperado em que é preciso contra-atacar com os recursos disponíveis no momento. Diversas tarefas estão presentes entre os níveis citados, e diferente técnicas e abordagens podem ser usadas para tentar solucionar cada uma delas.

Grande parte dos trabalhos desenvolvidos dentro da área de jogos RTS busca solucionar problemas relativos a diferentes tarefas dos níveis de abstração, contudo, poucos propõem abordagens para lidar com todas as tarefas do jogo. Trabalhos como os de (NAVES; LOPES, 2012a), (CHURCHIL; BURO, 2011), (YOUNG; HAWES, 2012) e (JAIDEE; MUNOZ; AHA, 2011) focam a resolução de problemas que envolvem as tarefas presentes em estratégia, como produção eficiente de recursos, escalonamento, tomada de decisão e predição de metas adversárias. Os trabalhos de (HALE; YOUNGBLOOD; DIXIT, 2008), (KABANZA et al., 2010), (CADENA; GARRIDO, 2011) e (SYNNAEVE; BESSIERE, 2012) focam a parte de táticas, com formações eficientes de unidades, análise de terreno, caminhamento pelo mapa do jogo, métodos de ataque e defesa. A respeito de controle reativo, as abordagens de (URIARTE; ONTANON, 2011), (AVERY; LOUIS; AVERY, 2009), (WENDER; WATSON, 2012) e (PREUSS et al., 2010) concentram-se em tarefas como caminhamento mais curto, resposta reativa a ataques inimigos e análise de ameaças presentes no mapa. Porém, buscar solucionar uma tarefa específica ou um nível de abstração dentro de um RTS não torna a abordagem apta a jogar de modo autônomo, como um *bot*/agente completo, capaz de lidar com todas as tarefas do jogo, proporcionando uma solução completa.

A arquitetura de um agente apto a jogar um jogo RTS é normalmente dividida em módulos, que são responsáveis por executar atividades específicas no jogo. Os módulos são integrados sob uma arquitetura que habilite comunicação entre eles, com o objetivo de melhorar a troca de informações, a tomada de decisões e a leitura do jogo em andamento. Contudo, há poucos trabalhos que apresentam abordagens completas, que

conseguem executar todas as tarefas presentes no jogo. Pequenas exceções são os trabalhos de (HAGELBACK, 2012) e (SYNNAEVE, 2012), que apresentam agentes aptos a desempenhar todas as exigências em uma partida de um jogo de RTS. No entanto, nessas abordagens citadas, as técnicas utilizadas permitem o controle do agente até determinado nível de abstração dentro do jogo, deixando tarefas serem executadas por blocos de regras simples ou scripts predefinidos. Esses trabalhos conseguem atender às exigências mínimas de um jogo RTS, porém não são arquiteturas baseadas totalmente em IA, em que todas as decisões e ações são definidas por meio de algoritmos que efetuam algum tipo de raciocínio sobre suas escolhas. Por definição, essas abordagens serão consideradas parciais, devido à natureza de suas arquiteturas, enquanto um agente que consegue gerenciar todas as tarefas do jogo com uso de IA será considerado completo. Muitos dos *bots* presentes nas competições têm como característica utilizar blocos de regras simples, uma vez que integrar diferentes técnicas sob uma arquitetura para um jogo com restrições de tempo real é um trabalho complexo.

Considerando-se a falta de abordagens completas na área de jogos RTS, esta tese propõe uma abordagem para um agente jogador completo, por meio de uma arquitetura baseada em planejamento probabilístico. Essa arquitetura envolve o uso de técnicas de planejamento probabilístico combinadas com mineração de dados, tomada de decisão, estruturas de busca, planejamento reativo e escalonamento de ações. Juntamente com o objetivo principal, foi construído um *bot* com a abordagem proposta, capaz de realizar todas as tarefas dentro de um jogo RTS, bem como raciocinar sob seus níveis de abstração. O *bot* deve ser capaz de tomar decisões utilizando informações obtidas diretamente do domínio do jogo e executar ações com uso de planejamento probabilístico e tomada de decisão, sem necessidade de nenhum tipo de codificação predefinida ou blocos de regras específicos de controle. Assim, a arquitetura pode ser reutilizada em qualquer outro jogo RTS, desde que as etapas de modelagem que serão descritas nessa tese sejam seguidas, utilizando uma base de dados advinda de um jogo RTS específico. Com essa característica, a abordagem proposta pode ser classificada como uma Arquitetura de Uso Geral (AUG) (CUNNINGHAM; WILKS; GAIZAUSKAS, 1996) autônoma. Uma AUG é uma arquitetura capaz de ser reutilizada em qualquer classe de cenários semelhantes àquele para o qual ela foi desenvolvida originalmente. A especificação do problema de planejamento deve ser feita por uma linguagem que defina todas as possibilidades de informações e interações que podem ser feitas no problema específico. A dinâmica das ações de qualquer jogo RTS pode ser descrita na arquitetura proposta utilizando Planning Domain Definition Language 2.1 (PDDL) (MARIA.; LONG, 2003), que é uma linguagem de planejamento, ou utilizando *Game Description Language* (GDL) (LOVE; PELL, 2005), que é uma linguagem baseada em lógica de primeira ordem.

A arquitetura proposta pode ser utilizada para diferentes finalidades, tanto em jogos RTS como em ambientes incertos, com processo decisório de execução de ações. Em jogos

RTS, a abordagem pode ser utilizada para:

- ❏ Verificar se diferentes raças têm seus recursos balanceados em relação ao poder de ataque e defesa.
- ❏ Atuar como um assistente para auxiliar o aprendizado de jogadores iniciantes, cobrindo as tarefas e os níveis de abstração do jogo.
- ❏ Permitir que outras abordagens possam comparar seus resultados no controle do jogo completo ou em tarefas específicas.

Em relação ao uso da abordagem proposta em problemas diversos, destacam-se:

- ❏ Ambientes com largos espaços de estados gerados sob intervalos de tempo. A abordagem de classificação aqui utilizada pode encontrar maior e mais preciso número de padrões, utilizando as estratégias de combinação de algoritmos que será apresentada no Capítulo 5.
- ❏ Tomada de decisão em ambientes reais e incertos com restrição de tempo, comum em tarefas de robótica. A abordagem de POMDP online (Capítulo 6) utiliza conceitos online, macroações e reatividade, que podem ampliar as recompensas obtidas e o tempo de resposta nesses ambientes.
- ❏ Escalonamento de tarefas em ambientes de tempo real, em que é preciso atingir o menor tempo para execução e satisfazer pré-condições das tarefas. A abordagem de escalonamento (Capítulo 7) lida com esses problemas, empregando conceitos de janela de intervalo e recurso próximo, que utilizam intervalos ociosos entre tarefas e recursos extras para melhorar o resultado do escalonamento.

1.1.1 Hipótese e Objetivo

Uma hipótese deste trabalho é que desenvolver um agente com uma arquitetura baseada em planejamento probabilístico, combinada com técnicas de IA, pode fazê-lo lidar com todas as tarefas de um jogo RTS, utilizando algum tipo de raciocínio em cada uma delas. Isso também possibilita que o agente exiba um comportamento mais próximo ao de um jogador humano, sem a necessidade de utilizar nenhum tipo de código estático ou regras *adhoc*.

Outra hipótese é que a análise de informações específicas do domínio por meio de *replays* de jogos pode fornecer todas as ações e os estados que serão utilizados na abordagem, caracterizando-a como uma AUG. Os dados obtidos via mineração e classificação possibilitam que ações e conjuntos de ações sejam mais bem descritos, quando comparados com as mesmas descrições feitas por humanos.

O objetivo geral deste trabalho é propor uma arquitetura baseada em planejamento probabilístico e técnicas de IA, para construção de um agente capaz de realizar todas as tarefas de um jogo RTS.

Para alcançar o objetivo geral deste trabalho, vários objetivos específicos são propostos e devem ser alcançados. São eles:

1. Explorar o uso de um arquitetura AUG voltada para ambientes de tempo real. Trata-se de uma combinação de planejamento, tomada de decisão, mineração de padrões sequenciais, busca e escalonamento. Essa arquitetura é responsável pelo controle total do jogo, em que sequências de ações nos diversos níveis de abstração são escolhidas com base em raciocínio e probabilidade.

2. Explorar o conhecimento específico do domínio de jogos do StarCraft, para inferir sobre ações e estados que um agente pode alcançar e executar durante o jogo. Efetuar análises em bases de *replays* com partidas disputadas entre jogadores profissionais para extrair padrões de ações e estados do jogo. Técnicas de mineração de dados e classificação são utilizadas nessas bases de dados, para inferir e encontrar padrões a respeito de ações, estados, estratégias de jogo e quaisquer outros parâmetros que possam melhorar a tomada de decisão da arquitetura. O objetivo é que os conjuntos de ações executados pelo agente sejam obtidos a partir de padrões reconhecidos em seu próprio ambiente, sem especificações *adhoc* ou feitas diretamente por código.

3. Propor a modelagem de um processo de decisão de Markov parcialmente observável (POMDP) para auxiliar a tomada de decisão do sistema de planejamento. As ações consideradas pela arquitetura terão suas recompensas e sequência de execução modeladas pelo POMDP, que utilizará, além das informações sensoriais da partida, os dados provenientes da análise de *replays* para especificar estados, calcular recompensas, probabilidades e fazer reajustes, sempre que for necessário.

4. Especificar os módulos necessários para gerenciamento das etapas que compõem a arquitetura de planejamento. Uma arquitetura de planejamento efetua diversos caminhos utilizando entradas e saídas de dados para executar todas as tarefas necessárias. Essas tarefas incluem desde capturar dados sensoriais em uma partida, até o escalonamento das ações escolhidas para execução. Para que o fluxo de dados seja correto e contínuo, é preciso definir a arquitetura de forma modular, em que cada tarefa é executada utilizando seus respectivos algoritmos, que ficam dentro do módulo responsável por aquela tarefa. Os módulos tornam a construção da arquitetura e troca de dados entre tarefas mais simples, além de permitir rápida modificação e adição de novos módulos.

5. Utilizar escalonamento de ações. O escalonamento garante que os conjuntos de ações selecionadas pela arquitetura tenham sua execução finalizada o mais breve possível.

O objetivo principal é paralelizar a execução das ações. Quanto mais eficiente for essa etapa, mais tempo de jogo fica disponível para produção de recursos e controle das unidades.

1.2 Visão Geral da Arquitetura Proposta e Contribuições

A arquitetura proposta é inteiramente descrita a partir do Capítulo 4, com todos os algoritmos utilizados e a integração entre cada módulo. Para melhor compreensão da composição e do funcionamento da arquitetura, uma visão geral dela é apresentada, com foco em seus três módulos, que operam em conjunto. A Figura 1 ilustra os módulos e a ordem em que eles trocam informações. O funcionamento é contínuo em relação à duração de uma partida com o ciclo de troca de informações formando um *loop*. Uma descrição desses módulos será feita nas seções que seguem.

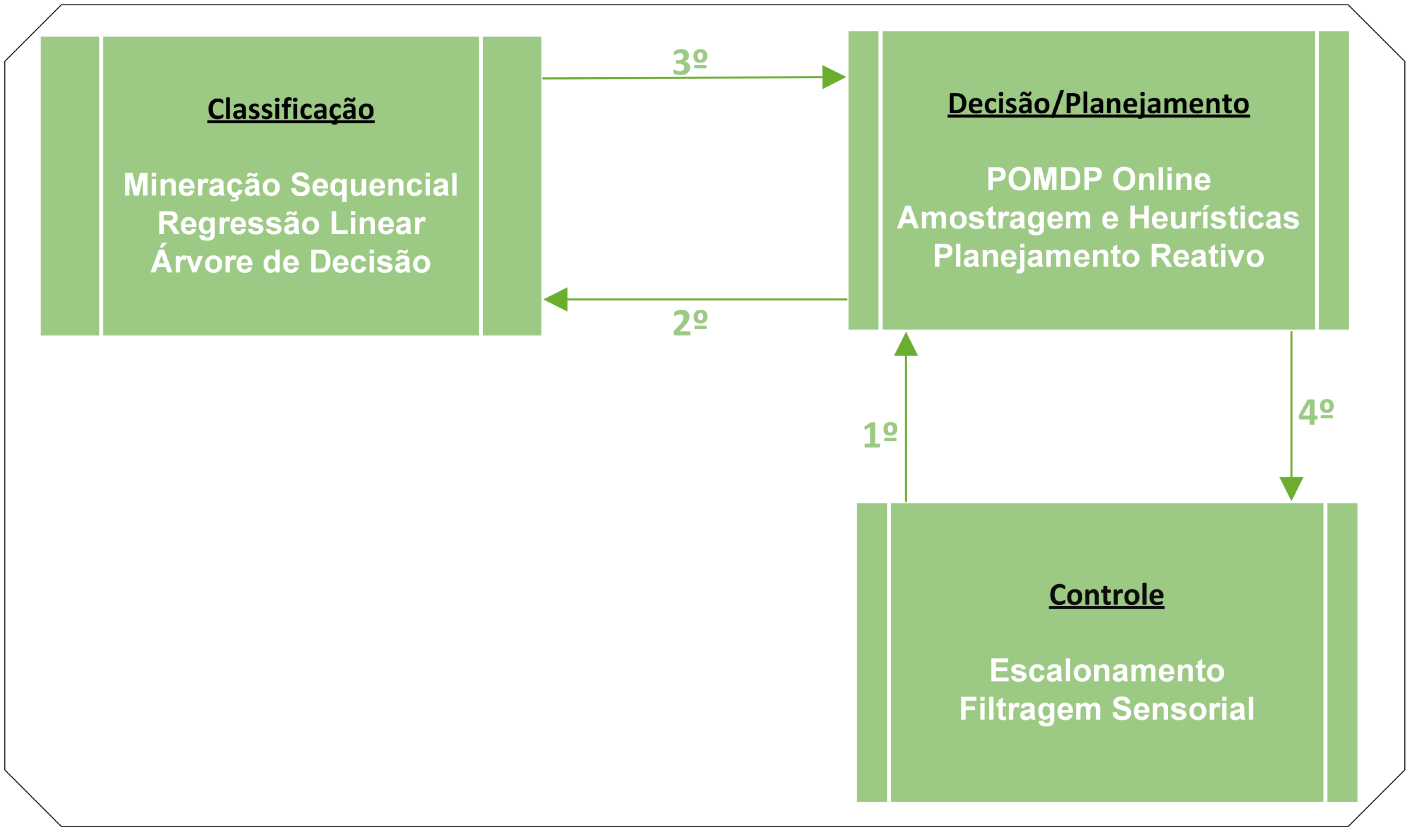

Figura 1 – Visão geral da arquitetura de AUG para jogos RTS, descrita com base nos três módulos.

1.2.1 Classificação e Mineração

O módulo de *Classificação* exibido na Figura 1 é descrito na Seção 5.1. Tal módulo é responsável por minerar bases de *replays* de jogos, definir as macroações utilizadas

em toda a abordagem e construir a árvore com todos os conjuntos de ações que serão conhecidos e considerados para escolha. O módulo de *Classificação* introduz um novo conceito ao classificar os conjuntos de ações com base na frequência em que são utilizados por jogadores humanos, considerando todos os níveis de abstração do jogo e as ações de cada um destes. A árvore é utilizada como base organizada para escolha de qual conjunto de ações executar em um dado momento da partida. Toda nova escolha de macroação é feita com o módulo *Decisão* que consulta informações contidas na *Classificação*. Esse compartilhamento de informações é representado na Figura 1, através das mensagens com rótulo 2º, disparadas pela *Decisão*, e mensagem 3º, feita pela *Classificação*, que retornam informações necessárias para auxiliar a escolha da macroação adequada.

A abordagem de classificação proposta possui várias propriedades desejáveis, tanto para uma arquitetura de AUG como para o ambiente de jogos RTS. Primeiramente, o uso de macroações permite explorar os conjuntos de ações que jogadores humanos tendem a executar, e não apenas ações isoladas ou lineares, fazendo o processo de decisão ser executado inúmeras vezes, para que um conjunto seja executado. A classificação explora ações em níveis de abstração diferentes, identificando conjuntos que controlam unidades em grupo, ataques combinados e comportamento de combate entre unidades isoladas. Esses tipos de ações não são catalogados ou identificados, e geralmente são definidos utilizando conhecimento *ad hoc*.

As contribuições do módulo de *Classificação* proposto incluem:

- Mineração sequencial de padrões para classificar macroações em todos os níveis de abstração do jogo. São combinados algoritmos de teste de candidatos com algoritmos de variante de crescimento de padrões. A abordagem identifica e classifica maior número de macroações ao combinar dois padrões sequenciais distintos. Assim, garante-se que todas as ações executadas pela arquitetura são obtidas por meio de análise de dados do jogo.

- Uso do tempo de jogo como parâmetro para identificação e escolha das macroações via regressão linear. Os valores da regressão são utilizados como um aditivo para montagem da árvore de decisão onde ficam as macroações. Esses valores também são utilizados para auxiliar a tomada de decisão, sendo adicionados nos cálculos das recompensas, com o tempo do jogo sendo um elemento importante nas escolhas de conjuntos de ações, assim como jogadores humanos fazem.

- Árvore de decisão como estrutura para suporte à execução de múltiplas macroações. Com a disposição hierárquica das macroações em função dos níveis de abstração, é possível gerenciar a execução de mais de um macro ao mesmo tempo. Com macroações de diferentes níveis sendo executadas, os cenários de criação, movimentação e gerenciamento das unidades do jogo aproximam-se daqueles praticados por jogadores humanos.

1.2.2 Decisão e Planejamento

O módulo de *Decisão* exibido na Figura 1 é descrito na Seção 6.1. Tal módulo é responsável por escolher a macroação a executar em determinado momento do jogo e por gerenciar mudanças no planejamento em curso na partida. O módulo inicia sua operação ao receber os dados sensoriais do jogo em forma de observações, representado na Figura 1 pela mensagem 1º. As informações obtidas e armazenadas no módulo de $Classificação$ são consultadas para auxiliar a tomada de decisão. POMDP online é a principal técnica, sendo modificada para operar com amostragem e heurísticas de forma integrada. Além de decidir qual macroação executar, o módulo também considera outros conjuntos de diferentes níveis que possam ser executados em paralelo com a atual macroação, ampliando o raciocínio e o alcance das estratégias em geral. A execução de uma macroação pode ser interrompida a qualquer momento, caso o ambiente sofra alguma mudança, e os recursos gerados até então são reutilizados na escolha de um nova macroação, com o planejamento reativo. A macroação escolhida é enviada para escalonamento e execução através da mensagem 4º.

O módulo de *Decisão* possui características vantajosas para ambientes de tempo real e adversário, como o de jogos RTS, além de operar com AUG. A tomada de decisão considera a incerteza do ambiente no cálculo de recompensas, mantendo as restrições de tempo no retorno das decisões. Geração de estados crença feita com duas técnicas de geração combinadas permite que o espaço de estados seja visitado em diversos pontos distintos, aumentando a chance de encontrar melhores macroações. Durante a partida, qualquer macroação em execução pode ser interrompida em relação a seu planejamento e a tomada de decisão refeita, para adaptar-se a situações de ataque do inimigo ou incontingências repentinas.

As contribuições do módulo de *Decisão* proposto incluem:

- Uso do POMDP na versão online com modificações específicas para melhor desempenho em ambientes de tempo real adversário com cenários dinâmicos.
- Uso de heurística que utiliza como base para cálculo de recompensas a pertinência de uma macroação, dado o tempo atual de jogo. A pertinência é obtida pela dependência do tempo de jogo em relação ao uso das variáveis representadas pelas ações que compõem a macroação. O valor é obtido pela relação de uso por tempo de jogo, que é definido com análises de partidas previamente disputadas. Além da heurística, há o uso de amostragem para geração de estados, quando o ambiente do jogo e do POMDP são modificados. O algoritmo de amostragem busca por estados crença que possam apresentar maior recompensa quando o ambiente do jogo é modificado, proporcionando alternativas para o tomador de decisões.
- Adaptação do módulo para uso de planejamento reativo. Esse uso inclui capacidade de suspender a execução do atual conjunto de ações, nova tomada de decisão com

parâmetros selecionados e uso de linguagem reativa com árvores de comportamentos para gerenciamento das ações de cada nível em execução. O comportamento do *bot* com a característica reativa fica mais próximo ao de jogadores humanos, com maior eficácia em responder a eventos inesperados na partida.

1.2.3 Controle

O módulo de *Controle* exibido na Figura 1 é descrito no Capítulo 7. Ele é responsável por captar os dados sensoriais do jogo e filtrá-los para que sejam transferidos para os outros módulos, além de escalonar as ações antes de executá-las dentro do jogo. Esse módulo é descrito por último, pois, apesar de iniciar a operação da arquitetura enviando os dados sensoriais filtrados, ele finaliza a operação ao enviar os comandos de execução de ações ao jogo. A filtragem sensorial é feita para verificar se o ambiente do jogo pode ter sido alterado devido às observações recebidas. Caso haja probabilidade de mudança, ela é passada ao módulo de *Decisão* para que o POMDP possa alterar entre suas políticas. O escalonamento é uma etapa importante na arquitetura. A técnica utilizada contém boa performance e utiliza um conceito de janela de intervalo, proposto para ambientes com ações que utilizam recursos renováveis. Essa técnica também é útil para reaproveitar intervalos criados quando uma macroação em execução é interrompida e uma nova é escolhida para execução.

As contribuições do módulo de *Controle* proposto incluem:

- Algoritmo de escalonamento eficiente para ambientes de tempo real. Uso de uma técnica que aproveita o surgimento de intervalos de tempo, que surgem devido a escalonamento de ações com tempos de execução diferentes ou devido a ações com precondições, que adiam sua execução até estarem satisfeitas no jogo. O reaproveitamento dos recursos gerados por ações interrompidas também gera muitos intervalos, que tentam ser preenchidos ao máximo pelo algoritmo de escalonamento.

- Arquitetura AUG através da integração do módulo de *Controle* com os demais, através da recuperação de dados do jogo e transformação deles em metadados para comunicação com a arquitetura. Todas as mensagens enviadas e recebidas entre os módulos possuem informações sensoriais advindas do jogo. A *Controle* filtra essas informações, transformando-as em dados genéricos que podem ser utilizados em qualquer outro jogo RTS ou cenário contendo ações e recursos. Assim, toda a arquitetura pode ser reutilizada como AUG.

1.3 Organização da Tese

Esta tese está organizada como segue.

- Capítulo 1: apresenta a introdução com a motivação para realização deste trabalho, uma visão geral da arquitetura proposta e suas principais contribuições.
- Capítulo 2: apresenta os principais conceitos ligados a este trabalho, como jogos RTS, POMDP, mineração de dados, planejamento e escalonamento.
- Capítulo 3: mostra uma revisão bibliográfica dos principais trabalhos envolvendo as técnicas aqui exploradas, com relação ao ambiente de jogos RTS e de tempo real em geral.
- Capítulo 4: descreve de forma geral a metodologia proposta neste trabalho, abordando o cumprimento dos objetivos e a integração de técnicas para gerar a arquitetura aqui proposta.
- Capítulo 5: descreve a metodologia de modelagem e classificação de dados, com detalhes do módulo de $Classifica$ção, os algoritmos utilizados e a definição das macroações.
- Capítulo 6: apresenta e descreve o módulo de *Decisão* com a metodologia de planejamento e tomada de decisão. Descreve o POMDP e suas políticas, reforçando a integração dele com o restante da arquitetura proposta.
- Capítulo 7: descreve o módulo de *Controle*, com o planejamento reativo e escalonamento de ações. Também apresenta os detalhes de como são gerenciados os dados provindos do ambiente do jogo StarCraft.
- Capítulo 8: apresenta os resultados experimentais obtidos com os métodos propostos, enfatizando cada módulo proposto, junto com outras abordagens disponíveis na literatura.
- Capítulo 9: encerra esta tese apresentando as conclusões, contribuições, limitações encontradas e sugestões de trabalhos futuros.

Capítulo 2

Fundamentação Teórica

2.1 Introdução

Neste capítulo é apresentado o referencial teórico importante para o entendimento desta tese. Os conceitos iniciais sobre Jogos RTS, Mineração de Dados, Planejamento Reativo, POMDP, Escalonamento, dentre outros serão abordados. A proposta do trabalho envolve combinar essas abordagens, para criar uma arquitetura de uso geral para gerenciamento das tarefas de um jogos RTS, sendo importante que o leitor tenha uma familiaridade com o tema. Dessa forma, esse capítulo foi dividido em quatro seções principais. Na Seção 2.2 são apresentados os conceitos sobre jogos de estratégia em tempo real, os desafios no gerenciamento da área e o uso do jogo StarCraft. Seção 2.3 apresenta um resumo sobre mineração de padrões com foco em mineração sequencial. A Seção 2.4, descreve os conceitos de árvore de decisão e o algoritmo que será utilizado nesta tese. Na seção 2.5, o conceito geral de planejamento reativo é apresentado. A seção 2.6, detalha o funcionamento do processo de decisão parcialmente observável, com foco no modo de operação textitonline. Na seção 2.7 o objetivo do escalonamento de ações com restrição de recursos é discutido, bem como seu funcionamento no domínio de jogos RTS. Por fim, as considerações finais são apresentadas na Seção 2.8.

2.2 Jogos RTS

Jogos de estratégia em tempo real ou Jogos RTS (um acrônimo para *Real Time Strategy*), são jogos de estratégia militar onde as decisões e ações ocorrem em um ambiente de tempo real (BURO; CHURCHILL, 2012). Esses jogos são um subgênero de jogos de estratégia, onde jogadores precisam desenvolver uma economia (reunir recursos e construir uma base) e seu poder militar (construir unidades de ataque e defesa e evoluir suas tecnologias), para conseguir derrotar os inimigos (destruir as bases inimigas). A Figura 2 exibe dois jogos do gênero RTS.

Figura 2 – Imagens dos jogos *Warcraft II* e *Warcraft III*.

Os jogos RTS trouxeram uma dinâmica diferente aos jogos de estratégia, uma vez que é necessário lidar com diversas restrições e precondições entre os recursos que deseja-se construir, além de gerenciar diversas unidades e suas peculiaridades de movimentação e ações de forma simultânea (YOUNG et al., 2012). Do ponto de vista teórico, as principais diferenças entre jogos RTS e jogos clássicos de tabuleiro ou baseados em turnos como o Xadrez são:

- Existem movimentos simultâneos, onde um ou mais jogadores podem executar ações ao mesmo tempo. As ações são durativas, ou seja, essas não acontecem de forma instantânea, levam um tempo até serem completadas.
- São jogos executados em tempo real, onde o jogador tem um intervalo de tempo muito curto para tomar uma decisão sobre o próximo movimento. Comparado ao Xadrez, onde o jogador possui tempo para pensar em uma jogada, RTS é executado a 24 frames por segundo, sendo assim o jogador precisa agir em intervalos de 42 ms, antes que os estados do jogo mudem.
- A maioria dos jogos RTS são parcialmente observáveis, ou seja, so é possível ver uma parte do mapa e do mundo onde o jogo está ocorrendo. Esse termo é referido como *fog of war*.
- Grande parte dos jogos RTS são não determinísticos, ou seja, algumas das ações possuem uma chance de ocorrerem com sucesso dentro do jogo.
- A complexidade tanto em termos de espaço de estados como em quantidade de ações disponíveis a cada ciclo é muito grande. Em uma breve comparação, o espaço de estados do Xadrez é algo em torno de 10^{50} e o do jogo Go em torno de 10^{170}. Em

jogos RTS, uma estimativa preliminar indicou que esse estado pode estar em algo em torno de $10^{200^{36000}}$, um valor bem superior e difícil de ser encontrado em outro domínio de testes (CABRERA; COTTA; LEIVA, 2013).

Por essas razões, as técnicas e abordagens clássicas utilizadas em outros tipos de jogos de estratégia não conseguem bons resultados em domínios de jogos RTS. A divisão básica das abstrações do jogo indicam que é preciso uma cooperação entre algoritmos para obter o controle desejados em todos os níveis de abstração do jogo. Jogadores humanos possuem grande facilidade para incorporar as regras do jogo e criar estratégias para o mesmo. De fato, esses jogadores ainda estão muito superiores em relação aos bots e projetos existentes atualmente.

2.2.1 StarCraft

StarCraft:Brood War (expansão) é um dos jogos mais populares de todos os tempos na categoria de RTS (CHURCHILL; SAFFIDINE; BURO, 2012) e foi lançado em 1998 pela Blizzard Entertainment. O jogo se passa em um mundo de ficção científica onde o jogador deve escolher entre três raças para combater os inimigos, essas são: Terran, Protoss e Zerg.

- Terran é a raça humana em um futuro distante, ela possui unidades que são versáteis sendo uma opção balanceada entre Protoss e Zerg.
- Protoss é uma raça alienígena, as unidades são mais pesadas e caras para serem construídas, contudo são fortes e resistentes no combate.
- Zerg é uma raça insectóide, as unidades são baratas e fracas, porém rápidas de serem construídas.

A figura 3, mostra uma imagem do jogo StarCraft onde um ataque da raça Protoss contra a Terran está sendo feito.

Para vencer uma partida o jogador deve derrotar cada inimigo presente no mapa executando ações e destruindo todos os recursos presentes nas suas respectivas bases. No StarCraft e nos demais jogos RTS ações são os comandos que podem ser executados dentro do jogo, enquanto recursos são itens gerados e construídos através de ações. Ações podem gerar novos recursos, reunir aqueles já existentes e controlá-los. Por exemplo, uma ação que constrói um recurso específico ou que reuni algum tipo de mineral ou mesmo que posiciona algum recurso em um local específico do mapa.

Recursos podem ser unidades de combate do jogo, edificações ou minérios. Unidades podem ser móveis, automobilísticas ou aéreas. Já edificações são estruturas construídas que produzem unidades, coletam minério, habilitam a construção de outras unidades e edificações mais avançadas. Minérios são recursos do tipo gás e mineral que são coletados

Figura 3 – Imagem do jogo StarCraft. Batalha entre as raças *Protoss* na parte mais a esquerda da figura e a raça *Terran* do lado mais direito representado na maioria por construções. Fonte: (ONTANON et al., 2013)

.

e utilizados como requisitos básicos para construção de qualquer recurso do jogo. A Figura 4, exibe algumas unidades e edificações do Starcraft. Neste trabalho, os recursos serão sempre citados utilizando o nome da sua classe, como unidades para recursos móveis e aéreos de ataque, edificações para construções em geral e minérios para os recursos não renováveis primários.

Figura 4 – Recursos do StarCraft, a esquerda unidades aéreas e terrestres, a direita edificações. Todos esses recursos são da classe *Terran*

.

No início de uma partida, o jogador começa reunindo minerais. Com esses recursos é possível construir novas unidades através de ações que precisam ser alocadas na ordem correta para que suas restrições sejam atendidas antes que possam ser executadas. Unidades podem executar diferentes tarefas como construir edificações. Entre as ações que as unidades podem executar temos: coletar recursos, caminhar sobre o mapa para descobrir novas áreas, construir novas edificações, atacar e defender. O raciocínio tempo-

ral é importante para que essas ações sejam feitas em uma ordem correta. Já o raciocínio espacial está presente no jogo o tempo todo, pois decidir onde construir uma unidade, movimentá-la pelo mapa, coordenar ataques e defesa em grupos ou individual.

No StarCraft todos os recursos são produzidos através de ações. Ações essas que possuem precondições e efeitos. As precondições de uma ação são os recursos que precisam estar disponíveis para que ela seja executada. Já o efeito de uma ação é a produção do seu respectivo recurso. Na Figura 5 é descrito de forma resumida o domínio de ações do StarCraft. Nessa, estão ilustradas as principais características das ações da classe *Terran*, que são usadas no jogo. As ações estão ligadas a quatro especificações de recursos: *Borrow*, *Consume*, *Require* e *Produce*.

Borrow significa que uma ação necessita pegar/tomar "emprestado" um recurso para ser executada. Esse recurso fica ocupado, ou seja, não pode ser utilizado enquanto a ação que está ocupando-o não for finalizada. *Require* são precondições onde os recursos devem estar disponíveis quando a ação for executada. Estar disponível quer dizer que o recurso já foi construído e está presente no jogo, não há necessidade de pegar o recurso "emprestado". *Produce* são os recursos produzidos por uma determinada ação. Em jogos RTS, uma ação sempre produz um recurso com seu respectivo nome. *Consume* são os recursos utilizados para que uma ação seja executada.

O mapa é representado por uma malha retangular, onde a proporção altura x largura é 32 x 32 pixeis chamado de *build tile*. Dentro de um *build tile* uma unidade pode ocupar um espaço de 8 x 8 pixeis, conhecido por *walk tile*. Já um mapa completo pode ter de 64 x 64 a 256 x 256 *build tiles* dependendo da escolha do jogador. Cada jogador pode controlar acima de 200 unidades mais uma quantidade ilimitada de construções. Cada raça contem em média 30 a 35 diferentes tipos de unidades e construções, a maioria dessas com uma quantidade de habilidades especiais que podem ser usadas.

2.3 Mineração de Dados

A mineração de dados ou *data mining* é uma etapa que integra o processo de Descoberta de Conhecimentos em Bancos de Dados (DBD) (WITTEN; FRANK, 2005). Brevemente, a mineração de dados corresponde em aplicar a uma base de dados adequadamente preparada, algoritmos e técnicas para descoberta de padrões interessantes nesses dados. A sistemática do DBD é o processo não trivial de identificação de padrões válidos, novos, potencialmente úteis e compreensíveis em dados. Dados são um conjunto de fatos, o chamado padrão corresponde a uma expressão em alguma linguagem descrevendo um subconjunto dos dados.

O termo "processo" informa que DBD compreende vários passos, envolvendo a preparação dos dados, a busca por padrões e a avaliação do conhecimento obtido. Esses passo podem ser repetidos em múltiplas iterações. O termo não trivial indica que o processo não é

```
resource CommandCenter
resource Barracks
resource Factory
resource CovertOps
resource Scv
resource Firebat
resource Ghost
resource SiegeTank
resource BatleCruiser
resource Minerals
resource Gas

action build-commandcenter :duração 75 seg.
 :borrow 1 Scv :consume 400 Minerals
 :produce 1 CommandCenter
action build-barracks :duração 50 seg.
 :require 1 CommandCenter :borrow 1 Scv :consume 150 Minerals
 :produce 1 Barracks
action build-factory :duração 50 seg.
 :require 1 Barracks :borrow 1 Scv
 :consume 200 Minerals 100 Gas :produce 1 Factory
action build-covertops :duração 25 seg.
 require:  1 ScienceFacility :borrow 1 Scv
 :consume 50 Minerals 50 Gas :produce 4 supply
action build-scv :duration 13 seg.
 :borrow 1 CommandCenter :consume 50 Minerals 1 Supply
 :produce 1 Scv
action build-firebat :duração 15 seg.
 require:  1 Academy :borrow 1 Barracks
 :consume 50 Minerals 25 Gas 1 Supply :produce 1 Firebat
action build-ghost :duração 32 seg.
 require:  1 Academy 1 CovertOps :borrow 1 Barracks
 :consume 75 Minerals 75 Gas 1 Supply :produce 1 Ghost
action build-siegetank :duração 32 seg.
 require:  1 MachineShop :borrow 1 Factory
 :consume 150 Minerals 100 Gas 2 Supply :produce 1 SiegeTank
action build-batlecruiser :duração 84 seg.
 require:  1 ControlTower 1 PhysicsLab :borrow 1 Starport
 :consume 400 Minerals 300 Gas :produce 1 BatleCruiser
action collect-minerals :duration 45 seg.
 :require 1 CommandCenter :borrow 1 Scv
 :produce 50 Minerals
action collect-gas :duration 20 seg.
 :require 1 Refinery :borrow 1 Scv
 :produce 25 Gas
```

Figura 5 – Algumas das principais ações do domínio de recursos do *Strarcraft*.

direto e pode envolver o uso de técnicas de busca e algoritmos. As várias etapas que constituem o processo DBD são: seleção dos dados, pré-processamento e transformação,

mineração de dados e por último interpretação e avaliação dos resultados. A Figura 6 ilustra esses passos.

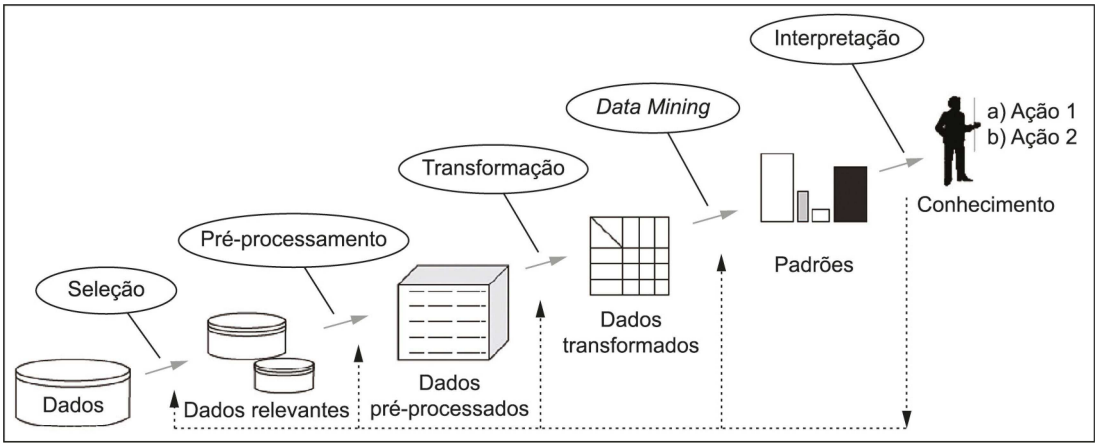

Figura 6 – Etapas do processo de descoberta de conhecimentos em bancos de dados (SILVA; COSTA, 2015).

A mineração de dados pode ser feita utilizando diferentes técnicas e estratégias. Dentre essas estratégias, as principais para mineração de dados podem ser resumidas em:

- **Classificação.** O procedimento de classificação corresponde em fazer a categorização de dados em um conjunto de classes conhecidas. Um banco de dados de treinamento, isto é, um conjunto de objetos em que se conhece as classes às quais os mesmos pertencem é dado e analisado, e um modelo de classificação é construído baseado nas características dos dados no banco de dados de treinamento. Um conjunto de regras de classificação é então gerado do modelo de classificação, as quais podem ser usadas para classificar novos dados. Por exemplo, regras de classificação sobre doenças podem ser extraídas de casos conhecidos, dados de treinamento e usadas para diagnosticar novos pacientes através de seus sintomas (E; RON, 1999).

- **Agrupamento (Clustering).** Agrupamento é o processo de agrupar um conjunto de objetos em classes de objetos semelhantes, de acordo com seus atributos. No agrupamento não existem classes pré-definidas como na classificação, estas são obtidas no decorrer do processo. Os grupos são formados de maneira que os objetos de um mesmo grupo possuam maior similaridade entre si e objetos de grupos diferentes possuam baixa similaridade. Por exemplo, os links retornados por um site de busca em resposta a uma consulta podem ser agrupados de acordo com o grau de similaridade entre os documentos encontrados (JAIN, 2010).

- **Mineração de Padrões Sequenciais.** A mineração de padrões sequenciais tem como objetivo identificar sequências de eventos (ou objetos) que ocorrem frequentemente em bancos de dados temporais. Os padrões sequenciais aparecem nos mais variados domínios de aplicações. Sua mineração pode ser utilizada para descobrir

a evolução de sintomas apresentados em pacientes, para descobrir a evolução de compras realizadas por clientes, na análise de sequências de DNA, na descoberta de caminhos (sequências de páginas acessadas) na Web frequentemente percorridos pelos usuários, etc.

Nessa tese, será dada ênfase ao procedimento de mineração de padrões sequenciais, cujo uso é feito nesta.

2.3.1 Mineração de padrões sequenciais

A Mineração de Padrões Sequenciais (MPS) é um importante problema na área de mineração de dados e possui amplas aplicações. Tal mineração, tem sido bastante utilizada no contexto de mineração de dados temporais, como por exemplo: análise de padrões de usuários em redes sociais; preferência de usuários a produtos em sites de compra; análise de sequências de DNA; etc. O problema de MPS foi introduzido por (AGRAWAL; SRIKANT, 1995). Nesse trabalho, os autores propõem, entre os outros, o algoritmo AprioriAll para mineração de padrões sequenciais em um banco de dados de transações de clientes.

Em um algoritmo de MPS os tipos de sequência e o formato do banco de dados são pontos importantes para obtenção de bons resultados. Podemos definir esses como:

- **Sequência**. Seja I um conjunto de itens. Uma sequência s é uma lista ordenada de itemsets e é denotada por $s =< s_1, s_2...s_n >$, onde cada s_i é um itemset, ou seja, $s_i \subseteq I$. O comprimento de uma sequência s, $comp(s)$, corresponde ao número total de itens dessa sequência, isto é, $comp(s) = \sum_{i=1}^{n} |si|$, onde $|si|$ denota o número de itens do i-ésimo itemset de s. Uma sequência de comprimento k é referenciada como uma k-sequência.

  Suponha um conjunto de itens $I = a, b, c, d, e$. Nesse caso, $s =< (a, b)(a)(b, c, d) >$ é um exemplo de uma sequência de comprimento 6 possuindo 3 itemsets. Um itemset pode representar, por exemplo, um conjunto de produtos comprados por um cliente em uma única transação. Repare que cada ocorrência de um item na sequência é contabilizada.

- **Sub-sequência**. Dizemos que uma sequência $\alpha = < a_1, a_2, ..., a_n >$ é uma sub-sequência de outra sequência $\beta = < b_1, b_2, ..., b_m >, m >= n$, denotado por $\alpha \subseteq \beta$, se existem inteiros $j_1 < j_2 < ... < j_n$ tais que $a_1 \subseteq b_{j1}, a_2 \subseteq b_{j2}, ..., a_n \subseteq b_{jn}$. Também chamamos β de uma super sequência de α e dizemos que β contém α ou que α está contida em β.

  As sequências $< (b)(a)(b, d) >$ e $< (b)(c) >$ são sub-sequências de $< (a, b)(a)(b, c, d) >$, mas $< (b)(b)(c) >$ não é uma sub-sequência de $< (a, b)(a)(b, c, d) >$.

- **Sequência maximal**. Dado um conjunto de sequências C, dizemos que a sequência $s \in C$ é maximal se não há nenhuma sequência em $C - s$ contendo s.

  Considere o conjunto de sequências $A = < (a)(b,c)(d) >, < (a,e)(b) >, < (a)(c)(d) >, < (a)(b) >$. A sequência $< (a,e)(b) >$ é maximal, pois nenhuma outra no conjunto a contém. Por outro lado, a sequência $< (a)(c)(d) >$ não é maximal, pois está contida na sequência $< (a)(b,c)(d) >$.

- **Banco de dados de sequências**. Um banco de dados de sequências D é um conjunto de tuplas$< ids, s >$, onde s é uma sequência e ids o identificador da sequência s. Dizemos que uma tupla $< ids, s >$ em um banco de dados de sequências D contém uma sequência α, se α é sub-sequência de s, ou seja, $\alpha \subseteq s$.

  Na Figura 7 temos um banco de dados de sequências com quatro tuplas

IdSeqs	**Sequências**
1	<(a,b)(c)(b)(d)>
2	<(c,d)(a)(a,b)>
3	<(d)(a,b)>
4	<(b,c,d)(c)(a,f)>

Figura 7 – Exemplo de um banco de dados de sequências

.

- **Suporte**. O suporte de uma sequência α em um banco de dados de sequências D, $sup(\alpha)$, corresponde à porcentagem de tuplas em D que contém α, isto é, $sup(\alpha) = \frac{|s|\exists id, <id,s>, \in D \epsilon \alpha \subseteq s|}{|D|}$.

- **Sequência frequente**. Dado um nível mínimo de suporte, sup_min, uma sequência α é dita frequente em um banco de dados de sequências D se a porcentagem de tuplas de D contendo α é maior ou igual a sup_min. Uma sequência frequente também é chamada de padrão sequencial.

  Considerando o banco de dados da Figura 7, a sequência $< (b)(c) >$ está contida nas sequências 1 e 4 desse banco de dados. Logo, o suporte de $< (b)(c) >$ é 2. Se consideramos um suporte mínimo igual a 1 (25%) então $< (b)(c) >$ é tida como uma sequência frequente desse banco de dados, visto que duas sequências contêm $< (b)(c) >$.

Existem duas principais estratégias para mineração sequencial. A primeira baseada na proposta do algoritmo AprioriAll chamada de Geração e Teste de Candidatos (GTC). A

segunda, baseada no algoritmo FPGrowth (HAN; PEI; YIN, 2000) com nome Crescimento de Padrões Frequentes (FPG).

Algoritmos GTC possuem estrutura baseada na execução de múltiplos passos sobre os dados. Em cada passo, começa-se com um conjunto semente de um número grande de sequências, chamadas de sequências candidatas. O suporte para essas sequências candidatas é achado durante a passada sobre os dados. Ao final do passo, são determinadas as maiores sequências candidatas. Essas, compõem a semente para o próximo passo (SRIKANT; AGRAWAL, 1996). Como exemplo podemos citar o Algoritmo de Classificações Sequenciais (GSP) (CHEUNG, 2002).

A abordagem FPG busca superar o GTC em situações onde o limite mínimo do suporte é baixo ou o tamanho dos padrões gerados é grande. Nesses casos, abordagens GTC tendem a manter altos custos de processamento, mesmo com uso de diferentes estruturas de dados ou modificações no algoritmo. Na abordagem baseada em GTC, a geração de candidatos é substituída pela análise com contagem de conjuntos frequentes de dados relevantes. Isso é possível porque o método preserva os agrupamentos essenciais dos elementos de dados originais para mineração. Também é utilizada uma metodologia de divisão e conquista, para reduzir o espaço de busca. O conjunto de dados e o conjunto de padrões a ser examinado em cada passo é particionado. Assim, o banco de dados sofre uma redução substancial no seu tamanho, devido à utilização de projeções do próprio banco na memória principal. Como exemplo dessa abordagem, podemos citar os algoritmos Frequent Pattern Growth (FPGrowth) (HAN; PEI; YIN, 2000) e Prefix-projected Sequential Pattern Mining (PrefixSpan) (HAN et al., 2001).

2.3.2 Algoritmo GSP

O algoritmo de mineração sequencial GSP foi proposto para a classificação de padrões sequenciais com base na propriedade Apriori. GSP é um algoritmo iterativo que encontra, a cada iteração, o conjunto dos padrões sequenciais frequentes de comprimento k, denotado por L_k. Cada iteração do GSP é realizada em duas fases: a fase da geração de candidatos e a fase do cálculo do suporte. A fase de geração de candidatos, por sua vez, é subdividida na etapa da junção e na etapa de poda. Na primeira iteração, GSP encontra o conjunto de todos os itens frequentes no banco de dados (1-sequências). Tal conjunto é utilizado na segunda iteração, para o cálculo das 2-sequências candidatas. Uma 2-sequência candidata pode conter um único itemset com 2 itens frequentes, ou dois itemsets, cada um contendo um item frequente. Por exemplo, se os itens 1 e 2 são frequentes, na segunda iteração serão geradas as 2-sequências candidatas $< (1), (2) >, < (2), (1) > e < (1, 2) >$. Em seguida, o banco de dados é percorrido e o suporte das 2-sequências candidatas é calculado.

GSP elimina as sequências candidatas com suporte inferior ao suporte mínimo. A partir da terceira iteração, o conjunto das $k - sequências$ candidatas é gerado através da

combinação de $(k-1)-sequências$ frequentes, em seguida as $k-sequências$ candidatas que não podem ser frequentes são podadas e então o banco de dados é percorrido para cálculo do suporte das sequências candidatas restantes.

A etapa de Geração de Candidatos é realizada em duas etapas. A primeira chamada de etapa da junção, onde um dado um par de $(k-1)-sequências(s1, s2)$, onde descartando o primeiro item de s_1 e o último item de s_2, obtêm-se sequências idênticas. Uma nova $k-sequência$ candidata acrescentando-se o último item de s_2 em s_1 é criada. O item adicionado estará em um itemset separado, caso o mesmo estiver em s_2, ou fará parte do último itemset de s_1, caso contrário. Por exemplo, dadas as $3-sequências < (1,3),(2) >$ $, < (3),(2),(4) >$ e $< (2,5),(4) >$, podemos realizar a junção da primeira sequência com a segunda, pois obtemos sequências iguais eliminando-se o item 1 de $< (1,3),(2) >$ e o item 4 de $< (3),(2),(4) >$. A nova sequência obtida é a $4-sequências$ candidata $< (1,3),(2),(4) >$.

Na segunda etapa de geração de candidatos acontece a chamada etapa da poda. Nesta etapa, são eliminadas as $k-sequências$ candidatas que possuem pelo menos uma $(k-1)$-subsequência não frequente. Isso acontece devido a uma $(k-1)-subsequência$ que não se encontra no conjunto $L_k - 1$ de sequências frequentes, gerado na iteração anterior.

No GSP a etapa de cálculo do suporte começa com um dado de um conjunto de sequências candidatas C_k, o banco de dados é percorrido uma vez e o suporte de cada $k-sequência$ candidata é calculado. As $k-sequências$ candidatas cujos suportes não atingem o suporte mínimo α são eliminadas e o conjunto L_k de sequências frequentes é obtido. Para cada sequência s do banco de dados, o algoritmo verifica quais $k-sequências$ candidatas são subsequências de s, incrementando o contador de suporte dessas sequências candidatas. Para reduzir o número de sequências candidatas que são testadas para cada sequência s do banco de dados, GSP armazena o conjuntos das sequências candidatas em uma árvore hash. Um nó folha dessa árvore armazena uma lista de sequências candidatas enquanto um nó interno mantém uma tabela hash com ponteiros para outros nós do nível seguinte. Maiores detalhes de como a árvore é utilizada nesta fase do algoritmo podem ser encontrados em (SRIKANT; AGRAWAL, 1996).

2.3.3 Algoritmo FPGrowth

O algoritmo FPGrowth é um método eficiente e escalável para a mineração de padrões frequentes, sejam eles curtos ou longos. Uma estrutura chamada FP-Tree é utilizada. Essa, baseia-se no crescimento de fragmentos de padrões, que armazena informações quantitativas sobre padrões frequentes de uma forma comprimida compacta (HAN et al., 2000). A partir do conceito do FPGrowth, outros algoritmos com modificações e adições na estratégia principal foram propostos, entre esses está o PrefixSapn (HAN et al., 2000).

O PrefixSpan explora a projeção de prefixos na mineração de padrões sequenciais. Sua ideia principal é que, ao invés de projetar sequências de bancos de dados considerando-se todas as ocorrências possíveis de subsequências frequentes, a projeção seja baseada apenas em prefixos frequentes. A ideia é que qualquer subsequência crescente pode sempre ser encontrada pelo crescimento de um prefixo crescente.

O PrefixSpan será utilizado para ilustrar o funcionamento de um algoritmo que utiliza abordagem de crescimento de padrões. Seja o banco de dados de sequências S dado pela Figura 8 e o suporte mínimo igual a 2. O conjunto de itens no banco de dados é a, b, c, d, e, f, g. É preciso encontrar o conjunto completo de padrões sequenciais no banco de dados.

IdSeqs	*Sequências*
10	<a(abc)(ac)d(cf)>
20	<(ad)c(bc)(ae)>
30	<(cf)(ab)(df)cb>
40	<eg(af)cbc>

Figura 8 – Exemplo de um banco de dados de sequências

.

- Passo 1: Encontrar padrões sequenciais de tamanho-1. Rastreamento de S uma vez para encontrar todos os itens frequentes em sequências. O resultado é: $< a >: 4$, $< b >: 4$, $< c >: 4$, $< d >: 3$, $< e >: 3$, e $< f >: 4$, onde $<$ *padrão* $>$: *contagem* representa o padrão e sua contagem de suporte associada.

- Passo 2: Dividir o espaço de busca. O conjunto completo de padrões sequenciais pode ser particionado em seis subconjuntos, de acordo com os prefixos: (1) aqueles que têm o prefixo $< a >$; ...; e (6) aqueles que têm o prefixo <f>.

- Passo 3: Encontrar subconjuntos de padrões sequenciais. Os subconjuntos de padrões sequenciais podem ser minerados pela construção de bancos de dados projetados e mineração de cada um recursivamente.

A Figura 9 exibe o resultado final dos passos executados sobre S.

2.4 Árvore de Decisão

A Árvore de Decisão (DT) é um eficiente método não paramétrico que pode ser aplicado a tarefas de classificação em *data mining* ou regressão. As DT são estrutura de dados hierárquicas para aprendizado supervisionado, onde a espaço de entrada é divido em regiões locais que habilitam a previsão de variáveis dependentes (SAFAVIAN; LANDGREBE, 1991).

Prefixo	*Banco de Dados Projetado*	*Padrões Sequenciais*
<a>	<(abc)(ac)d(cf)>, <(_d)c(bc)(ae)>,<(_b)(df)cb>, <(_f)cbc>	<a>, <aa>, <ab>, <a(bc)>, <a(bc)a>, <aba>, <abc>, <(ab)>, <(ab)c>, <(ab)d>, <(ab)f>, <(ab)dc>, <ac>, <aca>, <acb>, <acc>, <ad>, <adc>, <af>
<b>	<(_c)(ac)d(cf)>, <_c)(ae)>, <(dfcb>, <c>	<b>, <ba>, <bc>, <(bc)>, <(bc)a>, <bd>, <bdc>, <bf>
<c>	<(ac)d(cf)>, <(bc)(ae)>, <b>, <bc>	<c>, <ca>, <cb>, <cc>
<d>	<(cf)>, <c(bc)(ae)>, <(_f)cb>	<d>, <db>, <dc>, <dcb>
<e>	<(_f)(ab)(df)cb>, <(af)cbc>	<e>, <ea>, <eab>, <eac>, <eacb>, <eb>, <ebc>, <ec>, <ecb>, <ef>, <efb>, <efc>, <efcb>
<f>	<(ab)(df)cb>, <cbc>	<f>, <fb>, <fbc>, <fc>, <fcb>

Figura 9 – Bancos de dados projetados e padrões sequenciais.

Uma DT pode ser representada como um grafo $G = (V, E)$ em forma de árvore, com uma quantidade finita não nula de nós V e um conjunto de arestas E. O grafo deve satisfazer as seguintes regras:

- ❏ As arestas devem ser pares ordenados (v, w) de vértices, ou seja, o grafo deve ser direcionado.
- ❏ Não devem existir ciclos no grafo, ou seja, o grafo deve ser acíclico.
- ❏ Existe um caminho único, uma sequência de arestas na forma $(v1, v2), (v2, v3), ..., (v_{n-1}, v_n)$, para cada nó a partir da raiz.
- ❏ Quando existe um caminho a partir do nó v para o w, $v \neq w$, v é um ancestral de w e w é um descendente de v. Um nó que não possuo descentes é chamado de "folha" (ou terminal).

Nós internos e o nó raiz armazenam valores correspondentes a testes feitos sobre um conjunto de atributos, e as arestas correspondem aos possíveis resultados desses testes. Nós folhas podem armazenar rótulos de classificação, valores contínuos de regressão, modelos não lineares de regressão e modelos produzidos por qualquer algoritmo de mineração de dados. Para prever os valores de variáveis dependentes de um cenário qualquer, é preciso percorrer a árvore de decisão, iniciando pelo nó raiz e seguindo pelas arestas de acordo com o resultado dos testes de atributos. Quando um nó folha é atingido, o resultado preditivo é retornado, o que caracteriza a DT como uma disjunção de restrições conjuntas sob os valores de atributos (PRADHAN, 2013).

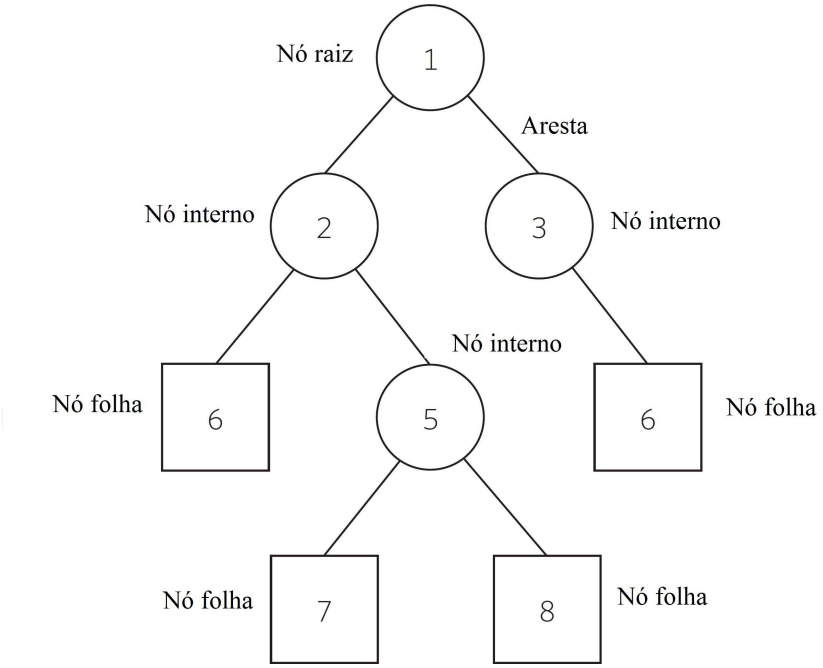

Figura 10 – Exemplo de uma árvore de decisão. Nó raiz e demais internos armazenam valor preditivo ou variável decisória, nós folhas com o valor da variável decisória. Arestas estabelecem as conexões entre os nós, conduzindo para os respectivos valores

Uma DT pode crescer e atingir uma grande quantidade de nós, bem como atingir um alto valor de altura para uma determinada base de dados. Construir uma DT binária mínima com todos os conjuntos de testes para identificar objetos ocultos, pode ser um problema NP-Completo (ZANTEMA; BODLAENDER, 2000). Assim, os algoritmos mais eficientes na construção e execução de DT são baseados em estratégias de particionamento recursivo (indução crescente) e com uso de heurísticas (ROKACH; MAIMON, 2005).

Os algoritmos com abordagem de indução crescente mais utilizados são ID3 (QUINLAN, 1986), C4.5 (CHAWLA, 2003), CART (STEINBERG; COLLA, 2009) e QUEST (WILKINSON, 1992). Nessas, a cada iteração o algoritmo considera a partição do conjunto de treino utilizando o retorno de uma função discreta dos atributos de entrada. A função reflete a estratégia de divisão que será tomada, a escolha da função mais adequada é feita observando os critérios de divisão de classes que é oferecida pelo conjunto de treino. Após a escolha do método de divisão, cada nó subdivide o conjunto de treino em subconjunto menores, até que nenhuma divisão apresente ganho de informação ou algum critério de parada seja atingido.

Para realizar uma divisão nos conjuntos de testes é preciso escolher qual atributo é mais apropriado. O atributo que melhor discrimine o dado de entrada é escolhido, uma regra de decisão é gerada e o dado de entrada é filtrado de acordo com os resultados dessa regra. O critério de escolha baseado em teoria da informação é chamado de entropia. Esse, representa a média de informação quando cada classe é rotulada com tamanho ideal de

acordo com sua probabilidade. Com base na entropia foi criado o critério de Informação Mútua Global (IMG), representado pela Equação 1.

$$IMG(a_i, X, y) = \frac{1}{N_x} \sum_{l=1}^{k} \sum_{j=1}^{|a_i|} N_{vj \bigcap yl} \log_e \frac{N_{vj \bigcap yl} N_x}{N_{vj,} \bullet N_{\bullet, yl}} \tag{1}$$

Sempre que uma partição é induzida por um conjunto X e um atributo a_j, existe um refinamento da partição induzido por a_i. A entropia de uma partição induzida por a_j nunca será maior que a entropia da partição induzia por a_i. Isso mostra que o refinamento contínuo de uma partição em subpartições, sempre irá decrescer continuamente o valor de entropia, independente da distribuição de classes obtida depois de particionar o conjunto original. O cálculo do GMI tem limite inferior igual a zero quando a_i e y são independentes, seu limite máximo é igual a $max(\log_2 |a_i|, \log_2 k)$, quando existe uma correlação entre a_i e y. Essa medida tende a acompanhar atributos com valores distintos, para eliminar esse viés a normalização chamada Redundância de Atributo Classe (RAC) da Equação 2 é utilizada:

$$RAC(a_i, X, y) = \frac{IMG}{-\sum_{j=1}^{|a_i|} \sum_{l=1}^{k} p_{vj \bigcap yl} \log_2 p_{vj \bigcap yl}} \tag{2}$$

onde GMI é divido pela entropia conjunta de a_i e y. No RAC, $(a_i, X, y) \geq 0$, uma vez que ambos IMG e a entropia conjunta são maiores ou igual a zero.

O objetivo do ganho de informação é maximizar a redução da entropia devido as divisões de conjuntos em subconjuntos. Na Equação 3 a entropia bem como a redução são exibidas.

$$\phi^{entropia}(X, y) = -\sum_{l=1}^{k} rac_{\bullet, yl} \times \log_2 rac_{\bullet, yl} \tag{3}$$

Com a entropia definida, é preciso calcular a medida do Ganho de Informação (GI). GI mede a qualidade da divisão de conjunto de instâncias X, de acordo com os valores do atributo a_i. A Equação 4 define o GI e a Equação 5 define a Proporção do Ganho (PG). PG normaliza o GI baseado na entropia do atributo que estiver sendo testado, essa normalização é devido a tendência de surgir diversos atributos com valores parecidos. O decaimento da entropia em múltiplas partições é compensado, dividindo o GI pelo atributo que representa a entropia $\phi^{entropia}(a_i, X)$. Esse valor, aumenta de forma logarítmica a medida que o número de partições sob a_i também aumenta, diminuindo o valor da PG.

$$\Delta\Phi(a_i, X, y) = \phi(y, X) - \sum_{j=1}^{|a_i|} rac_{j, \bullet} \times \phi(y, X_{ai=vj}) \tag{4}$$

$$\Delta\Phi^{proporcaoGanho}(a_i, X, y) = \frac{\Delta\Phi^{GI}}{\phi^{entropia}(a_i, X)} \tag{5}$$

2.4.1 Algoritmo CHAID

Chi-squared Automatic Interaction Detector (CHAID) (KASS, 1980) é um algoritmo que utiliza testes Qui-Quadrado de Pearson (LIU; SETIONO, 1995) em variáveis independentes. Esse, testa em tabelas a associação entre duas variáveis de categoria dependente e independente, onde quaisquer variáveis independentes são discretizadas com base em classes. CHAID é um algoritmo de construção de DT que não utiliza o princípio de indução crescente. Esse, foca em realizar múltiplas divisões em cada nó e esses são discretizados em função de valores estatísticos e preditivos. CHAID é um método estatístico, com boa performance para a segmentação e controle do crescimento da árvore em grandes base de dados.

CHAID utiliza recursões consecutivas, essas geram partições em uma população de dados através da criação de nós distintos representando classes. Esses nós, são particionados de tal forma que a variação da resposta categórica é minimizada dentro dos nós internos e maximizada entre os demais nós. O particionamento inicial da base de dados gera dois ou mais nós, que são definidos por valores de variáveis independentes ou de predição. O processo de particionamento é repetido em cada um dos nós restantes, onde cada um desses é tratado como uma nova subpopulação. Um novo particionamento definido dessa vez por uma variável preditiva em dois ou mais novos nós é feito, a variação da resposta é novamente minimizada internamente e maximizada externamente entre os conjuntos. Esse processo de repetição será interrompido caso alguma condição seja atingida, ou o valor de classe em uma partição repetir-se no momento da sua geração.

Um passo essencial na construção do modelo de predição do CHAID, é a seleção das características relevantes para classificação. Qui-Quadrado é o teste estatístico que mede a divergência entre a distribuição esperada, caso a ocorrência de características seja independente do valor de classe. O teste, avalia características individualmente calculando o Qui-Quadrado de cada nó estatisticamente em relação a sua classe e aos nós vizinhos. Se a variável independente Y é nominal categórica, a independência de X e Y é testada. Caso seja positivo o teste, a função Qui-Quadrado (X^2) representado pela Equação 6 é utilizado para calcular o valor do nó. Caso negativo, significa que a variável independente é contínua, assim o teste de Fisher (G^2) representado pela Equação 7 é utilizado:

$$X^2 = \sum_{j=1}^{J}\sum_{l=1}^{L}\frac{(n_{ij} - \hat{m}_{ij})}{\hat{m}_{ij}} \tag{6}$$

$$G^2 = 2\sum_{j=1}^{J}\sum_{l=1}^{L} n_{ij}\ ln(n_{ij}/\hat{m}_{ij}) \tag{7}$$

onde $n_{ij} = \sum_{n\in D} f_n I(x_n = i \wedge y_n = j)$ é a frequência observada de ocorrências da variável independente Y, utilizada para calcular o Qui-Quadrado. $\hat{m}_{ij}$ é frequência esperada de células com X e $(x_n = i, y_n = j)$, a partir do modelo independente. O valor final é representado por p sendo dado por $p = Pr(\chi_d^2 > X^2)$ para o teste de Qui-Quarado ou

$p = Pr(\chi^2_d > G^2)$ quando o teste de Fisher é utilizado. χ^2_d segue a distribuição da equação de Qui-Quadrado, com os respectivos graus de liberdade $d = (J-1)(I-1)$.

Com variável independente Y sendo categórica ordinal, a hipótese de independência de X e Y é testada contra o modelo de efeitos nas linhas dos dados (H^2). As linhas, são as categorias de X e as colunas são classes de Y. No modelo de efeitos o teste é feito utilizando Razão de Verosimilhança (H^2). Dois conjuntos de células frequentes são estimadas, o primeiro $\hat{m}_{ij}$, com hipótese de independência, e $\hat{\hat{m}}_{ij}$ com hipótese dos dados seguirem modelo de efeito nas linhas. Em ambos, $\hat{\hat{m}}_{ij} = \dfrac{n_i n_j}{n}$ e o valor de retorno é definido com base em p. A razão de verosimilhança e o retorno da função p são dados pela Equação 8 respectivamente.

$$\begin{aligned} H^2 &= 2\sum_{i=1}^{I}\sum_{j=1}^{J}\hat{\hat{m}}_{ij} ln(\hat{\hat{m}}_{ij}/\hat{m}_{ij}), \\ p &= Pr(\chi^2_{I-1} > H^2). \end{aligned} \tag{8}$$

No modelo de efeito nas linhas, são necessárias as pontuações (β) estabelecidas para as classes geradas a partir das variáveis de Y. Essas pontuações, complementam a frequência observada de Y, adicionando diferentes valores no uso de variáveis dependentes com um conjunto de variáveis independentes distintas. Esses valores são representados em forma de peso w_{ij}^{-1}, com os valores do nível α da árvore e com a atual frequência das células visitadas no passo anterior γ. Os valores de pontuação, podem ser configurados utilizando conhecimento *ad hoc*, contudo os resultados são configurados no início da árvore e não são alterados durante a execução do CHAID. Assim, para obter resultados com maior confiabilidade, é utilizado a frequência de células estimadas, dado pela Equação 9. Nessa, β_j^{k+1} é um parâmetro adicional a definição de $\hat{\hat{m}}_{ij} = \dfrac{n_i n_j}{n}\beta_j^{k+1}$, seu valor é calculado proceduralmente.

$$\beta_j^{k+1} = \frac{n_j}{\sum_i w_{ij}^{-1}\alpha_i^{(k+1)}(\gamma_i^{(k)})^{(s_j - s)}} \tag{9}$$

O algoritmo CHAID possui muitas propriedades vantajosas. O crescimento incorreto ou *overfitting* da árvore é controlado automaticamente, sem uso de poda e apenas com balanceio dos ramos. A predição é calculada junto aos dados de classificação, unindo duas tarefas em uma só, com garantia de melhor desempenho em grandes bases de dados. A árvore gerada pelo CHAID diferente de outros algoritmos de DT não é exclusivamente binária, assim é possível expandir a relação de classes de níveis superior com as demais presente na árvore. O teste de particionamento do algoritmo é simples de ser utilizado com base na variável independente, onde: caso essa seja contínua, o teste Fisher é utilizado; caso seja nominal então o teste do Qui-Quadrado é utilizado; em último caso, se a variável independente é ordinal a Razão de Verosimilhança é utilizada.

2.5 Planejamento Reativo

Planejamento Reativo (PR) (FIRBY, 1987), é um paradigma que permite que as tarefas de planejamento em um determinado ambiente, possam adaptar-se a incerteza e modificações inesperadas que venham a ocorrer. O planejamento considerado "clássico" é aquele onde a partir de um estado inicial e um objetivo bem definido, é preciso encontrar uma sequência de ações que serão executadas interruptamente, até que o estado objetivo seja alcançado. Em uma arquitetura com PR, a cada instante uma ação é escolhida para execução e nenhuma sequência futura de ações é planejada. Assim, o PR foca no instante atual em que o ambiente se encontra, permitindo que a arquitetura possa operar junto aos efeitos de incerteza e gerenciar eventos internos e externos que possam ocorrer (WOOLDRIDGE; JENNINGS, 1995).

O uso de PR é geralmente atribuído a uma arquitetura ou sistema com funcionamento baseado em módulos. Essa atribuição, remete a necessidade de mecanismos e dados sensoriais que são utilizados para controlar quando o sistema deve alterar seu funcionamento, quando o ambiente passa por mudanças ou quando eventos inesperados estão em ocorrência. Com módulos bem definidos, o PR ganha autonomia sobre partes específicas da arquitetura, gerenciando ações em relação a sua execução, seu cancelamento e a proposição de novas ações. A Figura 11, exibe um modelo de arquitetura geral para uso de PR no domínios de jogos RTS.

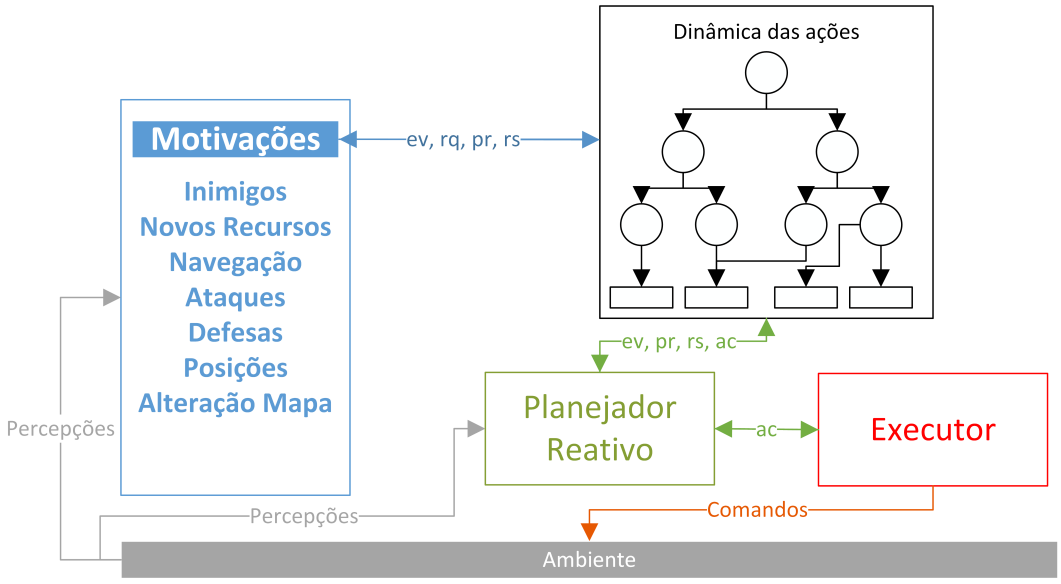

Figura 11 – Arquitetura geral para uso e planejamento reativo em jogos RTS

.

As percepções do ambiente são passadas em forma de dados sensoriais ao módulo *Motivações* e ao *PlanejadorReativo*, de acordo com a Figura 11. *Motivações* ao receber os dados, busca definir qual a motivação atual do(s) inimigo(s) que pode ser deduzida, determinando algum tipo de estratégia para mitigar essa intenção. Atacar o inimigo ou

tentar identificá-lo seriam exemplos de possíveis ações contra uma motivação identificada. Para executar qualquer ação é feita uma consulta a *Dinâmicadasações*, que determina quais recursos devem existir para que a execução possa acontecer. As informações utilizados nessa comunicação são: *ev* que representa os eventos que serão gerados no ambiente com a execução da ação escolhida; *rq* é uma requisição de informação, que será preenchida com a quantidade de recursos que já existem no ambiente e que podem ser utilizados para executar a nova ação; *pr* é uma lista de parâmetros que inclui o estado em que os recursos devem estar para executar a ação; *rs* é o resultado do comportamento da ação, o comportamento é uma descrição formal da dinâmica da ação, utilizando uma linguagem reativa.

A *Dinâmicadasações* na Figura 11 é consultada para que as informações sejam repassadas ao *PlanejadorReativo*. Esse, é responsável por escolher a ação que será executada e utiliza o parâmetro de dinâmica para configurar a mesma. O parâmetro *ac* representa as possíveis ações que podem ser executadas e suas respectivas sequências de execução. O *PlanejadorReativo* escolhe e determina como será a execução da ação no ambiente, levando em conta quaisquer ações adicionais necessárias para satisfazer as precondições da ação atual durante a execução. Toda a sequência de passos para execução da ação é representada através de um comportamento, que é entregue ao *Executor* através da adição de dados ao parâmetro *ac*. O *Executor*, executa os comandos dentro do ambiente mantendo a sequência de execução definida para a ação escolhida.

O *PlanejadorReativo* continua a receber as percepções do ambiente, enquanto a ação é executada. Utilizando o comportamento para guiar a execução da ação e rastrear o andamento dessa, o módulo verifica continuamente se a ação deve ser interrompida e uma nova considerada para execução. Caso as percepções indiquem que a ação deve ser cancelada, o módulo verifica quais recursos foram gerados até o momento, a situação do ambiente, e decide com base no conjunto de informações qual é a nova ação. No processo de escolha, o módulo *Motivações* e *Dinâmicadasações* são utilizados.

Para utilizar PR a arquitetura exibida na Figura 11 pode ser expandida com adição de novos módulos, além do uso de diferentes técnicas em cada um desses. Algoritmos de PR em muitos casos utilizam Máquina de Estados Finitos (FSM) (CHOW, 1978) e algoritmos conexionistas (MARESCHAL; SHULTZ, 1996) para representar o módulo *PlanejadorReativo*. Essa escolha é feita com base no nível de reatividade desejado para o sistema. Algoritmos de *Steering* (Direção) (HESS; MODJTAHEDZADEH, 1990) utilizados em navegação de agentes, por exemplo, utilizam lógica *fuzzy* (MENDEL, 2001) para obter um elevado nível de reatividade. Assim, o módulo *PlanejadorReativo* pode ser visto como um interpretador, que com base nas percepções pode julgar quando acionar a reatividade com base nos critérios de importância para o sistema. No exemplo de sistema reativo para Direção, os critérios de maior importância seriam, por exemplo, seguir em direção ao objetivo, evitar obstáculos e inimigos. O PR é executado para que o sistema

consiga atingir um objetivo dentro do ambiente, contudo esse critério não deve ser seguido de forma fixa como no planejamento clássico. Outros critérios e principalmente as mudanças ocorridas no ambiente, são utilizadas para condicionar o que o sistema deve fazer, quais ações devem ser escolhidas e quando o planejamento atual eve ser interrompido.

2.6 Processo de Decisão de Markov Parcialmente Observável

Um Processo de Decisão de Markov Parcialmente Observável (POMDP) (KAELBLING M. LITTMAN, 1998) é uma forma de modelar processos onde as transições entre estados são probabilísticas, no qual um agente executa ações sem ser capaz de observar diretamente o estado subjacente. Cada ação tem uma recompensa ou custo, que depende do estado em que o processo se encontra. As transições de estados realizadas por meio de ações também estão associadas a probabilidades específicas. O agente tem então que decidir suas ações com base em algumas observações locais e em distribuições de probabilidade sobre os estados do mundo que está sendo modelado. O conjunto de probabilidades que determina estar em um estado específico em um determinado momento é chamado de crença.

Um determinado processo é considerado "Markoviano" desde que os subprocessos modelados obedeçam a propriedade de Markov: o efeito de uma ação em um estado depende apenas da ação e do estado atual do sistema e não o modo como o processo chegou ao estado atual (CASSANDRA, 1998). São chamados de processos de decisão porque modelam a possibilidade de um agente ou tomador de decisões executar ações. Um POMDP é uma tupla (S, A, O, T, Z, R), onde:

- S é um conjunto de estados em que o processo pode estar;
- A é um conjunto de ações que podem ser executadas em diferentes épocas de decisão;
- O é um conjunto de observações que são obtidas em cada época de decisão e diretamente do ambiente;
- T: $SxAxS \mapsto [0, 1]$ é uma função que dá a probabilidade onde $T(s, a, s') = Pr(s'|s, a)$ representando a incerteza pela probabilidade do agente executar uma ação a em um estado s e ir para um estado s';
- Z: é uma função de observação $SxAxZ \mapsto [0, 1]$ onde $O(s', a, z) = Pr(z|a, s')$ gera a probabilidade do agente perceber z após executar a ação a e ir para o estado s', a probabilidade representa a incerteza na detecção da observação;
- R:$SxA \mapsto R$ é a função de recompensa ou custo imediato, dado que o agente executou uma ação a no estado s;

Como os estados de um sistema não estão acessíveis ao tomador de decisões, o mesmo utiliza apenas um conjunto $z \in Z$ de observações para determinar o provável estado atual. Representar o histórico de ações e observações seria muito dispendioso. Assim, o POMDP representa apenas informações relevantes, representando os elementos através de uma distribuição de probabilidade sobre o espaço de estados S, chamado de estado crença. Um estado crença b_t no tempo t é definido pela Equação 10.

$$b_t = Pr(s_t = s|h_t, b_0). \tag{10}$$

O agente pode tomar uma decisão com base no seu estado crença b_t ao invés de consultar todo as ações e observações passadas. A qualquer tempo t, o estado crença b_t pode ser recalculado a partir do estado anterior b_{t-1} utilizando a ação prévia a_{t-1} e a obervação atual z_t. Esse, é feito utilizando a função de atualização de estado crença $\tau(b, a, z)$, onde $b_t = \tau(b_{t-1}, a_{t-1}, z_t)$ é definido pela Equação 11.

$$b_t(s') = \tau(b_{t-1}, a_{t-1}, z_t)(s') = \frac{1}{Pr(z_t|b_{t-1}, a_{t-1})} O(s', a_{t-1}, z_t) \sum_{s \in S} T(s, a_{t-1}, s') b_{t-1}(s), \tag{11}$$

onde $Pr(z|b, a)$ é a probabilidade de observar z após executar a ação a no estado crença b. Essa, comporta-se como uma constante de normalização assim como b_t permanece como uma distribuição probabilística na Equação 12:

$$Pr(z|b, a) = \sum_{s' \in S} O(s', a, z) \sum_{s \in S} T(s, a, s') b(s) \tag{12}$$

Com as definições anteriores um agente está apto a computar suas crenças dentro do modelo do POMDP. Para escolher uma ação, é preciso utilizar uma política π, que especifica a probabilidade de executar qualquer ação em qualquer estado crença (SMITH; SIMMONS, 2012). Uma política define a estratégia do agente, a maximização da recompensa ganha sobre um horizonte finito é, por exemplo, uma das mais utilizadas. A recompensa é dada para cada ação e seu respectivo estado crença, seu critério de otimalidade é conseguir a o valor máximo dentre as recompensas possíveis. Uma política ótima π^* é definida pela Equação 13.

$$\pi^* = \underset{\pi \in \Pi}{argmax}\ E\left[\sum_{t=0}^{\infty} \gamma^t \sum_{s \in S} b_t(s) \sum_{a \in A} R(s, a) \pi(b_t, a) | b_0\right], \tag{13}$$

onde $\gamma \in [0, 1]$ é o fator de desconto e $\pi(b_t, a)$ é a probabilidade da ação a ser executada no estado crença b_t, como descrito pela política π.

O retorno obtido após executar uma política específica π, a partir de um determinado estado crença b, é definido pela função valor V^π Equação 14.

$$V^\pi = \sum_{a \in A} \left[R_B(b, a) + \gamma \sum_{z \in Z} PR(z|b, a) V^\pi(\tau(b, a, z))\right]. \tag{14}$$

O framework do POMDP utiliza as equações anteriores para definir seu processo operacional. Sempre que novas observações o são percebidas no ambiente, essas são enviadas ao estimador de estados τ, responsável por gerar os estados de crença. Sob os estados é aplicada a política π, que mapeia as ações nos respectivos estados calculando as recompensas descontadas. Uma ação é escolhida e executada no ambiente, que devolve novas observações para que o processo do POMDP seja novamente iniciado. A Figura 12 exibe o comportamento do POMDP em seu processo de escolha de uma ação.

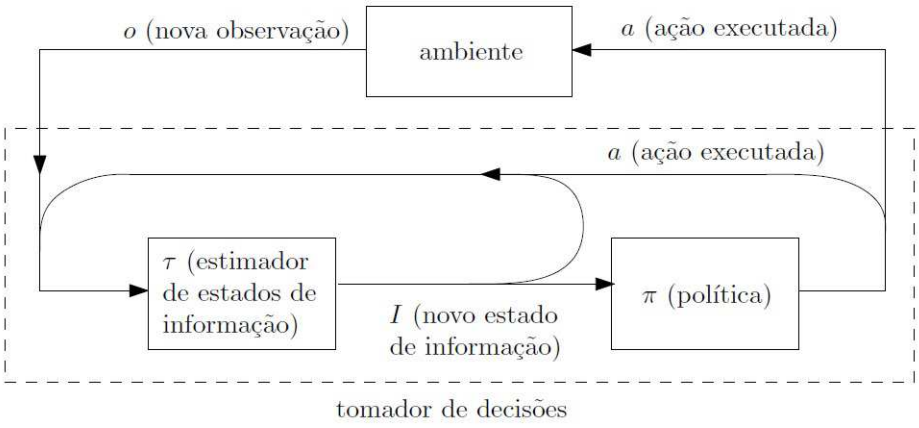

Figura 12 – Representação do processo de decisão do POMDP (PELLEGRINI; WAINER, 2007)

.

Solucionar problemas com extenso horizonte de eventos ou com grande espaço de estados utilizando POMDP é uma tarefa complexa. Em muitos casos, utilizar uma função valor e uma política ótima podem tornar o algoritmo impraticável. Para esses tipos de problemas existem duas abordagens, a primeira que utiliza de soluções *offline* aproximadas, e a segunda com uso de POMDP em sua versão *online*. Soluções aproximadas são utilizadas quando o intervalo para obter uma resposta é grande mas limitado. Esses algoritmos utilizam versões simplificadas do POMDP para calcular limites superiores e inferiores. Esses, são complementos que orientam a busca por estados em direções promissoras. Também são utilizados como base para aplicação de técnicas como poda de ramos na árvore de busca, e para estimar recompensas de longo prazo em estados crença.

Algoritmos *online* são focados em obter respostas em menos de segundos. O processo operacional do POMDP é alterado para que o mesmo consiga operar da forma mais rápida possível. Nesta tese, o POMDP *online* é utilizado e modificado. Sua estrutura de funcionamento é descrita a seguir.

2.6.1 POMDP Online

Algoritmos *online* focam em encontrar boas políticas locais para o atual estado crença em que o agente se encontra (PAQUET; TOBIN; CHAIB-DRAA, 2005). Algoritmos

aproximados retornam uma política que define qual ação executar em cada possível estado crença. Mesmo em ambientes de tamanho médio e com uma política simples, algoritmos aproximados tendem apresentar complexidade exponencial na quantidade combinada de observações, estados e ações. Políticas locais possuem a vantagem de focar apenas em estados crença que são alcançáveis a partir do atual estado. O esforço computacional é concentrado em pequenos conjuntos de crenças que varrem diversas partes do espaço de estados.

A execução do POMDP *online* é feita intercalando a etapa de planejamento e execução, diferente dos algoritmos aproximados que executam a fase de planejamento para todo o cenário e em seguida a execução, sem retorno a nenhuma das fases. A Figura 13 exibe a diferença no comportamento das abordagens de POMDP.

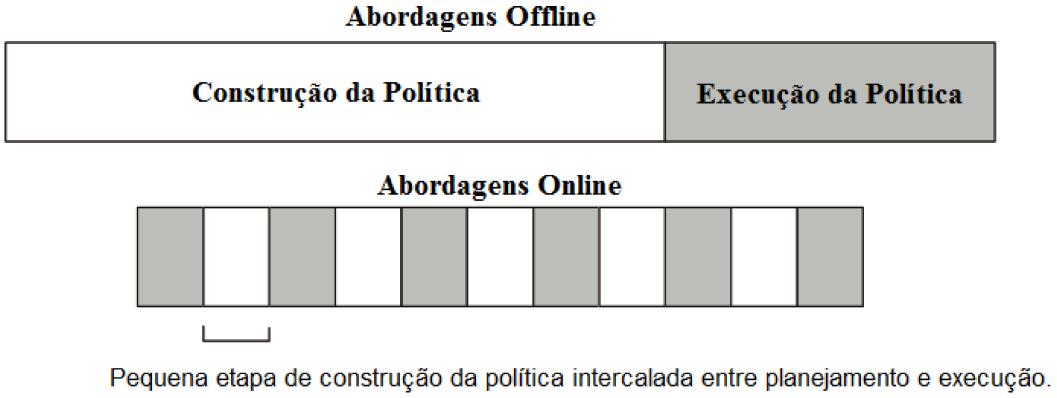

Figura 13 – Diferença no processo de execução das abordagens de POMDP *offline* e *online* (ROSS et al., 2008)

.

Na etapa de planejamento, o algoritmo a partir do estado crença atual define a melhor ação a ser executada. Nessa etapa são efetuados basicamente dois passos. O primeiro, uma árvore com estados crença alcançáveis a partir do atual é construída, com várias sequências de ações e observações que podem ser combinadas com esse estado. Nessa árvore, o estado atual é a raiz e os demais estados obtidos utilizando a função $\tau(b, a, z)$ da Equação 11, são adicionados a árvore como filhos dos estados anteriores. A árvore geralmente é uma *and-or* (KUMAR; KANAL, 1983), representando crenças como nós *or* e ações entre os nós crenças por nós *and*. A Figura 14 exibe um exemplo da árvore.

Na Figura 14, os estados crença são representados pelos nós triangulares, enquanto as ações pelos nós circulares. As recompensas $R_B(b, a)$ são representadas por valores fixados as setas que saem dos nós *or*. As probabilidades $PR(z|b, a)$ são exibidas em valores fixados nas setas dos nós *and*. Os valores dentro dos colchetes são os limites máximos e mínimos, que serão melhor descritos na seção 6.1. Assim que uma ação é escolhida, a fase de execução é iniciada e a ação escolhida como melhor é executada dentro do ambiente. O atual estado crença é atualizado com novas observações e o processo é

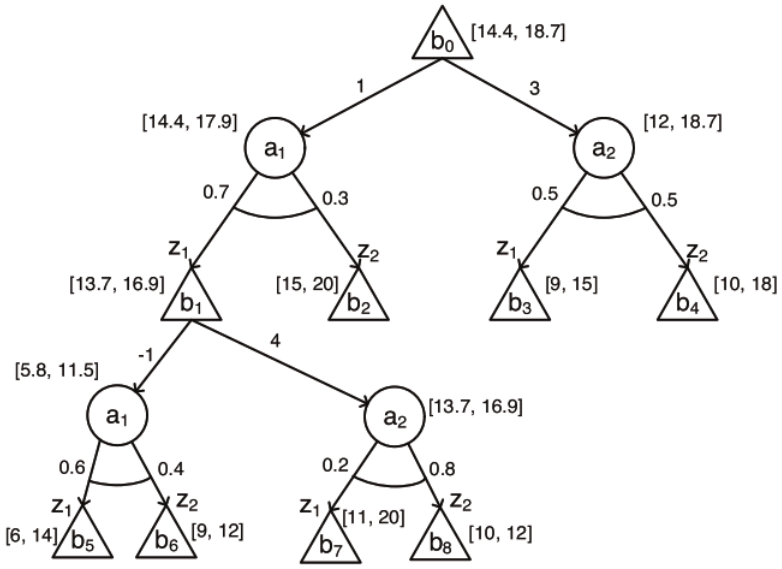

Figura 14 – Uma árvore *and-or* construída pelo processo de busca do POMDP *online*, com 2 ações e 2 observações (ROSS et al., 2008)

.

reiniciado novamente na etapa de planejamento. O Algoritmo 1 exibe uma implementação padrão de um POMDP online.

Algoritmo 1 $POMDPonline()$

1: b_c Estado crença atual
2: A Árvore and-or representando a atual árvore de busca dos estados crença
3: R Fator de ramificação
4: LI Limite inferior em V*
5: LS Limite superior em V*
6: $b_c \leftarrow b_0$ (solução atual recebe solução inicial)
7: Inicializa A com b_0 como nó raiz
8: **while** ExecucaoFim() não seja atingida **do**
9: **while** PlanejamentoFim() não seja atingida **do**
10: $b* \leftarrow$ ExpandeProxNo()
11: Expande($b*, D$)
12: AtualizaAncestrais($b*$)
13: **end while**
14: Executa melhor ação a para b_c
15: Recebe novas observações z
16: $b_c \leftarrow$StCrenca(b_c, a, z)
17: Atualiza A e coloca b_c como nova raiz
18: **end while**

O Algoritmo inicializa a árvore *and-or* com apenas o estado crença inicial (linhas 6 e 7). Com a árvore inicializada, o algoritmo começa a fase de planejamento selecionando o próximo nó a ser gerado (linha 10) com a função $ExpandeProxNo()$, que dá início a busca e construção do restante da árvore *and-or* utilizada pelo POMDP. A função

$Expande(b*, R)$ (linha 11) constrói os próximos estados crença. $AtualizaAncestrais(b*)$ (linha 12) propaga o valor aproximado de cada novo estado crença aos nós de seus antecessores, que também são estados crença. Em seguida é iniciada a fase de execução do POMDP, que executa a melhor ação encontrada (linha 14). Com a execução da ação o POMDP recebe novas observações (linha 15), a função $StCrenca(b_c, a, z)$ (linha 16) atualiza os estados de crença co as novas observações.

2.7 Escalonamento

A tarefa de um escalonador de ações é alterar a disposição de execução de um conjunto de ações para que as mesmas sejam executadas paralelamente e o mais cedo possível (ZHANG; DIETTERICH, 1995). O mesmo é resumido formalmente como: "a capacidade de distribuir, durante um intervalo de tempo, as tarefas a serem executadas sobre recursos limitados, buscando otimizar tanto tempo quanto a utilização de recursos". Formalmente um problema de escalonamento é definido como um conjunto de tarefas $J = J_1, ..., J_n$ e um conjunto de recursos $R = R_1, ..., R_n$, onde cada tarefa J_i consiste de um conjunto de operações $O^i = O^i_1, ..., O^i_n$ que devem ser realizadas entre o tempo de prontificação, T_{pi}, e o tempo de execução, $T_{e0}i$. Na execução de cada operação O^i_k é necessária a utilização de um conjunto de recursos $R^i_k \subseteq R$ durante determinado intervalo de tempo T_{ui}. O tempo de início T^i_k da operação O^i_k indica quando a operação deve utilizar o recurso R^i_k. Este tipo de problema pode ser considerado como um problema de satisfação de restrições.

Em jogos RTS, todas as ações possuem um tempo de execução, um tempo de prontificação e restrições de recursos em forma de precondições que devem ser atendidas. Nesse ambiente, um algoritmo de escalonamento de ações procura para cada ação A_i mover o seu tempo de início para o mais cedo possível, de tal forma que suas precondições estejam satisfeitas. Para uma ação A_i que começa no tempo t^s_i tem-se que $R^+(t^s_i)$ é o estado de recursos no tempo t^s_i após os efeitos adicionados ao estado do jogo para todas as ações que terminam em t^s_i, sendo que $R^-(t^s_i)$ é o estado de recursos do jogo antes que os efeitos sejam adicionados. O algoritmo busca por uma combinação onde as condições prévias de A_i sejam satisfeitas por $R^+(t^S)$,e não por $R^-(t^S)$, ou seja, a satisfação das condições de A_i é devido às ações que terminam com tempo t^s. Quando isso ocorre, dentro do conjunto de ações a ação A_i é remarcada com tempo de execução inicial igual ao tempo t^s. Em seguida as demais ações do plano vão tentar ser antecipadas também. O Algoritmo 2 mostra o pseudocódigo de um algoritmo de escalonamento de ações com restrição de recursos. Vamos utilizar a palavra *makespan* para referir ao menor tempo em que um conjunto de ações pode ser escalonado.

Algoritmo 2 Escalonamento($Ações$)

1: **for** $i \leftarrow 1, ..., k$ **do**
2: $t^s \leftarrow t_i^s$
3: $R^-(t^s) \leftarrow$ estado de recursos antes dos efeitos das ações que terminam em t são adicionados.
4: **while** pré-condições de A_i são satisfeitas por $R^-(t^s)$ **do**
5: $t^s \leftarrow$ época (início e/ou término de uma ação) de decisão anterior.
6: **end while**
7: $Ações \leftarrow Ações - (A_i, t_i^s, t_i^e) + (A_i, t^s, t_i^e - t_i^s + t^s)$
8: **end for**

2.7.1 Exemplo Escalonamento em Jogos RTS

A seguir será apresentado um exemplo do processo de escalonamento no domínio de jogos RTS, utilizando o Algoritmo 2 . Nesse, será utilizado um conjunto de ações comum em jogos RTS e descrito pela na Figura 15. O tempo em que as ações são escalonadas para execução será dado em ciclos de jogo e não em segundos como os demais exemplos dessa tese. O objetivo é paralelizar ao máximo as ações que serão executadas.

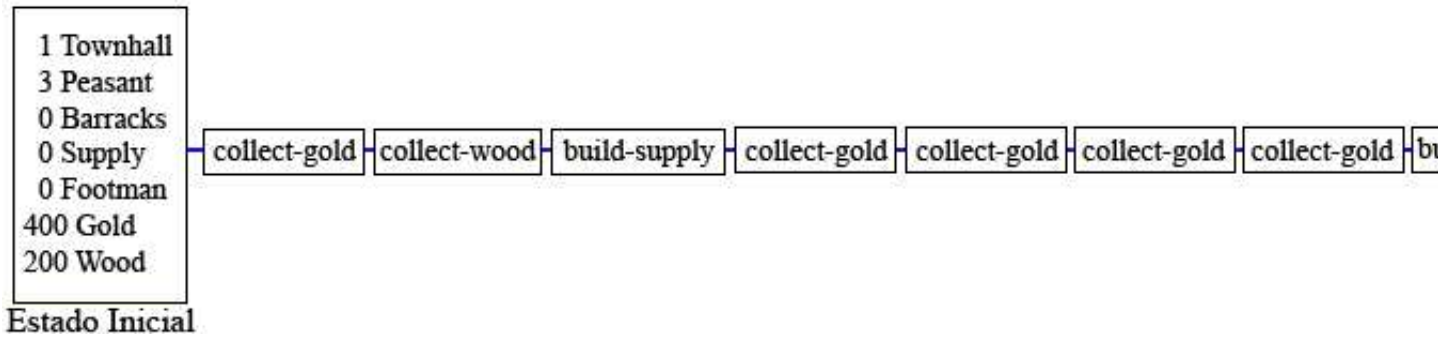

Figura 15 – Conjunto de ações em uma disposição sequencial que será escalonado.

O conjunto de ações é percorrido de forma linear em relação ao posicionamento das ações. O primeiro passo do escalonamento, exibido na Figura 16, descreve a tentativa de escalonar a ação *collect-gold* no tempo 0. Os recursos no tempo 0 (*1 Townhall*, *3 Peasant*, *0 Barracks*, *0 Supply*, *0 Footman*, *400 Gold*, *200 Wood*) satisfazem as precondições da ação *collect-gold* (**borrow***: 1 Peasant* **require***: 1 Townhall*). Não existe nenhum tempo anterior ao 0, sendo assim, *collect-gold* é mantida neste tempo.

No passo seguinte, a *collect-wood* é escolhida para ser escalonada. A ação é primeiramente posicionada no tempo mais adiante, que é igual a 510 (Figura 17). Como neste tempo a ação tem suas precondições satisfeitas, é feita a tentativa de posicioná-la no tempo anterior, igual a 0. Neste caso, a ação *collect-wood* tem suas precondições satisfeitas neste tempo também e como é o menor tempo possível ela é mantida nele. Agora temos duas ações em paralelo, que são mostradas na Figura 18).

Figura 16 – Tentativa de escalonar a ação *collect-gold* no tempo 0.

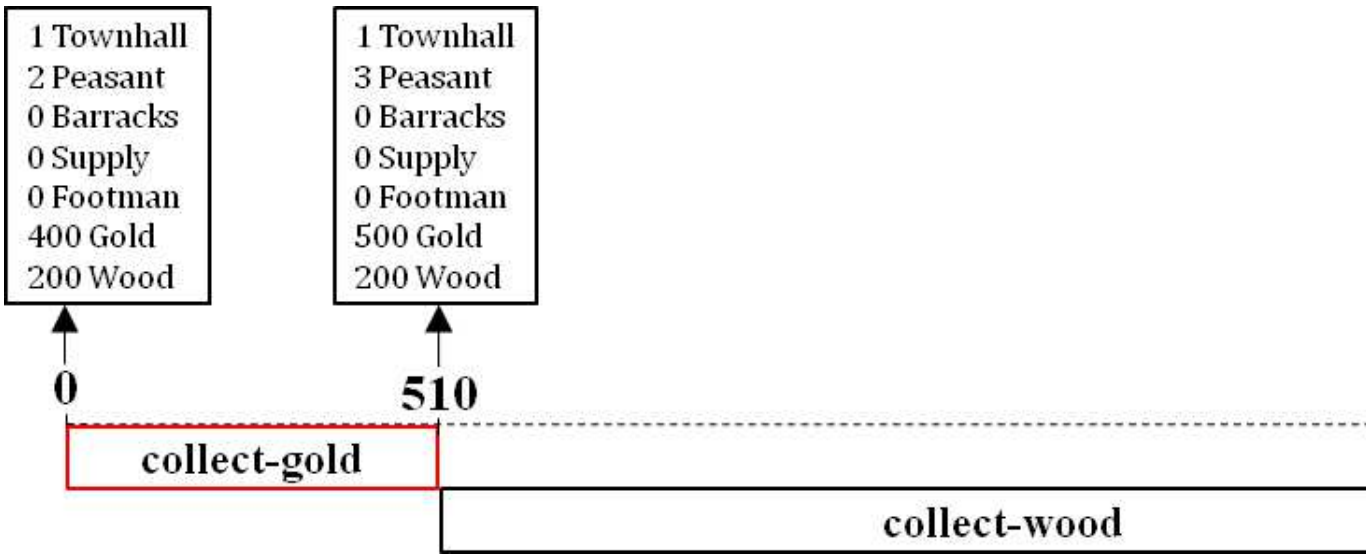

Figura 17 – Tentativa de escalonar a ação *collect-wood* no tempo 510.

Agora temos a escolha da ação *build-supply* para o escalonamento. A verificação de satisfação das precondições começa no tempo 1570 (Figura 19), que é o mais adiante no conjunto. Neste tempo ela pode ser mantida, deste modo, tentamos escaloná-la no tempo anterior 510. Entretanto, no tempo 510 não existem recursos suficientes que satisfaçam suas precondições (Figura 20). Com isso, a ação *build-supply* é escalonada no tempo 1570, que foi exatamente o último tempo em que teve suas precondições satisfeitas (Figura 21).

As demais ações irão passar pelo processo de escalonamento. O resultado final é apresentado na Figura 22. O *makespan* do conjunto sequencial e do conjunto escalonado são, respectivamente, 4915 e 2415. Isso demonstra que um conjunto pode ter melhor eficiência se for aplicado um procedimento de escalonamento.

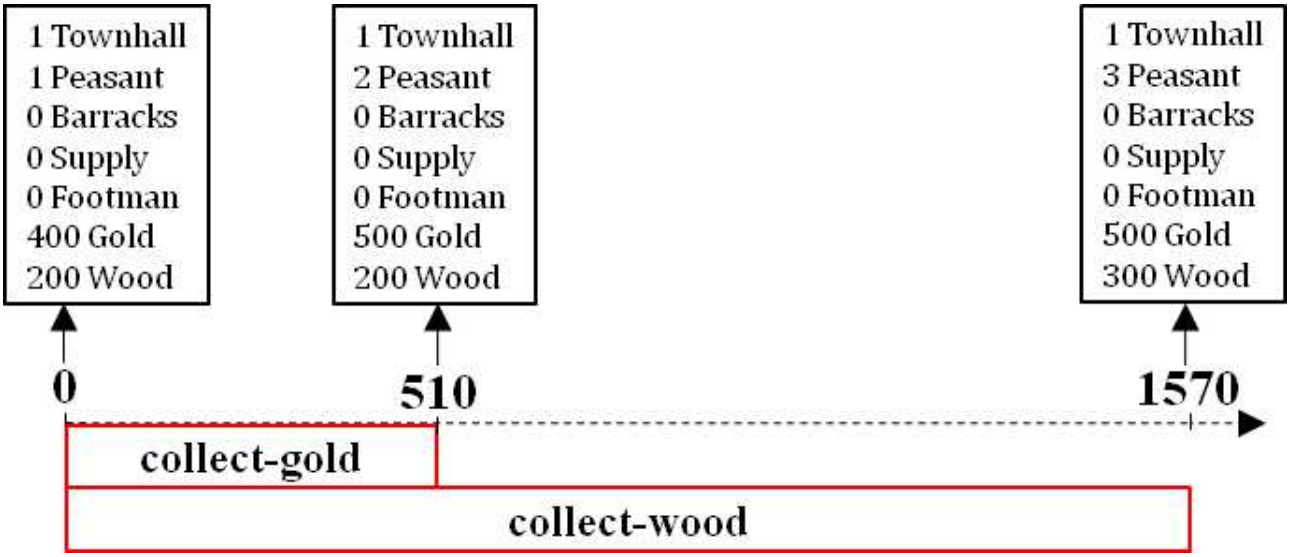

Figura 18 – A ação *collect-wood* é escalonada no tempo 0.

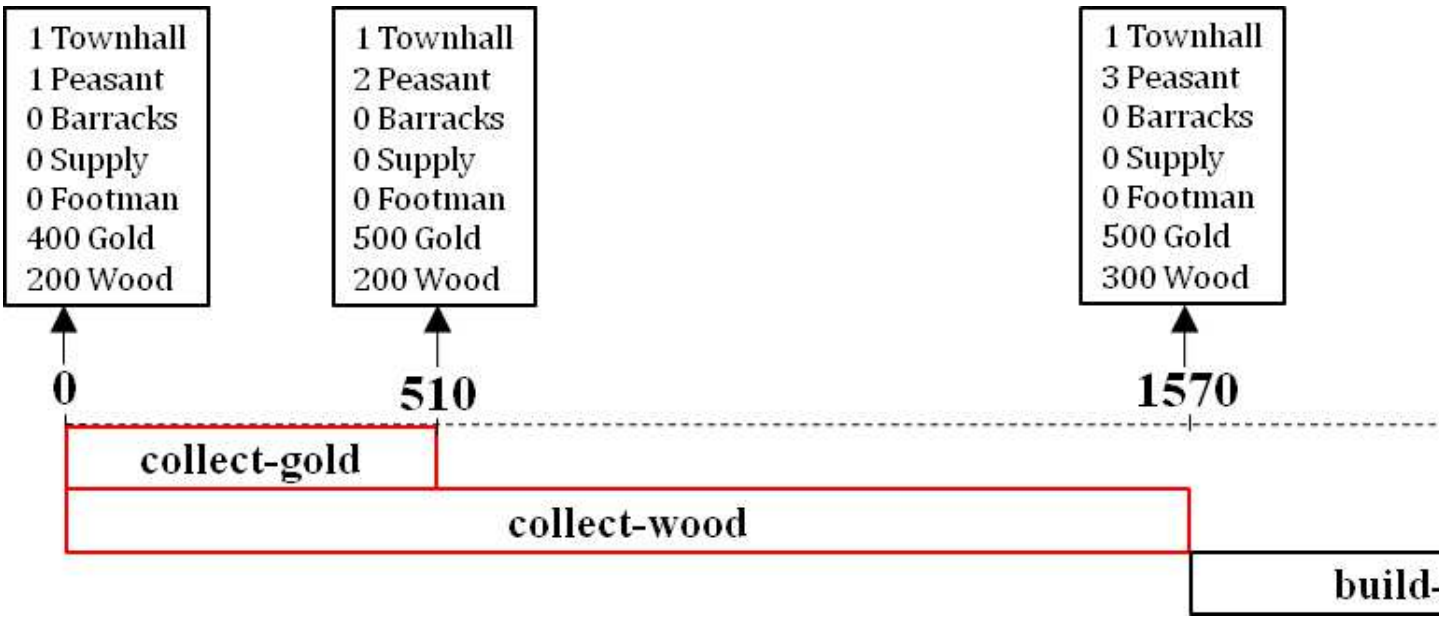

Figura 19 – Tentativa de escalonar a ação *build-supply* no tempo 1570.

Os problemas de escalonamento podem ser caracterizados em função de parâmetros tais como:

- ❏ determinismo, frente a estocasticidade (duração das tarefas fixas);
- ❏ requisitos de tempo real (restrições de tempo frente a processos *offline*);
- ❏ presença de restrições sobre os recursos;
- ❏ objetivos do escalonador (minimizar o espaço de solução, o custo, etc.);

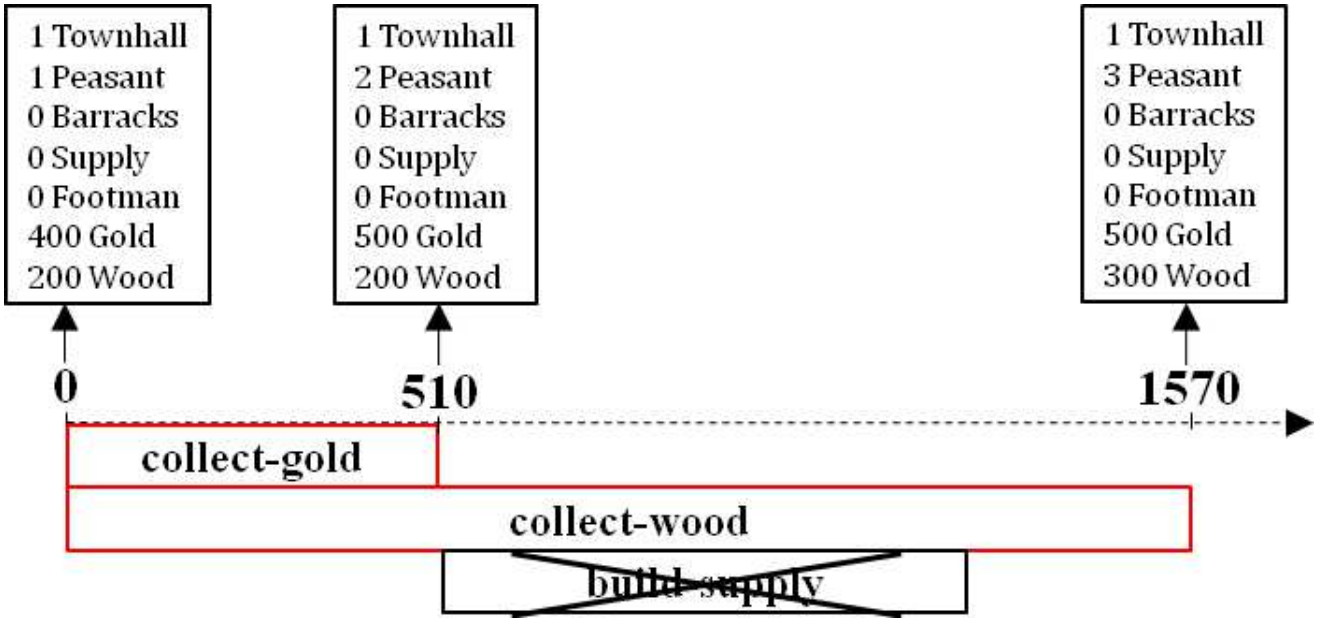

Figura 20 – Tentativa de escalonar a ação *build-supply* no tempo 510.

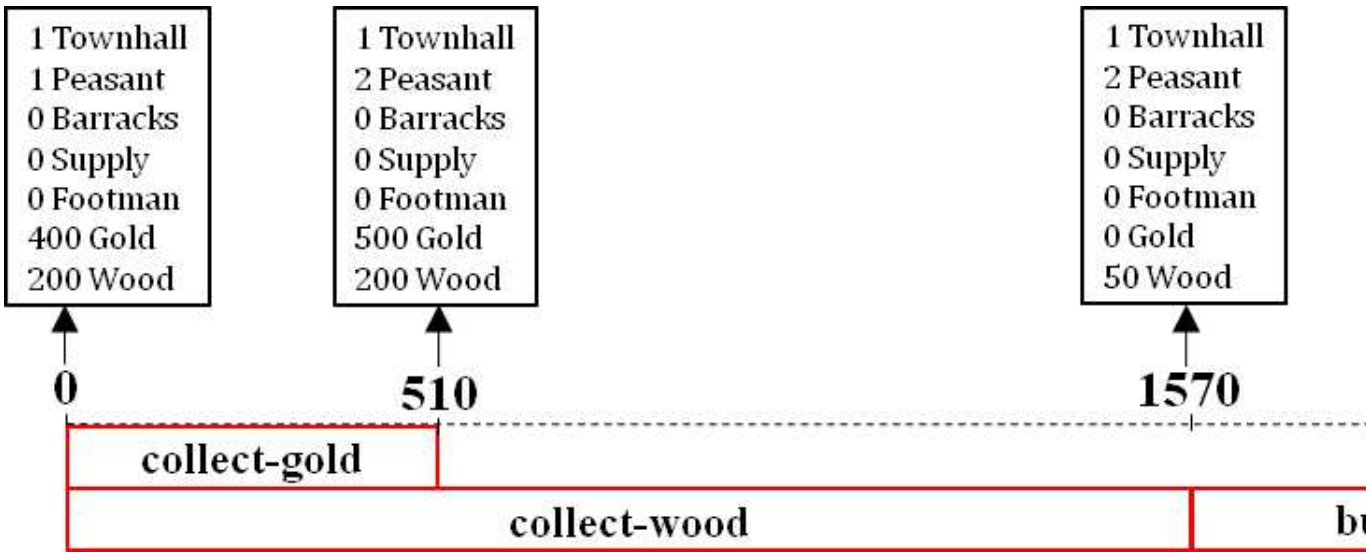

Figura 21 – A ação *build-supply* é escalonada no tempo 1570.

- ❑ requisitos de otimização.

2.8 Considerações Finais

Neste capítulo foram apresentados diversos conceitos e propriedades relacionados as técnicas que serão utilizadas para compor a arquitetura proposta. Foram enfatizados os principais conceitos envolvendo o domínio de jogos RTS, mineração de padrões sequenciais, árvore de decisão, tomada de decisão com incerteza, e escalonamento de ações com

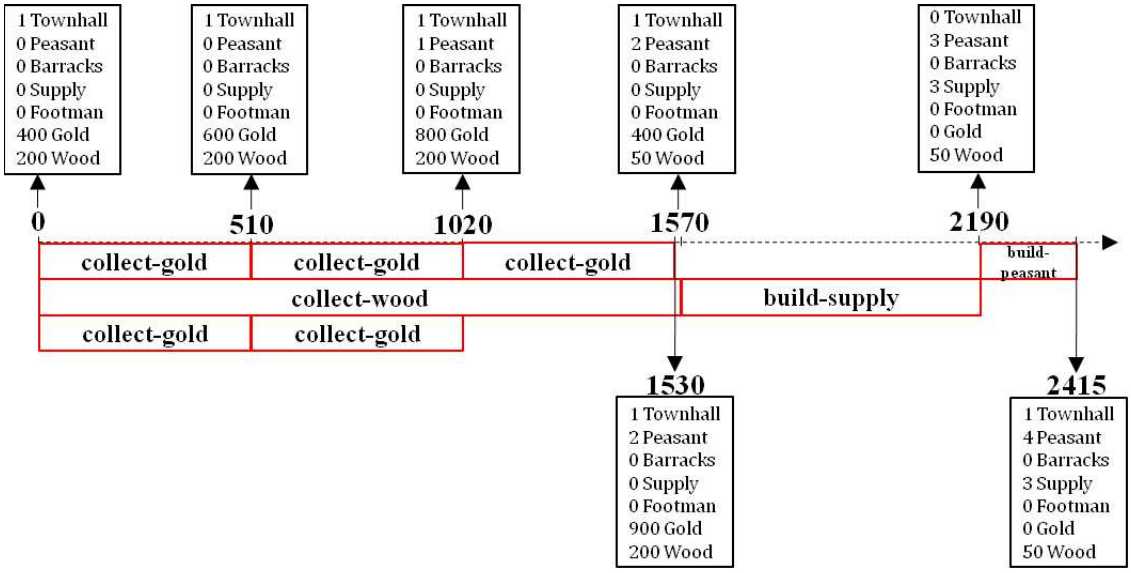

Figura 22 – Resultado do escalonamento do plano da Figura 15.

restrições de recurso. Tanto os conceitos apresentados quanto as nomenclaturas específicas abordadas são essencialmente importantes, pois serão utilizados e referenciados em diversos momentos nesta tese.

A mineração de padrões é utilizada para modelar todos os dados utilizados pela arquitetura. As técnicas presentes nessa etapa e principalmente o uso de árvore de decisão, viabilizaram o conceito de macroações, junto com a cobertura de todos os níveis de abstração do jogo. O formato desses dados ampliou a capacidade do tomador de decisões modelado com o POMDP. Esse, utiliza de conceitos atuais como versão online do algoritmo e geração de estados baseado em estruturas otimizadas. Seu funcionamento é utilizado em conjunto com o planejamento reativo, que amplia a capacidade do sistema de antecipar e gerenciar eventos inesperados. Também são propostas modificações nos algoritmos, para alcançar melhores resultados no ambiente de jogos RTS.

Nesta tese, ênfase será dada na descrição da arquitetura e assim como feito nesse capítulo as etapas serão apresentadas por ordem de execução na abordagem completa. Cada etapa é representada por um módulo, esses serão utilizados para descrever as técnicas apresentadas nessa seção e como são executadas em conjunto na arquitetura completa.

Capítulo 3

Trabalhos Correlatos

O objetivo principal deste trabalho é a criação de uma arquitetura de uso geral, para um agente capaz de gerenciar as tarefas de um jogo RTS. Para alcançar esse objetivo é preciso desenvolver e adaptar técnicas de: mineração de dados em *replays* de jogos; armazenamento e representação de macroações com predição estatística; planejamento e tomada de decisão em ambientes de tempo real com informação parcial. Neste capítulo, os trabalhos que utilizam as técnicas abordadas nesta tese, tanto em jogos RTS, como em outros domínios, serão apresentados. Trabalhos que são importantes para o entendimento da arquitetura proposta e que inspiraram sua construção em algum ponto, serão apresentados.

Este capítulo está dividido da seguinte forma: Seção 3.1 apresenta os principais trabalhos de mineração de dados com base em *replays* de partidas de jogos. Seção 3.2, discorre sobre o uso e modificação em algoritmos de árvores e busca para melhores resultados no domínio aqui investigado. Na Seção 3.3 são discutidas as abordagens que buscam usar planejamento para resolver algum desafio no domínio de jogos. Por fim, A Seção 3.4 apresenta as considerações finais sobre o capítulo.

3.1 Mineração de Replays de Partidas

O uso de dados provenientes de partidas disputadas por jogadores humanos em jogos RTS para descoberta de informações vem crescendo bastante. O objetivo é tentar descobrir padrões, jogadas e comportamentos de jogadores humanos e tentar aplicar esse a agentes jogadores. Na literatura, a mineração vem sendo utilizada principalmente no StarCraft, pois o jogo disponibiliza as informações de partidas em formato de *replays*, onde é possível ter acesso a dados como tempo de jogo, cliques do jogador e quantidades de recursos disponíveis na interface aos jogadores. Com esses dados é possível explorar informações do jogo em conjunto com técnicas de IA, para executar e gerenciar de modo racional tarefas dentro de jogos RTS. Com isso, serão apresentados nessa seção os tra-

balhos que utilizam mineração de dados no StarCraft para gerar informações, que são utilizadas para auxiliar a execução de tarefas do jogo.

(WEBER; MATEAS; JHALA, 2011) propõem o uso de mineração de dados para estratégia de predição de ações adversárias, utilizando aprendizado supervisionado em bases de dados de *replays* de partidas. São utilizados os classificadores disponíveis na ferramenta Weka (WITTEN; FRANK, 2005). Os comandos executados pelos jogadores nos *replays* são agrupados, para que possam ser identificados padrões de comandos de combate a partir da hierarquia de execução. Assim, caso um adversário comece a executar comandos que fazem parte de alguma hierarquia conhecida, sua estratégia pode ser predita e ações preventivas contra a mesma são tomadas. As ações estão dividas entre aquelas que geram dois tipos de recursos, sendo esses construções e unidades. Na hierarquia apresentada os comandos são feitos para viabilizar a geração de diferentes tipos de unidades.

Scout é uma tarefa importante jogos RTS, feita quando o jogador envia uma unidade para navegar pelo mapa, uma vez que o mesmo está coberto por uma neblina que serve para ocultar as intenções do adversário. Se executada de forma eficiente, detalhes importantes do mapa como posições de ataque, visão sobre novos acontecimentos e locais com recursos são revelados. Informações sobre a posição do inimigo e o que ele está construindo em sua base podem ser obtidas também. O trabalho de (CERTICKY, 2013) propõe a extração de informações de *replays* para criar rastreamento de unidades individuais do jogo e classificar as principais rotas utilizadas para realizar *Scout*. O objetivo é obter uma rota que seja executada sem que inimigos descubram e ataquem a unidade que está caminhando. Para a abordagem tornar-se válida são usadas técnicas de caminhamento (*pathfinding*) para execução de *scout* eficiente, uma vez que é preciso observar o que o adversário está fazendo para que os casos possam ser analisados. Além de fazer *scout*, o trabalho propõe classificação dos dados obtidos para identificar situações comuns de estados do jogo. A técnica de Raciocínio Baseado em Casos (CBR) (AAMODT; PLAZA, 1994) é utilizada, para construção de unidades de exército com base na composição de unidades do adversário. A abordagem identifica pelos dados obtidos um caso utilizando as unidades que compõe o exército do jogador e as unidades do exército adversário, em um mesmo instante de tempo, registrando também qual dos dois saiu vitorioso.

(SYNNAEVE; BESSIERE, 2011a) usam a base de dados formalizada em (WEBER; MATEAS; JHALA, 2011) com um modelo Bayeseano semi supervisionado. Esse, aprende a partir dos *replays* utilizados a realizar predição de *openings* (estratégias de início de jogo) no Starcraft. Os *openings* são rotulados por um cluster com o algoritmo de Maximização da Expectativa (EM) (MOON, 1996), que considera um conjunto de características apropriadas do jogo retiradas do conjunto de dados de *replays*. Com isso, um grupo unidades produzidas junto com as respectivas quantidades e ordem de produção indicam qual tipo de *opening* o jogador pode estar preparando, possibilitando a execução de uma contra-estratégia por parte do *bot*. (SYNNAEVE; BESSIERE, 2011b) apresentam um modelo

Bayeseano não supervisionado para predição de planos completos e não apenas de *openings*. O modelo é o mesmo discutido anteriormente, novas informações obtidas através de mineração e aprendizado são adicionados.

Em (SYNNAEVE; BESSIÈRE, 2016) o modelo Bayeseano é utilizado para identificar estratégias em diferentes níveis de abstração. Nesse modelo, o domínio de jogos RTS é simplificado e aceita qualquer sequência de ações, desde que estejam em um intervalo de tempo $t + 1$ em relação ao tempo atual do jogo. O modelo utiliza uma heursítica criada pelos autores chamada Heurística para Alvos Inteligentes (HOAI). Essa, leva em conta a intenção e disparos de fogo feito por unidades inimigas, para indicar situações onde o jogador deve fazer uma fuga, contra-ataque ou uma formação personalizada com suas próprias unidades. O modelo tenta correlacionar diferentes níveis de abstração do jogo, para identificar diferentes tipos de estratégias e oferecer indicações de contra-estratégias.

3.2 Árvores de Busca

Árvores de busca são bastante exploradas em trabalhos envolvendo jogos de todos as categorias, por proporcionarem uma estrutura eficiente para representação de estados do jogo e possibilidade de utilizar diferentes técnicas de busca por estados. Para jogos RTS, a tentativa de otimizar e adaptar algoritmos de árvores de busca é um ponto importante, uma vez que essas podem contribuir muito no cumprimento das restrições de tempo real. Também torna-se importante o estudo de árvores de busca devido a integração que essas podem ter com outras estruturas e técnicas necessárias para interagir com o ambiente do jogo. Alguns trabalhos do estado da arte buscam utilizar árvores e estender sua representação para estados mais complexos e até mesmo novas estruturas. Porém, ainda existem algumas lacunas não exploradas no uso dessa estrutura, principalmente na incorporação do conhecimento de jogadores humanos em forma de probabilidade. Essa incorporação é uma das contribuições dessa tese, além da exploração da estrutura de árvores para representação de conjuntos de macroações. Nessa seção serão discutidos os trabalhos que utilizam árvores de busca e estendem o uso dessa, trazendo novas técnicas de geração de nós e percorrimento.

(MILES; LOUIS, 2006) criam o conceito de *IMTrees*, uma árvore onde cada nó folha é um mapa de influência e cada nó intermediário é uma combinação de operações (soma, substração), que são usados para reavaliar o mapa após a simulação de ações sobre esse. Os autores usam Computação Evolucionária (CE) (RUSSELL; NORVIG, 2003) para simular operações e melhorar o aprendizado da árvore em relação ao que pode acontecer durante uma partida, naquele respectivo mapa. A figura 23, mostra um exemplo de mapa de influência representando um estado do jogo que seria um nó folha na árvore de *IMTrees*. Em vermelho estão representadas as unidades de ameaça ao agente e em azul as unidades do mesmo time; em branco o caminho que pode ser traçado para evitar ameaças. Na

Figura 24, os demais objetos presentes em um mapa e que podem ser identificados, como recursos minerais e pontos para construções de bases são representados pelo mapa de influência. Ambas as figuras 23 e 24 são retiradas do trabalho de (MILES; LOUIS, 2006).

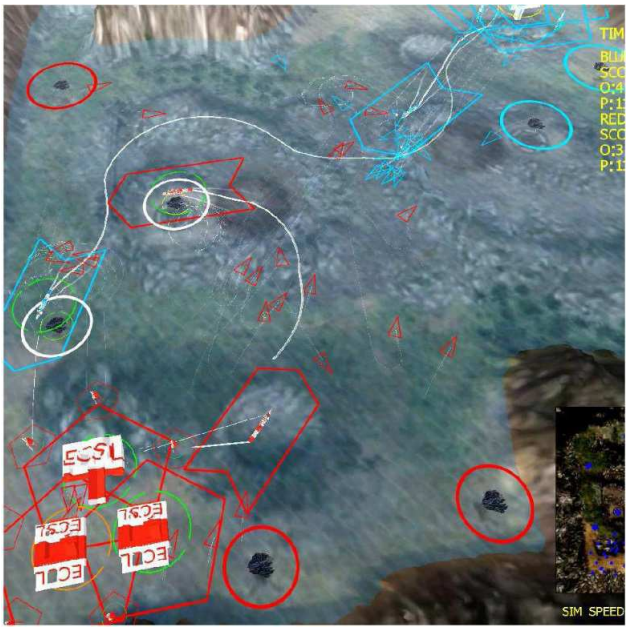

Figura 23 – Exemplo de um mapa de influência sobre o mundo do jogo em um determinado estado em que ele se encontra (MILES; LOUIS, 2006).

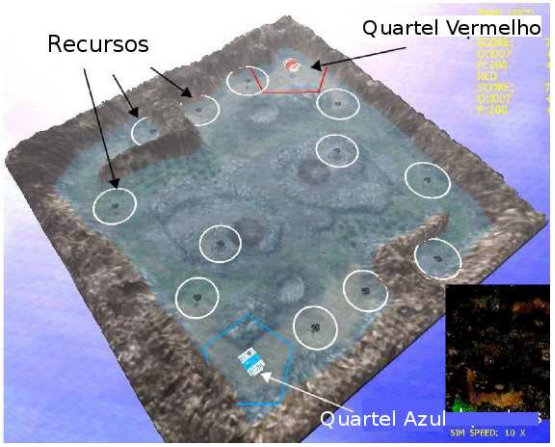

Figura 24 – Objetos presentes no mundo do jogo e que são representados pelo mapa de influência (MILES; LOUIS, 2006).

(CHUNG; BURO; SCHAEFFER, 2005) utiliza árvore de busca com lógica *fuzzy* para um modo de captura de bandeira do jogo Open RTS. O objetivo é maximar um ou um conjunto de critérios para que sejam gerados os melhores planos de ações possíveis. A árvore de busca avalia os possíveis de planos através de dados sensoriais do jogo que passam pelas etapas *fuzzy*. Os valores são retornados e alteram as probabilidades de cálculo da árvore, gerando diversas combinações de ações que possam maximizar as chances do time

conseguir capturar a bandeira do adversário. O plano que obtiver melhor probabilidade de sucesso no final da busca é utilizada no jogo. (BALLA; FERN, 2009) utilizam a versão Árvore de Busca de Monte Carlo (MCTS) (KOCSIS; SZEPESVARI, 2006), do algoritmo de busca em árvore de Monte Carlo (BREMAUD, 1999) para executar ataques no início do jogo em jogos RTS. Monte Carlo é uma técnica de análise combinatória, que utiliza busca para gerar amostras do espaço de estados reduzindo a quantidade de combinações e avaliações de resultados. A MCTS acelera a geração de novos cenários do algoritmo, dispondo o processo em formato de árvore, permitindo que técnicas de otimização e controle desse tipo de estrutura de dados sejam adicionadas. Contudo, todos esses trabalhos envolvendo algoritmos de busca necessitam de diversas simplificações e abstrações, para obterem resultados dentro do intervalo válido de jogos RTS.

(MARTHI et al., 2005) utiliza uma árvore com poda para representar a hierarquia concorrente de *Q-learning* (*Q-functions* representam movimentações de unidades individuais que podem se agrupar) (WATKINS; DAYAN, 1992) com Aprendizado por Reforço (RL) (LITTMAN; MOORE, 1996), para controle eficiente de agente em jogos de tiro. *Q-learning* é uma técnica de aprendizado por reforço, onde agentes podem encontrar a melhor política de escolha de ação em um domínio de estados finitos, modelado na forma de um modelo Markoviano. O aprendizado é incorporado através de funções, que avaliam ações e são memorizadas em forma de sequências que podem ser usadas pelo agente como estados. A árvore é expandida quando novos tabelas de *Q-functions* são adicionadas e executa a poda naqueles que já estão na estrutura e com valor de ganho menor que os novos que chegaram. Em (MARTHI et al., 2005) os movimentos são primeiramente retirados de observações e base de dados de outros agentes do mesmo tipo. O processo de movimentação é aprendido através de análise de movimentos prévios realizados em testes anteriores. As combinações de movimentações são representadas por *Q-functions*, que conseguem aprender melhores combinações a medida que o agente utiliza movimentos agrupados ou individuais escolhidos através do *Q-learning*.

(CHURCHILL; SAFFIDINE; BURO, 2012) explora o uso de técnicas de árvore de busca em jogos RTS. Nesse trabalho, é apresentado um algoritmo *Alpha-Beta* Considerando Durações (ABCD), uma árvore de busca para determinar estados de ordem tática (batalha) em jogos RTS. A árvore é utilizada para simular situações de combate, onde um nível possui nós que representam ações táticas do adversário e o nível logo abaixo ações do agente. Essa hierarquia, é repetida formando uma árvore com diversos níveis de ações do adversário e do agente. Tentar maximizar o ganho e diminuir a perda é o objetivo para determinar o melhor cenário de combate tático em um determinado momento. As ações que compõem a árvore são retiradas de banco de dados de sites especializados em StarCraft. O algoritmo busca podar não só os nós que representam ações sem ganhos, mas também aqueles que o jogador humano iniciou mas acabou descartando ou não finalizando durante a partida.

3.3 Planejamento e tomada de decisão

O uso de planejamento em jogos RTS pode ser estendido a diversas tarefas do jogo como: escolha de quais recursos devem ser produzidos, coordenação de ataque, caminhamento de unidades, entre outros. No geral, o objetivo é tomar decisões em todas as tarefas utilizando de algum tipo de planejamento a partir de informações do ambiente. Assim, as tarefas serão executadas com foco em obter o melhor resultado possível, utilizam de IA para tomada de decisão. O termo planejamento, será utilizado nessa seção, remetendo a qualquer tomada de decisão feita utilizando informações do jogo ou através do gerenciamento dessas. A maior parte dos trabalhos encontrados no estado da arte utilizam planejamento em apenas uma tarefa específica do jogo. São raros aqueles trabalhos que propõem utilizar planejamento ou mesmo IA na coordenação de todos os requisitos do jogo. Nesta seção vamos além de jogos RTS e apresentamos técnicas de planejamento utilizadas em outros estilos de jogos, uma vez que foi necessário olhar para outros ambientes para obter mais arquiteturas e algoritmos.

Abordagens baseadas em regras simples e scripts tem sido muito usadas no decorrer dos anos. A forma mais comum de modelagem dessas é através de MEF. A ideia é decompor a tomada de decisão em estados que representem ações essenciais dentro do jogo, como "Atacando" ou "Reunindo Recursos". Essas regras geralmente são simples transições e permitem mudanças rápidas entre os estados. Abordagens comerciais utilizavam MEF hierárquicas, onde cada estado de um MEF possui uma hierarquia na execução das ações (JAIDEE; MUNOZ, 2012). De fato, essas abordagens conseguiram sucesso em diversos jogos, mas por não adaptarem-se a comportamentos de diferentes adversários e não conseguirem dinâmica nas decisões estratégicas, são facilmente superadas com modelagem do adversário e predição de ações. Essas tarefas são desempenhadas por jogadores humanos e bots presentes nos trabalhos mais recentes da área.

Abordagens com planejamento tem sido pouco exploradas, principalmente pelo tamanho do espaço de estados e as restrições de tempo real de jogos RTS. Um dos primeiros trabalhos a considerar planejamento foi o de (ONTANON et al., 2008), que explora o uso da técnica Planejamento em Tempo Real Baseado em Casos (CBP) no domínio do jogo Wargus (clone do jogo Warcraft II). Nesse trabalho, eles fazem observações a partir de demonstrações humanas, onde dois ou mais jogadores disputam uma partida e dados da mesma são retidos em forma de anotações. Essas anotações são usadas com a técnica CBR para aprender sobre conjuntos de ações e os casos em que elas podem ser usados. O CBP utilizado é uma variante do CBR.

Em (ONTANON et al., 2008) os planos são decompostos durante o jogo pelo CBP e suas ações, que são armazenadas juntos com eles em forma de casos, são executadas dentro do jogo. A figura 25 foi retirada do trabalho de (ONTANON et al., 2008) e ilustra como é feita a extração de casos a partir das anotações. Cada caso corresponde a um conjunto de ações. A Figura 26 apresenta o ciclo de armazenamento de casos no CBR. Em

(MISHRA; ONTANON; RAM, 2008), eles ampliam o trabalho melhorando o modelo de CBP com o uso de reconhecimento de situações, que permite que a escolha dos conjuntos possa ser feita através da predição de casos futuros. Isso aumenta a velocidade da escolha dos conjuntos e consequentemente a quantidade de casos que são analisados.

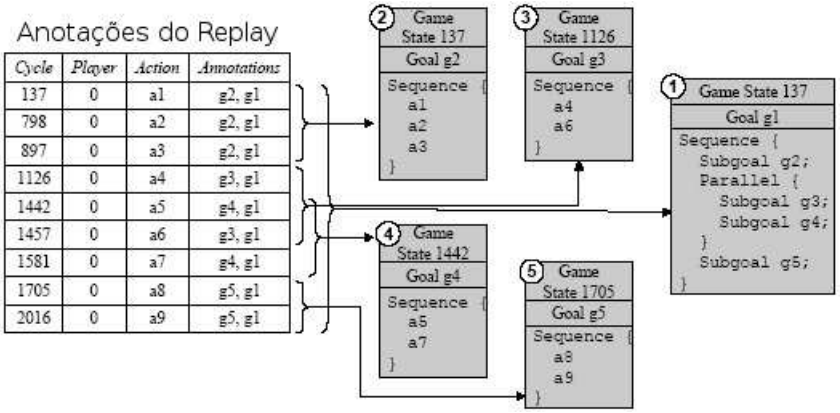

Figura 25 – Extração de casos base a partir da análise de conjuntos de ações que foram realizadas em uma partida (ONTANON et al., 2008).

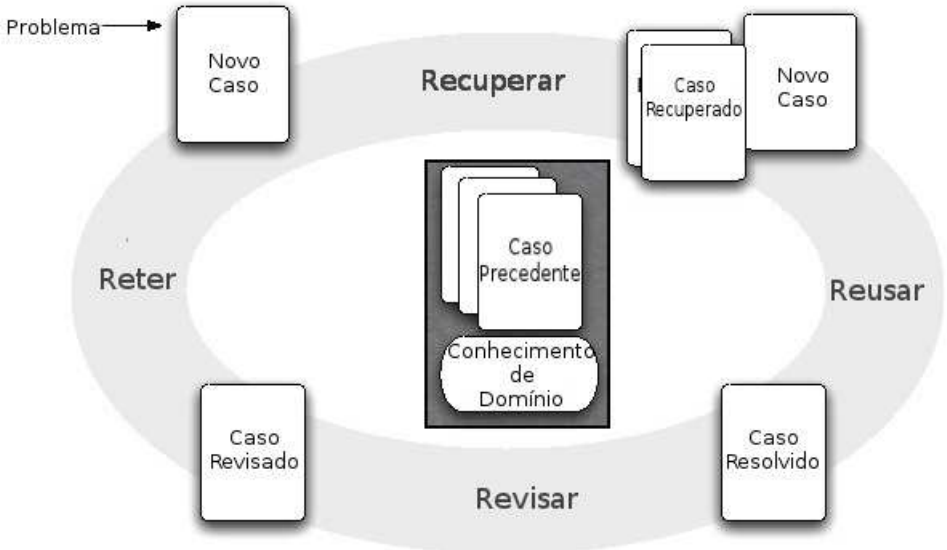

Figura 26 – Ciclo de armazenamento de novos casos na base de dados do CBR. Fonte: (CERTICKY, 2013).

Redes de Tarefas Hierárquicas (HTN) foram exploradas em jogos de tiro em primeira pessoa. Em (HOANG; LEE-URBAN; MUNOZ-AVILA, 2005), os autores afirmam que a adaptação desta abordagem para jogos RTS é possível. HTN é uma técnica de planejamento onde uma rede representa um conjunto de ações através de seus nós e as arestas modelam as restrições entre as ações. As ações são escolhidas para satisfazer tarefas em um determinado ambiente. Contudo para a execução de uma ação é preciso saber se as restrições estabelecidas para essa já foram satisfeitas. Uma HTN coordena a execução

de uma ação garantindo que suas restrições sejam satisfeitas também (KUTLUHAN; HENDLER; NAU, 2003). Redes de tarefas são montadas de acordo com a hierarquia de controle que é observada em jogadores humanos, sendo que seu comportamento de escolha de ações é distribuído pela rede e controla as decisões do agente dentro do ambiente do jogo.

(CHURCHIL; BURO, 2011) é um dos poucos trabalhos que apresentam bons resultados entre as abordagens que usam planejamento em um agente parcial para StarCraft. O agente é limitado a executar apenas algumas tarefas do jogo utilizando algoritmos de IA, as demais tarefas são feitas a partir de instruções predefinidas. Neste, é utilizado um construtor de recursos que determina as ações necessárias para construir as unidades presentes em grupos de recursos. Esses grupos são obtidas a partir da análise de *replays* de jogadores profissionais. Os grupos de recursos são consideradas de produção, uma vez que são descritas apenas por unidades e construções que devem ser desenvolvidas no jogo, não incluindo comandos ou movimentos para as unidades. O construtor consegue especificar e executar as ações necessárias para construir um determinado grupo dentro de um tempo limite inferior a 1 segundo. O construtor executa uma busca em profundidade primeira junto com uma heurística para poda de nós que representam ações que não auxiliam no alcance dos recursos do grupo.

Dentre os trabalhos que utilizam aprendizado de máquina para planejamento e tomada de decisão, as abordagens utilizando CBR são predominantes neste campo. O trabalho de (AHA; MATTHEW; PONSEN, 2005) foi um dos pioneiros, onde o CBR foi usado para executar recuperação dinâmica de planos de ações dentro do jogo de RTS Wargus. Cada caso construído com a técnica CBR representa um plano de ações retido a partir de *logs* de partidas. O trabalho de (HSIEH; SUN, 2008) é baseado no de (AHA; MATTHEW; PONSEN, 2005), onde o modelo CBR foi utilizado no jogo StarCraft para recuperar além de plano de ações, quais unidades do jogador foram utilizadas e se houve agrupamentos de construções. Já (JAIDEE; MUNOZ; AHA, 2011) propõe o uso de CBR para a escolha automática de metas em um jogo RTS. Nesse trabalho, o modelo CBR armazena ações em forma de casos e o algoritmo de Autonomia Guiada por Objetivos (GDA) é usado para escolha automática dessas durante o jogo. As ações são armazenas com *Q-tables* (LIU; LI, 2008) e cada uma possui uma pontuação que mede o quanto ela foi adequadamente usada. RL é usado para reavaliar as metas no momento em que são selecionadas. Caso a escolha de uma ação tenha sido positiva e o estado do jogo tornou-se melhor para o jogador, a mesma é melhor avaliada, caso contrário sua nota diminui.

(CUNHA; MACHADO; CHAIMOWICZ, 2015) apresentam o RTSMate, um sistema que auxilia jogadores indicando ações de nível estratégico e tático em jogos RTS. Uma DT é construída usando informações extraídas de guias e materiais, os estados do jogo são avaliados seguindo métricas pré-especificadas. A DT é percorrida com os estados captados e retorna ações em forma de dicas ao jogador. Em (TAVARES et al., 2016) a escolha de

uma estratégia adequada em um jogo RTS é feita através da análise da estratégia de diversos *bots*. A análise envolve o uso de técnicas de teoria dos jogos como Equilíbrio de Nash (ALMEIDA; KAJIN; VIEIRA, 2012), para avaliar a qualidade das estratégias em forma de um portfólio para escolha do *bot* mais adequado em relação as condições do jogo e dos inimigos. (DERESZYNSKI et al., 2011) utiliza Modelos Ocultos de Markov (HMM) (RABINER; JUANG, 1986) para calcular probabilidades de sequências entre construções e as unidades que são produzidas por essas. Com a restrição de visibilidade, uma rede com ações e probabilidades entre si utiliza uma distribuição de probabilidade de uma ação estar ligada a outra no modelo. A cada execução de uma ação no modelo as distribuições são ajustadas. Em (DERESZYNSKI et al., 2011) as construções mais prováveis são mantidas para criação de modelos probabilísticos de comportamento de jogadores no StarCraft. Os HMM são construídos de modo que os nós são ações ou construções do jogador e as arestas as probabilidades de que uma determinada sequência de ações seja tomada. As sequências mais usadas tornam-se clusters e indicam que algum tipo de unidade específica de ataque ou defesa será desenvolvida.

A Figura 27, mostra um diagrama de transição de estados que representa um modelo de predição do trabalho de (DERESZYNSKI et al., 2011). As arestas sólidas são probabilidades maiores que 0.25 em uma escala de até 1. As arestas pontilhadas com probabilidades de 0.5 a 0.15 demonstram que todo nó é acessível no modelo. Os conjuntos de nós cercados por quadrados são sequências de ações que indicam a criação de uma unidade específica, com o seu respectivo nome em cada quadrado. O HMM utilizado calcula todas as probabilidades com base de dados de partidas já disputadas e não utiliza nenhum tipo de aprendizado levando em conta o que ocorre durante uma partida. Para uma atualização no modelo, é preciso que uma nova base a partir das últimas partidas disputadas por essa seja criada e submetida novamente ao algoritmo. Para que os resultados tornem-se melhores os autores mencionam que vão trabalhar futuramente com técnicas que possam aumentar a quantidade de informação do jogo, uma vez que em uma partida o jogador consegue observar apenas a sua base, o restante do mapa está sob *fog-of-war*, uma neblina que cobre a visão dos jogadores e só desaparece quando o jogador envia unidades para desvendar essas áreas. Assim, o HMM de (DERESZYNSKI et al., 2011) trabalha considerando informação completa do jogo, mesmo quando essa não está disponível o que pode alterar a qualidade dos resultados e das escolhas feitas pela técnica.

Recentemente, os trabalhos estão voltando os esforços de planejamento e tomada de decisão em jogos RTS para o uso de algoritmos de Amostragem e Árvore de Busca em Jogos (TGS) (ANAGNOSTOPOULOS; BRODER; CARMEL, 2005). (URIARTE; ONTAÑÓN, 2014) propõe uma representação dos estados do jogo utilizando abstração de ações, onde as mesmas são descritas por objetivos no jogo. MCTS é utilizada para reduzir o espaço de busca por objetivos no jogo. Em (ONTAÑÓN, 2017), MCTS é utilizada em conjunto com o algoritmo de amostragem *naïve sampling*, para reduzir o espaço de

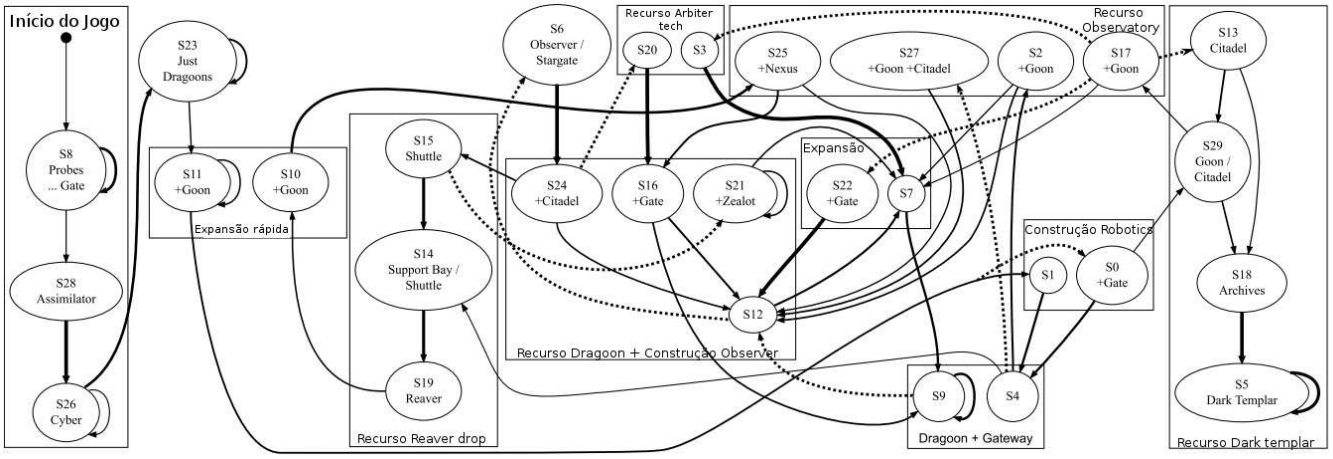

Figura 27 – Modelo de predição. Probabilidades são calculadas através da análise de partidas já disputadas e unidades específicas são descritas pelas ações necessárias para sua criação (DERESZYNSKI et al., 2011).

estados e permitir escalabilidade em outras tarefas do jogo. (BARRIGA; STANESCU; BURO, 2017) utiliza a versão TGS clássica para criar o algoritmo *puppet search*, que simula diferentes cenários e determina qual estratégia deve ser utilizada mesmo com grande quantidade de recursos e unidades. O algoritmo utiliza TGS para simular os combates e determinar a probabilidade de vitória. (URIARTE; ONTAÑÓN, 2016) e (ONTAÑÓN, 2016) combinam o uso de MCTS com algoritmos de aprendizado. No primeiro trabalho, os dados de *replays* são incorporados em um modelo bayesiano ingênuo, que guia busca na árvore e auxilia na redução do espaço de estados para tomada de decisão. No segundo trabalho, o uso do modelo bayesiano é melhorado incorporando probabilidades de distribuição de jogadores ao executar ações. O modelo de MCTS é incoporado ao bayesiano criando o *NaiveMCTS*, que é utilizado para escolha de ações no jogo, balanceado a escolha de estratégias ofensivas e defensivas de acordo com as probabilidades de distriuição incorporadas pelo modelo com informações de *replays*.

3.4 Considerações Finais

Neste capítulo foram discutidos os principais trabalhos que envolvam técnicas ou abordagens que são utilizadas nessa tese. Entre essas, mineração de *replays*, planejamento e árvores de busca. Em geral, os trabalhos da área utilizam as técnicas mencionadas com foco em gerenciar uma etapa ou desafio específico do jogo. Alguns trabalhos se destacam por estender o uso de uma técnica em particular para aumentar a sua representatividade em relação aos componentes do jogo. Contudo, poucas referências buscam criar uma abordagem com a integração de diversas técnicas.

A mineração de dados é uma das técnicas mais exploradas pelos trabalhos estudados,

sendo utilizada desde as primeiras publicações da área envolvendo StarCraft como domínio de testes. Nesta tese, um dos objetivos é encontrar ainda mais padrões do jogo a partir dos *replays*, ampliando a representação do jogo através de macroações e diferentes níveis de abstração. Técnicas de planejamento estão começando a ganhar mais destaque e serem exploradas no domínio de Jogos RTS. O uso de busca com algoritmos de aprendizado tem recebido muita ênfase atualmente. Mas, o uso de raciocínio para tomada de decisão e planejamento seja na integração de técnicas ou no uso de uma arquitetura, ainda têm sido pouco explorados.

O objetivo central desta tese é que o uso de planejamento através da integração de diversas técnicas pode ser uma arquitetura possível de ser construída. Essa sendo capaz de gerenciar todos as tarefas presentes em uma partida de jogos RTS. Não encontramos trabalhos que propõem o uso de raciocínio no controle total do jogo. Os poucos trabalhos que buscam controlar tarefas do jogo, ainda combinam uso de IA com códigos e estruturas predefinidas.

Capítulo 4

Arquitetura de Uso Geral Baseada em Planejamento Probabilístico para um Agente Completo

Neste capítulo, a arquitetura de uso geral baseada em planejamento probabilístico será descrita em relação a seu funcionamento geral. Os métodos utilizados para o desenvolvimento da arquitetura, a ordem de funcionamento e os algoritmos serão discutidos. Além destes, os termos específicos e importantes serão enfatizados para o entendimento dos próximos capítulos. A Seção 1.2 descreve brevemente os três módulos principais da arquitetura, juntamente com a Figura 1.2. Em resumo, a arquitetura é capaz de controlar um agente jogador cumprindo todas as tarefas de um jogo RTS. Com a característica de uso geral, a arquitetura proposta utiliza base de dados de um jogo RTS para obter informações específicas do ambiente. A partir delas, a arquitetura toma decisões e gerencia a execução de todas as tarefas diretamente no ambiente do jogo. As principais técnicas utilizadas na construção dessa arquitetura e que serão descritas neste e nos próximos capítulos são:

- Mineração de dados para modelagem dos principais dados do jogo. Foram utilizadas ferramentas para extrair dados de *replays* com partidas entre jogadores experientes. Esses dados foram catalogados e analisados para classificação com os algoritmos de mineração sequencial GSP e PrefixSpan. O objetivo foi classificar macroações em todos os níveis de abstração do jogo.

- Árvore de decisão com predição para execução de várias macroações. As macroações classificadas foram modeladas em uma árvore CHAID, a qual possui capacidade preditiva e consegue escolher macroações que são executadas em paralelo com a macroação principal, complementando sua execução. Essas macroações serão referidas nesta tese como macroações complementares.

- ❏ Modelagem de um POMDP *online* para Jogos RTS. Foi proposta uma modelagem e estruturação de processos para o funcionamento do POMDP no domínio de jogos RTS. São propostas duas políticas de execução. A primeira é o AEMS, que se baseia no uso de heurísticas para geração de estados de crença. A segunda é o RTSSample, que utiliza o conceito de amostragem de cenários e gerenciamento de mudanças no modelo do POMDP.

- ❏ Planejamento reativo para uso dinâmico de macroações. A arquitetura possui a capacidade de cancelar a execução de um planejamento, desconsiderando as macroações em execução e considerando a escolha de novas. A dinâmica de cancelamento ocorre em função de mudanças significativas no ambiente do jogo durante uma partida. Foram utilizadas as abordagens árvore de comportamentos e linguagem reativa ABL.

- ❏ Escalonamento de macroações em diferentes níveis de abstração. O escalonamento é feito para paralelizar a execução das ações internas de uma macroação. A execução de uma macroação dentro da partida utiliza menos tempo para completar suas ações internas, assim mais macroações podem ser consideradas e executadas na partida. Foi utilizado o escalonador de ações com estratégia de janela de intervalos (NAVES, 2012), o qual foi adaptado para operar com macroações que executam ações em diferentes níveis de abstração.

A arquitetura de uso geral baseada em planejamento probabilístico é sumarizada na Figura 28. A divisão baseada em módulos feita na Seção 1.2 é mantida na figura. Em cada módulo, estão os algoritmos e estruturas de controle utilizados. As setas na figura indicam o tipo de processamento de dado que é feito entre os algoritmos internos. As setas vermelhas que conectam os módulos mostram a ordem em que acontece o processo de troca dados entre eles. Esse processo é feito sempre que é preciso escolher macroações para execução no jogo.

De acordo com a Figura 1.2, os módulos existentes são $Classificação$, $Decisão$ e $Controle$. A arquitetura tem como objetivo controlar um agente jogador, gerenciando todas as tarefas durante uma partida de jogo RTS. Nesta tese, o jogo utilizado como ambiente de testes é o StarCraft. Todas as tarefas do jogo são gerenciadas via execução de ações, assim, a principal operação da arquitetura é decidir quais conjuntos de ações serão executados a cada instante. A arquitetura tem as macroações como o principal objeto de dados utilizado. Elas abstraem a execução de qualquer tarefa do jogo em conseguir encontrar e executar um conjunto de ações adequadas. Esses conjuntos de ações adequadas são as macroações, e cada uma executa algum tipo de tarefa dentro de seu nível de abstração. Em determinado momento da partida em que não foi detectada a presença de nenhum inimigo, a arquitetura poderá verificar as macroações disponíveis e calcular as probabilidades de executar uma que produza mais recursos ou ataque o inimigo. Esse comportamento surge justamente pela observação da ausência de inimigos na partida

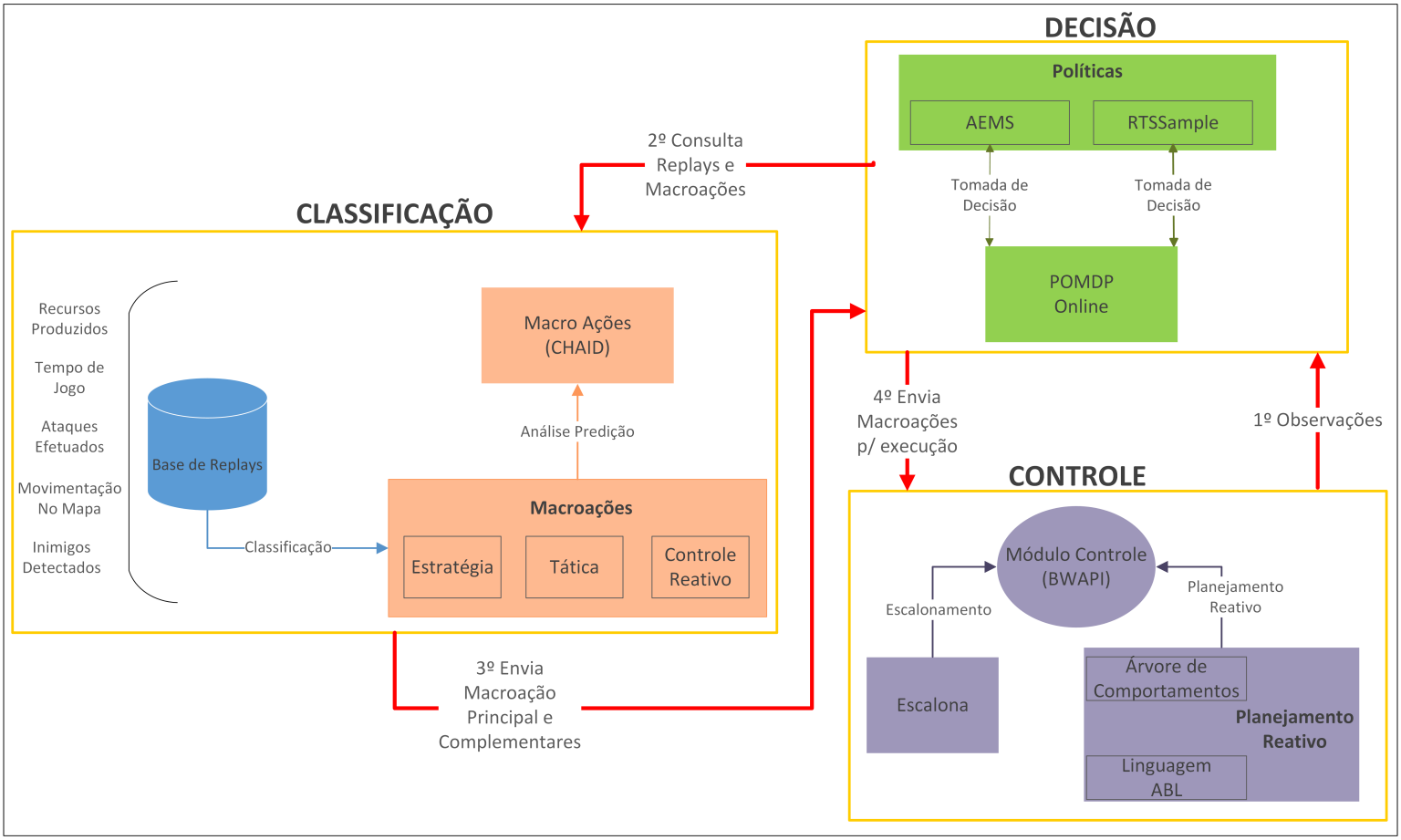

Figura 28 – Visão geral da arquitetura baseada em planejamento probabilístico a partir dos módulos e algoritmos. As setas em vermelho indicam a ordem em que os módulos trocam dados para gerenciar a arquitetura

.

até o momento. Para a tarefa de produzir recursos, uma macroação de *Estratégia* será utilizada para definir quais deles serão produzidos. Outra macroação de *Tática* será escolhida para complementar a produção de recursos, nesse caso, executando ações que posicionarão as unidades produzidas em locais estratégicos do mapa. Assim, a forma de operação da arquitetura é sumarizada em executar as ações corretas para cada tarefa do jogo. Esse modo de operação concede à arquitetura a capacidade de uso geral, sendo reutilizada em qualquer jogo RTS.

A arquitetura opera em uma repetição contínua que inicia junto com a partida e finaliza quando ela é encerrada. De acordo com as setas de fluxo de dados da Figura 1.2, para executar alguma tarefa dentro do jogo por meio da escolha de uma macroação, é preciso que o fluxo 1º*Observações* inicie o processo. As observações são obtidas direto da partida pelo módulo *Controle* e repassadas ao módulo *Decisão*. Esse último, para escolher uma macroação, consulta aquelas disponíveis no módulo $Classificação$ através do fluxo 2º$ConsultaMacroAções/Replays$. Nessa consulta, a macroação principal será escolhida

pelo módulo de *Decisão*, mas o módulo de *Classificação* utiliza essa escolha para decidir sobre as macroações complementares. As macroações complementares serão executadas junto com a principal, através do fluxo 3º*EnviaMacroAçõesComplementares*. Por fim, as macroações são enviadas de volta ao módulo de *Controle*, que processará as ações internas das macroações e as executará no jogo. Todo o processo de escolha de macroações descrito é iniciado novamente, sempre que novas observações são obtidas. Dentro da arquitetura proposta, não existem outras disposições de troca de dados ou ordem entre os módulos, apenas essa aqui descrita.

No módulo de *Classificação*, está contida a base de *replays* de partidas entre jogadores profissionais. Essa base é utilizada para que os algoritmos de classificação possam encontrar as macroações, as quais representam um conceito proposto nesta tese e que possibilita obter conjuntos de ações em todos os níveis de abstração do jogo. Com os *replays* classificados, as macroações dos níveis de abstração *Estratégia*, *Tática* e *ControleReativo* são obtidas. Elas são dispostas sob uma árvore com capacidade de modelagem preditiva. A predição é utilizada para que macroações complementares sejam escolhidas para execução junto com a macroação principal. Além da predição, a árvore mantém as macroações catalogadas para que o POMDP possa consultá-las, obtendo informação de quais estão disponíveis para ser utilizadas na tomada de decisão. O módulo de *Classificação* exerce papel importante dentro da arquitetura. Sua primeira contribuição é feita de forma *offline*, com a execução dos algoritmos de mineração sequencial e árvore preditiva. Dentro da operação *online* da arquitetura, a *Classificação* armazena todas as macroações e todos os dados do jogo obtidos com análise dos *replays*. Além disso, a *Classificação* decide sobre quais macroações de *Tática* e *ControleReativo* devem complementar a execução da macroação principal de *Estratégia*. O módulo atua tanto organizando os dados como escolhendo macroações para execução no jogo.

O módulo de *Decisão* para executar a tomada de decisão utiliza, junto com o POMDP *online*, as duas políticas propostas neste trabalho. Esse módulo concentra a maior parte do processamento da arquitetura, pois recebe as observações do jogo, consulta as macroações e retorna a escolhida para execução no jogo. Os algoritmos do módulo de *Decisão* foram construídos com diferentes estratégias de modelagem interna, uma vez que as restrições de tempo real de jogos RTS são complexas. Além disso, os algoritmos foram adaptados para operar com os conceitos da arquitetura, como as macroações. O módulo foi adaptado para responder a mudanças no modelo do jogo que ocorrem diversas vezes durante uma partida. Essas mudanças decorrem de eventos que mudam o comportamento da partida, como detecção de inimigos ou ataques eminentes. Essa mudança no modelo é uma característica específica de jogos RTS, em que o próprio modelo do POMDP também é afetado, sendo gerenciado nesse caso por uma das políticas de execução propostas.

O módulo de *Controle* é o que interage diretamente com o ambiente do jogo. Na Figura 1.2, o módulo possui um processo em formato circular, que representa a API uti-

lizada para receber as informações do ambiente e enviar os comandos para execução das macroações escolhidas. O módulo é responsável por escalonar as ações que compõem as macroações, para que elas sejam executadas no menor tempo de jogo possível dentro da partida. O planejamento reativo é capaz de gerenciar a execução das ações internas das macroações e utilizar as observações advindas da partida. Com as observações, o planejamento reativo decide se as macroações em execução devem ser interrompidas devido a novos acontecimentos na partida. Interromper uma macroação significa cancelar as ações ainda não executadas, gerenciar os recursos produzidos e escolher novas macroações para execução. Esse paradigma reativo rompe com o funcionamento tradicional de planejamento, adaptando o gerenciamento da arquitetura a situações que ocorrem naturalmente nas partidas e são gerenciadas por jogadores humanos. Nessas situações, os jogadores devem verificar se é necessário abrir mão de uma estratégia atual para considerar outras em efeito imediato.

No funcionamento geral da arquitetura e seus módulos, a macroação é um tipo de dado utilizado como objeto principal de decisão e comunicação entre os módulos. Macroação é um conceito utilizado nesta tese para definir um conjunto de ações que um jogador executa na partida. Essas ações produzem uma quantidade específica de recursos ou executam comandos sobre recursos já criados. Em ambos os casos, a macroação contém ações internas que, quando executadas, cumprem alguma tarefa no jogo. Na comunidade de jogadores de StarCraft, existe um conceito semelhante a macroação. Um conjunto de ações específicas que pode levar um jogador a vencer uma partida de forma rápida é chamado de *build order*. Os *build orders* são difundidos entre os jogadores, assemelhando-se a estratégias de abertura em jogos como Xadrez. Essas estratégias focam em momentos iniciais da partida e com execução rápida, utilizando um ou dois minutos do tempo de jogo. A Tabela 1 mostra um *build order* da classe *Terran* difundido pela comunidade de jogadores, o qual produz unidades de ataque do recurso $Vulture$ e unidades de defesa dos recursos $IonThrusters$ e $SpiderMines$. Os valores numéricos ao lado de cada recurso representam a relação de suprimentos gastos pela quantidade total disponível, para que novos suprimentos sejam produzidos. O indicador @ mostra quando deve ser feito um *upgrade* em alguma edificação, ou qual recurso deve ser gerado ininterruptamente durante a partida.

Diferentemente dos *build orders*, a macroação é um conjunto de ações em intervalos de execução que variam de poucos segundos (40 segundos) até muitos minutos (9 minutos). Além disso, as macroações são identificadas nos três níveis de abstração do jogo, fornecendo conjuntos de ações para a arquitetura que executa diferentes tarefas e com diferentes finalidades dentro do jogo. Assim, as decisões tomadas em uma partida são feitas utilizando somente conjuntos de ações identificados a partir de padrões de jogadores humanos. Não existe interferência externa ou uso de ações predefinidas nas decisões feitas

Tabela 1 – Exemplo de um *build order* da classe *Terran*, intitulado $2FactVult/Mines$.

Build Order: $2FactVult/Mines$
9/10 - Supply Depot
11/18 - Barracks
11/18 - Refinery
16/18 - Factory[1][2]
16/18 - Supply Depot
18/26 - Factory[2]
22/26 - Supply Depot[3]
@1st Machine Shop 100% - Upgrade para Ion Thrusters
@2nd Machine Shop 100% - Upgrade para Spider Mines
@ 100% Factory - produzir interruptamente Vulture
28/34 - Supply Depot

pela arquitetura. Uma macroação m segue a definição:

$$m \in M = (a_1, a_2, a_3, ..a_i) \forall i \in 1, N : a_i \in A \tag{15}$$

onde M é conjunto de todas as macroações identificadas e catalogadas na arquitetura. Ações são utilizadas para compor uma macroação em vez dos nomes dos recursos que essas ações gerenciam. Caso algum recurso que faça parte da macroação já exista ou tenha quantidade suficiente, a ação necessária para construí-lo é descartada, e o tempo de execução da macroação diminui. O uso de ações para compor a macroação faz com que ações de todos os níveis de abstração sejam executadas de forma simples, apenas replicando o padrão de ações que foi identificado nos *replays*.

Nos próximos capítulos, discorre-se com detalhes sobre cada um dos módulos inicialmente discutidos e aqui apresentados.

Capítulo 5

Modelagem e Classificação dos Dados

Neste capítulo, o módulo *Classificação* e seus algoritmos serão descritos. Esse módulo é responsável por classificar as macroações utilizando os dados obtidos por meio dos *replays* de jogos. Além da classificação, o módulo organiza as macroações em uma árvore de decisão com capacidade preditiva. Essa árvore é construída também com base nos dados de *replays*, sendo utilizada para escolher macroações complementares. Essas macroações são executadas em conjunto com a macroação principal escolhida no módulo *Decisão*.

Para classificação dos dados, são utilizados os algoritmos de mineração sequencial GSP e PrefixSpan. Cada um dos algoritmos foi utilizado para macroações de determinado nível de abstração, pois cada algoritmo obteve seus melhores resultados em níveis de abstração específicos. Para a construção e manutenção da árvore preditiva, foi utilizado o algoritmo CHAID, que consegue manter nós com diversos filhos, o que facilita a conexão entre macroações de diferentes níveis de abstração. Os valores preditivos da árvore são construídos utilizando partidas entre jogadores nos *replays* de jogos. Com os valores de predição, macroações complementares podem ser escolhidas com base na macroação principal e nas observações recebidas da partida. Este capítulo está dividido da seguinte forma: Análise dos *replays* de Partidas; Descoberta de Padrões sob a Base de Dados Gerada; Hierarquia de Predição para Macroações.

5.1 Análise dos *replays* de Partidas

Os *replays* do StarCraft são armazenados em um formato proprietário e binário. Nesse caso, é preciso utilizar uma ferramenta de análise para conseguir acessar os dados e interpretá-los. A informação contida nos *replays* pode ser limitada, dependendo das ferramentas utilizadas. Os dados disponíveis, fazendo-se apenas uma análise básica, são: cliques na tela de cada jogador; cliques nas ferramentas disponíveis na interface do usuário. Assim, foi feita uma modificação na ferramenta LordMartin Replay Browser [1]

[1] *Disponível em http://www.teamliquid.net/forum/brood-war/88501-lord-martins-replay-browser*

para converter os *replays* em *logs* de jogo, os quais contêm os cliques dos jogadores, quantidade de recursos durante a partida, tempo de jogo, unidades construídas e destruídas etc. Foi modificada também a ferramenta BWChart [2], que consegue recuperar estatísticas das partidas, sequências de ações e posicionamento das unidades dentro do jogo. Esses dados são importantes para que seja possível identificar macroações dentro do jogo via mineração de dados. A Tabela 2 exibe um *log* parcial de uma partida obtido pela análise de um *replay* com a ferramenta LordMartin Replay Browser. Já a Tabela 3 apresenta um log extraído com a ferramenta BWChart. Sempre que o tempo de jogo for mencionado nos parâmetros de tabelas ou gráficos, a letra 'm' será utilizada para representar minutos e a letra 's' para segundos.

Tabela 2 – Log de uma partida obtido após análise feita com a ferramenta LordMartin Replay Browser customizada.

Jogador	**Tempo de jogo**	**Ação**
Player 1	00m:00s	Drone
Player 2	00m:00s	SCV
Player 1	01m:18s	Overlord
Player 2	01m:22s	Supply Depot
Player 2	02m:04s	Barracks
Player 1	02m:25s	Hatchery
Player 2	02m:50s	Barracks
Player 1	02m:54s	Spawning Pool
Player 2	03m:18s	Marine
Player 1	04m:10s	Zergling

Tabela 3 – Log de uma partida obtido após análise feita com a ferramenta customizada BWChart sobre um dos jogadores da partida.

Recurso	**Tempo de jogo**	**Posição**	**Comando**	**Qtd. recursos**	**Qtd. inimigos**
SCV(1)	00m:01s	125 x 654	Ir(x)	5	0
SCV(2)	00m:01s	127 x 655	Ir(x)	5	0
SCV(3)	00m:01s	128 x 656	Ir(x)	5	0
SCV(4)	00m:01s	129 x 658	Ir(x)	5	0
Barracks	00m:44s	156 x 611	Construir(x)	5	0
Refinery	01m:28s	174 x 632	Construir(x)	6	0
SCV(5)	01m:47s	159 x 608	Produzir(x)	7	0
SCV(5)	02m:14s	141 x 687	Ir(x)	8	0
Factory	02m:51s	161 x 608	Construir(x)	8	0
Firebat	02m:55s	159 x 608	Produzir(x)	9	0

Para obter uma boa amostragem, foram adquiridos um total de 12.121 *replays* de sites e comunidades de jogadores. Esses *replays* são de diferentes partidas entre as raças

[2] *Disponível em http://bwchart.teamliquid.net*

Tabela 4 – Quantidade de Replays adquiridos para cada classe do StarCraft e seu respectivo confronto em partidas disputadas no jogo.

Tipo de Replay	**Quantidade**
Terran vs Terran	3500
Terran vs Protoss	4000
Terran vs Zerg	824
Protoss vs Zerg	3245
Protoss vs Protoss	765
Zerg vs Zerg	3387
Total	12121

disponíveis no jogo, como é exibido na Tabela 4. Todos os replays são de partidas entre jogadores humanos experientes, na maioria profissionais do StarCraft. Foi cogitado o uso de *replays* de partidas entre *bots* das competições entre agentes jogadores, além de partidas entre jogadores humanos nos níveis intermediário e iniciante. Nos *replays* entre agentes jogadores, muitas partidas terminavam com um dos *bots* sem executar nenhuma ação no jogo, por falha no código ou erro de execução do código. Outras partidas tinham duração de mais de 30 minutos, devido a nenhum dos agentes efetuar ataques e permanecer apenas produzindo recursos. Já para partidas entre jogadores humanos em diferentes níveis, é difícil encontrar bases de dados com esses tipos de *replays*, porque a comunidade de jogadores não coleta esses tipos de partidas.

Mediante as ferramentas de análise, foram extraídos vetores de características da base de *replays*, que são utilizados na próxima seção, em conjunto com os algoritmos de mineração para classificação de macroações. A base dos vetores contém representações de diversos recursos e ações em forma de atributos, inclusive temporais. Todos eles são armazenados quando um jogador executa ações individuais e em sequência. Cada atributo descreve quando determinado recurso teve sua construção iniciada e que tipo de recurso é esse, via identificação da ação, que é executada para construí-lo. Os atributos também identificam quando um recurso recebe uma ordem para movimentação no mapa, ataque ou coleta de recursos minerais. Além disso, a quantidade de recursos produzidos, o tempo de jogo e o jogador vencedor também são descritos no vetor de características. Para cada *log* de uma partida são extraídos dois vetores de características iniciais. Cada um desses vetores contém todos os atributos descritos anteriormente executados por determinado jogador na partida. Esses vetores considerados genéricos são divididos em vetores com características específicas, que por sua vez serão utilizados para mineração e classificação dos dados.

A representação de um vetor genérico, em que cada característica c pertence a um

jogador J, é definida como:

$$c(x)_J = \begin{cases} r & \text{quando x foi construído ou executado pela primeira vez por J} \\ u & \text{se x é um } \textit{upgrade}\text{, u é o número de unidades atualizadas} \\ 0 & \text{x ainda não foi construído ou executado por J} \end{cases}$$

onde x é uma ação executada no jogo que representa vários tipos de comandos, que podem ser para construir um recurso, movimentar uma unidade ou ordenar um ataque. Contudo, x pode representar também uma ação que foi colocada em execução na partida, mas não foi finalizada. Os tipos de informação que são representados por x são definidos pelo atributo r que pode ser: o tempo de jogo, uma ação que foi executada, um recurso que foi apenas selecionado ou uma posição destino no mapa. Todos os valores são atribuídos a partir dos *logs* extraídos das partidas. É importante notar que os atributos do *log* da Tabela 2 são específicos para ações executadas por ambos os jogadores em uma partida. Já os atributos do *log* da Tabela 3 são de apenas um dos jogadores e contêm tanto ações que criam novos recursos, como $Produzir(x)$, quanto ações que utilizam recursos já existentes, como $(Ir(x))$. Mesmo com essas diferenças, os atributos de ambos os *logs* são utilizados para construir os vetores genéricos de características. A forma de criar esses vetores difere da maioria dos trabalhos investigados, que focam apenas as ações e o tempo de jogo em que foram executadas. Extrair mais atributos da partida e tentar identificar padrões de uso mesmo com representações distintas é interessante para expandir o conceito de macroação. Com todos os atributos utilizados, é possível identificar macroações em diferentes níveis de abstração do jogo, como, por exemplo, uma macroação de estratégia que descreve a criação de recursos e uma macroação de tática que descreve uma movimentação sobre os recursos criados.

Cada raça do jogo possui um diferente número de características, que é baseado em sua árvore de recursos e *upgrades*. A partir do vetor genérico de características, são criados quatro vetores específicos de características para cada um dos jogadores e sua respectiva raça. O primeiro vetor contém apenas ações que constroem recursos dentro do jogo; o segundo, apenas ações de movimentações, ataques e comandos diretos a unidades; o terceiro contém todas as ações executadas na partida, sendo esse uma junção dos dois primeiros; o quarto contém todas as posições do mapa em coordenadas bidimensionais que cada unidade assumiu após ser criada. Com os atributos que descrevem a criação dos recursos, é possível classificar diferentes macroações que constroem recursos semelhantes. Assim, construir determinado grupo de recursos pode ser uma macroação. Mas, construir a maioria desses mesmos recursos variando apenas alguns pode ser outra macroação diferente. Com os atributos de ações/comandos executados sobre unidades, é possível classificar diferentes macroações, sejam elas identificadas em grupos ou de forma individual. As combinações de recursos criadas e ações executadas geram padrões de sequências que indicam novas macroações.

As informações dos vetores de características são utilizadas para classificar as macroações, mas essas informações são utilizadas também nas próximas etapas da arquitetura. O POMDP *online*, por exemplo, utiliza os padrões de informações obtidos para calcular heurísticas *offline* e atualizar estados de crença. Em relação aos vetores de características, a representação que foi estabelecida para a raça Protoss, por exemplo, conta com: 42 características no primeiro vetor; 12 características no segundo; 54 no terceiro; uma quantidade maior no quarto vetor, dependendo da movimentação das unidades do jogador. A Tabela 5 mostra um conjunto parcial do vetor de características de recursos da raça *Protoss*. A ação $TrainHighTemplar$, que produz um recurso $HighTemplar$, possui valor temporal de 00m:00s, pois ele não foi produzido durante a partida. Apesar de a ação ter sido criada pelo jogador, a partida pode ter sido encerrada antes de ela ser completada ou cancelada pelo próprio jogador. Já a Tabela 6 mostra um conjunto parcial do segundo vetor de características, com os comandos diretos a unidades.

Tabela 5 – Exemplo de um conjunto parcial de dados do vetor de características contendo ações que produzem recursos na classe *Protoss*

Ação	**Tempo de jogo**	**Qtd. recursos**	**Qtd. inimigos**
Build Pylon	01m:22s	6	0
Build Gateway	02m:37s	7	0
Train Lair	03m:58s	9	1
Train Zealot	04m:33s	10	1
Train Zealot	05m:01s	12	4
Train High Templar	05m:52s	15	4
Build Assimilator	06m:47s	18	6
Build Stargate	07m:56s	22	11
Train Zealot	08m:33s	23	9
Build Supply Depots	09m:41s	25	2
Train High Templar	00:00	-	-

As distribuições de tempo foram incorporadas aos vetores como uma das características principais. É possível determinar padrões de tempo associados às ações que produzem recursos e executam comandos sobre recursos já existentes. Distribuições de posicionamento foram adicionadas para identificar padrões de combate, movimentações individuais e terrenos do mapa. Com isso, é possível detectar onde as unidades podem posicionar-se para obter vantagens de combate. Para algumas características, existem padrões que a maior parte dos jogadores segue, como por exemplo, o tempo de jogo quando é feita à construção de uma $Factory$. A Figura 29 mostra esse padrão. A $Factory$ possibilita a construção de unidades motorizadas de combate e tende a ser construída no quarto minuto de jogo. Isso indica que o recurso é gerado quando já existe pelo menos uma unidade terrestre construída, o que favorece a proteção da base enquanto a $Factory$ é construída. Para outras características, como movimentos sobre unidades, há uma var-

Tabela 6 – Exemplo de um conjunto parcial de dados do vetor de características contendo ações diretas a unidades na classe *Protoss*

Comando	Tempo de jogo	Posição
Mover(Probe)	02:05	785 x 451
Mover(Probe)	02:06	777 x 464
Mover(Probe)	02:08	794 x 447
Mover(Probe)	02:09	821 x 430
AtacarMover(Zealot)	02:10	580 x 321
AtacarMover(Dragoon)	03:40	588 x 334
Mover(Probe)	03:41	751 x 473
Construir(Gateway)	03:42	776 x 442
AtacarMover(Zealot)	03:44	583 x 322
Mover(Probe)	03:45	785 x 451
Mover(Probe)	04:06	777 x 464
Mover(Probe)	04:07	791 x 449
Mover(Probe)	04:09	779 x 466
AtacarMover(Zealot)	04:10	587 x 319
AtacarMover(Dragoon)	04:12	585 x 339
AtacarMover(Zealot)	04:13	584 x 454

iedade de padrões, devido às ramificações táticas que o jogador pode seguir. Coordenadas revelam posicionamento e controle de unidades individuais.

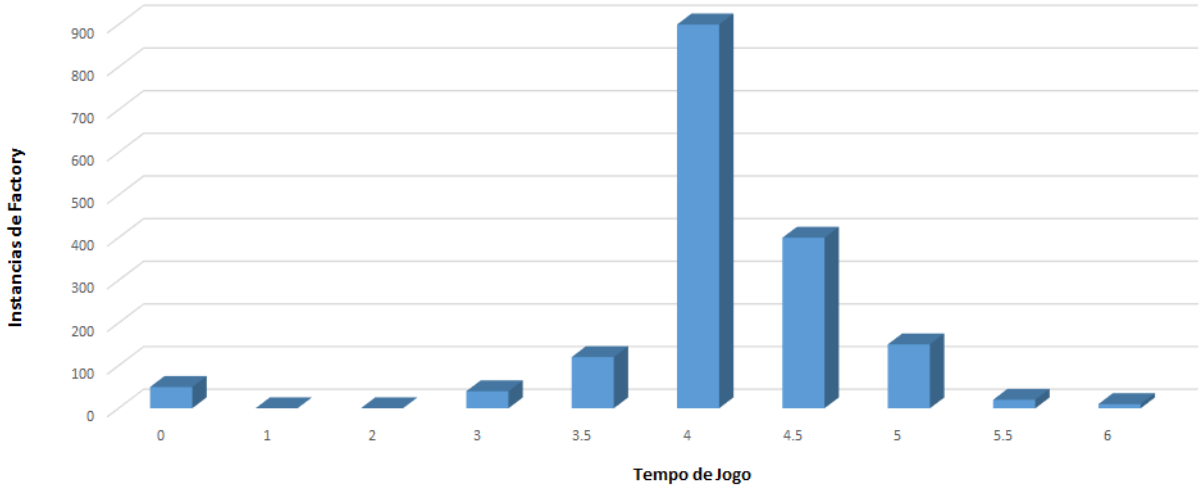

Figura 29 – Padrão de construção do recurso *Factory* com base nas distribuições de tempo analisadas a partir dos vetores de características.

Mesmo com a abordagem definida para os vetores de características, eles não conseguem capturar o estado do jogo completo. Isso ocorre devido às limitações das ferramentas de análise dos *replays*. Além disso, jogos RTS não possuem uma descrição do

estado de jogo. Um jogo RTS possui um estado contínuo que dura do início da partida até seu fim. Essa espécie de estado único é uma repetição dinâmica e iterativa em que todos os eventos do jogo ocorrem. Porém, com as representações obtidas nos vetores, é possível observar a sequência de ações e os conjuntos formados por elas. Esses conjuntos, quando comparados com os padrões sequenciais de tempo, ações, posições e demais atributos, apresentam modelos da ordem em as ações são executadas. Essas ações repetem-se em várias partidas com os mesmos arquétipos e ordem de execução, definindo assim uma possível macroação naquele nível de abstração analisado. A estratégia de análise proposta nesta tese combina variações de trabalhos que usam desde simples representações de ações (WENDER; WATSON, 2012), até casos e histórias de jogadas completas (SYNNAEVE; BESSIERE, 2011a), (SYNNAEVE; BESSIERE, 2011b), (CERTICKY, 2013).

5.2 Descoberta de Padrões sob a Base de Dados Gerada

Nessa etapa, o objetivo é encontrar o máximo de macroações no jogo StarCraft via mineração da base de dados com *replays*. Os classificadores são utilizados com os vetores de características construídos na etapa anterior. Uma macroação pode ser a produção de um único recurso, de um conjunto de recursos ou a execução de comandos diretamente relacionados aos recursos já produzidos. É importante classificar uma macroação e rotulá-la, para que haja controle da quantidade de vezes que ela é identificada durante as partidas. Essa etapa é executada de forma *offline*, devido ao dispendioso processamento de dados. Para execução da arquitetura de planejamento, as macroações devem ser identificadas nessa etapa. Cada macroação encontrada deve se encaixar em um dos três níveis de abstração definidos nesta tese, que são $Estratégia$, $Tática$ e $ControleReativo$.

O conceito de macroação apresenta vantagens diretas para a arquitetura aqui proposta. A primeira é habilitar o agente para considerar ações individuais e em conjunto em sua tomada de decisão. Ações em conjunto são definidas com base em comportamento identificado de jogadores experientes. O espaço de estados é consideravelmente reduzido com uso de um conjunto predefinido de macroações, em vez de tentar combinar conjuntos utilizando algoritmos durante uma partida em execução. Isso permite que a arquitetura consiga responder dentro das restrições de tempo real. Outra vantagem importante é classificar macroações em três níveis de abstração diferentes. Essa abordagem permite que as macroações identificadas cubram todos os tipos de tarefas do jogo, o que torna a arquitetura proposta uma AUG.

Foram utilizados dois algoritmos de classificação sobre a base de dados para obter o máximo de macroações possíveis. Para classificação de macroações de $Tática$ e $ControleReativo$, foi utilizado o algoritmo de classificações sequenciais GSP (CHEUNG, 2002). Para classificar macroações de $Estratégia$, utiliza-se o algoritmo de classificação de padrões fre-

quentes PrefixSpan (HAN et al., 2001). Cada algoritmo foi definido para o tipo de macroação em que alcançou os melhores resultados nos experimentos executados. Outros algoritmos de mineração e classificação, como *k-Means (KMN)* (JAIN, 2010) e *Non-nested Generalized Exemplars (NNGE)* (MARTIN, 1995), foram testados para essa etapa, no entanto, a flexibilidade e o conceito de mínimo e máximo intervalo entre elementos da base fizeram das abordagens de padrões frequentes a melhor opção. Os *replays* do StarCraft organizam as informações dentro de sequências que refletem a execução de ações por parte dos jogadores. Uma macroação pode ser identificada por características presentes em diferentes possíveis sequências, com ações em todos os níveis de abstração.

Foi definido um alfabeto G para as sequências que serão utilizadas pelos classificadores. Considere um conjunto de ações A, tempo de jogo T quando ocorre uma ação e a finalização dela F, onde $F = completo, incompleto$. O alfabeto de sequências é definido como $G = A \times T \times F$. Com essa definição, é possível obter um padrão sequencial do tipo: $s = (BuildRefinery, 00m\colon 41s, completo)$, $(BuildBarracks, 01m\colon 01s, completo)$, $(BuildMarine, 01m\colon 26s, completo)$, $(BuildSCV, 01m\colon 45s, completo)$, $(BuildMarine, 0m\colon 0s, incompleto)$, $(BuildSupply, 01m\colon 54s, completo)$. Esse padrão mostra que uma macroação pode ser determinada, dado um intervalo de tempo qualquer e se esse mesmo subconjunto do alfabeto continuar a surgir de forma frequente. Também é possível constatar que a ação $BuildMarine$ aos $01m : 51s$ não foi concluída. Outra ação de $BuildSuplly$ que produz suprimentos foi iniciada, indicando que a quantidade de suprimentos estava esgotada, por esse motivo, a ação foi cancelada. O alfabeto G foi definido para que macroações de $Estratégia$ fossem encontradas a partir dos padrões das ações. Esse alfabeto será referido como G_{est}. Foi definido um alfabeto para macroações de $Tática$, onde $G_{tat} = U[] \times C \times T \times P \times Q \times F$. Os novos itens desse alfabeto são: o conjunto de unidades que sofreram comandos $U[]$; comando utilizado C; a posição média para onde as unidades vão P; quantidade de recursos existentes Q. Para as macroações de $ControleReativo$, o seguinte alfabeto é definido $G_{ctr} = U \times C \times T \times P \times R \times F$. Nele o único novo item é o recurso ao qual o comando será aplicado R. Apenas uma única unidade U que executará o comando é identificada nesse caso, ao contrário do alfabeto de $Tática$, que verifica o conjunto de unidades $U[]$.

Para ampliar o alcance da classificação das macroações foi definido o conceito de Sequência Entre Jogadores (SEJ) $< J1, J2 >$, a qual reflete a troca de ações em sequência feita por dois jogadores adversários. O objetivo é identificar séries frequentes quando os jogadores alternam suas ações. Sequências com padrões de intervalo entre os jogadores tendem a apresentar possíveis macroações. Identificar qual dessas macroações foi vitoriosa aumenta a contribuição desse padrão, sendo ele colocado com maior frequência na classificação. Uma SEJ $< J1, J2 >$ a partir de uma sequência s composta pelos itens $(U \times C \times T \times P \times R \times F) \in G_{ctr}$ é a mesma sequência em que os itens são substituídos por $(U/u \times C/c \times T/t \times P/p \times R/r \times F/f)$. Por

exemplo, com o padrão $s = (Firebat, mover(x), 03m\colon 04s, 241 \times 492, Probe, completo)$, é indicado um ataque individual do $Firebat$ em um $Probe$. Há então outra sequência $s\tilde{} = \langle(Probe, mover(x), 03m\colon 07s, 248 \times 484, Firebat, incompleto)\rangle$ que caracteriza o cenário oposto. Dois agentes executam ações contra os recursos de seu inimigo, utilizando itens com valores semelhantes. No entanto, um dos agentes obtém sucesso e outro não, indicando que um dos recursos foi destruído ou recuou, e não houve confronto. Uma SEJ $< J1, J2 >$ pode ser identificada em macroações de todos os níveis de abstração.

O PrefixSpan para classificar macroações de *Estratégia* executa diversas iterações sobre dois vetores específicos de características. O primeiro contém a produção de recursos e o segundo, todas as ações executadas, sendo ambos para um dos jogadores. O PrefixSpan utiliza os dois vetores como uma base D, os itens da sequência G_{est} e um limite mínimo de frequência σ. O valor de σ para macroações de estratégia foi definido em 35% em relação a todas as sequências encontradas. Com esse valor, consideram-se para macroações apenas sequências que sejam capturas acima desse limiar. Inicialmente, a base D é examinada para que todos os itens frequentes de G_{est} sejam encontrados. Esse é o padrão sequencial com suporte tamanho 1, composto por cada recurso individual, tempo e finalização do alfabeto de macroações de *Estratégia*. As macroações encontradas com prefixo tamanho 1 são importantes, pois a construção de cada recurso de forma individual pode ser considerada a menor macroação. Essas macroações são utilizadas quando se deseja um recurso específico em vez de todo o conjunto composto com ela. Por exemplo, muitas vezes, é importante construir apenas um SCV, que é uma unidade que coleta minerais e acelera o acúmulo desses recursos. Selecionar a macroação que gera apenas a ação $BuildSCV$ é mais rápido e eficiente do que executar uma macroação que produz algumas edificações, e durante esse processo produz um SCV.

Com suporte tamanho 1, o PrefixSpan verifica todas as possíveis macroações que tenham pelo menos um item $< a_i >, i \in I$ de G_{est} contido em D como prefixo. Para todas as sequências, admite-se uma diferença de 12 segundos entre os tempos de execução, para aumentar a frequência de macroações com tamanhos de prefixo acima de 4. Durante a busca por padrões de prefixo 1, todas as macroações encontradas são utilizadas para dividir o espaço de busca. Aquelas com o prefixo $< a >$ já estão classificadas, e todas as remanescentes com o mesmo prefixo formam o banco de dados projeto $D_{<a>}$. Esse banco é vasculhado para gerar o padrão com suporte tamanho 2, tendo $< a >$ como prefixo. Nesse caso, uma macroação que contém apenas *Barracks* será usada como prefixo para encontrar outras macroações que contenham essa e mais outra ação qualquer. Faz-se a busca por padrões construindo-se uma estrutura de árvore chamada FP-Tree. Nela cada nó é um padrão sequencial frequente, associado a algum banco de dados com pelo menos $[\sigma \times |D|]$ sequências. A Figura 30 exibe uma busca por macroações com suporte tamanho 1. Já a Figura 31 apresenta a FP-Tree em construção com suporte 2 e 3, utilizando $< a >$ em cada possível sequência parcial. Para adicionar um item i em uma sequência de

tamanho k, i, pode ser alocado como novo item sufixo de $< a >$, formando $< a > o_a i$, ou alocado com o último item sufixo formando $< a > o_n i$. Caso uma sequência parcial não atinja a frequência de suporte mínima necessária, esta não é considerada uma macroação. A árvore parcial associada a essa macroação é eliminada.

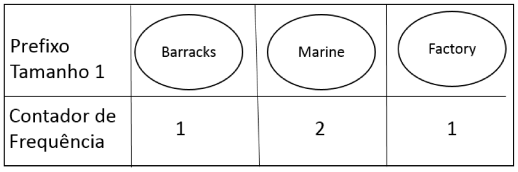

Figura 30 – Nós isolados classificados como macroações e que contém apenas um item da sequência G_{ctr}

.

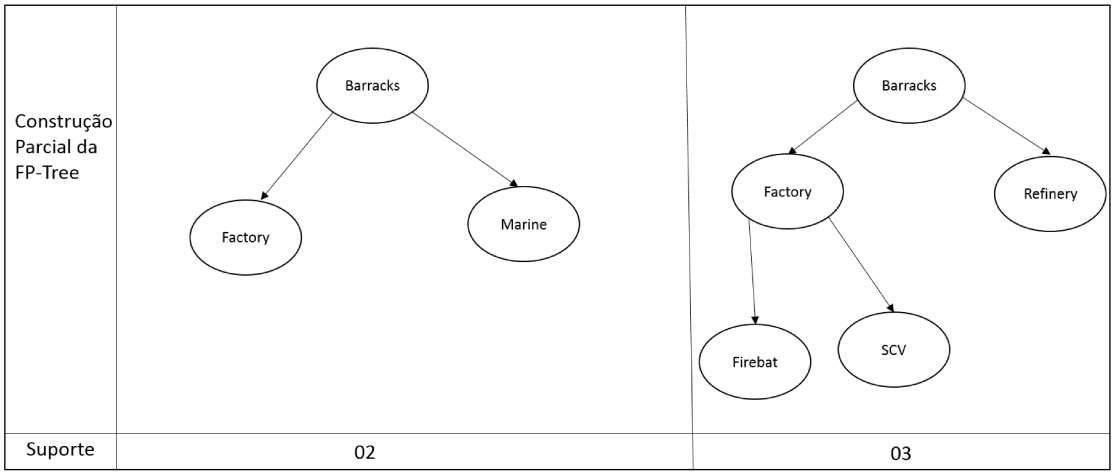

Figura 31 – FP-Tree em construção para classificar padrões sequências em que as primeiras macroações são combinadas com outros conjuntos de ações sequenciais.

As sequências parciais da FP-Tree utilizam as SEJ $< J1, J2 >$, contudo não identificam a capacidade de uma macroação ser utilizada de forma frequente. Em jogos RTS, uma macro tende a ser frequentemente utilizada, se seu suporte em razão de uma macroação utilizada contra ela tiver valor médio entre 0 e 1. Por exemplo, o suporte de uma macroação $ma1 = (BuildMarine, 01m : 22s, completo), (BuildFirebat, 01m : 41s, completo)$ é 2, e para uma macroação $ma2 = (BuildAcademy, 01m : 39s, incompleto)$ é 1. Comparando a razão dos suportes de $ma1/ma2$, o maior valor de $ma1$ indica que essa macroação prevaleceu na partida, mas $ma2$ continha menos recursos ou não utilizava os recursos apropriados em sua estratégia. Isso faz $ma1$ não se caracterizar como uma macroação de uso frequente. Assim, foi proposto um índice de frequência de utilidade $Util(s) = [0; 1]$ para as sequências s no formato $< J1, J2 >$. O índice mede a razão do suporte de uma sequência s e a sequência oposta do jogador $\tilde{s}$, dada pela

Equação 16. Quando $Util(s) = 0.5$, significa que a macroação que prevaleceu possui utilidade de frequência estável. A quantidade de recursos utilizados em ambas era próximo e essa macroação pode ser utilizada em diferentes momentos da partida. Macroações com $Util(s) <= 0.6$ recebem 4 pontos de frequência a mais, quando construídas e catalogadas durante o PrefixSpan. O Algoritmo 3 mostra os detalhes da versão utilizada para mineração de macroações de estratégia.

$$Util(s) = \frac{|suporte(s)|}{|suporte(s)| + |suporte(\tilde{s})|} \tag{16}$$

Algoritmo 3 $PrefixSpan(Sequência\ s, BancoProjetado\ D_s, Limite\ \alpha)$

1: $S \leftarrow s$
2: $Scaneia(D_s)$ encontrar cada item a frequente, onde s seja estendido $s\ o_a\ < a >$ ou $s\ o_n\ < a >$
3: **if** Se nenhum item a foi estendido **then**
4: **return** S
5: **end if**
6: **for all** a_i validos **do**
7: $BuscaSeq(s\ o_a\ < a_i >)$
8: **if** $sequência(s\ o_a\ < a_i >)$ **then**
9: $Conexão()\ sequência(s\ o_a\ < a_i >)$ em $(s\ o_a\ < a_i >)$
10: $S \leftarrow PrefixSpan(s\ o_a\ < a >, D_{s\ o_a\ <a>}, \alpha)$
11: **end if**
12: **if** $Fim(sequência(s\ o_a\ < a_i >))$ caso recursao não consiga aumentar a sequência com o suporte atual **then**
13: **if** $DuplaSeq(sequência(s\ o_n\ < a_i >))$ **then**
14: $S \leftarrow PrefixSpan(s\ o_a\ < a_i >, D_{s\ o_n\ <a_i>}, \alpha)$
15: **end if**
16: $Util(s)$
17: **end if**
18: **end for**
19: **return** S

O Algoritmo 3 recebe uma sequência s vazia no início (devido ao suporte tamanho 1), o banco projetado e o limite de frequência α. Na linha 2, a função *Scaneia* executa a varredura inicial no banco D_s, para definir as macroações com apenas uma única ação. Nas próximas iterações, serão encontradas macroações que tenham o item a combinado com outras ações. Caso nenhuma ação em sequência seja encontrada na função, o algoritmo interrompe aquela iteração e retorna a macroação s definida até o momento. Entre as linhas 7 e 10, é feita a busca por padrões sequenciais, utilizando cada variação da ação no item a_i e suporte definido na atual iteração. A função $BuscaSeq()$ gera nós filhos da sequência iniciada apenas com a ação do item a, adicionando ações que tenham sido identificadas com padrão sequencial de execução. Elas formam um novo nível na FP-tree, que é verificado na linha 8. Caso o novo nível tenha sido formado com ações inseridas logo após o item a, isso significa que foi gerada uma extensão com sucesso. A função

$Conexão()$ verifica as conexões entre os novos itens da FP-Tree e o algoritmo é chamado recursivamente. Uma nova iteração é iniciada para tentar criar mais um novo nível na árvore, utilizando ações com o mesmo padrão de alfabeto G utilizado na última iteração.

A condição da linha 12 do Algoritmo 3 é acionada quando uma sequência máxima de macroações para determinado suporte for atingida. Nesse momento, a verificação de SEJ $< J1, J2 >$ é iniciada. A função $DuplaSeq$ tenta encontrar uma nova ação para a sequência construída até o momento, a qual aumenta o raio de busca para ações que possam ter sido utilizadas e espelhadas por ações do adversário. Se até o momento uma sequência $s = (BuildBarracks, 01m : 22s, completo), (BuildFirebat, 01m : 54s, completo)$ foi encontrada e existe uma sequência $\tilde{s} = (BuildMarine, 01m : 17s, incompleto), (BuildMarine, 01m : 38s, completo), (BuildMarine, 01m : 50s, incompleto)$, isso significa que pode existir mais uma ação vinculada a s feita pelo jogador e que pode ser acrescentada. Caso essa frequência seja encontrada, o PrefixSpan é chamado recursivamente para tentar acrescentar mais uma nova ação à sequência. Caso a frequência não seja encontrada, nenhum novo item de ação é incluído e o índice de utilidade da sequência s é calculado. A frequência como macroação é registrada com base no valor do índice. Com o PrefixSpan, foram classificadas 250 macroações de estratégia para as três raças do StarCraft.

Para classificar as macroações de $Tática$ e $ControleReativo$, foi utilizado o algoritmo GSP, que efetua uma busca sobre a base de dados, investigando o intervalo entre a execução das ações para encontrar padrões de macroações. Como base de dados, são usados todos os vetores de características exceto o primeiro, que contém a produção de recursos. Após testes, foi definido como intervalo mínimo de tempo 2 segundos e máximo de 1 minuto e 30 segundos para macroações de $Tática$ que envolvem comandos a unidades. Para macroações de $ControleReativo$, intervalo mínimo de 0,5 segundos e máximo de 2 segundos, com ações que envolvem comandos a unidades individuais. Com a estratégia de teste por candidatos do GSP, os corretos intervalos de tempo habilitam a classificação dos dois níveis de abstração desejados. Macroações de $Tática$ com maiores intervalos e $ControleReativo$ com menores intervalos. O GSP busca pelo menos duas ações com possibilidade de formar uma macroação ou unir-se a outras para tornar-se tal. Caso as encontre, ele adiciona essa sequência, criando uma ramificação separada em uma árvore $and-or$. Caso a sequência já tenha sido criada em outra iteração, ele aumenta o valor de probabilidade dela na ramificação principal. Se as ações consideradas não formarem uma macroação, a ramificação é descartada. Caso formem uma macroação que, em futuras iterações, não supere um limite α de frequência, ela é descartada na fase de poda do GSP.

Para estabelecer o padrão de uma macroação, o GSP seleciona um item de ação e os limites de tempo dela. Em seguida, verifica outras ações que influenciam os recursos a priori da ação selecionada ou a relação dessa ação com outras executadas imediatamente depois. Caso haja uma sequência possível, a ação e sua tabela de frequências são adi-

cionadas, sendo incrementada sempre que essa sequência é encontrada novamente. Para determinar se uma sequência é possível e pode tornar-se uma macroação, eliminando quantidades de ações muito grandes ou desconexas, foi adicionada ao algoritmo a propriedade de sequenciamento máximo e mínimo. Essa propriedade é descrita pela Equação 17, onde: N é a quantidade de ações ainda não avaliadas nos vetores de características; Max e Min são os limites do GSP; δ é um valor decrescente proporcional ao valor já atribuído pelo classificador do GSP. Com o δ, à medida que mais ações são consideradas para uma única macroação, o critério de aceitação fica mais rígido, não permitindo que um conjunto de ações desconexas seja sequenciado. A equação de sequenciamento é usada sempre que uma ação é considerada para juntar-se com outra, a priori formando uma macroação.

$$Txsequencia = \frac{1}{N} \sum_{t=Min}^{Max} acuracia(t) * \delta \tag{17}$$

O uso da propriedade de sequenciamento é necessário para resolver três classes de problemas, que surgem na classificação de macroações de tática e controle reativo. As classes são:

- **Movimento de unidades**: essa classe de padrão é muito frequente nos vetores de características utilizados. O movimento de duas ou mais unidades diferentes aparece como uma sequência de ações primitivas. Diferenciar os movimentos que são para movimentação ou ataque é difícil, devido à falta de parâmetro ou marcadores. A taxa de sequenciamento pode criar distanciamento entre as ações de movimentação, colocando os padrões de distância no mapa x, y como fator para padronizar posições de destino próximas ao inimigo como movimentações de ataque e próximas à própria base como movimentações simples.

- **Ciclos de ações**: esses ciclos estão ligados ao fato de jogadores profissionais tentarem sincronizar seus ataques com conjuntos de unidades. Por exemplo, se o jogador possui 3 $Marines$ e está esperando a produção de 1 $Firebat$ terminar para começar a atacar, ele fica sincronizando os $Marines$ em posições que formam um ciclo de ida e volta no mapa. Esses ciclos são movimentos que vão de um ponto de ida até um de volta, até o recurso ficar pronto, contudo, para identificar essas movimentações como não necessárias e utilizar apenas a posição do mapa onde as unidades se reunirão para começar o ataque, a taxa de sequenciamento é usada. Nesse caso, é observada a quantidade de ações N ainda não verificadas, e caso elas utilizem as mesmas ações já consideradas em forma de ciclo, o valor de N é zerado e a taxa de sequenciamento não as considera.

- **Ações fantasmas**: esse conceito é específico dos *replays* de jogadores profissionais de RTS. Ações fantasmas referem-se ao hábito de executar ações desnecessárias ou ações que não possuem retorno a nenhuma unidade. Essas ações são feitas nos

primeiros minutos do jogo, quando os jogadores não têm muitas unidades produzidas e aquecem os dedos, reordenando comandos já em execução ou escolhendo recursos já produzidos. Essas ações estão presentes nos vetores de características e podem dificultar a identificação de macroações ou estar presentes nelas. Nesse caso, o intervalo máximo e mínimo é configurado para que não sejam consideradas ações com intervalos menores do que 1 segundo e que tenham a mesma descrição. Assim, as ações fantasmas não são consideradas na busca por padrões sequenciais, uma vez que outros tipos de ação não podem ser executados dentro desse intervalo mínimo estabelecido.

O algoritmo 4 exibe o GSP utilizado no trabalho. Nas três primeiras linhas, o algoritmo recebe todas as ações individuais que são aquelas com tamanho 1, extrai todas as sequências frequentes de ações em L e configura o suporte com K em 2. Na linha 5, a função *Sequencia* testa possíveis combinações de ações de movimentação, ataque e outros tipos, com base nos padrões da base de dados. Caso existam ações na sequência montada que sejam de alguma das classes de problemas, a função $TSequencia$, na linha 6, aplica a taxa de sequência, eliminando-as. Nas linha 7 e 8, a macroação parcial formada até o momento tem sua frequência de uso verificada com a função $Frequencia$. Caso a sequência com a adição de novas ações não tenha atingido valor maior ou igual a σ, parte dela ou ela toda pode sofrer poda. O valor de σ é proporcional a 20% do tamanho da sequência S_k atual. Na linha 12, todas as macroações parciais têm seu contador interno incrementado em relação à frequência, caso não tenham sido podadas. Nas linhas 14 e 15, cada macroação parcial é adicionada como um ramo na árvore *and-or* onde ficam as macroações totais, ou seja, aquelas que já foram classificadas em L_k. Enquanto uma macroação está sendo considerada no processo de classificação, as ações que a compõem são colocadas em uma ramificação temporária da árvore *and-or*. Uma vez que a macroação é considerada um padrão correto pelo GSP, ela é adicionada como um nó à ramificação principal. Essas macroações são passadas até a lista que armazena todas as macroações L, desde que não tenham sido armazenadas em iterações anteriores.

Utilizando o classificador GSP, foi encontrado um total de 60 diferentes macroações de tática e 49 macroações de controle reativo, em todas as raças do jogo. A Tabela 7 apresenta a quantidade de macroações para cada raça e nível de abstração. Em todas as raças, foi encontrada uma quantidade maior de macroações de estratégia. Isso ocorre devido à quantidade de combinações que podem ser feitas, utilizando-se os recursos de edificações, unidades e atualizações. A construção de apenas um recurso individual ou um conjunto de recursos pode ser considerada uma macroação de estratégia. Macroações de tática são comandos diretos a unidades, formando combinações ou sendo unitários. Macroações de controle reativo são comandos diretos com intervalos de tempo muito pequenos.

Algoritmo 4 $GSP(Bancodedados\ D,\ Banco\ 1Frequência\ D_1,\ Limite\ \alpha)$

1: $F \leftarrow D_1$ Todas as sequências tamanho 1
2: $L \leftarrow Extrai(D)$ Conjunto de todas as sequências frequentes do banco de dados
3: $K \leftarrow 2$
4: **for** k *até* $L_{k-1} \neq 0$ **do**
5: $S_k \leftarrow Sequencia(S_{k-1},\ k)$ Testa combinações de sequências de S_k
6: $S_{k-1} \leftarrow TSequencia(S_{k-1})$
7: **if** $S_k \leftarrow Frequencia(S_{k-1}, \sigma)$ Verifica a frequência das novas ações para podar ou não a macroação **then**
8: $S_k \leftarrow Poda(S_k)$
9: **end if**
10: **end for**
11: **for all** $c \in D$ Para cada sequência do banco de dados D **do**
12: $s_c \leftarrow +1$ Incrementa o contador de todas as sequências
13: **if** $c_\alpha > \alpha$ **then**
14: $L_k \leftarrow c$
15: $L \leftarrow L \bigcup L_k$
16: **end if**
17: **end for**
18: **return** L

Tabela 7 – Quantidade de metas encontradas com o GSP para cada raça do StarCraft.

Raça	**Estratégia**	**Tática**	**C. Reativo**	Total
Terran	96	21	16	133
Protoss	93	20	18	131
Zerg	61	18	15	94

5.3 Hierarquia de Predição para Macroações

Além de encontrar as macroações, é preciso armazená-las em uma estrutura que promova organização por nível de abstração e conexão com outras macroações. Para isso, é utilizada uma árvore de decisão preditiva com testes Qui-Quadrado de Pearson (CHAID) (KASS, 1980), para modelar as macroações encontradas e estabelecer uma cadeia de conexões entre seus respectivos níveis de abstração. A diferença dessa abordagem para a clássica de árvores é que a CHAID permite múltiplas divisões em um nó, além de incorporar informações de regressão junto com a classificação, bem como permite que um nó tenha quantos filhos forem necessários. Uma macroação em um nível de abstração tende a possuir conexão com várias macroações de um nível inferior. Uma macroação de estratégia possui macroações de tática conectadas a ela, que podem ser executadas logo em seguida, complementando a estratégia escolhida. Com o uso de regressão, CHAID pode indicar quando utilizar uma macroação para complementar outra já em execução e escolhida pelo módulo de *Decisão*. Detalhes de como é feita a execução de múltiplas macroações serão apresentados no Capítulo 6.

A CHAID tem papel importante na arquitetura proposta. Todas as macroações encontradas estão presentes na árvore, em uma disposição em que macroações de estratégia estão no primeiro nível da árvore. O nó raiz é nulo, pois não é necessário representá-lo, uma vez que a tomada de decisão é feita sempre para uma macroação de estratégia. A CHAID, em um primeiro momento, opera como estrutura para organizar as macroações. No segundo nível da árvore, estão todas as macroações de tática, conectadas com as macroações de estratégia que possuem alguma probabilidade de ser executadas em conjunto, ou seja, uma estratégia com uma tática observada com frequência nos dados analisados dos *replays*. No terceiro nível, estão as macroações de controle reativo, também conectadas àquelas de tática logo acima e que possuem alguma probabilidade de execução conjunta, ou seja, uma tática seguida de um controle reativo. A CHAID cria então uma representação entre todos os níveis de abstração do jogo, junto com as macroações de cada um deles, replicando o comportamento de execução de ações dos jogadores presentes na base de dados. Assim, em um segundo momento, a CHAID opera executando predição para escolha de macroações de tática e controle reativo, quando uma estratégia já foi escolhida pelo módulo de *Decisão*. A Figura 32 mostra um exemplo da CHAID, com 1 macroação de estratégia no primeiro nível, 4 de tática no segundo nível e 5 de controle reativo nos demais.

Na Figura 32, cada macroação tem seu valor de Chi-Square (X_k^2) e $p-value$ calculados. Esses valores indicam de forma preditiva qual macroação de tática deve ser escolhia, dada a atual macroação de estratégia em execução e as observações recebidas da partida em andamento. Logo em seguida, os valores de predição são atualizados para a escolha de uma macroação de controle reativo, dada a tática escolhida e em execução. Dessa forma, a escolha de macroações pelo CHAID é feita enquanto a partida é disputada, utilizando os valores obtidos na montagem da árvore *offline* e atualizando-os com dados obtidos *online*. Na Figura 32, foi utilizado um exemplo com apenas uma macroação de estratégia no primeiro nível da árvore, contudo, a árvore completa possui a quantidade de macroações indicadas por raça na Tabela 7. Por exemplo, a macroação de estratégia $BarrackToMarine$ será executada após ser escolhida pelo módulo de *Decisão* na Figura 32. A macroação de tática $ReagroupFast$ é aquela com o melhor valor de predição para ser escolhida, complementando a ação de estratégia. Em cada conexão entre uma macroação e sua respectiva de nível superior, os valores de agrupamento da CHAID podem estar em algum intervalo $< 100\&50 >$ ou ser maiores ≥ 50 ou menores ≤ 50, de acordo com o valor obtido. Além do valor de agrupamento, o $p - value$ é calculado para cada macroação, sendo obtido mediante o agrupamento das variáveis independentes que vêm em forma de parâmetros do jogo durante a partida. Os valores utilizam os mesmos intervalos que o Chi-Square.

Com todas as macroações já classificadas, o processo de construção da CHAID com o modelo preditivo é iniciado. As variáveis dependentes são compostas por todas as

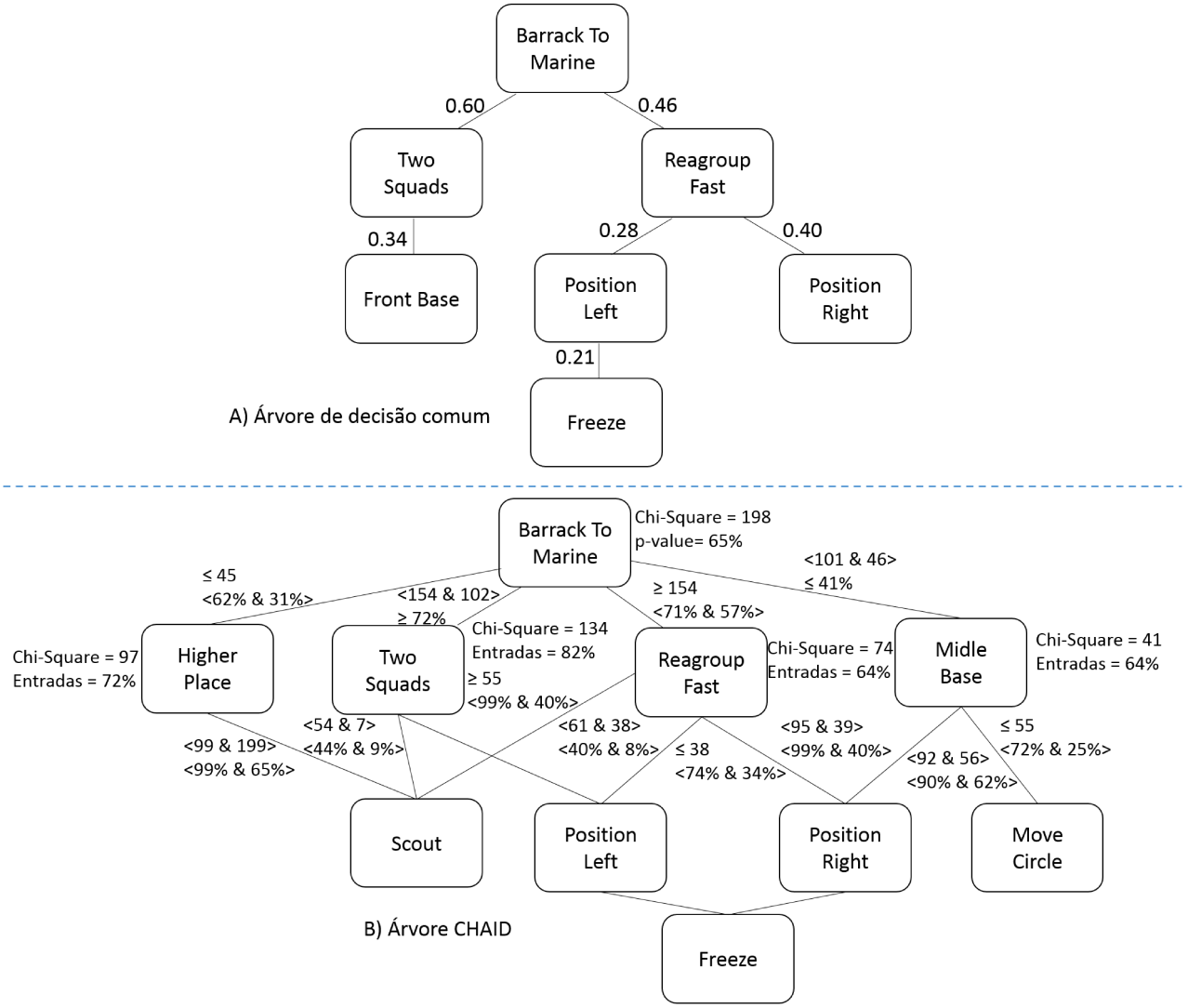

Figura 32 – Exemplo de CHAID em relação a abordagem de árvore convencional. CHAID utiliza do valor preditivo das classes de variáveis para escolher uma macroação.

macroações de tática e controle reativo de uma determinada classe. As variáveis independentes e, nesse caso, preditivas são a macroação de estratégia atualmente escolhida, junto com os dados sensoriais obtidos do ambiente no jogo naquele momento. Esses dados sensoriais incluem a quantidade de recursos de minerais e gás, quantidade de construções e unidades, quantidade de inimigos, posição dos inimigos, recursos destruídos e tempo do jogo. Assim, a CHAID prediz qual macroação escolher sempre que uma estratégia é escolhida. O mesmo processo de escolha é feito para uma macroação de controle reativo, com base na atual macroação de tática e nos dados sensoriais recebidos. A CHAID é criada utilizando como base de treino os mesmos *replays* utilizados na classificação, com o aditivo das macroações. Durante o treino, as frequências estatísticas de predição são calculadas, verificando a escolha de macroações feita pelos jogadores, com base nas variáveis

preditivas descritas anteriormente. O formato de construção da CHAID foi desenvolvido para este trabalho, levando em conta a hierarquia na execução de ações e os níveis de abstração que foram observados nos dados obtidos de jogadores humanos.

Cada macroação de tática e controle reativo é representada por uma variável dependente Y. O conjunto de todas as variáveis preditoras é dado por X_m, onde $m = 1, ..., N$ e N é a quantidade de preditores. cada preditor é uma macroação de estratégia junto com os dados sensoriais capturados quando a macroação foi executada no jogo. A primeira etapa da CHAID é a *junção*, em que, para cada preditor X, aqueles com pouca significância em seu valor preditivo são acoplados. Cada preditor X, individual ou proveniente de uma união, será um nó filho na árvore caso seja usado na próxima etapa de divisão de nós. Para isso, nessa etapa é calculado o $p - value$ de cada preditor, que mede o quanto este é apropriado para escolha de uma macroação de nível de abstração inferior. Inicialmente, é verificado se o preditor possui apenas uma, duas ou mais classes. Uma macroação, quando utilizada como preditor, possui apenas duas possíveis classes, estratégia ou tática, denominando essa variável como categórica nominal. Nesse caso, o cálculo do $p-value$ deve ser feito utilizando o ajuste de Bonferroni $B(i)$. Já os dados sensoriais da partida são variáveis quantitativas contínuas, sendo necessário utilizar um Modelo Classe Latente (CL) (HAGENAARS; MCCUTCHEON, 2002) $P(X = j)$, que permite trabalhar com uma análise mista, em que haja preditores categóricos e quantitativos. O cálculo do $p - value$ é dado pela Equação 18, utilizada para determinar a melhor divisão dos nós da árvore, e a Equação 19, utilizada para calcular a verossimilhança de proximidade no uso de macroações em sequência.

$$X^2 = \sum_{j=1}^{J} \sum_{i=1}^{I} P(X = j) \frac{(n_{ij} - \hat{m}_{ij})}{\hat{m}_{ij}} B(i) \tag{18}$$

$$G^2 = 2 \sum_{j=1}^{J} \sum_{i=1}^{I} n_{ij} P(X = j)\ ln(n_{ij}/\hat{m}_{ij}) B(i) \tag{19}$$

No cálculo do $p - value$, uma tabela de contingência é criada a partir das classes de Y, que são tática e controle reativo como colunas, e as categorias do preditor X, que são os dados sensoriais como coluna. A frequência de células obtidas n_{ij} na tabela é dada por $n_{ij} = \sum_{n \in D} f_n I(x_n = i \wedge y_n = j) w_{ij}$, com f_n sendo o peso de frequência para cada preditor, utilizando o valor 0,6 obtido empiricamente. D é a quantidade de dados utilizados para união e separação no atual nó, delimitando o valor de i, j em cada iteração. O valor de n_{ij} representa a relação de vezes em que o jogador utilizou uma macroação de estratégia e, em seguida, uma de tática ou uma macroação de tática e uma de controle reativo. A frequência de células estimada $\hat{m}_{ij}$ é dada por $(x_n = i, y_n = j)$, utilizando um modelo independente. Para cada caso de teste entre variáveis, foi utilizada uma constante de peso

w_{ij}, obtida por testes com valores estimados com a Equação 20.

$$w_{ij} = \frac{w_{ij}}{n_{ij}}, w_{ij} = \sum_{n=D} w_n f_n I(x_n = i \wedge y_n = j). \tag{20}$$

Voltando à etapa de *junção*, caso o teste de classes aponte que o preditor X possui apenas uma classe, o $p - value$ é definido como 1 e a CHAID segue para a etapa de separação. Caso X tenha duas classes ou mais, o $p - value$ é calculado para as classes que se unirão, utilizando a Equação 18 e o ajuste de Bonferroni, dado pela Equação 21.

$$B = \sum_{v=0}^{r-1} (-1)^v \frac{(r-v)^I}{v!(r-v)!}, \tag{21}$$

onde I é a quantidade de classes do preditor e r é a quantidade resultante de classes depois que a etapa de *junção* termina. O multiplicador B é a combinação de possibilidades em que as I classes podem ser unidas em r classes. Quando o ajuste B é utilizado, o valor da CL $P(X = j)$ presente na equação é configurado com o valor 1, pois não há valor quantitativo no preditor.

Quando o preditor corresponde aos dados sensoriais do jogo, eles possuem duas ou mais classes e valores quantitativos. A união de classes é feita com o cálculo do $p - value$ na Equação 18 e o modelo CL dado pela Equação 22.

$$P(X = j) = \sum_{k=1}^{K} P(X = k) \prod_{m=1}^{M} P(X_m = j_m | Y = k), \tag{22}$$

onde o valor de K cresce iterativamente até quando for possível criar novas categorias; Y_m é uma das M macroações nominais que funcionam como variáveis de resposta com $m = 1, 2..., M$;; j_m é o nome e uma categoria em particular definida; e J_m é o número total de categorias criadas para a variável Y_m. Nesse modelo, são obtidos $k - categorias$ latentes dos preditores X para descrever a associação entre cada valor contínuo de dados do jogo e macroação dependente Y. Para os conjuntos numéricos, tais como quantidade de unidades ou inimigos, são criados agrupamentos entre aqueles que aparecem com maior frequência, formando pares ou trios de valores que passam a ser uma categoria. Cada conjunto de valores responde por sua classe, e os dados da partida podem ser descritos pela quantidade de classes que possuem quando são recebidos. Assim, o cálculo do $p - value$ é uniforme para quaisquer macroações independentes Y. $P(X = k)$ representa a probabilidade de estar em uma classe latente $k = 1, 2, ..., K$, e $P(X_m = j_m | Y = k)$ é a probabilidade condicional de obter j_m em relação a Y. Essa probabilidade é obtida utilizando o teorema de Bayes, dado pela Equação 23.

$$P(X_m = j_m | Y = j) = \frac{P(Y = j, X_m = j_m)}{P(X_m = j_m)} \tag{23}$$

Com o cálculo do $p - value$ definido para macroações e dados sensoriais do jogo, como variáveis independentes, a segunda etapa da CHAID, denominada *separação*, é iniciada.

Os valores $p - value$ e verossimilhança gerados com os devidos ajustes indicam a melhor divisão para cada preditor. Na *separação*, o melhor preditor, seja uma macroação ou um dado sensorial, é escolhido para gerar uma divisão no nó. A escolha é feita utilizando dois passos: selecionar a macroação ou dado sensorial com menor $p - value$ ajustado (mais significante); caso o $p - value$ seja menor ou igual a um fator $\alpha_{divisão}$, dividir esse nó. Caso não seja, não haverá divisão e o nó é considerado terminal na árvore. O valor de $\alpha_{divisão}$ foi configurado em 190 e determina quando um preditor não consegue indicar a escolha de uma macroação. O processo de crescimento da CHAID é feito durante a *separação*, em que cada macroação de estratégia ou um conjunto de dados sensoriais que foram unidos na etapa anterior pode formar uma divisão, gerando uma nova conexão preditiva com uma macroação de tática ou controle reativo. O processo de *separação* é feito para cada conjunto de X de variáveis preditivas por cada variável dependente Y. Devido às múltiplas divisões que a CHAID faz, cada macroação de tática contém diversos possíveis preditores que podem determinar sua escolha, como mostra a Figura 32.

O processo de *separação* e crescimento da CHAID é interrompido, quando alguma das condições descritas a seguir é satisfeita.

1. Caso um nó de uma macroação obtenha valores idênticos para diferentes preditores, esse nó não será mais dividido, além de não gerar mais conexões com macroações de estratégia e conjuntos de dados sensoriais.

2. Caso uma macroação independente gere apenas uma divisão e estabeleça apenas uma conexão com uma macroação dependente de tática ou controle reativo.

3. Caso o tamanho de um novo nó gerado seja menor do que dois, ou seja, sua predição seja definida com base em dois ou menos dados sensoriais mais a macroação de estratégia usada nessa divisão.

Quando a CHAID está totalmente construída, sua hierarquia possui todas as macroações de estratégia e os possíveis conjuntos de dados sensoriais conectados às macroações de estratégia, e estas conectadas às macroações de controle reativo. O processo para decidir por alguma macro começa após a imediata escolha da estratégia pelo tomador de decisão do módulo *Decisão*. Os preditores têm seu valor de $Chi - Square$ e $p - value$ calculados, e aquele com melhor probabilidade com os valores já definidos na CHAID é escolhido, levando à escolha de uma macroação de tática. Logo em seguida, o mesmo processo é feito colocando a macroação de tática como preditor e decidindo por uma macroação de controle reativo. Essa estratégia de execução de múltiplas macroações será detalhada no próximo capítulo.

Com a etapa de classificação concluída, o próximo passo é utilizar o POMDP *online* e outras técnicas em conjunto com as macroações encontradas, para tomar decisões e

gerenciar as ações que serão enviadas para execução no ambiente do jogo durante uma partida.

5.4 Considerações Finais

Neste capítulo, foram apresentados os conteúdos relativos à metodologia de análise e uso de dados do StarCraft com a arquitetura proposta nesta tese. Os métodos propostos são interessantes em várias etapas, que vão desde a obtenção dos dados até a classificação e organização hierárquica deles. Os dados obtidos são essenciais para o controle de um agente jogador em ambientes de tempo real, pois eles guiam o planejamento do agente utilizando informações coletadas do próprio ambiente, possibilitando o gerenciamento de múltiplas tarefas presentes em jogos RTS por meio do conceito e hierarquia de macroações que são propostas.

A etapa de coleta de informações e classificação dos dados é responsável por gerar toda a base de informações e conhecimento, a priori, que a arquitetura de planejamento utilizará. O uso de abordagens de mineração sequencial junto com uma estrutura de árvore preditiva é uma estratégia que apresentou bons resultados na exploração de macroações. Em relação a outros trabalhos da literatura, eles concentram-se em obter apenas padrões de estratégias já catalogadas nas bases de dados de comunidades do jogo. A proposta de classificar macroações em diferentes níveis de abstração ainda não foi explorada. O uso da árvore CHAID no auxílio à escolha de macroações habilitou a execução simultânea de ações em diferentes níveis de abstração do jogo, assemelhando-se ao que é feito por jogadores humanos. O conceito de macroação foi uma proposta para esta tese e que proporcionou toda a metodologia de análise e mineração de dados.

Capítulo 6

Planejamento e Tomada de Decisão

Neste capítulo, será apresentado e discutido o módulo de *Decisão*, que é responsável pela tomada de decisão da arquitetura em relação às macroações. Além disso, o módulo precisa prover uma dinâmica de planejamento que mantém a tomada de decisão como um processo contínuo e iterativo, até o fim da partida. Sempre que for necessário, é preciso fazer novas escolhas. Após a tomada de decisão desse módulo em relação à macroação de *Estratégia*, são escolhidas macroações complementares a partir da CHAID. O módulo de *Decisão* funciona como um sistema integrado, com algoritmos e técnicas operando sobre uma ordem específica. Todos os detalhes serão apresentados no decorrer deste capítulo.

Como algoritmo principal desse módulo é utilizado o processo de decisão de Markov parcialmente observável (POMDP) (MONAHAN, 1982), o qual é responsável pela tomada de decisão em relação às macroações durante a partida. POMDP é um algoritmo com alto custo computacional em domínios com grande número de observações e ações, como o de jogos RTS. Assim, ele foi modificado em alguns aspectos de sua execução e adaptado para operar sob a modelagem de dados construída nesta tese. Além disso, são propostas duas políticas para geração de estados de crença. Uma dessas políticas foi desenvolvida especificamente para este trabalho e para o domínio de jogos RTS. Este capítulo está dividido da seguinte forma: POMDP *Online* para Jogos RTS; AEMS: Política Baseada em Heurísticas; RTS Sample: Política para o Domínio de Jogos RTS.

6.1 POMDP *Online* para Jogos RTS

POMDP foi escolhido para a tomada de decisão, devido a diversas vantagens, tais como: capacidade de lidar com ambientes incertos; explorar grande parte do espaço de estados de crença; possibilidade de modificação e adaptação do algoritmo. Para viabilizar o uso do POMDP na arquitetura aqui proposta, foi preciso propor uma abordagem *online* do algoritmo que considera as restrições de ambientes dinâmicos e de tempo real. Versões *online* do POMDP vêm sendo exploradas há pouco tempo, e grande parte da literatura é concentrada em versões *offline* do algoritmo. Existem métodos *offline* recentes baseados

em pontos de atualização (SMITH; SIMMONS, 2012) e agentes descentralizados (AMATO et al., 2015) que obtêm bons resultados em grandes domínios. Porém, eles tornam-se intratáveis quando existem restrições no tempo de resposta, como em jogos RTS.

Dentro do módulo *Decisão*, o POMDP é responsável por escolher uma macroação de *Estratégia* para execução no jogo, com base nas macroações de *Estratégia* que estão descritas na Tabela 7. Para a escolha, as macroações disponíveis, as observações recebidas da partida e o atual estado são usados na geração de novos estados de crença. O processo de gerar novos estados de crença inclui simulações destes com outras macroações. Nessas simulações, várias jogadas à frente do tempo de jogo atual são disputadas, e diversos estados e macroações são executados. Esse processo é chamado de *lookahead*, e sua execução é feita em uma árvore com horizonte de estados finitos. Nessa execução, probabilidades entre os estados de crença são geradas e recompensas para ações são calculadas, como descrito na Figura 14. A macroação presente no ramo da árvore que gerou a recompensa máxima, ou seja, a maior dentre todas as recompensas somadas de cada ramo, é escolhida para execução.

A Figura 33 mostra um exemplo de um *lookahead* do POMDP dentro do ambiente de jogos RTS. Nessa figura, são utilizadas três macroações e uma observação para efetuar o *lookahead*. O estado de crença b_0 representa o início do jogo. Os demais estados b_1, b_2 e b_3 são gerados após a execução de alguma macroação e uma observação recebida. Os estados de crença são compostos por diversos estados do jogo s_n. Cada estado do jogo possui uma descrição de recursos, inimigos e uma probabilidade de o ambiente estar nesse estado do jogo (identificados na Figura 33 pelas expressões $recursos()$, $inimigos()$ e $prob()$, respectivamente). Cada macroação é composta por ações do jogo, como $CollectMinerals$ e $BuildMarinem_n$, e existe uma recompensa R por sua execução. As macroações recebem as observações o_n e as utilizam para gerar a probabilidade de alcançar outros estados de crença. Os novos estados de crença atualizam as probabilidades dos estados internos do jogo, além de gerar novos estados deles também. O estado de crença b_1 atualiza as probabilidades dos estados s_1 a s_3. Cada estado de crença pode conter até 15 estados do jogo s. Os valores são divididos com os novos estados s' gerados com a execução da macroação m_1 e da observação o_1. Essa geração de estados e escolha de macroações continua sempre que novas observações são recebidas. Para recomeçar o *lookahead*, o estado de crença atual é colocado como nó raiz b_0 e uma nova macroação será escolhida.

É importante salientar a definição e as diferenças entre o estado do jogo $s \in S$ e o estado de crença b do POMDP. O estado do jogo s descreve a situação do ambiente do jogo em um dado momento t da partida em disputa. Esse estado precisa descrever a transição entre a atual situação de recursos do agente e o acréscimo de novos recursos via execução de ações. Essa descrição deve ser composta por variações dentro da representação do estado do jogo. Por exemplo, descrever a quantidade de unidades e minerais em um dado momento da partida é parte da representação necessária de um estado do jogo. Caso

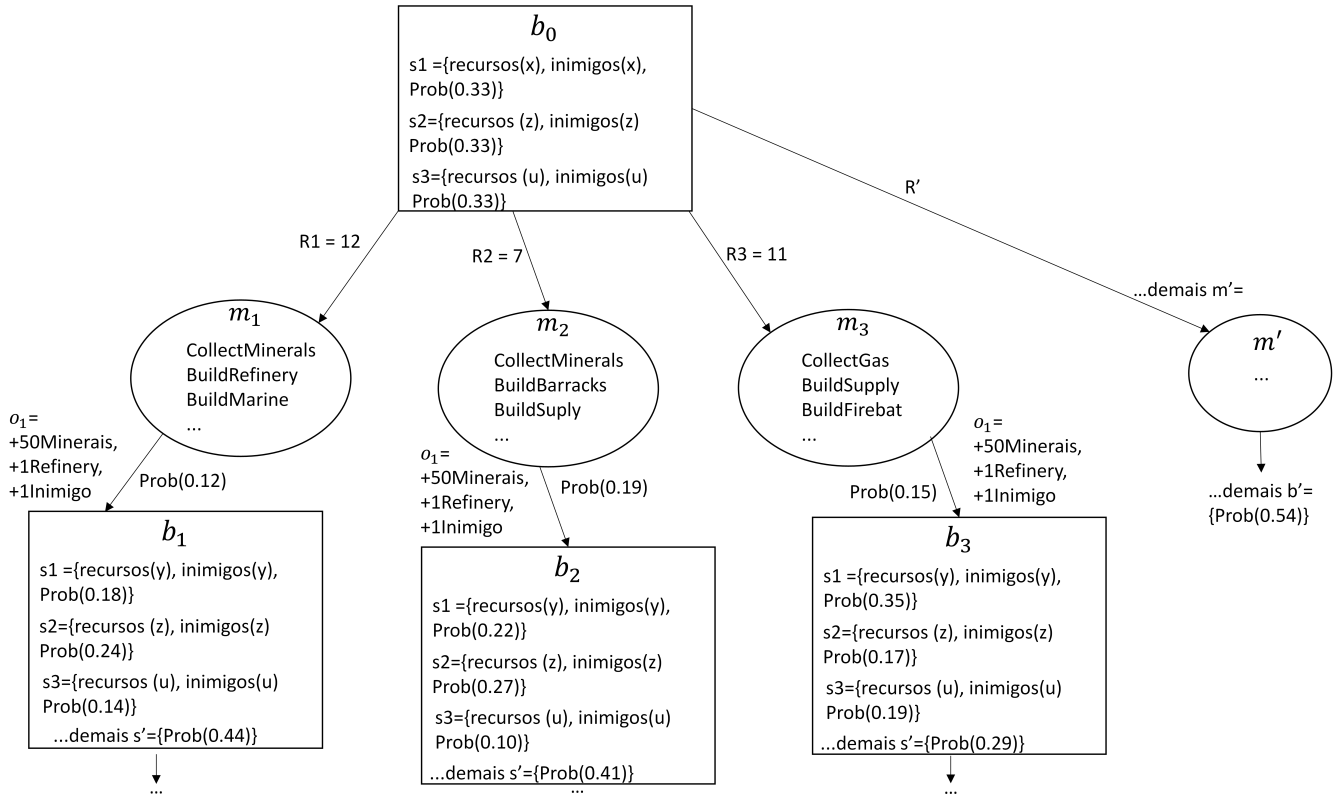

Figura 33 – Modelagem do POMDP *online* para jogos RTS.

novas unidades sejam produzidas em um momento à frente na partida, o estado do jogo precisa modificar sua descrição para representá-las. Já o estado de crença do POMDP é uma descrição da probabilidade de a partida encontrar-se em algum dos possíveis estados do jogo $b(s)$, em um dado momento da partida. O estado de crença é uma função que mapeia a probabilidade de estar em cada um dos estados do jogo $\sum_{s'=1}^{15} b(s) = 1$. Nesse, 15 é a quantidade máxima de estados de jogo que são utilizados para gerar um estado de crença. Nessa função $0 \leq b(s) \leq 1$, dado que uma macroação foi executada e novas observações foram recebidas. Para que o POMDP consiga mapear as probabilidades e atualizá-las, o estado do jogo s precisa ser definido.

Em jogos RTS existe uma dificuldade em definir quais atributos ou informações do jogo devem ser utilizadas para representar o estado do jogo. Nesses tipos de jogos, cada partida conta com inúmeras ações que são executadas e seus efeitos aplicados ao ambiente. Assim, é possível capturar diversos estados do jogo em vários momentos. Para estabelecer um padrão para descrição de um estado de crença, foi proposta uma definição que considerando capturar o máximo de informações possíveis do mundo do jogo num determinado instante de tempo. A descrição é dada por um conjunto v de variáveis de estado que juntas formam um estado completo de jogo s. Um total de $n = 94$ variáveis v foram escolhidas, e são elas:

- $v_1 \in \mathbb{N}$ é o tempo de jogo atual ou tempo de jogo já marcado desde que a partida começou
- $v_2 \in \mathbb{N}$ é o total de unidades produzidas pelo agente jogador
- $v_3 \in \mathbb{N}$ é a quantidade de SCV produzidos pelo agente jogador, essa unidade é responsável por construir recursos e coletar minerais, sendo importante para o desenvolvimento dos recursos do agente
- $v_4 \in \mathbb{N}$ é a quantidade de $Mineral$ colhido pelo agente jogador
- $v_5 \in \mathbb{N}$ é a quantidade de Gas colhido pelo agente jogador
- $v_u \in \mathbb{N}, u \in 6, ..., 17$ é a quantidade acumulada de cada tipo de unidade produzida pelo agente jogador
- $v_e \in \mathbb{N}, e \in 18, ..., 36$ é a quantidade de cada tipo de edificação construída pelo agente jogador
- $v_i \in \mathbb{N}, i \in 37, ..., 81$ é a quantidade de cada tipo de unidade, edificação ou tecnologia inimiga que foi encontrada
- $v_i \in \mathbb{N}, i \in 82, ..., 93$ é a última posição do mapa em que cada tipo de unidade inimiga foi vista pela última vez
- $v_r \in \mathbb{N}, i \in 94$ indica a quantidade de recursos do agente jogador que está sob ataque do inimigo

Com a descrição do estado do jogo, é possível mensurar quando uma batalha foi disputada ou quantos recursos foram destruídos. Durante as transições de estados, basta verificar as quantidades de v_u, v_e, v_i, v_r armazenadas. Com esses valores, é possível efetuar operações simples para obter diferenças entre os estados do jogo a cada diferente instante na partida. Cada variação simples no estado do jogo pode gerar diversos outros estados que são representados pelo estado de crença.

Para funcionamento do módulo de *Decisão*, o POMDP consulta o módulo de $Classificação$ e recebe dados do módulo de *Controle*. Existe uma modelagem específica de processos em torno da tomada de decisão do POMDP. A Figura 34 mostra a modelagem do POMDP dentro da arquitetura de planejamento aqui proposta. Na figura, estão os processos, técnicas e módulos que operam junto com o POMDP para tomada de decisão. O objetivo do POMDP é tomar decisão sobre uma macroação e executá-la toda vez que observações sejam recebidas do ambiente do jogo. Assim que novas observações são recebidas, o processo de tomada de decisão é reiniciado. Em cada processo, existe uma seta saindo dele ou chegando nele, que indica qual tipo de dado esse processo fornece ou requisita. A

ordem em que ocorre a tomada de decisão é indicada pelos algarismos (I, II, II) respectivamente, que estão presentes na parte final do nome em cada seta. As cores, como mostra a legenda, indicam quais módulos enviam dados ao POMDP e vice-versa.

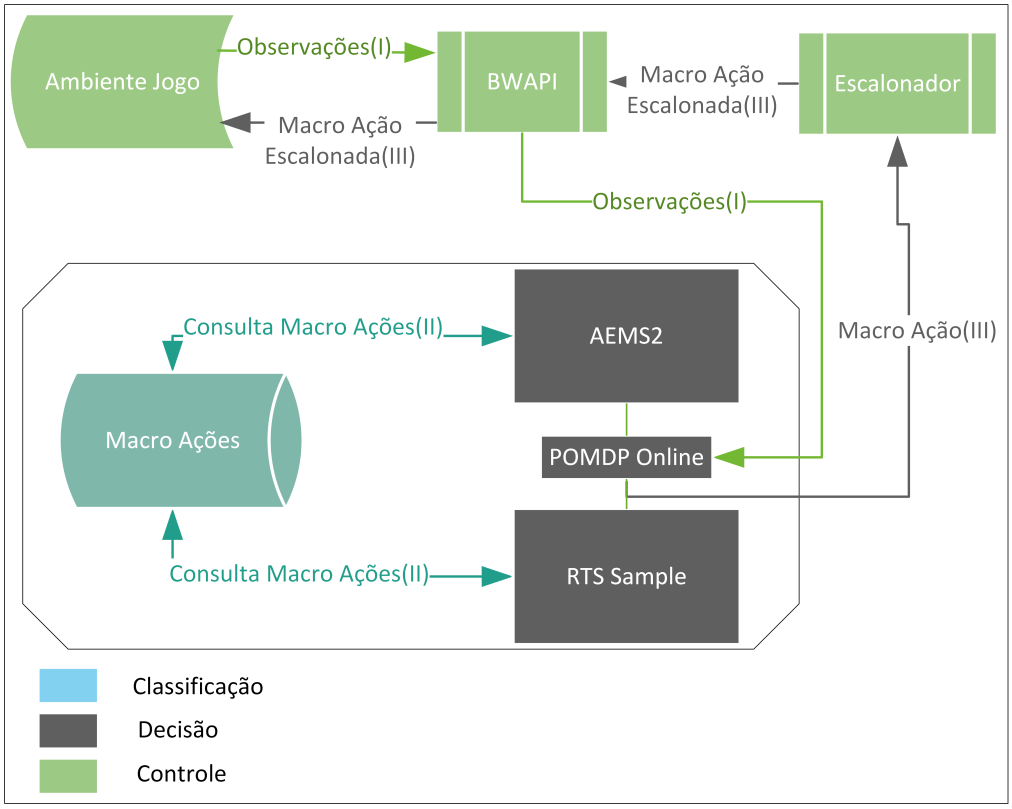

Figura 34 – Modelagem do POMDP *online* para jogos RTS.

De acordo com a Figura 34, o processo de tomada de decisão começa com o *AmbientedoJogo*, pois ele representa o próprio jogo StarCraft em execução e onde a partida está ocorrendo. Os dados sensoriais do jogo, como quantidade de recursos, unidades, informações dos inimigos, são capturados pela BWAPI[1], que é capaz de capturar os dados do jogo e executar comandos dentro do ambiente, sendo a responsável pela comunicação entre a arquitetura proposta e o StarCraft. A BWAPI encaminha os dados do jogo em forma de observações que são armazenadas em um vetor ϱ, onde $\varrho[q] = [0, n]$, q é a quantidade máxima de observações que podem ser recebidas (considerou-se $q = 48$) e n, a quantidade de recursos ou unidades de cada observação. Cada vetor de observações é da forma:

$$\varrho_t = (0, 150, 5, 1, 0, 0..), \varrho_{t+1} = (1, 50, 6, 1, 0, 2..), ... \tag{24}$$

onde t é o tempo de jogo em que o vetor foi capturado. Quando um índice do vetor é construído com valor 0, isso significa que a respectiva observação daquele índice não foi observada. Qualquer outro valor atribuído significa a quantidade total daquela observação atualmente. De acordo com o exemplo na Equação 24, em um primeiro vetor de

[1] https://github.com/bwapi/bwapi

observações ϱ no tempo t nenhuma unidade foi destruída (primeiro índice com valor de 0) e possuía 150$Minerais$ já coletados (segundo índice com valor de 150). Em um tempo $t + 1$ no jogo, foi observado que uma unidade foi destruída e a quantidade de $Minerais$ diminuiu, porém uma nova unidade foi construída (terceiro índice com valor 6) e duas unidades inimigas foram vistas (último índice com valor de 2). Nessa situação, presume-se que a base do agente jogador sofreu um ataque, pois a presença de inimigos foi detectada no mapa e uma unidade do agente foi destruída.

Com as observações sendo entregues ao POMDP pela BWAPI, o processo de decisão é iniciado, o qual é representado na Figura 34 com suas duas políticas de execução AEMS (ROSS; CHAIB-DRAA, 2007) e RTSSample. Foi proposto no desenvolvimento de uma solução para o POMDP o uso de duas políticas, e cada uma delas possui vantagens específicas dentro do domínio de jogos RTS. As políticas serão descritas com detalhes nas próximas seções. As políticas executam a geração de estados de crença por meio do *lookahead*, utilizando as observações, macroações e o estado de crença atual. As macroações estão armazenadas na árvore CHAID, representadas como $MacroAcoes$ na Figura 34. O POMDP faz consultas na CHAID para recuperar as macroações e utilizá-las no *lookahead*. Macroações estão conectadas aos estados de crença com as probabilidades e recompensas calculadas. O cálculo de recompensas é feito de forma diferente em cada uma das políticas, contudo ambas utilizam os seguintes parâmetros para adicionar e remover valores:

- $+1$ para cada recurso qualquer construído
- $+1$ para cada centena de $Minerals$ coletado
- $+3$ para cada centena de Gas coletado
- $+3$ para cada unidade construída
- $+3$ para cada atualização de edificação executada
- $+3$ para cada grupo com mais de quatro unidades reunidas em um local fora da base
- $+4$ para cada unidade do inimigo detectada
- $+4$ para cada SCV construído
- $+5$ para cada unidade inimiga destruída
- -1 para cada minuto sem coletar $Minerals$ ou Gas
- -2 para cada dois minutos sem produzir nenhuma unidade
- -2 para cada SCV ocioso
- -3 para cada edificação do inimigo existente

- ❏ -3 para cada minuto que uma unidade existente não se movimentar no mapa
- ❏ -4 para cada unidade do agente jogador destruída
- ❏ -5 para cada edificação do agente jogador destruída

Os valores para base do cálculo das recompensas foi definido empiricamente, depois de experimentos e testes com diversos valores. As recompensas positivas tendem a manter o agente produzindo unidades de ataque e unidades de *SCV* que aceleram a produção e coleta de minerais. Em relação a inimigos, cada unidade destruída retorna uma recompensa positiva para o agente. Essa recompensa equilibra a estratégia do agente, para que não haja apenas produção de recursos e para que ofensivas contra o inimigo sejam feitas. As penalidades dadas pelas recompensas negativas também direcionam o agente em pontos importantes do jogo. Inicialmente, o agente não deve ficar muito tempo sem produzir novas unidades e coletar recursos, redistribuindo pelo mapa as unidades produzidas. Quando uma unidade do agente é destruída, é importante focar em macroações que executem uma defesa ou contra-ataque. Edificações do inimigo quando são vistas indicam também a posição da base dele. Destruir a base inimiga deixa o agente muito próximo da vitória, assim, o maior valor negativo está atrelado às edificações inimigas.

Em relação à Figura 34, quando a macroação é escolhida pelo POMDP, ele a retorna para o escalonador. Antes desse retorno, com a macroação de estratégia escolhida, a CHAID escolhe uma macroação de *Tática* e uma de *ControleReativo* e as envia ao POMDP. As três macroações são enviadas ao escalonador e a CHAID atualiza seus valores de probabilidade em relação às macroações. O escalonador é responsável por colocar as ações que compõem as macroações em ordem paralela para execução. Elas serão executadas no menor intervalo de tempo possível dentro da partida. O escalonador utilizado e detalhes do processo de escalonamento das macroações serão descritos no próximo capítulo. Após o escalonamento, todas as ações são repassadas de volta para a BWAPI, que executa cada ação dentro do ambiente do jogo e recebe novas observações. O processo de tomada de decisão é reiniciado. O Algoritmo 5 apresenta o POMDP *online* utilizado na arquitetura de planejamento probabilístico.

Nas linhas 1 e 2 do algoritmo 5 estão listadas as variáveis constantes ou com valor obtido via processamento prévio *offline*. G é a árvore de expansão utilizada para gerar estados de crença e fazer *lookahead* no AEMS2. Essa árvore é uma *and-or*. G é enviado como parâmetro para o RTSSample, que utiliza somente o estado de crença representado pela raiz da árvore. A raiz representa o estado atual do jogo. O RTSSample, a partir do estado atual, utiliza uma abordagem baseada em cenários para expandir a árvore e gerar o *lookahead*. D é o limite máximo de profundidade para expansão da árvore G. K é o limite máximo de amostras que podem ser geradas a partir de um ramo dá árvore G. Ambos D e K são utilizado pelo RTSSample. L é o limite inferior e U o limite superior utilizados pelo AEMS2. Esses limites foram obtidos com o processamento de algoritmos

Algoritmo 5 Online POMDP()

```
1: Variáveis Estáticas : G, D, K, b_0, ϱ, amostra;
2: Valores Processados offline : L, U;
3: G ← b_0
4: while FimPartida() ≠ verdadeiro do
5:    amostra ← MudancaModelo(ϱ, amostra)
6:    while TempoDecisao() = verdadeiro do
7:       b1 ← AEMS2(ϱ, G, L, U)
8:       if amostra = verdadeiro then
9:          b2 ← RTSSample(ϱ, G, D, K)
10:      end if
11:   end while
12:   Executa(MelhorMacroAcao(b1, b2))
13:   ϱ ← ObtemObservacoes()
14:   NovaRaiz(G, b1, b2, amostra)
15: end while
```

em modo *offline*. Na linha 3, G recebe o estado inicial nulo, que é um estado neutro com todos os valores de probabilidades iguais. Ele representa o início da partida. Na linha 4 está a repetição central do algoritmo, que dura até o fim da partida.

Na linha 5 do algoritmo 5, a função $MudancaModelo()$ verifica se o modelo do ambiente na partida sofreu mudanças devido a novas observações. Em caso positivo, é habilitado o uso do algoritmo RTSSample. O algoritmo é utilizado sobre condições específicas das observações e do ambiente do jogo. Detalhes serão descritos na Seção 6.3. Na linha 6, está a repetição que é acionada toda vez que novas observações são recebidas e o *lookahead* deve ser executado. Essa repetição dura enquanto a função $TempoDecisao$ for verdadeira. Esse valor é alterado quando novas observações são recebidas ou quando as políticas atingem condição de parada. Na linha 7, o AEMS é executado. Nas linhas de 8 a 10, caso o modelo do ambiente tenha sofridos mudanças, a condição é acionada e o RTSSAmple é executado. Na linha 12, a função $Executa()$ verifica entre as duas políticas qual gerou a macroação com maior recompensa, e a envia ao escalonador. Caso o RTSSample não tenha sido utilizado para gerar estados de crença naquela iteração, o valor de $b1$ é zero. A função $Executa()$ também consulta a CHAID para obter as macroações complementares. Na linha 13, o vetor de observações é atualizado, caso novas observações sejam recebidas, e o processo de tomada de decisão é reiniciado na linha 5. A função $NovaRaiz$, na linha 14, coloca o estado de crença gerado junto com a macroação escolhida como raiz da árvore G. Assim, um novo *lookahead* pode ser iniciado a partir de um novo estado raiz b_0.

As observações do jogo são enviadas ao POMDP sempre que uma nova observação é detectada ou a cada 5 segundos corridos na partida. Nos minutos iniciais do jogo, geralmente são poucas observações recebidas em relação às 16 possíveis no vetor ϱ. Nesses minutos iniciais, os jogadores focam no desenvolvimento de recursos, e quase não existem

confrontos. Já em momentos de confronto, as observações são recebidas geralmente com todos os 16 parâmetros possíveis. Na tomada de decisão, a razão observações por ações é o que determinada a complexidade da geração de estados de crença. Para alcançar máxima eficiência no processo de tomada de decisão dentro do domínio de jogos RTS, foi proposto o uso do AEMS e RTSSample. AEMS é um algoritmo de POMDP que utiliza heurísticas para gerar estados de crença e direcionar a busca por estados promissores. Esse algoritmo representa a política executada continuamente no POMDP. Sua estratégia foi analisada e modificada para este trabalho. É proposto o uso de um aditivo na geração dos estados e cálculo da recompensa utilizando o tempo de jogo da partida como parâmetro.

Mesmo com o AEMS, há situações durante a partida em que uma boa macroação pode não ser encontrada. Isso ocorre devido a mudanças que podem acontecer no modelo do jogo e, consequentemente, no POMDP. Essas mudanças ocorrem quando são iniciados confrontos entre as unidades ou quando é preciso iniciar ações ofensivas e defensivas previamente contra o inimigo. Essas mudanças alteram as observações, o cálculo de estados e geralmente requerem mudança de estratégia do agente jogador. Foi proposta então uma política baseada em amostragem de estados intitulada RTSSample, a qual é especificamente criada para o ambiente de jogos RTS utilizando as características do jogo. O RTSSample não é executado continuamente no POMDP. Sua execução depende da percepção de mudanças no ambiente do jogo e no modelo do POMDP via observações. RTSSample utiliza diferentes estratégias para geração do *lookahead* e tomada de decisão, as quais aceleram sua execução e melhoram a eficiência na escolha de macroações.

Como exemplo de execução do POMDP, suponha-se que a partida esteja em execução e o atual estado de crença seja dado por b_0. O vetor ϱ_t é enviado pela BWAPI ao POMDP no tempo t, com as seguintes observações $(1, 200, 8, 1, 0, 3..)$. Nelas foi indicado que uma unidade do agente jogador foi destruída ($\varrho_t[0] = 1$) e três inimigos foram vistos ($\varrho_t[5] = 3$). Com essas observações, o POMDP verifica se o ambiente sofreu mudanças. Com a destruição de uma unidade e inimigos detectados, existe uma grande probabilidade de confronto, a mudança no domínio do POMDP torna-se verdadeira. Para iniciar a escolha de uma macroação diante dessas observações o POMDP executa o AEMS e, com a mudança confirmada, o RTSSample também. A árvore G começa a ser construída em ambos os algoritmos com b_0 como raiz. Os algoritmos consultam a CHAID e começam a gerar probabilidades e recompensas para execução das macroações de estratégia. O intervalo para executar o *lookahead* é curto, pois aquelas observações chegam em intervalos de poucos segundos, devido à presença de inimigos.

Diante do cenário de exemplo, os algoritmos vão gerar maiores recompensas para macroações que tenham ações de ataque e que produzam recursos de unidades ofensivas. Entre os algoritmos, será escolhida aquela macroação com maior recompensa calculada. Quando a escolha é feita, a CHAID é consultada para a macroação de $Tática$ e $ControleReativo$. Supondo que as macroações conduzam unidades até o local de batalha,

estas são enviadas e o escalonador organiza a execução. O vetor ϱ é atualizado com novas observações, e o processo de escolha é iniciado novamente. O POMDP não precisa escolher uma macroação, seja em qualquer nível, sempre que novas observações são recebidas. Durante a tomada decisão, uma macroação precisa de uma recompensa superior à atual em execução, para ser executada. O algoritmo pode optar por uma macroação nula, que não executa nenhuma estratégia, apenas para gerar o *lookahead*. Nenhuma escolha será feita também para as macroações complementares, caso nenhuma tenha probabilidade maior que aquelas atuais em execução.

A seguir, as duas políticas do POMDP *online* são apresentadas com seus respectivos detalhes.

6.2 AEMS: Política Baseada em Heurísticas

AEMS2 é uma das políticas de execução utilizadas no POMDP *online*. Sua execução é feita de modo contínuo, sempre que novas observações são recebidas. AEMS2 utiliza heurística baseada na minimização do erro, dado o uso de soluções aproximadas *offline*. Essa heurística busca minimizar a diferença entre os limites superior e inferior que surgem à medida que o *lookahead* é feito. A partir de um nível da árvore no *lookahead*, a heurística escolhe apenas um nó, que representa o estado de crença que possui o maior valor esperado de erro, dado o estado raiz b_0. Esse erro é medido com relação à diferença dos valores nos limites superior e inferior. O estado com maior erro gera novos estados que tendem a minimizar o erro em um comportamento aplicado a cada nova geração. O algoritmo foi escolhido por conter diversas propriedades desejáveis para operar no domínio de jogos RTS. A principal é o bom desempenho em ambientes de tempo real com grande quantidade de ações e observações. AEMS2 é o nome dado à versão do algoritmo que utiliza uma simplificação na estimativa da probabilidade de uma ação ser ótima $P(a|b)$. Essa probabilidade é igual ao limite superior da ação, quando o *lookahead* de estados de crença é executado. No AEMS, o valor da função $P()$ é utilizado no valor da heurística com diferentes parâmetros, e isso será descrito no decorrer desta seção. Uma mudança na forma de cálculo da heurística é proposta e também será discutida. Por conveniência, o algoritmo será mencionado apenas como AEMS.

Antes de utilizar o algoritmo, é feito um processamento do POMDP de modo *offline*, utilizando dois algoritmos. O objetivo é definir o limite superior e o limite inferior de cada possível combinação de estado de crença e macroação. Ambos os limites são utilizados no cálculo da heurística do AEMS. Esses limites estabelecem uma margem do menor e maior valor de recompensa já esperados para estados e macroações. O algoritmo então tenta obter resultados que superem o limite superior e que não sejam menores do que o limite inferior. Além disso, os limites são utilizados como base para efetuar poda sobre estados não promissores. Os limites são calculados de modo *offline*, com os *replays*

utilizados como base para a execução do POMDP. Utilizando políticas específicas, são obtidos valores prévios de recompensa e probabilidades. As partidas são simuladas com partidas entre raças específicas, como *Terran x Protoss* ou *Terran x Zerg*. Os limites são definidos para cada tipo de confronto.

O limite inferior é obtido com execução de uma política cega, a qual apenas repete a mesma macroação independentemente de em qual estado de crença o algoritmo esteja. O valor função que reflete a recompensa com essa política é o menor valor possível que pode ser obtido. A política cega é executada por meio da Equação 25. Em cenários simulados de partidas do jogo, a política calcula exatamente a recompensa mínima esperada para cada tomada de decisão. Para o limite superior, foi utilizado o algoritmo *Fast Informed Bound* (FIB) (SMITH; SIMMONS, 2012), que não leva em conta a total incerteza do ambiente. A incerteza é considerada até um determinado grau ajustado no algoritmo. Esse grau deve ser necessário apenas para calcular os limites dos estados de crença com informações baseadas em diferentes macroações. O algoritmo tem menor complexidade de execução e consegue estimar com melhor probabilidade o atual estado em que o ambiente se encontra.

$$\alpha^a_{t+1}(s) = R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')\alpha^a_t(s) \tag{25}$$

$$\alpha^a_{t+1}(s) = R(s,a) + \gamma \sum_{o \in O} \max_{\alpha_t \in \Gamma} \sum_{s' \in S} O(s',a,o)T(s,a,s')\alpha^a_t(s') \tag{26}$$

Com o FIB, o limite superior possui um valor consistente que direciona a heurística para estados mais promissores, sem estabelecer um alto valor que possa tendenciar a busca. A atualização dos estados de crença pela FIB é executada com a Equação 26, em que o operador α representa os limites. O limite de profundidade para geração de estados de crença foi configurado entre 10 e 5 segundos. Esse valor é o intervalo disponível para o cálculo, a fim de garantir um bom limite superior. Outros algoritmos que alcançam melhores resultados poderiam ser utilizados no cálculos dos limites, por exemplo, um POMDP com política de ponto de atualização como Sarsop (KURNIAWATI; HSU; LEE, 2008). Contudo, algoritmos com melhor resultado tendem a obter limites apertados entre os valores mínimo e máximo. Esses valores não são vantajosos para o cálculo heurístico, que pode não explorar porções distantes do espaço de estados.

Os limites superior e inferior são utilizados no cálculo da heurística de erro máximo do AEMS. Essa heurística é responsável por determinar qual estado de crença é escolhido em cada nível da árvore. A Equação 27 descreve essa heurística. Nessa equação: γ^d é um fator de desconto; $P(b^d)$ é a probabilidade de uma macroação tornar-se a melhor; $epsilon(b^d)$ é a diferença entre o limite superior e inferior de cada estado que pode ser gerado na árvore. Todos os valores da heurística são calculados para cada nível d da árvore. Em cada nível gerado durante o *lookahead*, deve ser escolhido sempre o nó do ramo que tenha o maior

valor heurístico. Escolher sempre esse nó faz com que o algoritmo busque sempre a maior recompensa em cada macroação gerada. Ao final, o algoritmo considera o ramo da árvore máximo que possui maior recompensa total.

$$E(b^d) = \gamma^d P(b^d)\epsilon(b^d) \tag{27}$$

No AEMS, quando um *lookahead* é feito, primeiramente o melhor estado de crença na borda da árvore é escolhido para expansão, de acordo com seu valor de heurística. Em seguida, a árvore é expandida sob o estado de crença escolhido. Por último, os nós ancestrais do estado de crença que foi expandido têm seus limites atualizados. Para a expansão e atualização dos estados de crença, os valores dos limites são importantes. Os limites são mantidos na árvore durante o *lookahead* para todos os nós, ou seja, estados de crença e macroações. Os limites dos nós da borda representados por $L(b)$ (limite inferior) e $U(b)$ (limite superior) são aqueles computados *offline*. Eles são utilizados para calcular os limites de cada macroação m por cada estado de crença b que representa o *Q-value* desse conjunto, dado por $Q*(b,m)$. Todos esses valores são então utilizados para obter os limites de cada estado de crença que representa o melhor limite possível $V*(b)$ naquele nível da árvore. O melhor limite possível é na verdade o maior $Q*(b,m)$ obtido, considerando todos os estados de crença pai e suas macroações filhos. Os valores dos limites dos conjuntos de *Q-value* são propagados para os estados de crença pai da árvore, usando as Equações 28 a 29, nas quais $F(G)$ é o conjunto dos nós de borda da árvore G; $L_t(b)$ e $U_t(b)$ representam os limites inferior e superior $V*(b)$ que são atribuídos a cada estado de crença; $L_t(b,m)$ e $U_t(b,m)$ são os valores de limites calculados para cada macroação m e estado de crença b dado pelo $Q*(b,m)$. O valor de $Q*(b,m)$ é calculado utilizando a equação de retorno de recompensa ou Equação 14 de Bellman (PENG, 1992). Assim, o valor de limite $L_t(b), U_t(b)$ atribuído a cada estado de crença b é igual ao limite obtido *offline* $L(b), U(b)$, caso o b seja um nó de borda, ou é igual ao maior limite $Q*(b,m)$ dado por $L_t(b,m), U_t(b,m)$, que utiliza a equação de cálculo da recompensa com cada macroação e estado de crença daquele nível da árvore.

$$L_t(b) = \begin{cases} L(b) & \text{se b} \in F(G) \\ max_{m \in M} L_t(b,m) & \text{senão} \end{cases} \tag{28}$$

$$L_t(b,m) = R_B(b,a) + \gamma \sum_{o \in O} PR(o|b,m) L_t(\pi(b,m,o)) \tag{29}$$

$$U_t(b) = \begin{cases} UL(b) & seb \in F(G) \\ max_{m \in M} U_t(b,m) & \text{senão} \end{cases} \tag{30}$$

$$U_t(b,m) = R_B(b,a) + \gamma \sum_{o \in O} PR(o|b,m) U_t(\pi(b,m,o)) \tag{31}$$

Considere um *lookahead* que gere dois níveis na árvore utilizando duas macroações e duas observações. Ele calculará recompensas à frente do tempo atual da partida. Suponha-se que as observações recebidas indiquem que nenhum inimigo foi visto e a produção de unidades cresceu. No primeiro nível da árvore, o estado atual será combinado com macroações diversas, como mostra a Figura 35. Nessa figura, tem-se: os nós no formato triangular representam estados de crença; os nós em formato circular especificam as macroações; valores nos arcos dos estados são recompensas; valores nos arcos das macroações são as probabilidades de atingir o estado; os valores em colchetes são os limites. Na primeira iteração da Figura 35, o único nó existente é o atual estado de crença que possui como limites valores $[10, 20]$, padrões de início do algoritmo. Logo em seguida, as macroações são utilizadas e os limites do estado de crença b_0 são recalculados, utilizando as Equações de 28 a 29. A macroação m_1 provavelmente possui características de produção de recursos equilibrada, de acordo com os limites e observações recebidas. Na segunda iteração, é preciso calcular a heurística para escolher o estado de crença que será expandido. Os estados de crença são estimados com base na macroação m_1 escolhida anteriormente. A Figura 36 mostra os valores que são obtidos com o cálculo da heurística e atribuídos aos estados de crença. Os detalhes do cálculo da heurística serão descritos no decorrer da seção. O valor heurístico é obtido utilizando a diferença no erro dos limites e a probabilidade da melhor macroação, dada pela Equação 27. Uma vez escolhido o estado de crença, sua expansão é feita com testes que utilizam macroações e as novas observações. A macroação m_2 possui limites superiores ao estado b_1 expandido. Assim, o estado tem seus limites atualizados com base nos limites da macroação e nos valores da heurística calculados anteriormente.

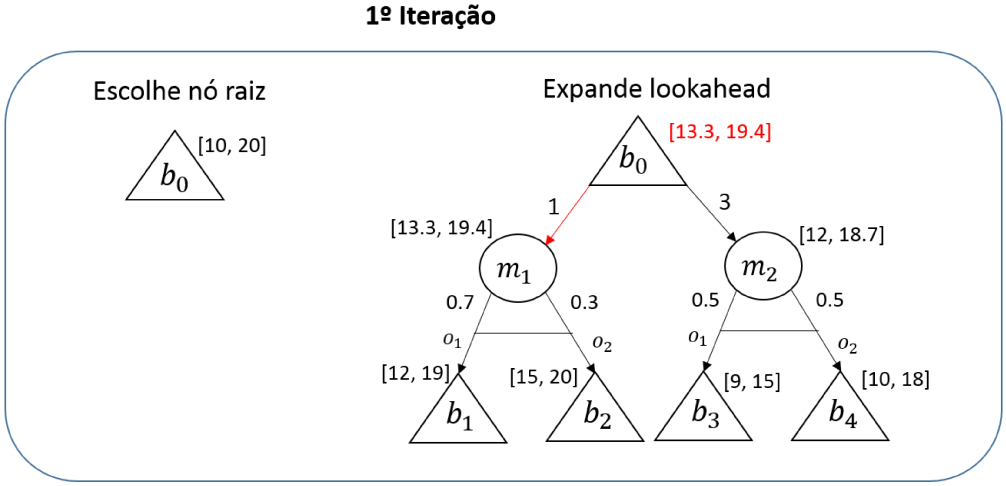

Figura 35 – 1º iteração do AEMS na execução do *lookahead* e cálculo das heurísticas.

A atualização iniciada com a macroação m_2 é propagada para os níveis superiores da árvore até a raiz, como mostra a Figura 37. A atualização deixa os limites com valores mais distantes em relação ao valor superior e inferior. Essa diferença nos limites cria novas

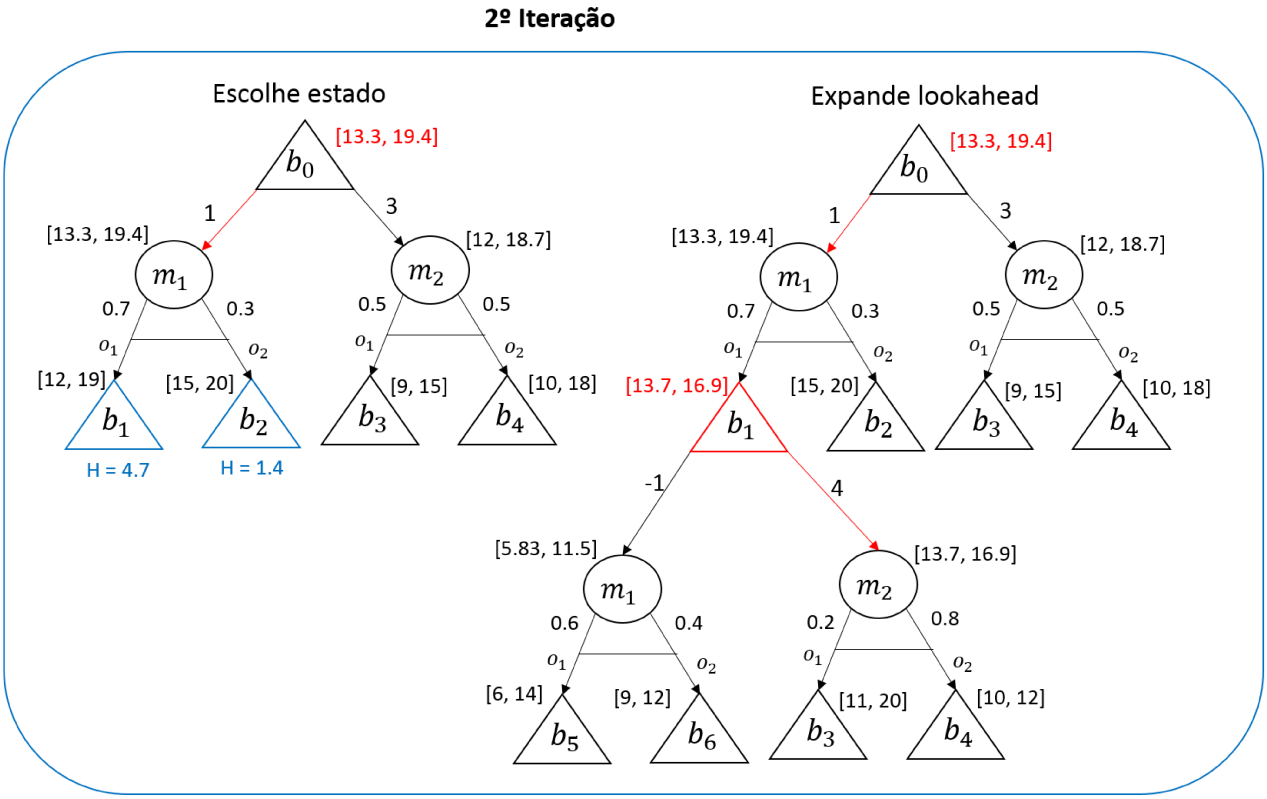

Figura 36 – 2º iteração do AEMS na execução do *lookahead* e cálculo das heurísticas.

possibilidades de expansão dos estados de crença por meio de maior exploração do espaço de estados. A segunda iteração termina com a atualização do melhor ramo da árvore até sua raiz, com isso a terceira iteração é iniciada, e recomeça o processo de cálculo da heurística, já que existe uma nova decisão em relação ao estado que pode maximizar o erro. Esse estado leva até um ramo com melhor recompensa total, como mostra a Figura 38. O processo da AEMS sempre inclui a atualização dos limites para que a tomada de decisão siga pelo ramo que utiliza as macroações mais promissoras. Essa estratégia de fato funciona como uma espécie de poda de estados. Com base nos limites e no cálculo do erro, nenhum estado que não seja o melhor ou não tenha um valor aproximado do melhor é escolhido. Em cada nível da árvore, apenas um estado ou macroação gera novos sucessores, e não é feita nenhuma expansão para outros nós naquele mesmo nível da árvore. A tendência é que esses estados promissores com bons valores gerados pela heurística do AEMS conduzam à escolha de macroações promissoras também.

Com o processo de *lookahead* do AEMS descrito, será feito a seguir um detalhamento da Equação 27. O primeiro elemento é o fator de desconto γ responsável por garantir que o *lookahead* seja feito dentro de um horizonte finito de estados. O valor foi definido empiricamente em 0.95, que é suficiente para garantir um retorno do AEMS até em situações em que o intervalo de novas observações está abaixo de 2 segundos. O segundo elemento é o erro local $\epsilon(b)$ descrito na Equação 32, na qual $U(b)$ é o limite superior e

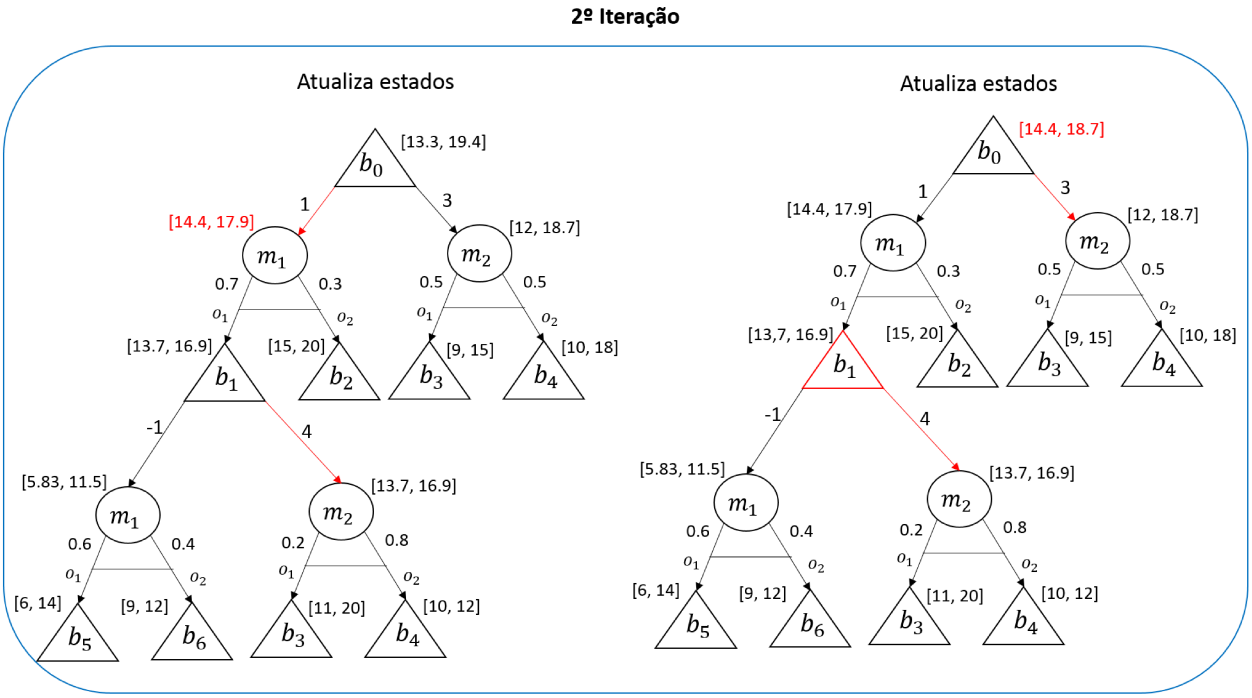

Figura 37 – 2º iteração do AEMS na atualização dos limites.

$L(b)$, o limite inferior. O erro local $\epsilon(b)$ captura a diferença na precisão dos limites, uma vez que o limite inferior é calculado com uma equação com retorno mínimo, e não com uma função valor exata. Essa metodologia gera valores que distorcem o resultado das recompensas, dado o limite superior do estado. O erro então é utilizado para substituir o limite inferior no cálculo da heurística.

$$\epsilon(b) = U(b) - L(b) \tag{32}$$

O terceiro elemento no cálculo da heurística é $P(b^d)$, que representa a probabilidade de um nó folha na árvore expandir um ramo com macroações ótimas. Para obter $P(b^d)$, é preciso inicialmente utilizar uma função objetivo aproximada. Essa função calcula a probabilidade de uma macroação ser a melhor em determinado nível da árvore. AEMS utiliza uma definição binária para essa escolha, dada por $P(a|b)$, na qual o valor 1 é atribuído, caso a ação tenha o maior limite superior daquele nível da árvore no *look-ahead*. Caso a ação não tenha o maior valor daquele nível da árvore, seu valor será 0. Foi proposta uma alteração a essa definição, utilizando o tempo do jogo, que é cronometrado durante a partida. Utilizar apenas o limite superior na estimativa da probabilidade faz com que macroações normalmente executadas em determinado tempo do jogo sejam escolhidas em tempos que outras macroações seriam melhores. Isso ocorre porque as ações do jogo disponíveis para formar as macroações estão presentes em diversas e diferentes macroações. O cálculo de recompensa pode ser próximo ou maior para macroações com

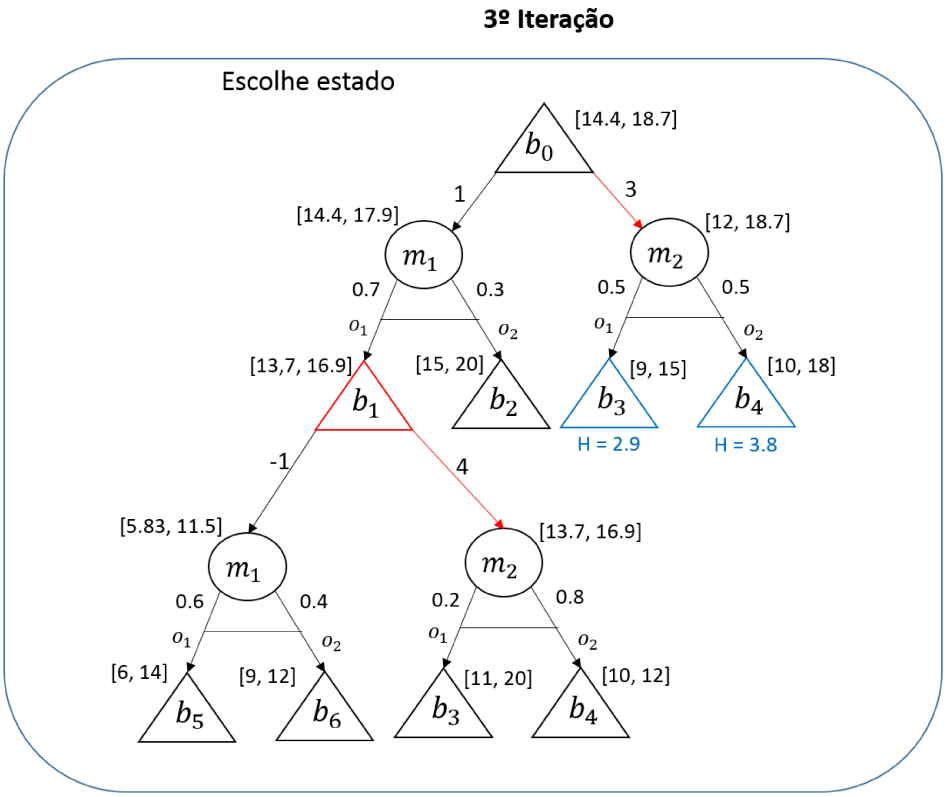

Figura 38 – 3º iteração do AEMS no cálculo da heurística.

ações individuais iguais. A Equação 33 mostra a definição proposta:

$$P(a|b) = \begin{cases} 0 & \text{se } U(m',b) < (minorB) \\ Ulr(m',b) & \text{senão} \end{cases} \tag{33}$$

Na Equação 33, tem-se: $U(m',b)$ representa o limite superior no valor da ação m'; $minorB$ é uma variável que armazena o maior limite inferior encontrado a cada estado de crença gerado; $Ulr(m',b) = U(m',b) * Lr(m',b)$, onde $Lr(m',b)$ é a razão da macroação utilizada na atualização do estado de crença mais o tempo do jogo atual obtido na partida. Essa razão é dada pela estimativa de uso da macroação pelo atual tempo de jogo. Essa estimativa dada por Lr foi obtida com uma regressão linear de quadrados mínimos (NETER; KUTNER; NACHTSHEIM, 1996), em que o tempo do jogo (x) é a variável independente e as macroações representam variáveis dependentes (y). Foram utilizados os vetores de característica definidos na Seção 5.1 e que possuíam dados sobre os recursos e o tempo do jogo em que eles foram colocados em produção. Os dados dos vetores são transformados utilizando a Equação 34, que simula o estado do jogo (com base no tempo x) anterior à produção de cada macroação y. Assim, um vetor de características $c = (0, 20, 55, 1 : 24)$ corresponde ao tempo de execução de três macroações diferentes. Uma transformação para uma macroação específica $y = TwoSaquads = 1 : 24$ resultaria no vetor de características $c = (0, 20, 0, 1 : 24)$, que representa o tempo de início e fim utilizado para produzir a macroação $TwoSaquads$. Com isso, a estimativa de uma

macroação ser executada em um determinado intervalo de jogo foi medida. Sempre que uma macroação for escolhida, a sua estimativa com o atual tempo do jogo vai influenciar a probabilidade de ela ser a melhor macroação daquele nível da árvore. Quanto maior for o valor esperado da macroação, maior será o valor de $Lr(m', b)$.

$$c(x, y) = \begin{cases} c(x) : y \neq x, c(x) < c(y) \\ 0 \qquad \text{senão} \end{cases} \tag{34}$$

Os conjuntos de treinamento e teste para a regressão linear foram os *replays*, utilizando as macroações já classificadas e o tempo de jogo. A regressão foi executada de modo *offline*, com 100 iterações sobre a base de dados com taxa de encolhimento de 0.5. Assim, o limite superior é utilizado juntamente com a pertinência da macroação e o tempo do jogo atual, na escolha da melhor macroação. Tomando um exemplo, macroações com alto valor de limite superior mas com uso frequente apenas no início do jogo terão seu valor compensado quando consideradas em uma partida com alguns minutos já jogados. Quando $Ulr(m', b)$ é calculado, uma macroação pode no máximo aumentar seu limite em 30% de seu valor original, utilizando o retorno esperado da regressão linear. Assim, a probabilidade retornada pela regressão não terá maior responsabilidade na escolha do que os limites de cada estado de crença.

Com o valor de $P(a|b)$ definido, o AEMS pode calcular a probabilidade de alcançar um estado de crença. Esse estado é um nó folha em uma profundidade específica, dado por $P(b^d)$. Essa definição utiliza a probabilidade de uma macroação ser considerada ótima em cada nível da árvore. Isso indica as chances de um estado de crença ser atingido através do *lookahead* em uma profundidade d. O cálculo da probabilidade de $P(b^d)$ é dado pela Equação 35. Nessa equação, o^i, a^i e b^i denotam a observação, macroação e estado de crença encontrado na profundidade i. Esse caminho leva ao estado de crença b^d na profundidade d. O *lookahead* é executado até um horizonte finito de tempo. A árvore será expandida com simulações envolvendo estados de crença e macroações, até uma profundidade d. Com a probabilidade $P(b^d)$ de alcançar o nó folha na profundidade d, é possível calcular o erro estimado. Essa estimativa começa pelo nó folha do ramo e propaga o valor estimado, ajustando o cálculo dos estados de crença na volta pela árvore até o nó raiz.

$$P(b^d) = \prod_{i=0}^{d-1} P(o^i|b^i, a^i)P(a^i|b^i) \tag{35}$$

Com todos os elementos necessários, a heurística para cálculo do erro em estados de crença pode ser feita. Dentre as propriedades desejáveis da heurística $E(b^d)$ para o domínio de jogos RTS, é possível destacar: favorece estados de crença com maiores diferenças nos valores dos limites, os quais normalmente fornecem caminhos na árvore que levam a estados futuros com maior recompensa por explorar partes diferentes do espaço de estados; e favorece a exploração de estados de crença com maior probabilidade

de ser encontrados no futuro, evitando retrabalho. A definição de $P(a|b)$ aumenta a performance do algoritmo. Com a alteração proposta, é possível também considerar mais macroações com bons limites superiores e pertinência com o tempo do jogo atual.

6.3 RTSSample: Política para o Domínio de Jogos RTS

O RTSSample foi proposto para gerar uma política de tomada de decisão para o POMDP *online* aplicado ao problema de jogos RTS. O algoritmo utiliza técnicas que adaptam sua performance para as características do ambiente de jogos. O algoritmo concentra-se em explorar estados e macroações que podem não ser considerados pelo AEMS. Isso ocorre devido às dinâmicas dos jogos RTS e às mudanças no modelo do ambiente durante a partida. A estratégia principal do algoritmo é o uso de amostragem (DOSHI; GMYTRASIEWICZ, 2009) na geração dos estados de crença, que são gerados a partir de amostras finitas e menores dos estados já existentes, combinados com partes das observações.

RTSSample utiliza o conceito de cenários para representar cada estado de crença. Nessa abordagem, a distribuição de probabilidade não é utilizada. Um cenário $e \in E$ é uma tupla $\langle s, m, o, r, c(e)\rangle$ onde: r é a recompensa imediata do estado; e $c(e)$ é um contador que marca quantas vezes o cenário e foi visitado ou selecionado para cada macroação a ($c(e) \sum_m c(e, m)$). As demais variáveis são estado do jogo (s), macroação (m), observações (o). A árvore de expansão e execução do *lookahead* representa a aplicação da política do algoritmo gerando amostras de cenários armazenadas em nós de árvore. A representação da expansão e geração de cenários é feita por meio do par formado por macroação e observação. Essa definição ainda mantém a geração de estados sobre a estrutura de uma árvore, porém o conceito é distinto daquele utilizado no AEMS. Cada nó contém uma amostragem de cenários com geração de diferentes estados do jogo, macroações e recompensas em cada um deles.

A árvore de expansão G do RTSSample mantém um conjunto E de cenários amostrados. A rastreabilidade dos cenários e uso das macroações são identificados como uma única representação na árvore através dos pares. Os pares auxiliam a identificação de mudanças no modelo do POMDP. Quando novas observações são obtidas da partida, basta rastrear o nó com a amostragem de cenários utilizados. Esse rastreio é feito por meio do par que contém a última macroação e as observações utilizadas. O atual cenário em execução na partida é utilizado como nó raiz para uma nova amostragem, a qual escolherá uma nova macroação, de acordo com as mudanças no modelo do jogo e do POMDP. A Figura 39 mostra como é a representação dos cenários na árvore G com amostragem de cenários efetuando o *lookahead*.

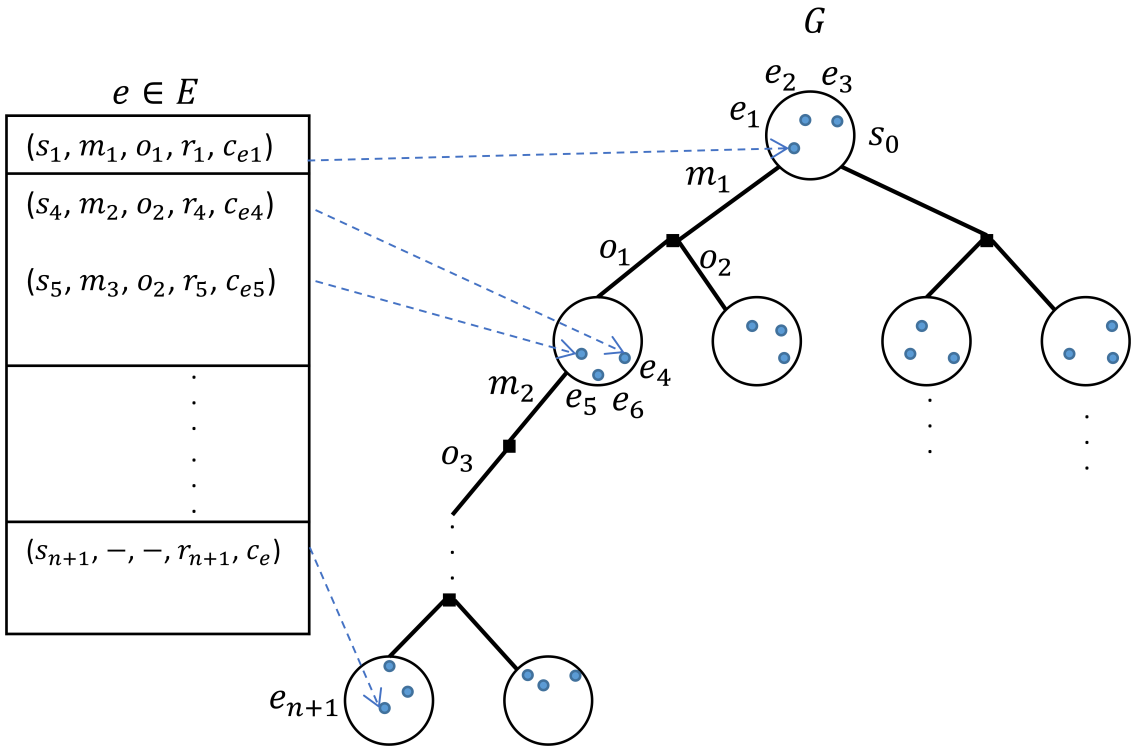

Figura 39 – Processo de amostragem com criação dos pares de cenários e macroações.

As amostras dos cenários são representadas por meio de partículas que ficam armazenadas dentro dos nós da árvore na Figura 39. As partículas são geradas com um filtro, utilizando um modelo generativo. Esse modelo funciona como uma função genérica que utiliza o estado do jogo atual s_0 e uma macroação m. Em seguida, retorna uma amostra dos próximos cenários com o estado s_1, uma observação utilizada o_1 e uma recompensa r_1. A Figura 39 mostra todos os elementos presentes na árvore de expansão. Na figura, o nó raiz s_0 é o estado atual da partida e inicia a expansão. Cada nó da árvore G representa uma crença e contém uma amostragem de vários cenários possíveis. Ambos os cenários e_5 e e_6 estão no mesmo nó crença e são duas partículas distintas, porém ambas utilizam a observação o_2 advinda da partida, com duas macroações de *Estratégia* diferentes, gerando duas recompensas distintas r. As amostragens em cada nó acompanham o padrão de observações e macroações da árvore G. Os nós crença e suas respectivas partículas formam caminhos pela árvore que são identificados pelos pares macroação e observação. Assim, os caminhos são gerados naturalmente quando o modelo generativo que amostra os cenários é utilizado.

O modelo generativo é modelado como $g : SxMxR \rightarrow SxO$. É utilizada uma sequência de macroações (m_1, m_2, am_3) sob uma amostragem de cenários $(s_0, o_1..)$ onde $s0 \in S$, $o_n \in O$ para $n \in [0, N]$. Então, um caminho é obtido $(s_0, m_0, o_0, r_0, c_0, ..., s_{n+1}, m_n, o_n, r_n, c_n,)$, o qual é adicionado à árvore a partir do nó raiz s_0. Esse nó contém várias partículas representadas por cenários conectados, esse processo de amostragem é rápido e não exige

um modelo específico do POMDP. No modelo proposto, $(s', o') = g(s, a, \varphi)$ é a representação para $p(s', o'|s, a) = T(s, a, s')O(s', a, o')$, onde φ é uma distribuição entre $[0, 1]$. Sem as funções T e Z presentes no modelo tradicional do POMD, o RTSSample não utiliza distribuição de probabilidade. Os cenários são representados pelos caminhos entre as partículas. Assim, a atualização dos estados consegue um extenso *lookahead* e explora diferentes partes do espaço de estados. Na Figura 39, o ramo mais à esquerda da árvore simula o caminho de um *lookahead*, partindo do nó raiz s_0 até o nó folha s_n, que contém o cenário e_{n+1}.

Os conceitos sobre expansão e uso dos cenários pelo RTSSample serão utilizados como base para a descrição da política utilizada no algoritmo proposto. Assim como no AEMS, a política do RTSSample tenta maximizar a recompensa total na escolha das macroações. A política $\pi(s)$ do RTSSample pode ser sumarizada na Equação 36. Nessa equação: s é estado atual; M o conjunto de macroações; E os cenários representando os pares formados na árvore G; $Q(s, m)$ representa a função *Q-value* do POMDP clássico, que retorna o valor de executar uma ação ótima a partir de um estado e continuar fazendo isso repetidamente. A estimativa da função para o uso de cenários e amostragem é dada pela Equação 37. Nessa equação, $E_{s,m} \subseteq E$ é o conjunto de todos os cenários associados e todos os caminhos na árvore G, que começam em s_0 e contêm a sequência (s, m). A profundidade do estado s na árvore G é dada por l.

$$\pi(b) = argmax \underset{(m \in M(E,s))}{Q(s, m)} \tag{36}$$

$$Q(s, m) = \frac{1}{|E_{s,m}|} \sum_{e \in E} V(e, l) \tag{37}$$

$V(e, l)$ é o valor de um cenário e, que começa a partir do elemento na profundidade l. O valor de $V(e, l)$ é calculado utilizando a Equação 38, em que γ é o mesmo valor de desconto do AEMS (0.95) e R é a função para cálculo da recompensa. Com a política $Q(s, m)$, é preciso calcular uma boa recompensa R e garantir uma boa amostragem em $|E_{s,m}|$. Esses valores são os mais importantes para a função maximizar as decisões ao longo dos caminhos da árvore. Com uma função apropriada para escolha da melhor macroação em cada conjunto de cenários, é garantido que os valores da política sempre melhorem. Com a escolha adequada, as amostragens em novos nós aumentarão, devido à chegada de mais observações com o passar do jogo. Com esse aumento, escolher sempre uma ação, a melhor macroação maximiza as recompensas acumuladas.

$$V(e, l) = \sum_{i=l}^{|e|} \gamma R(e_i.s, e_i.m) \tag{38}$$

Para a escolha da melhor macroação, a cada nível da árvore foi escolhido o algoritmo *The Upper Confidence Bound* (UCB) (AUER; BIANCHI; FISCHER, 2002). Esse algoritmo considera cada nível da árvore como um *multi-arm bandit problem*. O UCB

calcula a recompensa de cada macroação em seu respectivo cenário e escolhe a sequência de macroações que maximize a recompensa total. A equação 39 mostra o UCB. $V(e,m) = \frac{Ulr(m,e)}{c(e,m)}$ é o valor obtido com simulações em um cenário e, junto com o número de vezes que este foi visitado. Os valores de $Url(e,m)$ são obtidos através de computação *offline*, assim como no AEMS. A diferença é que a política escolhida para simular partidas do jogo foi o Rollout (ROSS et al., 2008), pois obtém valores acima das políticas básicas utilizadas no AEMS. O RTSSample não precisa de limites esparsos, já que sua execução é sob condições específicas do jogo. A aproximação proposta com $V(e,m)$ apresenta bons resultados no domínio de jogos RTS, e é feita com uma estratégia de amostragem de cenários. Os valores são obtidos utilizando o conceito de pertinência no tempo do jogo para escolha de uma macroação. Esse conceito é o uso dos valores da regressão linear pelo tempo do jogo, feito na seção anterior com o AEMS. A constante c é a proporção de exploração e uso utilizada pelo UCB, e foi determinada empiricamente em 0.7. A quantidade máxima de cenários amostrados K e profundidade máxima da árvore G foram definidos como 40 e 8, respectivamente. Todas essas constantes são utilizadas no algoritmo 5. O RTSSample e Rollout também utilizam essas constantes.

$$argmax_m \quad V^*(e,m) = V(e,m) + c\sqrt{\frac{log\ c(e)}{log\ c(e,m)}} \tag{39}$$

Na arquitetura do POMDP *online*, o RTSSample é utilizado quando o ambiente do POMDP sofre alguma mudança. Essa mudança detectada é a condição necessária para execução do algoritmo. Para identificar uma mudança, foram propostas duas condições, que se baseiam no fato de jogadores experientes apresentarem comportamentos muito distintos durante as duas fases existentes em uma partida. A primeira fase é a produção recursos e desenvolvimento de unidades, o que aumenta os recursos do próprio jogador. Na segunda, é necessário controlar as unidades em batalhas e executar ações de ataque e defesa. Essas duas fases são dinâmicas e alteram-se constantemente durante a partida e entre os jogadores. O ambiente do jogo muda quando há confronto entre unidades ou mesmo um princípio de possível confronto. Com isso, a primeira condição para detectar uma mudança no ambiente do POMDP é verificar se nas observações recebidas $\exists o_i | [o \in O, i \in \{|O/2|, ..., |O|\}, o_i \geq limit]$, ou seja, se existe alguma observação de confronto/ataque que está com valor maior ou igual ao limite. Essa condição verifica, entre as observações que refletem combate, a presença de inimigos ou quantidade de unidades destruídas com valores acima do limite são exemplos de indicadores de combate. Existe um total de 9 observações que indicam confrontos, e elas ficam na metade final do vetor de observações ($O/2$). O limite é uma constante que indica se existe a possibilidade de início ou andamento de confrontos. O valor do limite foi definido empiricamente em 3.

A segunda condição para detectar mudanças no ambiente é quando a primeira política o AEMS escolhe uma macroação ofensiva. O POMDP percebe que a macroação é ofensiva

a partir da identificação do par estado de crença e macroação. Com uma macroação ofensiva sendo executada, o ambiente do POMDP é alterado. O nó crença que estabelece um par com a macroação é passado como nó raiz para inicializar a árvore do RTSSample. A execução do RTSSample pode ser mantida em novas iterações do POMDP *online*, caso uma das duas condições a seguir sejam satisfeitas:

1. para a primeira condição, as observações recebidas anteriormente continuam com valor acima do limite ou novas observações de confronto são percebidas.

2. a segunda condição é quando ocorre a escolha de uma nova macroação de confronto/ataque. Todas essas condições são verificadas pela função $ModelChange()$ do algoritmo 5.

Foi definido que uma mudança no ambiente do POMDP em jogos RTS pode ser percebida pela existência de confrontos diretos com o inimigo. Os confrontos podem acontecer a qualquer momento e várias vezes durante uma partida. Para responder a esses tipos de mudanças, geralmente os algoritmos computam novamente toda a solução ou fazem buscas para encontrar estados afetados. Essa estratégia pode sofrer problemas de performance em ambientes de tempo real ou encontrar soluções com baixas recompensas, devido às restrições de tempo. Computar os mesmos estados ou tentar encontrar estados específicos gera retrabalho. A abordagem proposta com o RTSSample considera a geração de novos estados com base na nova dinâmica do ambiente. Quando as políticas são executadas em paralelo, a macroação com maior recompensa entre os algoritmos é escolhida para execução. O algoritmo 5 apresenta o RTSSample.

Algoritmo 6 RTSSample(ϱ, G, D, K)

```
1: G' ← ConstroiArvore(G)
2: if D = 0 or Vazio(G) then
3:    Return erro
4: end if
5: while !TempoLimite(ϱ, D, G') do
6:    for each MacroAction a ∈ A do
7:       a* ← UCB_Acao(G', b, ϱ)
8:    end for
9:    for i = 1 to K do
10:      e[i] ← Generativo(a*, b, ϱ)
11:   end for
12:   AtualizaValores(e[], ϱ)
13:   InstaParticulas(G', e[])
14: end while
15: a* ← argmax  V(e[], a*)
16: return a*
```

A função $ConstroiArvore()$ na linha 1 recebe a árvore utilizada pelo AEMS e a reconstrói na variável G'. Para isso, é configurado como nó raiz o estado crença que forma

par com a macroação executada em uma iteração anterior do POMDP. Essa macroação foi escolhida pelo AEMS. Caso a última macroação tenha sido escolhida pelo próprio RTSSample e sua execução continue, essa macroação será configurada como raiz para o RTSSample e para o AEMS. A função $TempoLimite()$ na linha 5 controla o tempo disponível para o algoritmo retornar uma macroação. Ela utiliza o tempo restante em segundos do jogo para a macroação em execução ser finalizada. Caso alguma das condições de confronto seja captada na partida ou a profundidade máxima para expansão da árvore D seja atingida, a função finaliza o tempo, e o algoritmo faz um retorno imediato.

Nas linhas 9 e 10 do algoritmo, o modelo generativo gera partículas de cenários, as quais são geradas dentro do limite de amostras definido por K em cada conjunto de macroação e estado. As partículas geradas são armazenadas no vetor de cenários $e[]$. A função $AtualizaValores()$ na linha 12 atualiza parâmetros dos cenários gerados anteriormente. Para cada cenário e_i a função atualiza o contador $c(e)$, caso o cenário tenha sido visitado novamente. Ela também atualiza o valor de recompensa e verifica quais observações de confronto do conjunto ϱ não foram amostradas ainda. Aquelas não amostradas são realocadas no vetor para ser utilizadas nas próximas iterações. Na linha 13, a função $InstaParticulas()$ insere os novos cenários e suas respectivas partículas na árvore G'. O processo é feito inserindo os conjuntos de partículas nas ramificações com nós que representem o par estado de crença e macroação. Por fim, na linha 15 depois que o processo de amostragem da árvore foi finalizado, é feita a escolha da macroação que está no ramo que alcançou a maior recompensa total, e esta é retornada ao POMDP *online* para verificar qual macroação escolhida entre as duas políticas de execução possui maior recompensa.

6.4 Considerações Finais

Neste capítulo, descreveram-se o módulo *Decisão* e seus algoritmos utilizados para tomada de decisão na arquitetura proposta. Esse módulo exerce papel importante na arquitetura, uma vez que esta executa tarefas do jogo por meio da execução de macroações. Decidir com qualidade e eficiência quais macroações serão executadas garante o sucesso do agente em garantir uma vitória na partida. O POMDP *online* foi utilizado e duas políticas foram incorporadas a sua execução. AEMS é uma boa política *online* que foi modificada para aumentar a qualidade de seu resultado no domínio de jogos RTS. A RTSSample foi proposta exclusivamente para este trabalho, e possui estratégias que utilizam a dinâmica dos jogos RTS para aumentar a qualidade na tomada de decisão e obter melhor performance.

A tomada de decisões com uso do POMDP *online*, apesar das restrições, mostrou-se funcional com as estratégias heurísticas e de amostragem. O sucesso da abordagem depende muito da performance e reatividade do POMDP. Decidir rápido e conseguir

analisar o máximo de estados são os fatores que mais impactam a qualidade dos resultados. Isso justifica a importância dos métodos propostos, que complementam o POMDP e sua integração neste trabalho.

Capítulo 7

Controle e Escalonamento

Neste capítulo será descrito o módulo de *Controle*. Esse é responsável por: intermediar a troca de informações entre a arquitetura e o ambiente do jogo; gerenciar o planejamento de forma reativa; escalonar as macroações que serão executadas. O módulo obtêm os dados necessários do ambiente em tempo real para que o processo de tomada de decisão seja executado As macroações escolhidas precisam ser tratadas e passarem por uma verificação contínua para sua correta execução no jogo. Para controle do módulo sobre suas tarefas alguns algoritmos e técnicas são utilizados. O primeiro destes a BWAPI é uma API para comunicação da arquitetura com o StarCraft. O Planejamento reativo (PR) é utilizado para controlar o processo de execução das macroações da arquitetura. Essa execução torna-se dinâmica, pois qualquer macroação em execução pode ser cancelada caso novos acontecimentos na partida sejam detectados. O Escalonador de macroações é utilizado para diminuir o tempo de execução das macroações. o algoritmo utilizado é o escalonador para jogos RTS chamado ESCALONA (NAVES, 2012). Esse foi desenvolvido durante o trabalho de mestrado do autor desta tese.

O módulo de *Controle* comunica-se com o POMDP enviando as observações percebidas na partida e recebendo quais macroações devem ser executadas. O módulo opera sobre uma verificação contínua no ambiente da partida. Assim sempre que novos dados sensoriais são captados o processamento destes é feito de imediato e enviados como observações para o POMDP. A BWAPI captura informações do jogo como quantidades de recursos e executa as macroações em forma de comandos dentro do jogo. Toda informação sensorial captada é transformada em um vetor para representar as observações. Variáveis numéricas traduzem as quantidades exatas de determinados recursos e unidades. O escalonamento é executado para todas as macroações e foram definidas representações para uma execução unificada dos diferentes níveis de abstração destas. O escalonador verifica a integridade das macroações. Caso faltem ações ou parâmetros para a execução o escalonador adiciona os itens necessários e gerencia a execução. O escalonamento é feito em conjunto com o PR. As ações que compõe uma macroação são armazenadas pelo PR em uma árvore que controla a ordem de execução e cancelamento dessas ações. Além da

dinâmica na execução do planejamento, o PR permite que várias macroações em diferentes níveis possam ser executada ao mesmo tempo. Este capítulo está divido da seguinte forma: Escalonamento de Macro Ações; Planejamento Reativo com Macro Ações.

7.1 Escalonamento de Macro Ações

O escalonamento é primeiro algoritmo utilizado no módulo *Controle* quando as macroações são enviadas para execução pelo módulo *Decisão*. A macroação descreve quais recursos deseja construir caso seja de *Estratégia*. Quais grupos de unidades vão movimentar-se ou atacar caso sejam *Tática*. Qual unidade individual irá executar algum comando caso seja *ControleReativo*. Assim cabe ao escalonador definir quais ações são necessárias para que a macroação seja executada por completa. Cada ação que compõe uma macroação possui uma série de precondições que devem estar disponíveis na partida para sua execução. As precondições são sumarizadas na Figura 5. Para executar a ação $BuildFirebat$ é preciso que esteja disponível $(1Barracks, 1Academy, 50Mineralse25Gas)$. Caso $BuildFirebat$ esteja em uma macroação para ser produzido, o escalonador verifica se todas as precondições estão atendidas antes de escalonar a sua execução. Para tornar o processo de verificação eficiente as características das ações e suas precondições são descritas por uma linguagem de planejamento. A linguagem utilizada é a *Planning Domain Definition Language (PDDL)* (EDELKAMP; HOFFMANN, 2004). A PDD, possibilita estender as características das ações para descrever precondições de macroações de *Tática* e *ControleReativo*. Essas que utilizam na maioria das vezes ações que movimentam unidades e ordenam ataques. A Figura 40 mostra a ação $BuildFirebat$ e $Move$ descritas com PDDL.

Na Figura 40 a ação $Move$ que move uma unidade para qualquer posição no mapa utiliza de parâmetros adicionais para uma execução completa. Como parâmetros da ação são utilizados a posição no mapa $(?x?y)$ para onde deslocar a unidade, a unidade (z) que será deslocada e quantas vezes o comando foi alocado na macroação (q). Como precondições a unidade (z) deve existir e nenhum recurso deve estar bloqueando a posição que esse irá assumir $(not(atresource?x?y))$. A ação de movimentação quando presente em macroações de *Tática* executa vários deslocamentos. Esses geralmente movem várias unidades para posições adjacentes umas as outras para reuni-las em um mesmo local. Assim a execução da ação $Move$ pode ser estendida a todas as unidades de uma só vez utilizando $(forall(?z)(when(and(not?q = 1))(andz(at??)))$. Essa extensão indica que todas as unidades z irão executar o mesmo movimento. Isso desde que a quantidade q seja maior que 1 e as unidades (z) estejam em alguma posição qualquer $(at??)$ do mapa. Com as extensões da PDDL as ações do jogo em todos os níveis de abstração podem ser descritas sob um mesmo padrão. Macroações enviadas para execução com precondições não satisfeitas são identificadas pelo escalonador. Para isso a descrição

```
(:action Build Firebat
:parameters (?x ?y - location)
:precondition (and (at ?x ?y) (not (= ?x ?y)) exist(Barrack) exist(Academy)
 have(Minerals value(150)) have(Gas value(25)))
:effect (and (Firebat(at ?x ?y)) (not (Gas value(25))) (not (Minerals value(50)))
)

(:action $Move$
:parameters (?x ?y - location ?z - unit ?q - quantity)
:precondition (and exist (?z) (not (at resource ?x ?y))
:effect (and z (at ?x ?y))
(forall (?z)
(when (and (not ?q = 1))
(and z(at ? ?)))
)
```

Figura 40 – Duas ações de diferentes níveis de abstração descritas com a linguagem PDDL.

das ações internas da macroação que estão descritas em PDDL são verificadas. Ações necessárias para satisfazer as precondições são adicionadas a macroação e escalonadas junto com essa. Esse processo aumenta o tempo de execução da macroação devido a adição de novas ações.

Além da especificação em PDDL as macroações e ações individuais precisam de uma formalização em suas estruturas. Essa formalização é utilizada pelo escalonador e pelo PR para ajustar a execução das ações. A formalização determina quais informações complementares cada uma dessas deve ter. A Tabela 8 exibe as informações atribuídas a macroações e ações individuais. Dentre as informações algumas são utilizadas para ampliar os resultados da execução de uma macroação. Como exemplo as informações de "quantidade de unidades e recursos por comando". Essas quando são maiores que 1 alteram a descrição em PDDL da ação que é executada repetida vezes. Uma fila para execução da ação é feita e sempre que um recurso ou unidade daquele tipo ficar disponível a ação já é direcionada a esse. Isso evita a necessidade de uma verificação contínua sobre as unidades e recursos na espera que algum desses fique disponível para executar a ação. Além disso a fila de execução evita que sejam criadas várias instancias da mesma ação na árvore do planejamento reativo.

Outra informação que pode maximizar os efeitos da execução de uma macroação é a "classe de recurso". Algumas ações como *BuildFirebat* ou *BuildBarrack* só podem ser executadas por um tipo específico cada. Esses tipos são *Barrack* e *SCV* respectivamente. Essas ações portanto possuem apenas uma classe de recurso que pode executá-las. Algumas ações permitem que outros recursos de uma mesma classe possam executá-la, principalmente nos níveis de *Tática* e *ControleReativo*. Uma ação para movimentar uma

Tabela 8 – Informações necessárias para gerenciamento do planejamento reativo e escalonamento, contidas em macroações e ações.

Informação	**Descrição da Informação**	**Aplicado**
Nível de Abstração	Necessário para que as ações internas sejam configuradas de forma correta pelo planejamento reativo.	MACRO AÇÃO E AÇÃO
Quantidade de unidades por comando	Indica quantos comandos iguais são feitos a diferentes unidades. Esse parâmetro é utilizado para configurar múltiplas execuções no PDDL e no escalonador também.	MACRO AÇÃO
Quantidade de comandos a edificações	Parâmetro igual ao anterior, apenas direcionado a edificações, com construção de unidades e outras edificações.	MACRO AÇÃO
Tempo de execução	Necessário para que o escalonador alocar o tempo inicial e tempo final de execução total. Esses tempos são utilizados para configurar o escalonamento e cancelar macroações com o PR.	MACRO AÇÃO E AÇÃO
Recurso destino	Indica qual recurso seja edificação ou unidade que deve ser o alvo da ação.	AÇÃO
Classe de recurso	Determina qual classe do recursos destino, esse parâmetro serve para que uma ação possa ser executada mesmo se o recurso destino dela não estiver disponível ainda, mas sim um recurso de mesma classe. As classes de recurso são: unidade ofensiva terrestre, unidade ofensiva aérea, unidade construção, edificação produz unidades, edificação produz tecnologia.	AÇÃO
Prioridade de execução	A prioridade indica o quanto a execução de uma ação é importante para o planejamento da partida. Uma ação com maior prioridade pode cancelar a execução de outra de menor prioridade, desde que essa esteja utilizando recursos que são necessários para a execução daquela de maior prioridade. Os valores de prioridades para as ações são: ações de ataque = 4, ações de produção de unidades e ações de coleta de recursos = 3, ações de produção de edificações = 2, Ações deslocamento no mapa e ações upgrade de tecnologia = 1.	AÇÃO

unidade têm sempre o objetivo de posicionar essa em um local para que seja utilizada futuramente. Considere um exemplo onde uma macroação tenha uma ação de movimentação para um *Firebat* que está em produção. Essa unidade é do tipo ofensiva terrestre mas não está disponível ainda. O escalonador verifica se existe outra da mesma classe como um *Marine* por exemplo. Caso o *Marine* exista o recurso *Firebat* é colocado como uma nova ação para ser produzido. Ao invés de aguardar essa produção para executar a ação de movimentação a mesma é executada inicialmente sob a unidade disponível *Marine*. Isso mantém a estratégia definida por qualquer macroação de *Tática*. Quando o *Firebat* finalizar sua produção o mesmo irá também executar a ação de movimentação.

O processo de escalonamento envolve colocar as ações internas de uma macroação para execução em paralelo onde essas utilizarão o menor tempo do jogo possível. Uma macroação m possui k ações a. Cada ação a possui precondições p na forma $p[l], l \in [0...N]$ que serão escalonadas no menor tempo t cada. O processo de escalonamento necessário para a arquitetura pode ser resumido na Equação 40.

$$m = \sum_{i=1}^{k} a_i \leq a_{max}, \sum_{j=1}^{l} a_i \neq p_l; a_i = \min_{t} \tag{40}$$

onde a_{max} é uma constante de tempo. Essa armazena o tempo de execução máximo para uma ação. A constante evita que alguma ação acumule sua execução com outras ações que não satisfazem nenhuma de suas precondições. $\min_{t}$ representa a alocação da ação a_i para sua execução no menor tempo do jogo possível t.

Com a definição da função de escalonamento da Equação 40 surge a necessidade de definir a ordem em que macroações de diferentes níveis de abstração devem ser escalonadas. A arquitetura escolhe uma macroação de *Estratégia* através do POMDP. Com base nessa e nos valores de predição a CHAID escolhe duas macroações complementares. Quando essas são enviadas ao escalonador é preciso definir uma ordem para execução devido aos níveis de abstração. O caso geral dessa situação é que a macroação principal de *Estratégia* vai produzir recursos enquanto as demais irão gerenciar a posição e coordenar ataques e defesas. Contudo existe a possibilidade de executar previamente as macroações complementares através de recursos já existentes. Também é possível que sejam consideradas macroações complementares mesmo quando nenhuma *Estratégia* tenha sido escolhida pelo módulo *Decisão*. Foi definida uma ordem para escalonamento quando existem macroações de diferentes níveis de abstração. Essa ordem é feita da seguinte forma:

- macroação de *Estratégia*: as ações que compõe essa são escalonadas o quanto antes no tempo do jogo disponível;
- macroação de *Tática*: as ações são verificadas para conferir se existe apenas ações de posicionamento sem nenhuma de ataque. Em caso positivo verifica se existem

unidades na mesma quantidade que a informação "quantidade de unidades por comando". Em caso positivo novamente escalona as unidades já existentes com a ação a ser executada. Em caso negativo aplica a ação apenas as unidades existentes e replica essa quando as unidades terminarem de serem produzidas. Caso contenha alguma ação de ataque verifica se existe pelo menos uma unidade ofensiva e replica a ação para essa e outras unidades que existam da mesma classe;

- macroações de *ControleReativo*: as ações que compõe essa são escalonadas o quanto antes. Caso a unidade que executa a ação não exista é verificado se outra de mesma classe exista na partida. Em caso positivo essa unidade executa a nova ação interrompendo qualquer execução em que já estivesse alocada.

O algoritmo utilizado para escalonar as macroações com todas as definições já mencionadas é o ESCALONA. Esse obteve bons resultados em trabalhos anteriores (NAVES; LOPES, 2012a) e (NAVES; LOPES, 2012b) inclusive utilizando a estratégia de janelas abertas. Essa estratégia utiliza uma característica de escalonamento presente em jogos RTS que são as janelas de tempo. Uma janela de tempo é um intervalo de tempo livre dentro da linha do tempo total da partida ou entre a execução de duas ações já escalonadas. Toda nova ação que será escalonada deve ser alocada em um intervalo livre de tempo inicial e final para ser executada. A Figura 41 mostra um exemplo de intervalo entre ações escalonadas na linha de tempo da partida. Quando ações que constroem edificações em macroações de *Estratégia* são escalonadas intervalos como o da Figura 41 entre duas ações tendem a não surgir. Isso ocorre porque ações de construção utilizam apenas um recurso construtor e possuem alto tempo de execução. Esse em média de 40 segundos.

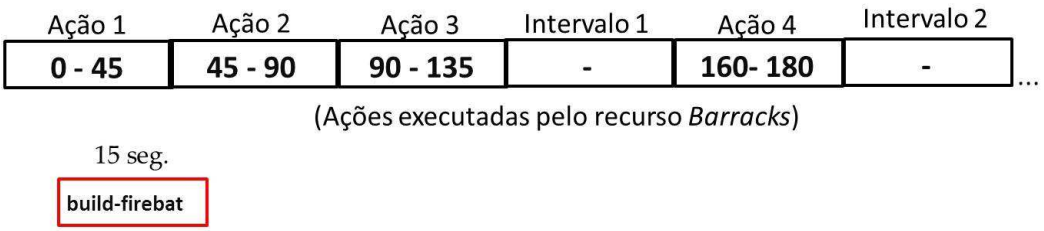

Figura 41 – Exemplos de intervalos em um escalonamento de ações.

Com o uso de macroações de *Tática* e *ControleReativo* as ações utilizam diversas unidades ao mesmo tempo. Essas executam ações com tempo médio de execução de 5 segundos ou menos. Assim surgem intervalos entre as ações quando existem os escalonamentos são feitos em grande frequência. Esses intervalos geralmente não são considerados pelos escalonadores devido a duração de poucos segundos. A estratégia de janelas abertas consiste verificar por intervalos no tempo de jogo anterior aquele da última ação que foi escalonada. Essa verificação é feita sempre que uma ação será escalonada. Caso exista o

intervalo a ação é alocada nesse. O escalonamento aloca diversas ações utilizando muitos segundos a frente do tempo atual do jogo. Isso cria janelas de tempo não utilizadas no tempo anterior a última ação escalonada. Com a abordagem de ações nos três níveis de abstração, até mesmo as janelas de tempo mínimas de 2 segundos ou menos podem ser reutilizadas. As Figuras 42 e 43 mostram como manter uma verificação contínua que sempre confere o tempo de jogo já escalonado que descobre janelas de tempo. Nas figuras a ação $BuildFirebat$ seria escalonada no intervalo 2 caso o escalonador verificasse a partir da última ação alocada. A verificação contínua faz o mesmo utilizar o intervalo 2. O algoritmo possui complexidade assintótica de $O(n^2)$ onde n é quantidade de ações que serão escalonadas durante toda a partida. A verificação contínua de janelas é feita toda vez que uma ação deve ser escalonada. Essa verificação é em tempo linear pela quantidade de ações já escalonadas m sendo essa complexidade $O(m)$. O algoritmo ESCALONA então tem complexidade total de $n^2 \cdot m$ que é igual $O(n^2)$. A estratégia de janela de tempo não diminui a performance do ESCALONA que consegue sua execução dentro das restrições de tempo real do jogo.

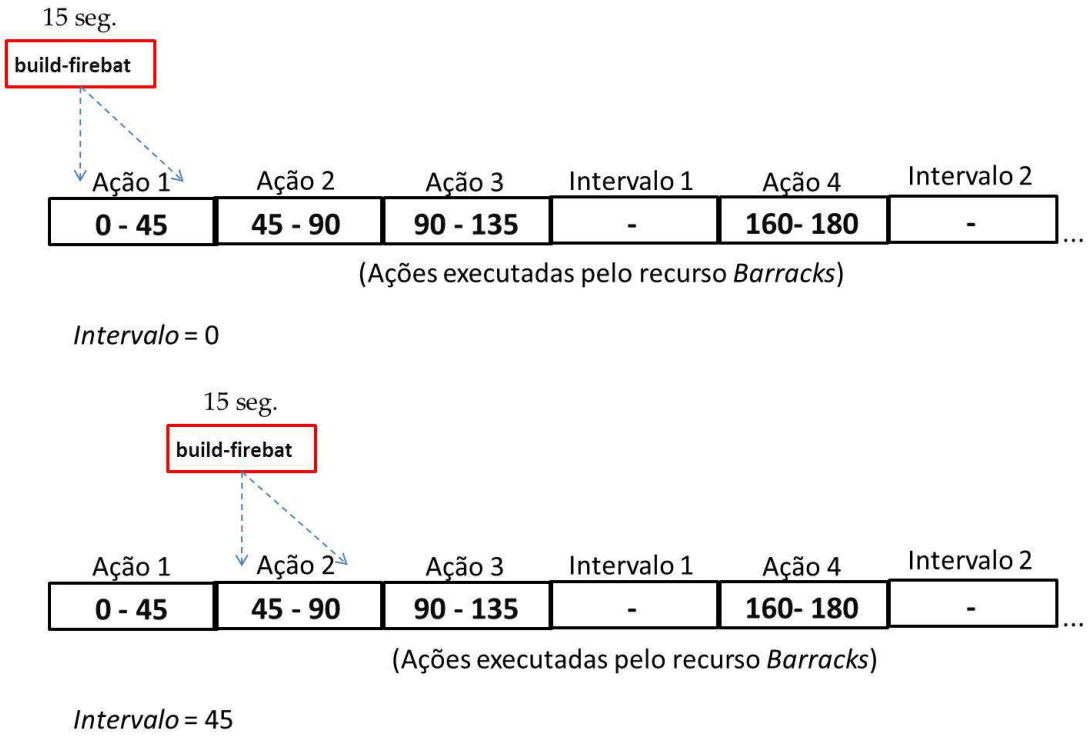

Figura 42 – Verificação contínua por janelas de tempo.

O Algoritmo 7 mostra as principais operações do ESCALONA. Como parâmetro são enviados a macroação e uma variável $melhorTempo$. Essa variável é utilizada para armazenar a última ação escalonada com menor tempo de jogo. Na linha 1 a função $Precond()$ é responsável por verificar todas as ações da macroação. A verificação inicialmente procura pelas precondições de cada ação e confere quais não são satisfeitas.

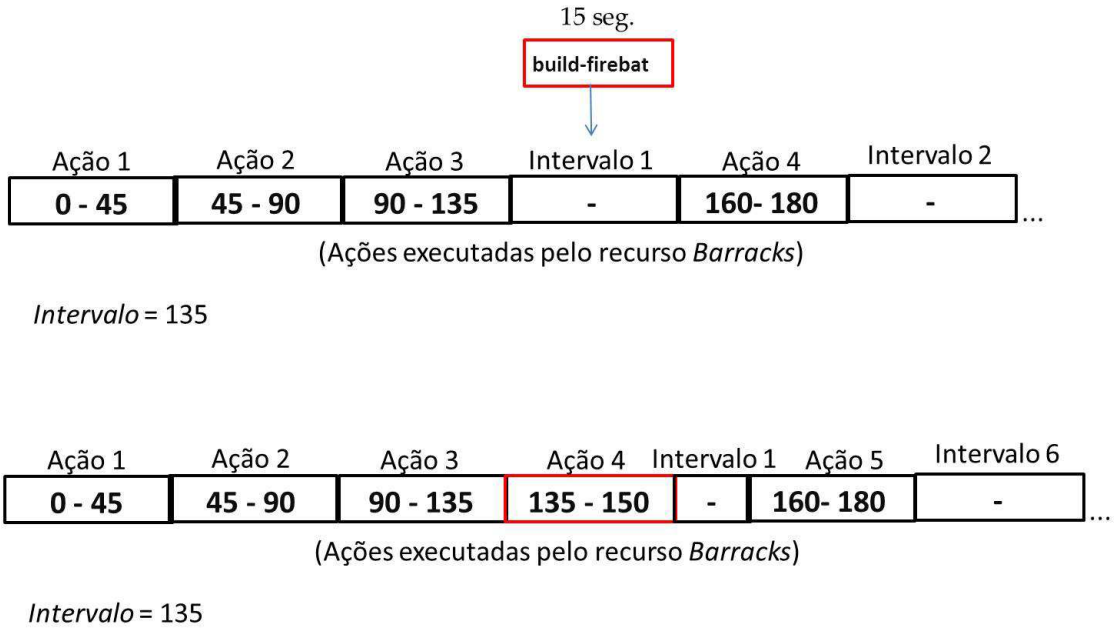

Figura 43 – Janela de tempo presente no intervalo 1 é encontrada.

Para cada precondição é adicionada a macroação as informações e ações necessárias para satisfazer essa. A função também verifica as informações da macroação e das ações. As especificações de execução prioridade e demais parâmetros definidos são criados. $Precond()$ deixa a macroação pronta para começar o processo de escalonamento e essa função é que possui complexidade $O(n^2)$. A linha 4 começa a repetição que define o tempo de inicio e fim de cada ação escalonando essa. Na linha 5 as ações já escalonadas e o tempo de jogo da partida são verificados através da variável *intervalo*. Caso exista um intervalo em que o tempo de execução inicial e final da ação se encaixem, a mesma é alocada na linha 6. Caso não seja possível alocar a ação em uma janela nas linhas 8 e 9 a ação é alocada. Nesse caso é alocada depois da última ação que satisfaz suas precondições terminar sua execução. A linha 11 verifica se a ação escalonada foi alocada em um tempo menor que a última ação escalonada na iteração anterior. Em caso positivo o valor de $melhorTempo$ é atualizado para ser utilizado na próxima iteração com outra ação. O ESCALONA sempre que aloca o tempo de uma ação envia essa para o planejamento reativo que organiza as ações para execução.

7.2 Planejamento Reativo com Macro Ações

Planejamento reativo é utilizado para adicionar dinamismo ao processo de tomada de decisão e planejamento da arquitetura proposta. A principal dinâmica é a possibilidade de cancelar e reconsiderar a execução de macroações. Além disso essa abordagem complementa a execução e gerenciamento de macroações junto ao escalonador. Junto com o PR é utilizada a linguagem reativa ABL (MATEAS; STERN, 2014) e a árvore de

Algoritmo 7 ESCALONA(*Macroação m*, *melhorTempo*)

```
1: m ← Precond(m)
2: intervalo ← tmpInicio
3: tempoEscalona ← 0
4: for each Ação a ∈ m do
5:    if intervalo+a.tempoFim <= a.tempoTotal ou intervalo >= a.tempoFim then
6:       a.tempoInicio ← intervalo
7:    else
8:       intervalo ← a.tempoFim
9:       tempoEscalona ← intervalo
10:   end if
11:   if tempoEscalona < melhorTempo e tempoEscalona <= a.tempoFim then
12:      melhorTempo ← tempoEscalona
13:   end if
14: end for
15: return melhorTempo
```

comportamentos (MATEAS, 2002). ABL é uma linguagem onde um agente possui um conjunto de metas a atingir. Agentes atingem as metas selecionando e executando comportamentos predefinidos ou a partir uma biblioteca destes. Um comportamento possui um conjunto de zero ou mais precondições. Essas especificam quando esse comportamento pode ser executado dependendo do estado do jogo. Durante o uso da ABL todas as ações de uma macroação que serão executadas e já foram escalonadas são armazenadas em uma Árvore de Comportamentos Ativos (ABT). Nessa árvore um nó raiz é sempre um comportamento que representa a macroação que foi escalonada. Os demais nós são os componentes que podem ser representados por qualquer tipo de objeto como ações, scripts e até mesmo outras macroações. Sempre que um nó é selecionado na ABT seus componentes são adicionados como seus nós filhos na árvore. A Figura 44 mostra um exemplo de uma ABT.

Na Figura 44 o comportamento representado pela ABT é a execução da macroação *Estratégia* chamada $OfensivaRápida1$. Do lado esquerdo da ABT estão as ações que compõe a macroação, essas já estão escalonadas e ficam na árvore até serem executadas no jogo. Quando a execução ocorre o nó que representa aquela ação é removido. Do lado direito da ABT está uma macroação de *Tática* chamada $MoverMeio2$ que foi escolhida pela CHAID e enviada como macroação complementar. Abaixo da $MoverMeio2$ estão as ações necessárias para executar essa. Entre essas dois nós paralelos mostram uma ação que foi replicada para as duas unidades já existentes na partida. Por enquanto as unidades ainda serão produzidas na macroação $OfensivaRápida1$. A montagem da ABT permite agregar as ações da macroação principal e das macroações complementares em uma única estrutura. Isso facilita o controle e execução dessas na partida. A árvore funciona como uma camada para as macroações transformando em comportamento cada uma e em componentes as suas precondições.

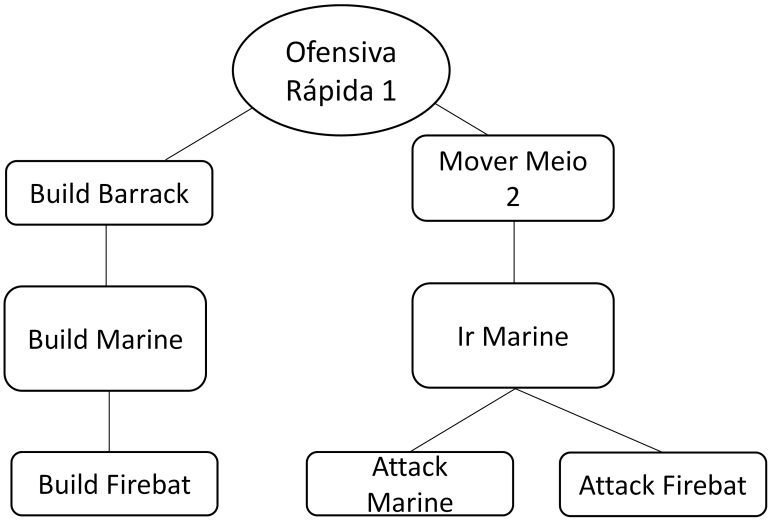

Figura 44 – Exemplo de uma árvore de comportamentos.

As restrições presentes no conceito de comportamento permitem uma verificação mais rápida de quais precondições devem ser satisfeitas para a macroação ser executada. Recursos que já existem tem suas respectivas ações retiradas do componente. Permanecem apenas aqueles necessários para produzir os recursos faltantes e cumprir as restrições. A reatividade afeta diretamente as macroações pois a ABL permite que comportamentos sejam descartados durante sua execução. Nesse caso a entrada de outros novos é feita dependendo da situação do jogo. Assim uma macroação pode ser descartada e outra colocada em seu lugar imediatamente sem comprometer o planejamento feito de acordo com a macroação escolhida.

A execução das macroações principal e complementares em paralelo é gerenciada pela ABT como mostra a Figura 44. Caso POMDP selecione uma nova macroação com recompensa maior que aquela atual em execução na ABT, a mesma é descartada tendo suas ações em execução dentro da partida canceladas. A nova macroação selecionada é colocada na árvore ABT em forma de comportamento. As macroações complementares conectadas com aquela removida tem suas ações canceladas e seus comportamentos retirados da ABT. As ações já executadas dentro do jogo e que pertenciam as macroações canceladas têm seus recursos gerados reaproveitados.

Sempre que um comportamento é colocado em execução é feita a tradução da linguagem ABL em código para execução dentro do jogo. O POMDP assim que escolhe uma macroação envia essa para a ABT que coloca essa como o comportamento inicial e começa sua execução. A Figura 45 mostra os códigos gerados quando a ABT é construída e comportamentos são colocados em execução. Na figura a ação $Marine$ é colocada como nó raiz na ABT por ser a primeira da macroação. Para conseguir executar a ação é preciso que existam os recursos $CommandCenter$, $Barrack$ e 250 $Minerais$. O parâmetro 4 equivale a executar quatro vezes a ação para conseguir a quantidade necessária de

Minerais. Esses recursos serão obtidos através de ações executadas em paralelo como mostra o parâmetro do comportamento *Marine*. Novos comportamentos serão criados para construir os recursos necessários e esses serão colocados conectados ao comportamento *Marine*. Os novos recursos provavelmente terão suas próprias precondições que irão gerar novos outros comportamentos populando ainda mais a ABT.

```
initial_tree{
subgoal Marine()
}

parallel_behavior Marine(){
act set Minerals(4)
act set build Command Center(1)
act set build Barrack(1)
}

sequential_behavior Agrupa 1() {
precondition {
(PlayerUnit type == Marine || Firebat ID::unitID)
(Resources == 250)
}
act select(unitID, 4);
act move(units[], positionx,positiony)
}
```

Figura 45 – Árvore de comportamentos convertida em código para execução.

Ainda na Figura 45 o comportamento *Agrupa*1 tem suas precondições colocadas de forma explícita no comportamento criado. Essa macroação não é proveniente da raiz e não é uma *Estratégia* por isso os recursos necessários e a quantidade de unidades podem já existir dentro do jogo. A Figura 46 apresenta o código de uma ABT onde o comportamento *Padrao*2 deve ser executado. O comando *spawngoal* utilizado com o comportamento $Mineral_Gas$ serve para iniciar uma nova execução. Essa utiliza a macroação em paralelo junto a atual com o $Mineral_Gas$ alocado na ABT como um nó de mesmo nível da macroação *Padrao*2. A execução de mais de uma macroação ao mesmo tempo é confirmado com o uso do comando *persistent* no comportamento $Mineral_Gas$. Esse indica que as macroações devem ser executadas em paralelo. Com o uso de PR o módulo de *Decisao* fica completo para gerenciar a execução das macroações e suas respectivas ações dentro do jogo.

7.3 Considerações Finais

O escalonamento e uso de planejamento reativo são feitos no módulo *Controle*. Esse é responsável pela comunicação com o jogo recebendo as observações e enviando as ações

```
sequential_behavior Padrão2() {
spawngoal Mineral_Gas();
mental_act {
System.out.println("Produção de gás iniciada");
}
}


sequential_behavior Mineral_Gas()() {
with (persistent) subgoal SCV();
}
```

Figura 46 – Macroação da árvore convertida em código para execução.

para execução. O escalonamento é realizado com as macroações que são enviadas pelo módulo *Decisão* utilizando os padrões e informações definidos para as ações. Com essas definições é possível que ações em diferentes níveis de abstração sejam escalonadas no mesmo processo. O algoritmo ESCALONA com a estratégia de janelas abertas foi adaptado para a arquitetura aqui proposta explorando a características de intervalos entre execuções de múltiplas ações.

O planejamento reativo foi utilizado para habilitar um comportamento dinâmico em relação a execução das macroações. Quando o ambiente sofre alterações e novas macroações mostram-se mais promissoras é possível cancelar a execução atual e considerar o replanejamento. Além disso o PR auxilia o escalonador na execução e controle das ações antes de enviar essas para a partida. O módulo *Decisão* junto com aqueles descritos nos capítulos anteriores formam a arquitetura de uso geral baseada em planejamento probabilístico.

Capítulo 8

Experimentos e Análise dos Resultados

Neste capítulo são apresentados os resultados obtidos a partir da validação da metodologia proposta. Os experimentos serão apresentados por etapas. Cada módulo passa por experimentos específicos e ao fim a arquitetura completa é testada e validada. Para cada módulo são utilizados diferentes métodos de avaliação e esses são descritos em cada seção antes dos resultados serem apresentados.

São executados experimentos para: avaliar a qualidade do processo de classificação e obtenção de macroações; avaliar a qualidade da tomada de decisão e performance dos algoritmos *online*; avaliar a capacidade e qualidade da arquitetura completa contra *bots* de competições e jogadores humanos. Os experimentos mostram que cada módulo da arquitetura desempenha de forma satisfatória suas tarefas superando os algoritmos da literatura. A arquitetura completa consegue superar a maioria dos *bots* e estabelecer um desafio sólido para jogadores humanos.

Todos os experimentos aqui apresentados foram feitos em um computador *Intel Core i7 1.73 Ghz* com 8Gb de memória *ram*, sendo executado com o sistema operacional *Windows 7*.

8.1 Análise da Classificação de Dados

Nesta seção serão descritos os experimentos e resultados obtidos com as técnicas do módulo $Classificação$. Os valores obtidos com os testes serão discutidos utilizando os algoritmos PrefixSpan e GSP na classificação de macroações.

Para os algoritmos de classificação os testes serão divididos em duas etapas sendo a primeira de análise estatística e a segunda de qualidade e performance. A análise estatística é feita para avaliar o modelo de classificação gerado. Os resultados obtidos serão comparados com outros métodos de classificação em relação ao problema de encontrar macroações. Os resultados são tabulados na matriz de confusão e utilizados com métricas de desempenho e avaliações estatísticas de modelos de classificação. Os testes de análise estatística foram feitos utilizando a técnica de validação cruzada (*k-foldcross-validation*).

Nesse $k = 10$ sendo que a base de dados de testes foi dividia em 10 conjuntos $C_1, ..., C_k$ de tamanhos iguais ou aproximados. Em seguida k iterações de treinamento e validação são executadas, em cada iteração i um diferente conjunto C_1 é usado para validação enquanto os restantes são utilizados para aprendizagem.

Para a validação cruzada foi criada uma base de testes utilizando *replays* que possuem macroações que representam *build orders* de *Estratégia*. Essas macroações representam estratégias catalogadas pela comunidade de jogadores de StarCraft, ou seja, suas descrições são conhecidas. Na arquitetura do trabalho as macroações já catalogadas na comunidade foram classificadas. Contudo aquelas não catalogadas também foram classificadas para que novas estratégias possam ser testadas aumentando a capacidade da arquitetura. Para a análise estatística é interessante utilizar apenas as macroações catalogadas para que o parâmetro de comparação entre algoritmos seja conhecido e sem nenhum viés. Ao todo são utilizados 300 *replays* sendo 50 para cada tipo de confronto entre as raças do jogo: *Terran x Protoss; Terran x Zerg; Terran x Terran; Protoss x Zerg; Protoss x Protoss; Zerg x Zerg*. Para cada raça foram escolhidas aleatoriamente 8 macroações onde cada representa uma classe em que os algoritmos irão classificar. O tamanho das macroações varia de 5 a 19 itens sendo cada um desses itens uma ação que compõe essa. A Tabela 9 mostra um exemplo com macroações da raça *Protoss* presente na base de testes. O tempo de jogo marca a quantidade de tempo necessária para produzir os recursos da macroação.

Tabela 9 – Exemplo de tuplas contidas na base de testes.

Macro Ação	**Itens**	**Tempo de Jogo**
One Gateway Dragoon	8 - Pylon10 - Gate12 - Assimilator13 - Zealot16 - Pylon18 - Cybernetics Core	02m:34s
Dark Templar Fast	8/9 — Pylon10/17 - Gateway12/17 - Assimilator14/17 - Cybernetics Core14/17 - Zealot 16/17 - Pylon18/25 - Citadel of Adun18/25 - Dragoon23/25 - Pylon24/33 - Templar Archives24/33 - Dark Templar24/33 - Nexus at Natural Expansion	03m:274s
2 Gate Range Expand	8 - Pylon10 - Gateway12 - Assimilator13 - Cybernetics Core15 - Pylon17 - Dragoon Range18 - Gateway20 - Nexus20 - Dragoon22 - Dragoon24 - Pylon25 - 2 Dragoons31 - Robotics Facility	4m:17s
Speed Zealot	8/9 - Pylon10/17 - Gateway12/17 - Assimilator13/17 - Zealot	01m:54s

As métricas para avaliação do desempenho utilizadas são encontradas com descrição detalhada em (FERREIRA, 2006). As matrizes de confusão utilizadas nos experimentos são multiclasse de tamanho MxM. Assim os valores serão descritos em função de cada

classe C_i considerada como principal e as demais classes representadas por C_j, onde $j = 1, ..., M, j \neq i$. Para cada classe C_i: o valor de VP_i corresponde a quantidade de verdadeiros positivos; FP_i é a quantidade de exemplos da classe C_j incorretamente classificados na classe C_i; FN_i é a quantidade de exemplos da classe C_i incorretamente classificados na classe C_j; VN_i é a quantidade de exemplos da classe C_j corretamente classificados na classe C_j. As métricas utilizadas são:

- Precisão (*accuracy*), porcentagem de exemplos que foram classificados corretamente pelo classificador.

$$precisão = \frac{VP_i}{VP_i + FP_i} \tag{41}$$

- Cobertura (*recall*), corresponde ao valor da cobertura de casos para uma determinada classe, isto é, o número de instâncias corretamente classificadas em relação ao total de exemplos pertencentes a esta classe

$$cobertura = \frac{VP_i}{VP_i + FN_i} \tag{42}$$

- *F-mesasure* (F1), é a média harmônica ponderada entre a $Precisão$ e a $Cobertura$. Essa, mede a eficiência do sistema levando em conta a diferença dessas métricas, quanto mais próximo de 1 melhor o resultado. Nessa, $\alpha = 1$, $p_i = Precisão$ e $c_i = Cobertura$.

$$F1 = \frac{(1 + \alpha)p_i c_i}{\alpha p_i + c_i} \tag{43}$$

- Erro médio, corresponde a média dos erros cometidos por cada classe na classificação. Quanto maior o seu valor, maior a chance de erro em qualquer uma das classes do problema.

$$ErroMedio = \frac{\sum_{i=1}^{m} \frac{FP_i + FN_i}{FP_i + FN_i + VP_i + VF_i}}{M} \tag{44}$$

Os algoritmos utilizados nos experimentos de análise estatística são: PrefixSpan com a versão e configuração descrita nessa tese; GSP com a versão e configuração descrita nessa tese; *Non-nested generalized exemplars* (NNGE) com a versão padrão do algoritmo; *k-Means* (KMN) onde o valor de k é dado pela raiz quadrada da quantidade de características $k = n^{1/2}$ e $n = 6$. Os algoritmos de classificação adicionais NNGE e KMN são amplamente utilizados na literatura como classificadores gerais com bom desempenho e resultado. A seguir são apresentadas as matrizes de confusão. É utilizada uma matriz para cada raça do jogo e com 8 macroações escolhidas aleatoriamente para classificação. Nas matrizes os valores apresentados em cada célula no formato $xx/xx/xx/xx$, representam a quantidade de classificações feitas pelos algoritmos na ordem PrefixSpan/GSP/NNGE/KMN respectivamente.

Na Tabela 10 os valores na primeira linha 80/73/65/66 representam a quantidade de macroações *Two Gateway Range* classificadas de forma correta pelos algoritmos na ordem mencionada. Na última coluna está o *Total* que representa a quantidade total de cada macroação na base de dados utilizada. Assim pode-se concluir que o classificador PrefixSpan classificou 80 macroações *Two Gateway Range* de forma correta de um total de 83 que estão contidas na base de dados. As Tabelas 10, 11 e 12 apresentam as matrizes das raças *Terran*, *Protoss* e *Zerg* respectivamente.

Tabela 10 – Matriz de confusão para classificação de macroações da raça *Terran*. Os resultados de cada célula, exibem valores para os classificadores PrefixSpan/GSP/NNGE/KMN respectivamente.

Macro Ação Real	**Macro Ação Prevista**							**Total**
	One Factory Double	One Barracks Fast	14 Command Center	Fake Double Attack	Barracks Barracks Supply	Fast Nuke	Proxy 5 Rax	
One Factory Double	**26/23/17/15**	0/2/6/9	0/0/0/0	0/0/1/0	0/1/2/1	0/0/0/0	0/0/0/0	26/26/26/26
One Barracks Fast	2/5/11/13	**92/87/78/79**	0/0/0/1	0/0/0/0	0/2/5/0	0/0/0/0	0/0/0/1	94/94/94/94
14 Command Center	0/0/7/11	0/1/5/6	**64/62/46/39**	0/0/6/4	0/1/4/4	0/0/0/0	0/0/0/0	64/64/64/64
Fake Double Attack	0/1/0/3	0/1/2/0	0/0/2/2	**37/31/20/18**	1/3/6/2	0/0/3/8	1/3/8/4	39/39/39/39
Barracks Barracks Supply	0/1/4/4	1/2/9/12	0/1/5/0	0/0/0/0	**87/83/69/71**	0/0/0/0	0/0/0/0	88/88/88/88
Fast Nuke	0/0/0/2	0/0/0/0	0/2/6/5	0/0/0/0	0/0/0/0	**32/30/26/25**	0/0/0/0	32/32/32/32
Proxy 5 Rax	0/0/0/1	0/0/2/0	0/0/0/0	0/0/0/0	0/4/6/10	0/0/0/0	**14/10/6/3**	14/14/14/14

Na Tabela 10 o classificador PrefixSpan conseguiu os melhores resultados de classificação superando todos os demais algoritmos. Em 4 das 8 classes o PrefixSpan foi capaz de classificar todas as amostras sem nenhuma instância incorreta. A classe *One Barracks Fast* possui a maior quantidade de classificações incorretas sendo 2 no total. Isso ocorre

devido as ações semelhantes que essa classe possui com a classe *One Factory Double*, ambas possuem mais da metade das ações iguais. O bom desempenho do PrefixSpan é devido as definições propostas de alfabeto e itens frequentes elaboradas na Seção 5.2. Essas definições foram propostas levando em conta as característica do ambiente de jogos RTS. O GSP obteve bom desempenho também utilizando das mesmas definições do PrefixSpan. No entanto o uso de prefixos a partir do próprio banco de dados projetado é uma estratégia com melhor retorno para descoberta de padrões na base de dados. Esse é o motivo do PrefixSpan ser utilizado como algoritmo para classificação de macroações de *Estratégia*. Os demais algoritmos alternam entre alguns resultados satisfatórios e outros regulares. A heurística de escolha de itens do NNGE a partir da base de treinamento mostra-se ineficaz quando é preciso levar em conta repetições contínuas de itens frequentes. A heurística consegue um ajuste para ações que repetem-se em intervalos entre 5 e 6 itens, contudo aquelas contínuas com mais itens geralmente não são consideradas. O KMN ao agrupar as médias tende a repetir os itens frequentes sem conseguir discretizar algumas macroações que possuem 60% ou 80% de ações iguais.

Na Tabela 11 os resultados da matriz da raça *Protoss* são semelhantes aqueles da *Terran*. As duas raças têm dinâmica de construção de recursos e estratégias com características próximas. Ambas possuem um equilíbrio entre unidades terrestres e aéreas sendo que macroações com maior duração de tempo baseiam-se em realizar *upgrades* nos recursos já construídos. O PrefixSpan continua com os melhores resultados alcançando um total de 5 classificações sem nenhum erro nas amostras utilizadas. O desempenho do NNGE e GSP são inferiores aos obtidos na raça *Terran*. As macroações da raça *Protoss* tendem a produzir um número elevado de uma mesma unidade devido ao bom custo benefício que essas apresentam. Macroações como *12 Nexus 5 Zealots*, *12 Nexus* e *5 Gateway Dragoons* evidenciam esse comportamento. Esse tipo de macroação é identificado através da execução contínua e em pequenos intervalos de tempo das ações de coletar minerais e de produção da unidade específica. Esses tipos de padrões são os que apresentam menor performance de qualidade para o NNGE e KMN.

A Tabela 12 mostra os resultados da matriz na raça *Zerg*. Nessas os algoritmos utilizados apresentam o pior resultado de classificação. A raça *Zerg* apresenta uma quantidade reduzida de macroações em relação as outras raças e sua dinâmica de produção de recursos e estratégia é diferentes das demais. Em geral a produção de recursos utiliza de um recurso e um processo chamado *morph*. Com esse qualquer produção de unidades ou edificações deve ser preparada por uma determinada unidade por um tempo e depois finalizado por outra unidade diferente utilizando *morph*. O *morph* difere a *Zerg* da forma como as demais raças operam. Isso dificulta a classificação correta quando a quantidade de ações na macroação é médio ou grande. O PrefixSpan obteve os melhores resultados mas com uma média de acerto na classificação inferior as anteriores. Com menos macroações disponíveis as classes da *Zerg* repetem-se com maior frequência e a matriz têm maior quantidade de

Tabela 11 – Matriz de confusão para classificação de macroações da raça *Protoss*. Os resultados de cada célula, exibem valores para os classificadores PrefixSpan/GSP/NNGE/KMN respectivamente.

Macro Ação Real	**Macro Ação Prevista**							**Total**
	Two Gateway Range	Dark Templar Fast	Two Base Arbiters	9/10 Gateway	5 Gateway Dragoons	12 Nexus 5 Zealot	12 Nexus	
Two Gateway Range	**80/73/ 65/66**	0/0/ 0/0	0/0/ 1/0	2/4/ 9/10	1/3/ 8/6	0/0/ 0/0	0/0/ 0/2	83/83/ 83/83
Dark Templar Fast	0/0/ 3/2	**101/94/ 88/85**	0/0/ 0/0	0/0/ 0/3	0/9/ 10/9	0/0/ 0/0	0/0/ 0/0	101/101/ 101/101
Two Base Arbiters	0/1/ 5/6	0/0/ 0/0	**37/36/ 28/28**	0/0/ 0/0	0/0/ 0/0	0/0/ 0/0	0/0/ 4/3	37/37/ 37/37
9/10 Gateway	1/3/ 5/7	0/0/ 0/0	0/0/ 0/2	**21/15/ 13/14**	0/4/ 4/1	0/0/ 0/8	0/0/ 0/0	22/22/ 22/22
5 Gateway Dragoons	0/2/ 5/6	0/0/ 0/0	0/0/ 0/0	0/3/ 7/8	**44/39/ 32/30**	0/0/ 0/0	0/0/ 0/0	44/44/ 44/44
12 Nexus 5 Zealot	0/0/ 1/0	0/0/ 1/0	0/0/ 0/0	0/0/ 0/0	0/0/ 0/0	**46/40/ 34/33**	0/6/ 10/13	46/46/ 46/46
12 Nexus	0/0/ 2/0	0/0/ 0/0	0/0/ 0/0	0/0/ 0/0	0/0/ 0/0	0/9/ 17/19	**51/42/ 32/32**	51/51/ 51/51

amostras para as macroações. O desempenho do NNGE e KMN foi insatisfatório devido as características da raça tornado o seu uso inviável com essa raça e respectivas amostras de classificação.

A Tabela 13, apresenta a aplicação das métricas nos valores presentes nas matrizes de confusão. Em todas as raças o PrefixSpan obteve maiores medidas e menor erro médio. A precisão obtida nas raças *Terran* e *Protoss* foi acima de 98% com poucos casos de classificação incorreta. Na classe *Zerg* devido as características diferentes na produção de recursos todos os classificadores tiveram um resultado abaixo da média das outras duas raças. O GSP obteve bons resultados explorando as mesmas configurações de alfabeto que o PrefixSpan superando os classificadores NNGE e KMN. Esses valores evidenciam a eficiência da estratégia de mineração sequencial com os dados de jogos RTS que possuem

Tabela 12 – Matriz de confusão para classificação de macroações da raça *Zerg*. Os resultados de cada célula, exibem valores para os classificadores PrefixSpan/GSP/NNGE/KMN respectivamente.

Macro Ação Real	**Macro Ação Prevista**							**Total**
	9 Pool Speed	Over pool	12 Hatch	Two Hatchery	Mutalisk Harass	Lurker Contain	8 Extractor	
9 Pool Speed	**124/99 101/98**	19/33/ 35/41	0/0/ 0/0	0/0/ 0/0	0/3/ 5/4	0/8/ 2/0	0/0/ 0/0	143/143/ 143/143
Overpool	19/27/ 42/45	**107/88/ 72/75**	0/0/ 0/0	0/0/ 0/0	0/5/ 9/4	0/6/ 3/2	0/0/ 0/0	126/126/ 126/126
12 Hatch	0/1/ 5/6	0/0/ 0/0	**19/11/ 06/09**	0/0/ 0/0	0/0/ 0/0	0/0/ 0/0	0/0/ 4/3	32/32/ 32/32
Two Hatchery	0/0/ 0/0	0/0/ 1/0	7/10/ 17/20	**60/56/ 47/42**	0/1/ 2/4	0/0/ 1/0	0/0/ 0/0	67/67/ 67/67
Mutalisk Harass	2/4/ 7/5	3/7/ 11/15	0/0/ 2/0	0/2/ 0/2	**69/61/ 51/52**	0/0/ 3/0	0/0/ 0/0	74/74/ 74/74
Lurker Contain	0/0/ 0/0	0/0/ 0/0	0/0/ 7/4	0/0/ 2/5	3/10/ 10/12	**79/66/ 56/52**	9/15/ 16/18	91/91/ 91/91
8 Extractor	0/0/ 0/0	0/0/ 0/0	0/0/ 3/4	0/0/ 0/2	2/2/ 0/0	7/16/ 18/14	**15/06/ 03/04**	24/24/ 24/24

muitas repetições de ações e ciclos entre essas. O NNGE foi superior ao KMN na maioria dos testes. O conseguiu particionar mais sequências do que a estratégia de comparação de vizinhos do KMN. A Figura 47, mostra um gráfico com as médias dos valores das métricas de desempenho nas três raças do jogo. Com a média geral os resultados do NNGE o aproximam do GSP pois na raça *Zerg* o NNGE consegui superar o KMN na classificação de macroações que continham grandes quantidades de ações.

A seguir serão apresentados os experimentos de qualidade no desempenho da classificação de dados. Esses são focados em descrever a qualidade do uso de mineração sequencial na tarefa de classificar macroações. Os algoritmos utilizados nos experimentos são: PrefixSpan com todas as características e técnicas descritas nessa tese; GSP com todas as características e técnicas descritas nessa tese; FPGrowth algoritmo eficiente e escalável para mineração sequencial; FreeSpan (**??**) um dos algoritmos de mineração sequencial mais utilizados junto com o PrefixSpan, este projeta recursivamente bancos de dados de sequências em um conjunto menor crescendo os fragmentos em cada banco de dados projetado. Foram escolhidos apenas algoritmos de mineração sequencial para explorar o padrão de operação desses algoritmos evidenciando as vantagens das estratégias

Tabela 13 – Métricas de desempenho aplicadas aos classificadores, utilizando os valores das matrizes de confusão.

	Precisão	**Cobertura**	**F-measure**	**Erro Médio**
		Terran		
PrefixSpan	0,983	0,948	0,964	0,0235
GSP	0,917	0,885	0,900	0,0286
NNGE	0,792	0,738	0,751	0,0367
KMN	0,734	0,688	0,715	0,0431
		Protoss		
PrefixSpan	0,992	0,977	0,983	0,0185
GSP	0,835	0,792	0,807	0,0303
NNGE	0,779	0,704	0,721	0,389
KMN	0,734	0,498	0,580	0,437
		Zerg		
PrefixSpan	0,859	0,867	0,851	0,0297
GSP	0,692	0,701	0,711	0,0452
NNGE	0,712	0,679	0,723	0,0446
KMN	0,655	0,676	0,631	0,518

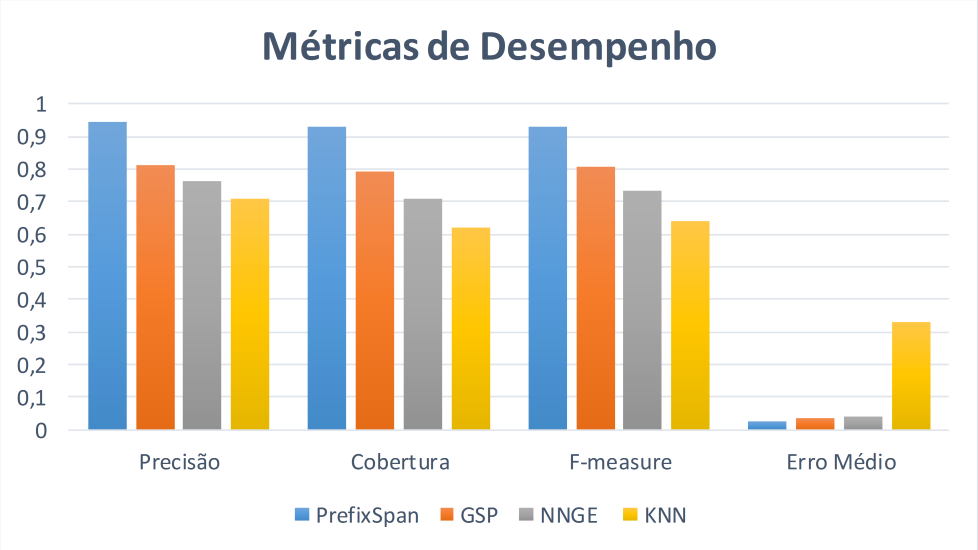

Figura 47 – Média dos valores obtidos com a aplicação das métricas de desempenho.

escolhidas nesta tese.

A Figura 48 mostra os resultados na classificação de macroações das três raças do jogo. Nessa o suporte mínimo utilizado para identificar padrões de sequência de ações varia em função do limite. O PrefixSpan consegue classificar o maior número de macroações. Na raça *Terran* esse consegue classificar em média 18 macroações com o suporte variando de 20% a 10%. O Freespan consegue um bom resultado dividindo as ações da base em subgrupos e procurando por padrões entre esses. Na raça *Protoss* o resultado apresenta um padrão em todos os classificadores, com a maioria das macroações classificadas com o suporte em 15%. Na raça *Zerg* o comportamento dos classificadores é diferente devido a quantidade menor de macroações existentes e o tamanho dessas. A maioria das macroações são classificadas com 30% do valor do suporte, a partir de 20% já são encontradas poucas macroações. O PrefixSpan conseguiu um bom resultado frente aos demais classificadores nessa raça. O uso de prefixos de ações para encontrar padrões com base neste prefixo facilita a identificação da produção de recursos utilizando o *morph* da raça *Zerg*.

O PrefixSpan supera os classificadores testados na quantidade de macroações por variação do suporte. O algoritmo também consegue identificar macroações em intervalos de suporte que os demais não conseguem. Especialmente nas raças *Terran* e *Protoss*. Algumas macroações são encontradas com suporte variando de 70% a 50% o que não é comum em algoritmos de mineração sequencial. Esses tendem a estabilizar os itens e definir padrões com baixas porcentagens de suporte.

A Figura 49 mostra os resultados do experimento que mede o tamanho das sequências obtidas com o PrefixSpan na classificação das macroações. O tamanho da sequência é a quantidade de ações que essa possui. Para que a arquitetura tenha opções com diferentes intervalos de tempo é importante que as macroações tenham diferentes tamanhos. Nesse experimento foram feitas 20 classificações completas na base de dados gerando todas as macroações encontradas pelo PrefixSpan e utilizadas nessa tese. Essas macroações vão além daquelas que foram classificadas e são conhecidas como *build-orders*. Essa classificação inclui as novas macroações encontradas nesse trabalho e não catalogadas pelas comunidades de jogadores.

Na Figura 49 para a raça *Terran* existe uma boa variação no tamanho das sequências entre 2 até 50 ações. O tipo mais comum de macroação encontrada é com 14 ações sendo que foram encontradas 16 desse tipo. Foram classificadas também 3 macroações com 40 ações e 2 macroações com 50 ações. Essas possuem uma estratégia longa que demora e média 10 minutos para ser executada por completo no jogo. Na raça *Protoss* a concentração foi em macroações com tamanho de 16 a 20 itens. Já na raça *Zerg* a grande maioria das macroações possui de 20 a 25 itens deixando poucas estratégias com curtos intervalos de tempo. Uma distribuição regular entre o tamanho das sequências e maior frequência de classificação em macroações com tamanhos entre 12 e 18 ações é o ideal.

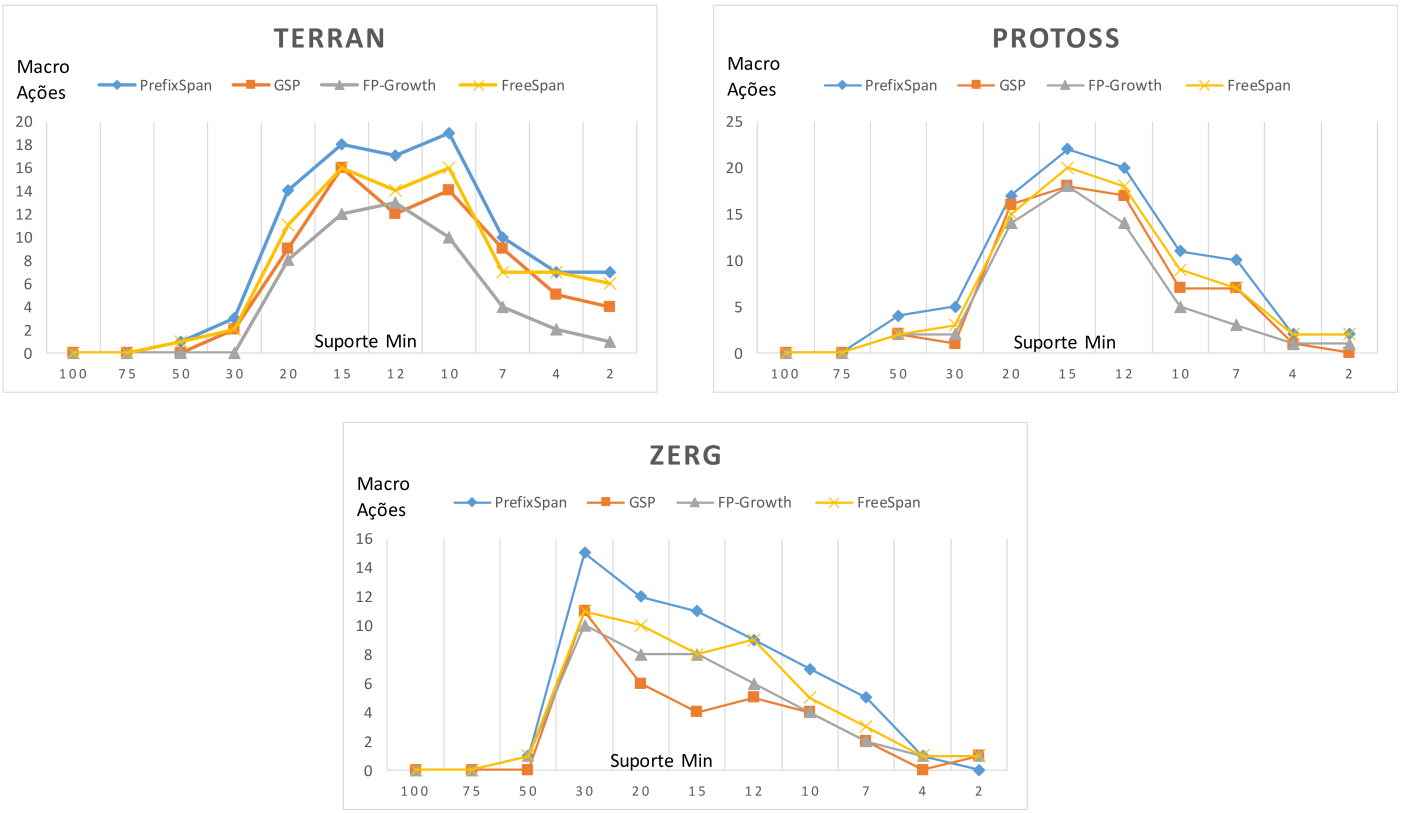

Figura 48 – Resultado dos experimentos de quantidade de macroações classificadas com base na variação do valor do suporte mínimo.

Isso porque macroações com esses tamanhos criam estratégias que levam de 2 a 4 minutos para serem executadas. Esse é um bom intervalo para produção recursos com estratégias bem definidas de ataque ou de expansão dos recursos. Assim a distribuição de tamanho obtida pelo PrefixSpan consegue um bom resultado frente as necessidades da arquitetura proposta.

A Figura 50 mostra os resultados do experimento que mede a quantidade de itens que cada classificador consegue sequenciar em uma macroação. Essa quantidade é com base na variação do suporte mínimo nas execuções. Os itens são as ações e foram feitos dois experimentos com 20 execuções dos classificadores. No primeiro experimento (a) foi escolhida uma base de dados que contém 4 macroações com sequência de 20 ações. No experimento (b) foi escolhida uma base que contém 2 macroações com sequência de 50 ações. Ambos os experimentos foram feitos com a raça *Terran*.

No experimento a da Figura 50 todos os classificadores identificaram pelo menos uma macroação com 20 itens. O PrefixSpan foi o único a identificar todas as 4 macroações, o GSP e FreeSpan conseguiram identificar 3 macroações e o FPGrowth identificou 2 macroações. Em relação ao desempenho o PrefixSpan apresenta a curva mais bem distribuída no gráfico, onde um item foi identificado com suporte em 75% e outros dois itens com suporte em 50%. O algoritmo não apresenta mudança abrupta na quantidade de

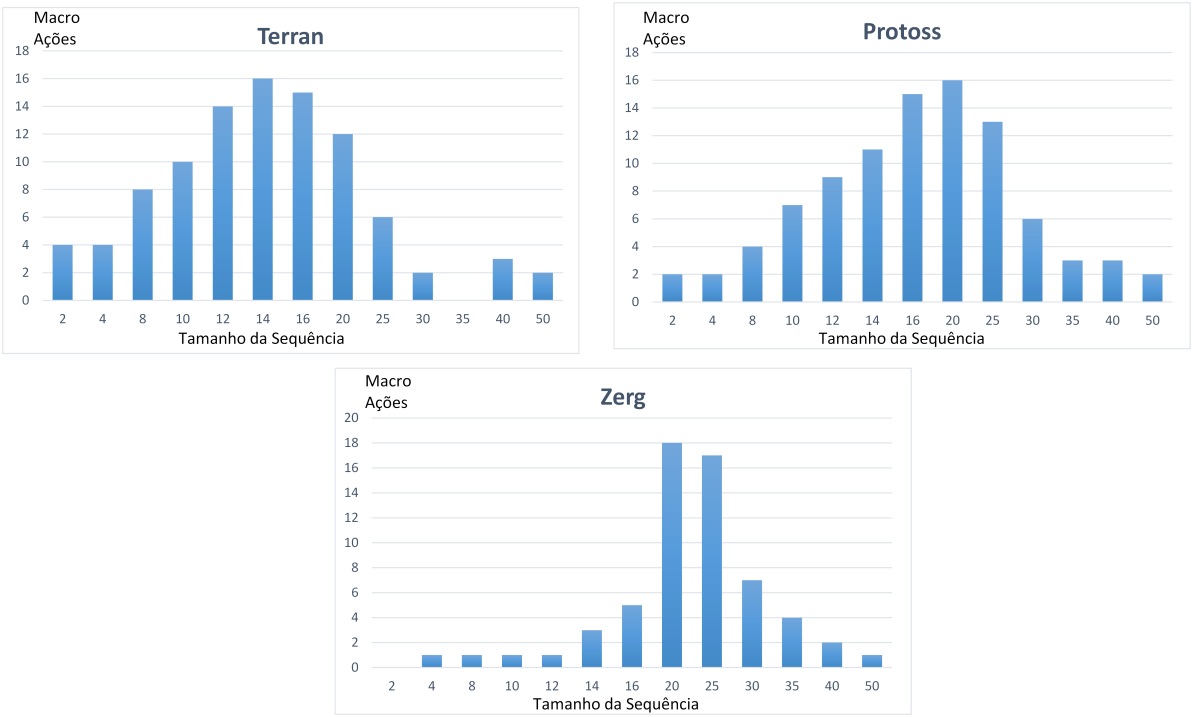

Figura 49 – Resultado dos experimentos que medem o tamanho das macroações classificadas com base no tamanho de suas sequencias de ações.

itens reconhecidos com a variação do suporte, sendo esse o comportamento ideal para identificar um número maior de macroações. O Freespan apresenta uma curva interessante para o problema porém com uma pequena queda quando o suporte está em 4%. Caso o último item não fosse encontrado no suporte em 2% a macroação não teria as 20 ações que a compõem e não seria identificada. O GSP apresenta o comportamento menos uniforme encontrando 14 dos 20 itens com os limites de 15% a 10% do suporte. Isso faz com que o algoritmo não consiga sequenciar todas as macroações possíveis. Quanto menor o limite mais rígido o algoritmo fica na escolha da sequência ignorando possíveis itens que seriam necessários.

No experimento *b* da Figura 50 foram classificadas duas macroações com 50 itens cada. O GSP e FPGrowth não conseguiram classificar as macroações. PrefixSpan classificou as duas e o FreeSpan apenas uma. Novamente a curva do PrefixSpan mostra-se melhor distribuída em relação ao FreeSpan. Os primeiros itens são encontrados e sequenciados com o suporte em 75%. O FreeSpan mesmo encontrando itens a partir de 50% do limite do suporte constrói as sequências com mais da metade das ações que compõe a macroação entre os limítes 20% e 10%. Considerando que sequenciar uma macroação com 50 itens é uma tarefa complexa o Freespan obteve um bom resultado. Mas ao não encontrar itens em valores maiores de suporte uma das macroações do experimento não foi classificada. Assim o PrefixSpan com as estratégias propostas nesta tese mostrou-se o melhor classificador para macroações de *Estratégia* em ambientes de jogos RTS. O algoritmo foi capaz de

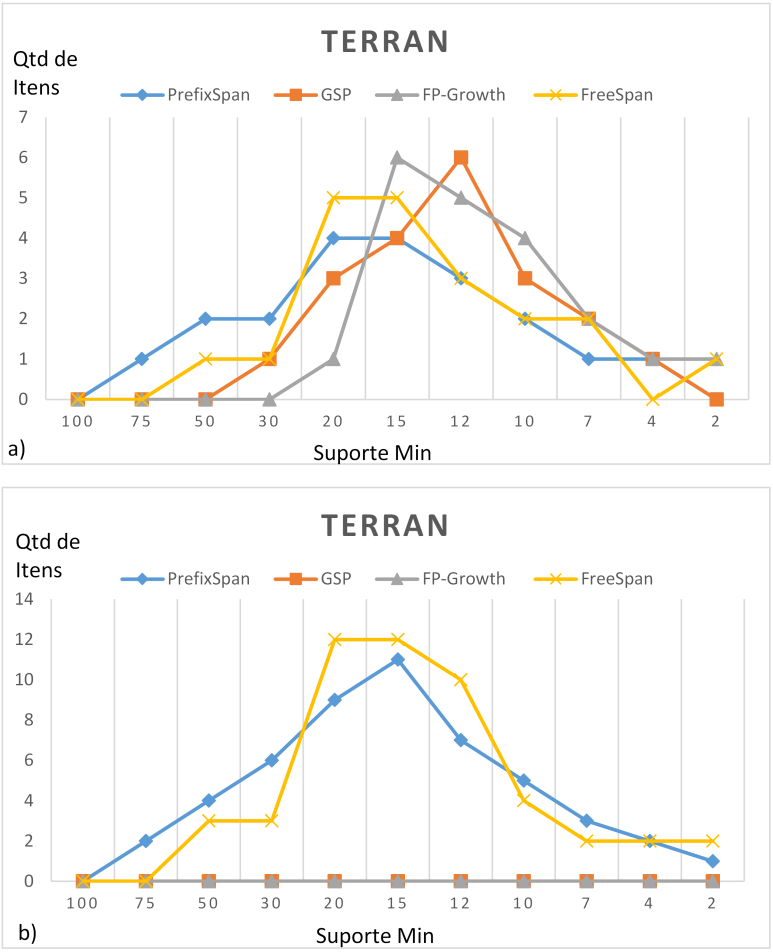

Figura 50 – Macroação da árvore convertida em código para execução.

superar classificadores comumente utilizados como KMN e outros algoritmos de mineração sequencial.

8.2 Análise da Tomada de Decisão e Planejamento

Para avaliar o módulo *Decisão* da arquitetura são feitos testes com a abordagem de POMDP *online* proposta. Experimentos de desempenho são feitos junto ao StarCraft. Os métodos utilizados nos experimentos são: POMDP *online* que é a arquitetura completa proposta nesta tese; AEMS(tempo) que é política utilizada na arquitetura do POMDP *online* com aditivo da regressão linear com tempo do jogo ao cálculo do $P(a|b)$; AEMS é o uso do algoritmo tradicional sem o aditivo de tempo do jogo; DESPOT (SOMANI et al., 2013) é uma política para POMDP *online* que utiliza o conceito de cenários e regularização da árvore de expansão; RTSSample é o algoritmo com amostragem de cenários para jogos RTS proposto nesta tese. Em todos os experimentos foram utilizadas as três raças do jogo de forma aleatória em cada execução de uma partida.

Tabela 14 – Teste de perfomance com 50 execuções do ambiente do StarCraft utilizando mapas aleatórios. Todos os resultados com intervalo médio de confiança de ± 0.95

Algoritmo	Tempo (seg.)	Estados de Crença	Recompensa	Tempo Offline	Redução Erro dos Limites
N. Ações $\|A\| = 26$		N. Observações $\|O\| = 48$		N. Oponentes = 1	
POMDP Online	0,688	6532	19,5 ± 0,7	387,07	23,9%
AEMS	0,896	4256	17,2 ± 0,7	272,23	23,6%
AEMS (tempo)	0,981	4785	19,0 ± 0,7	272,23	27,2%
DESPOT	0,934	2143	17,5 ± 0,7	368,74	26,1%
RTSSample	0,489	1747	20,4 ± 0,7	114,87	16,4%

A Tabela 14 apresenta os resultados para o teste de performance e qualidade utilizando os métodos propostos em partidas do StarCraft. Todos os resultados estão descritos em relação à média dos valores, exceto os estados de crença e tempo *offline*. A equação de redução do erro dos limites é dada por $BR(b) = 1 - \frac{U_T(b) - L_T(b)}{U(b) - L(b)}$, sendo essa a diferença entre os valores de limite obtidos *offline* e os obtidos durante a execução *online*. Em todas as execuções o inimigo era a IA padrão do jogo configura com o mais alto nível de dificuldade e nenhuma partida teve duração maior que 9 minutos. O POMDP *online* conseguiu uma boa média de tempo de execução para escolha de cada macroação. Esse desempenho é reflexo dos valores obtidos pelos algoritmos internos AEMS(tempo) e RTSSample. O melhor desempenho de tempo foi do RTSSample utilizando o limite de amostras e profundidade para explorar a árvore de expansão. O AEMS com estratégia padrão conseguiu melhor tempo que o AEMS(tempo), pois a estratégia padrão reduz o número de comparações em relação a abordagem com aditivo de tempo proposta. O DESPOT conseguiu um bom resultado superando o AEMS. Com a proposta de árvore regularizada do DESPOT processar grandes quantidades de observações não afeta a performance do algoritmo. Diferente do AEMS que diminui o valor médio das recompensas obtidas a medida que o número de observações aumenta.

Em relação a quantidade de estados de crença o RTSSample foi o mais eficiente devido a estratégia de amostragem e os valores de K e D utilizados. Além disso o algoritmo é utilizado apenas em situações que o modelo do POMDP muda durante a partida. Isso faz com que sua utilização seja de 30% a 40% menor que o AEMS(tempo) dependo da

partida. Em relação à média da recompensa total o RTSSample conseguiu o melhor resultado devido a integração dos valores da regressão de tempo do jogo ao cálculo do UCB. Esses valores geram amostragem com observações de confrontos que ainda não foram exploradas durante a partida. Essas estratégias tornam o algoritmo eficiente em seu propósito de uso quando uma partida muda sua dinâmica. O AEMS(tempo) conseguiu valores um pouco melhores que o AEMS tradicional nas recompensas e na redução do erro dos limites. Em ambos os casos foi devido a adição do tempo de jogo a escolha da melhor macroação na árvore. No cálculo $P(a|b)$ as recompensas obtidas são maiores quando a partida está em uma dinâmica de confronto. Os valores obtidos conseguem superar os limites obtidos *offline* aumentando a redução do erro. O gasto de tempo de execução para calcular os limites iniciais é dado pelo tempo *offline*. O RTSSample utilizou menos tempo pois calcula apenas o limite máximo e utiliza o algoritmo de Rollout para essa tarefa.

Na Tabela 15 o objetivo foi testar os métodos em partidas onde o ambiente muda constantemente e o número de observações é alto. A quantidade de oponentes foi definida em 3 sendo cada um desses uma base de recursos em uma posição do mapa onde as unidades desenvolvidas não podem ser vistas pelo agente. Esse deve atacar e defender a partir de qualquer um dos oponentes. O tempo para escolha de cada ação foi definido em 1 segundo pois com a quantidade maior de confrontos as decisões precisam ser feitas com maior rapidez. O RTSSample foi utilizado em mais de 70% do tempo e algumas partidas tiveram duração acima de 14 minutos. O POMDP *online* conseguiu obter respostas dentro das restrições de tempo em todas as partidas, o RTSSample escolheu a maioria das macroações. A média das recompensas diminuiu, porém o AEMS(tempo) e DESPOT não conseguiram retornar respostas na maior parte das partidas. O algoritmo obteve boas recompensas nos minutos iniciais das partidas. Porém quando o ambiente muda e unidades de diferentes oponentes começam a surgir no mapa o algoritmo não consegue retornar uma resposta dentro do limite de tempo esperado. Esse cenário demonstra a importância de uma estratégia para ambientes dinâmicos e quando esses sofrem mudanças. RTSSample manteve uma quantidade regular de amostras e nós crenças com a redução do erro próxima dos 15%.

A Figura 51 mostra o gráfico do experimento entre AEMS(tempo) e RTSSample. Nesse foram testados os valores de recompensa de ambos em partidas onde o ambiente do jogo sofre poucas ou nenhuma mudança. Foram disputadas 30 partidas e cada uma teve duração de 450 segundos com o oponente não fazendo nenhuma intervenção que alterasse o modelo do POMDP. O AEMS(tempo) conseguiu atingir uma maior recompensa de 19,6 e o RTSSAmple de 14,2. Essa vantagem acontece porque o AEMS(tempo) têm seus limites calculados com base em diversos *replays* partidas de modo *offline*, sem nenhuma tendência a situações específicas. O cálculo heurístico do AEMS(tempo) conduz para escolha de macroações melhores e mais promissoras no futuro. O algoritmo também leva vantagem no tempo gasto para escolher cada ação quando o jogo não está em dinâmica

Tabela 15 – Teste de perfomance com 25 execuções do ambiente do StarCraft utilizando mapas aleatórios. Todos os resultados com intervalo médio de confiança de ± 0.92

Algoritmo	Tempo (seg.)	Estados de Crença	Recompensa	Tempo Offline	Limite Redução do Erro
N. Ações $\|A\| = 32$		N. Observações $\|O\| = 48$		N. Oponentes = 4	
POMDP Online	0,846	9873	14,2 ± 0,29	403,29	11,1%
AEMS (tempo)	0,967	4805	18,4 ± 0,6	278,02	27,6%
DESPOT	0,894	2896	16,7 ± 0,9	377,98	25,4%
RTSSample	0,812	3341	13,6 ± 0,32	125,27	13,3%

de confronto. O RTSSample acaba perdendo tempo tentando amostrar estados a partir de observações de confronto que ainda não foram testadas. Mesmo quando o modelo do POMDP muda o AEMS consegue bons resultados e em algumas situações melhores que o RTSSample. Isso confirma o bom desempenho e a importância do algoritmo quando o jogo não está em dinâmicas de confronto.

A Figura 52 mostra outro experimento entre o AEMS(tempo) e RTSSample. Foram capturadas as médias de recompensa sem descontos durante as partidas disputadas com as três raças do jogo. Ao todo foram 30 partidas disputadas com duração de 450 segundos cada. Ambos os algoritmos atingiram bons valores de recompensa ao final das partidas. AEMS(tempo) é executado desde o início da partida e acumula recompensas positivas até 180 segundos de jogo. A partir desse momento os recursos do inimigo começam a ser detectados no mapa. Mesmo sem confronto direto o valor das recompensas diminui com recursos inimigos identificados e não destruídos. O decaimento das recompensas atinge o menor valor entre 240 e 260 segundos quando o confronto direto é iniciado e recursos do agente são destruídos. O aumento nas recompensas até o final da partida reflete as vitórias do agente após o confronto. Na parte final das partidas vários recursos do inimigo são destruídos e as recompensas atingem o maior valor. Nas partidas onde o agente saiu derrotado o AEMS(tempo) quando começa a atingir recompensas negativas não consegue mais recuperar esses valores. Uma derrota geralmente significa que nos primeiros confrontos o agente já perde recursos e em seguida o inimigo ataca sua base diretamente destruindo todos os recursos restantes.

O RTSSample atinge um equilíbrio em relação ao AEMS(tempo) na Figura 52. Os valores de recompensa do algoritmo ficam em 0 até os 180 segundos de partida. Isso

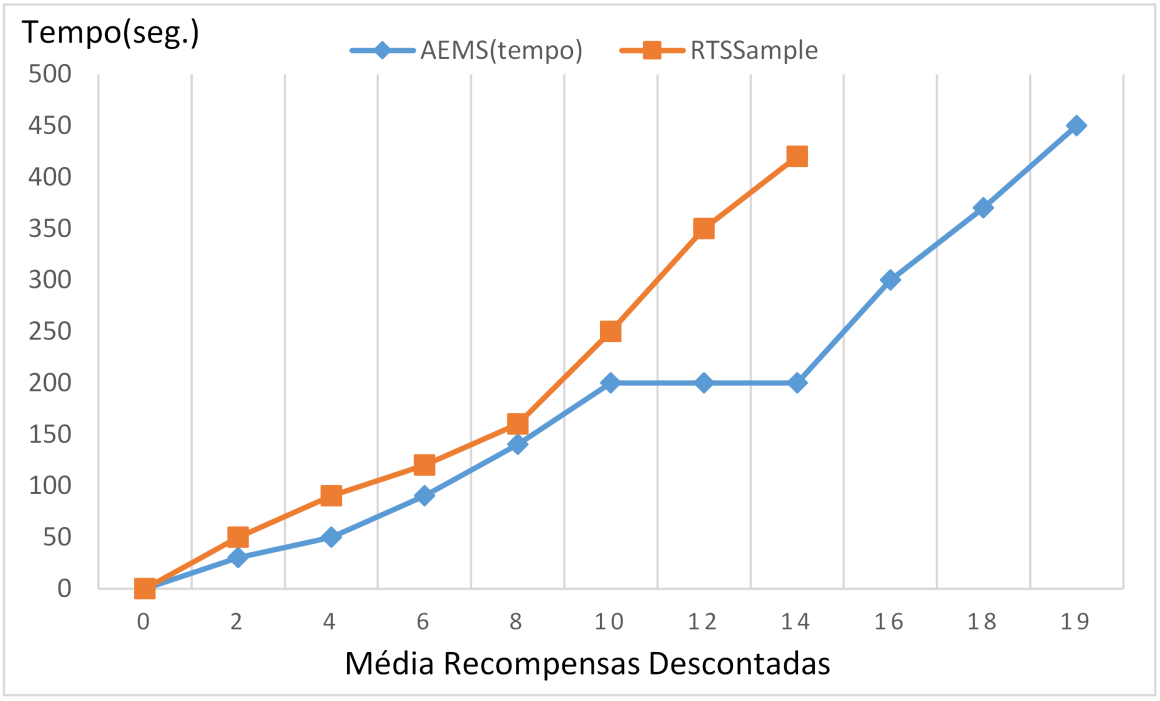

Figura 51 – Resultados dos testes de perfomance em 30 execuções no ambiente do StarCraft utilizando mapas aleatórios. Todos os resultados com intervalo médio de confiança de ± 0.95.

ocorre devido as mudanças no ambiente da partida que até 180 segundos não haviam sido identificadas. A partir desse momento o RTSSample inicia sua execução. Com situação de confronto eminente ou imediato o RTSSample consegue recompensas melhores mesmo com os descontos aplicados pelas unidades inimigas detectadas. O algoritmo diferente do AEMS(tempo) gera amostras de cenários que simulam diversas possibilidades de ataques ou defesas contra o inimigo. Com essas estratégias efetuadas com sucesso a recompensa aumenta quando unidades do inimigo são destruídas. O AEMS(tempo) explora principalmente estados defensivos e de contra-ataque devido as recompensas negativas já adquiridas. Essas estratégias geralmente resultam em perda de unidades pois os confrontos são feitos com o inimigo chegando até a base do agente. Nessas situações a exposição dos recursos de edificações do agente geram valores baixos de recompensas. O RTSSample obtêm recompensas menores entre 230 e 260 segundos. Nesse intervalo confrontos diretos acontecem e algumas unidades são destruídas, mas a estratégia ainda prevalece com o aumento das recompensas no restante da partida. Esse aumento reflete também as vitórias do agente nas partidas.

A Figura 53 mostra os resultados do experimento envolvendo os três algoritmos de POMDP. Nesse foram disputadas 300 partidas do jogo utilizando as raças *Protoss* e *Terran*. O objetivo do experimento é avaliar a evolução e estabilidade dos algoritmos em relação a recompensa a medida que mais partidas vão sendo disputadas. As partidas são

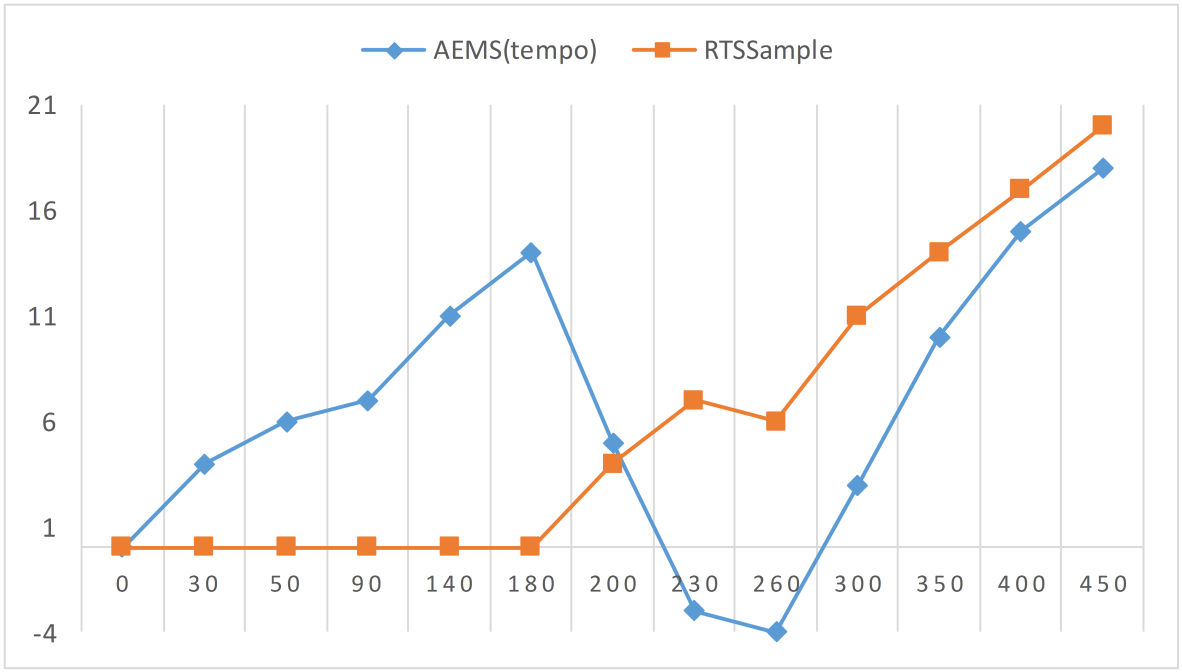

Figura 52 – Resultados dos testes de perfomance em 30 execuções no ambiente do StarCraft utilizando mapas aleatórios. Todos os resultados com intervalo médio de confiança de ± 0.95.

disputadas em conjunto entre as duas raças e diferentes resultados em relação a vitória ou derrota surgem. POMDP *online* proposto alcançou a melhor média de recompensas e maior estabilidade entre os resultados. A variação entre essas foi pequena e o valor médio final superou os valores iniciais e intermediários de recompensas. O DESPOT e AEMS mantiveram um equilíbrio entre os resultados mas com menor estabilidade. DESPOT com média de resultados maiores que o AEMS e com uma queda maior ná média das partidas entre 220 e 250. O AEMS obteve mais variações com menores intervalos, chegando ao final com uma média inferior a inicial.

8.3 Análise da Arquitetura Proposta em Relação ao Estado da Arte

Nesta seção a arquitetura proposta (POMDP *online*) será avaliada em relação a outras abordagens do estado da arte e jogadores humanos. No estado da arte existem *bots* desenvolvidos para competições de IA junto ao jogo StarCraft, como mencionado no Capítulo 1. Para os experimentos serão utilizados 3 *bots* que são: Tscmoo[1] o campeão da edição 2015 da AIIDE; UAlbertaBot[2] 4º na competição de 2015 e campeão da edição 2013;

[1] https://github.com/tscmoo/tsc-bwai
[2] https://github.com/davechurchill/ualbertabot

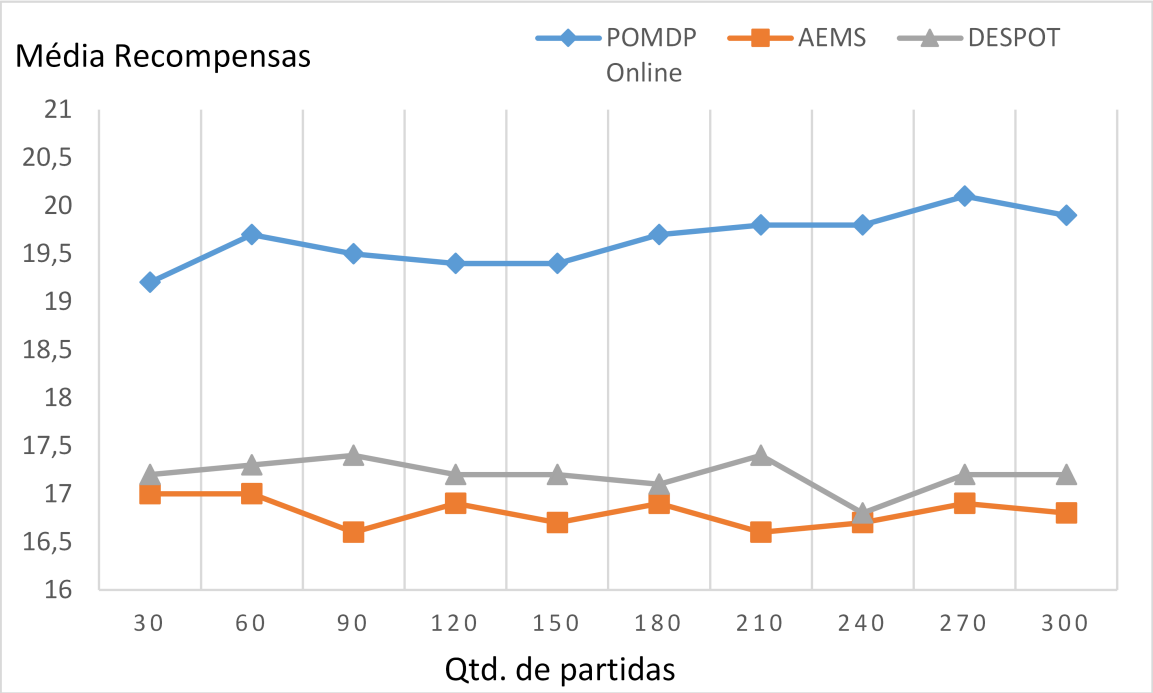

Figura 53 – Resultados dos testes de perfomance com 300 execuções no ambiente do StarCraft utilizando mapas aleatórios. Todos os resultados com intervalo médio de confiança de ± 0.95.

Nova [3] 13º colocado da edição de 2015 da competição. O Tscmoo foi escolhido por ter sido o campeão. UAlbertaBot foi escolhido por ser o *bot* que possui a maior quantidade de módulos construídos com técnicas de IA em relação aos demais da competição. O *bot* Nova foi escolhido pois possui módulos de controle reativo com uso de IA tornando-se um confronto interessante para arquitetura proposta. A arquitetura proposta será referenciada nos próximos experimentos pelo nome POMDP *online*. Serão utilizadas as versões da competição de 2015 pois não houve edição do torneio em 2016 e até o momento da escrita dessa tese não foi realizada a edição 2017. Além dos *bots* serão feitos experimentos contra jogadores humanos e esses irão avaliar a arquitetura proposta em diversos aspectos de sua funcionalidade.

Nos experimentos são feitas dez execuções da arquitetura proposta contra cada um dos *bots* em mapas aleatórios do jogo. Em cada execução são mostradas as seguintes informações sobre o POMDP *online*: (RP) a quantidade de recursos produzidos na partida; (RD) a quantidade de recursos inimigos que foram destruídos na partida; (ME) quantidade de macroações executadas pelo POMDP *online*; (MC) a quantidade de macroações canceladas pelo planejamento reativo; (RM) a média de recompensa obtida pelo POMDP *online*; (Duração) o tempo de duração da partida; (Venceu) se o POMDP *online* venceu ou não a partida. Em cada experimento nas 5 primeiras partidas a arquitetura jogou

[3] https://bitbucket.org/auriarte/nova/overview

com a raça *Terran* e nas demais com a raça *Protoss*. Não será utilizada a raça *Zerg* pelo POMDP *online* devido a dinâmica dos recursos e baixas recompensas conseguidas com essa. A grande maioria dos *bots* de competições possuem poucas funções executadas com técnicas de IA. Esses executam scripts que simulam algum *build order* da raça que é utilizada. Mesmo com essas limitações no comportamento dos *bots* esses ainda representam um desafio interessante para testes.

Tabela 16 – Confrontos entre POMDP *Online* e *bot* Tscmoo.

POMDP *Online* x Tscmoo							
	Duração	**RP**	**RD**	**ME**	**MC**	**RM**	**Venceu**
1	13m.25s	23	7	25	17	3,4	Não
2	15m.04s	24	5	23	17	-1,2	Não
3	13m.413s	24	9	28	19	4,0	Não
4	14m.41s	20	23	22	7	17,2	Sim
5	14m.17s	25	8	26	17	3,7	Não
6	13m.21s	24	22	20	9	17,5	Sim
7	12m.52s	22	4	29	14	-0,5	Não
8	14m.36s	25	21	23	9	16,8	Sim
9	15m.07s	25	4	27	16	0,15	Não
10	15m.19s	25	23	23	11	17,2	Sim

A Tabela 16 mostra os resultados do POMDP *online* contra o Tscmoo. Das dez partidas a arquitetura foi capaz de vencer quatro. O Tscmoo joga com a raça *Zerg* e possui uma série de estratégias de *build orders* em sua estrutura. Durante a partida ele opta por um desses e troca sua estratégia escolhida apenas se inimigos forem vistos. O *bot* utiliza como técnicas de IA mapas de risco para realizar reconhecimento do mapa e heurísticas de posição para escolher as unidades que serão atacadas.

Na maioria das partidas em que o Tscmoo saiu vitorioso ele conseguiu executar o primeiro ataque na base de recursos do POMDP. A estratégia envolve a produção consecutiva e rápida de um mesmo tipo de unidade e uso de um *build order* para movimentação. Nas execuções 1, 3 e 9 o POMDP *online* efetuou a primeira investida de ataque mas o adversário já tinha unidades suficientes para estabelecer uma defesa. Na execução 6 o POMDP *online* efetuou o ataque com uma maior quantidade de unidades e utilizando movimentação em grupo pelo mapa. Já na execução 4 o POMDP optou por duas estratégias de construção de unidades que efetuam ataques aéreos. Esse tipo de ataque é um ponto fraco do Tscmoo que não possui estratégias de contra-ataque para essa situação. Nessa execução a quantidade de RP foi menor que as demais devido ao custo de produção das unidades aéreas que é maior. Em relação as macroações o POMDP manteve uma média de execução em todas as partidas. Naquelas em que saiu vitorioso o número de ME foi menor pois as macroações escolhidas estavam retornando boas recompensas e foram executadas até o fim. Reflexo disso é a quantidade de macro ações canceladas que também é menor nas partidas com vitória.

Tabela 17 – Confrontos entre POMDP *Online* e *bot UAlbertaBot*.

POMDP *Online* x UAlbertaBot							
	Duração	**RP**	**RD**	**ME**	**MC**	**RM**	**Venceu**
1	10m.14s	21	18	19	12	17,1	Sim
2	8m.30s	19	17	18	14	16,6	Sim
3	8m.13s	19	5	26	17	-5,0	Não
4	8m.48s	18	5	26	16	-3,0	Não
5	10m.24s	22	18	20	11	16,8	Sim
6	9m.36s	23	6	25	17	1,0	Não
7	8m.44s	23	16	18	12	16,9	Sim
8	9m.25s	24	6	26	18	-4,3	Não
9	8m.21s	22	18	17	11	16,8	Sim
10	10m.12s	25	6	25	16	-3,41	Não

Na Tabela 17 estão os resultados dos confrontos contra o UAlbertaBot. Nesse experimento o POMDP foi capaz de vencer cinco das dez partidas disputadas. O UAlbertaBot joga com a raça *Protoss* e utiliza algoritmos de busca na montagem das estratégias. O *bot* também possui um escalonador de ações. O *bot* utiliza um simulador de combate para predição de resultados com um modelo de predição de estratégia do adversário. Em todas as execuções em que o UAlbertaBot saiu vencedor ele efetuou o primeiro ataque utilizando um *build order* rápido. Em todas essas partidas o ataque começa em média aos 3m:40s de partida sendo o *bot* mais rápido em iniciar um ataque. Na execução 1 da Tabela 17 o POMDP sofreu a primeira investida de ataque, mas com um número maior de unidades e uma *Tática* de agrupamento saiu vencedor. Na execução de número 9 o POMDP disputou a partida com a raça *Protoss*. Nessa foi utilizada uma *Estratégia* de ataque rápido com muitas unidades do recurso *Zealot* que surpreendeu o adversário. O POMDP obteve menores quantidades de RP e ME devido ao caráter ofensivo do UAlbertaBot que deixou as partidas com uma menor duração de tempo. As partidas tiveram uma maior quantidade de confrontos entre as unidades no seu decorrer. Com isso as RM ficaram maiores que no experimento anterior principalmente nas partidas em que o POMDP saiu vitorioso. Destruir mais unidades inimigas em uma partida faz com que o valor médio da recompensa aumente.

Os resultados das partidas contra o *bot* Nova estão na Tabela 18. Nesse experimento o POMDP conseguiu superar o inimigo em 80% das partidas com oito vitórias. O Nova joga com a raça *Terran* e utiliza uma máquina de estados finitos para controlar 3 possíveis *build orders*. Os *build orders* são para executar uma estratégia de ataque, ou para um contra-ataque ou para uma defesa. Para controle reativo é utilizada uma técnica de campos potenciais para gerenciar as unidades em combate. Com essa técnica cada unidade se aproxima do adversário e dispara um ataque recuando ao mesmo tempo para que o adversário não possa atingi-la. Nas partidas em que o POMDP não venceu ele foi derrotado em confrontos diretos devido a maior produção de unidades do adversário.

Tabela 18 – Confrontos entre POMDP *Online* e *bot* Nova.

POMDP *Online* x Nova							
	Duração	**RP**	**RD**	**ME**	**MC**	**RM**	**Venceu**
1	10m.35s	20	15	17	9	19,2	Sim
2	09m.44s	18	14	18	14	7	Não
3	10m.53s	19	15	14	9	19,0	Sim
4	11m.21s	21	17	15	11	18,8	Sim
5	10m.36s	19	6	19	15	8	Não
6	10m.54s	24	17	14	10	18,6	Sim
7	11m.25s	24	18	15	8	19,1	Sim
8	09m.42s	20	15	14	9	18,7	Sim
9	10m.50s	22	16	15	11	18,9	Sim
10	11m.14s	23	17	14	10	19,2	Sim

Os próximos experimentos serão feitos entre a arquitetura proposta POMDP *online* e jogadores humanos. Serão utilizados três jogadores sendo cada um desses no nível iniciante, intermediário, experiente. Para definir a experiência de cada jogador em seu respectivo nível de jogo os seguintes critérios foram adotados: iniciante é o jogador que teve pouco contato o StarCraft jogando no máximo uma vez por semana por pelo menos 3 meses; intermediário é o jogador que interage com o jogo três vezes por semana por pelo menos 1 ano; experiente é o jogador que interage com o jogo mais de três vezes por semana durante pelo menos 2 anos. Esses serão referenciados no texto como iniciantes ($jogadorI$), intermediário ($jogadorT$) e experiente ($jogadorE$). O objetivo dos experimentos com jogadores é mensurar o quanto a arquitetura proposta pode competir com esses e avaliação que eles fazem sobre o agente jogador. As técnicas nesta tese foram escolhidas para aproximar a forma de controlar as tarefas do jogo daquela feitas por um jogador humano. Como nos experimentos anteriores serão executadas 10 partidas entre o POMDP *online* e cada jogador. Como métrica de avaliação serão utilizados os mesmos parâmetros dos experimentos contras os *bots*.

A Tabela 19 mostra os resultados das partidas contra o $jogadorI$. Nessa o POMDP *online* foi capaz de vencer todas as disputas. Em todas as partidas exceto nas execuções 4, 5 e 8 a arquitetura efetuou o primeiro ataque. Nas execuções 3 e 10 o $jogadorI$ não foi abatido no primeiro ataque, o POMDP *online* produziu novas unidades e efetuou um segundo ataque para sair vencedor. Nas execuções onde o $jogadorI$ atacou primeiro a arquitetura conseguiu defender-se efetuando um contra-ataque para vencer. Nas partidas contra o $jogadorI$ foi possível observar diversos comportamentos desejados da arquitetura, tais como efetuar uma defesa e contra-atacar em seguida. As macroações de $Tática$ foram muito efetivas criando grupos de unidades e movimentando essas para derrotar as unidades adversárias. O POMDP *online* conseguiu vencer as partidas relativamente rápido e isso reflete nos valores dos parâmetros. A quantidade de RP, RD e ME foram menores que em outros experimentos. Foi preciso executar poucas macroações para vencer

e o adversário não produzia muitos recursos. Diante disso poucas macroações foram canceladas e na execução 4 nenhuma foi cancelada. A recompensa média de cada execução foi maior que nos experimentos anteriores devido ao *jogadorI* quase não destruir unidades da arquitetura. É possível perceber uma curva de aprendizado do *jogadorI* que foi abatido rapidamente nas 5 primeiras execuções e conseguiu partidas mais longas nas demais execuções. Com isso a recompensa média das execuções 7, 8 e 10 foram as menores devido a algumas unidades do POMDP *online* que foram destruídas.

Tabela 19 – Confrontos entre POMDP *Online* e jogador iniciante.

POMDP *Online* x *jogadorI*							
	Duração	**RP**	**RD**	**ME**	**MC**	**RM**	**Venceu**
1	07m:32s	16	9	8	3	27	Sim
2	06m.58s	14	8	7	1	25	Sim
3	07m.01s	15	8	7	2	26	Sim
4	06m.15s	13	7	6	0	29	Sim
5	06m.44s	13	7	7	1	24	Sim
6	07m.39s	16	8	8	2	26	Sim
7	08m.11s	17	10	10	3	23	Sim
8	08m.21s	16	10	11	4	22	Sim
9	07m.37s	16	9	9	2	25	Sim
10	08m.09s	17	10	10	3	24	Sim

A Tabela 20 mostra os resultados das partidas com o *jogadorT*. Nesse experimento a arquitetura foi capaz de vencer três das dez partidas disputadas. O jogador com nível intermediário mostrou-se capaz de superar o POMDP *online*. Nas partidas em que a arquitetura não venceu houve alternância dos jogadores em relação ao primeiro ataque. Mesmo com a arquitetura atacando primeiro o *jogadorT* foi capaz de defender utilizando em seguida um contra-ataque para vencer. A principal habilidade do *jogadorT* que influenciou diretamente nos resultados foi a sua capacidade de predição em relação as estratégias da arquitetura. Mesmo com variações nos ataques feitos o *jogadorT* conseguia prever e observar a chegada de um ataque posicionando suas unidades para defesa. O *jogdorT* também conseguiu produzir mais recursos que a arquitetura na maioria das partidas. Em todas as partidas com vitória a arquitetura efetuou mais de um ataque para vencer. A média das recompensas nas partidas com vitória não foi maior porque o *jogadorT* conseguiu abater diversas unidades do agente. Os recursos produzidos e macroações executadas mantiveram-se constantes em quase todas as partidas. Mesmo quando o *jogadorT* efetuava o primeiro ataque a arquitetura conseguia montar uma defesa aumentando o tempo de duração das partidas.

Na Tabela 21 estão os resultados das partidas contra o *jogadorE*. Nesse experimento a arquitetura não foi capaz de vencer nenhuma partida. O *jogadorE* possui diversas habilidades que fizeram o mesmo vencer todas as partidas. Entre essas habilidades destaque

Tabela 20 – Confrontos entre POMDP *Online* e jogador intermediário.

POMDP Online x *jogadorT*							
	Duração	**RP**	**RD**	**ME**	**MC**	**RM**	**Venceu**
1	11m:45s	21	12	21	10	-3,3	Não
2	12m.08s	22	7	20	12	-2,1	Não
3	11m.54s	22	24	23	9	12,3	Sim
4	10m.47s	20	16	17	12	-5,7	Não
5	12m.21s	21	19	20	10	11,8	Sim
6	10m.58s	21	9	18	14	-6,8	Não
7	11m.03s	19	13	23	11	-3,0	Não
8	11m.31s	19	12	21	9	-1,7	Não
9	11m.53s	20	23	21	11	15,4	Sim
10	12m.14s	23	17	18	15	-3,6	Não

para a capacidade de predição, velocidade na produção de recursos e uso de estratégias. O *jogadorE* superou a produção de recursos do POMDP *online* e sempre efetuava o primeiro ataque com pouco tempo de jogo. A quantidade de unidades do adversário sempre era maior e os ataques feitos com grupos desconexos que atacavam em locais diferentes. Isso dificulta a tarefa de defesa da arquitetura que precisa raciocinar sob os vários espaços da base que estão sofrendo ataque. A quantidade de recursos produzidos e macroações executadas foi baixo em relação aos outros experimentos. Com ataque rápido do *jogadorE* a arquitetura não conseguiu produzir muitos recursos interrompendo as macroações e reconsiderando outras de defesa. A quantidade macroações canceladas também foi baixo sendo todos os cancelamentos feitos para lidar com ataque do adversário. Em algumas partidas não houveram cancelamentos por derrota precoce do POMDP *online*. Ao concentrar-se em uma estratégia de produção rápida utilizando unidades com um ataque surpresa com poucos minutos o *jogadorE* conseguia surpreender a arquitetura. O POMDP *online* mesmo quando optava por uma estratégia de produção rápida de unidades ainda estava em produção de recursos quando os ataques ocorriam. Assim o número de unidades de combate sempre era inferior ao do *jogadorE*.

O próximo experimento foi realizado para mensurar a qualidade da arquitetura proposta como um agente jogador em relação ao entendimento dos jogadores humanos. Foi proposto um questionário com perguntas em relação aos principais aspectos técnicos e comportamentais desejáveis em um jogo RTS. Foram entrevistados 10 novos jogadores de cada nível entre iniciante, intermediário e experiente. Para cada pergunta é possível atribuir uma nota que varia de 1 a 5 sendo essas: 1 (ótimo); 2 (bom); 3 (regular); 4(ruim); 5(péssimo). Foram calculadas as médias de resposta de cada pergunta para cada grupo de jogadores. O valor dessas respostas foram atribuídas de forma qualitativa na Tabela 22.

Em relação as avaliações o *jogadorI* atribuiu as melhores notas classificando todas

Tabela 21 – Confrontos entre POMDP *Online* e jogador experiente.

POMDP Online x *jogadorE*							
	Duração	**RP**	**RD**	**ME**	**MC**	**RM**	**Venceu**
1	08m:13s	15	5	13	4	-5,6	Não
2	07m.33s	14	6	13	3	–4,2	Não
3	07m.24s	14	4	10	3	-4,4	Não
4	07m.05s	13	5	11	0	-3,8	Não
5	07m.42s	14	6	9	1	-3,6	Não
6	06m.49s	12	6	9	1	-4,1	Não
7	07m.17s	13	5	10	3	-3,6	Não
8	08m03s	13	4	12	2	-2,2	Não
9	06m.57s	12	5	13	0	-3,8	Não
10	07m.31s	14	5	11	1	-4,9	Não

as perguntas com ótimo ou bom. Com um conhecimento limitado do jogo e disputando partidas contra o POMDP *online* o grupo de jogadores iniciantes se surpreenderam com a capacidade da arquitetura. Ao avaliar o item de "qualidade como jogador" com valor ótimo, o grupo *jogadorI* entendeu que o nível de dificuldade ao confrontar a arquitetura é alto. Reflexo disso é atribuição de ótimo também ao item eficiência em atacar. O grupo *jogadorT* possui uma visão mais ampla do jogo e seus desafios. Isso reflete nas notas atribuídas que foram mais rigorosas que no grupo anterior de jogadores. Esse grupo não atribuiu nenhuma avaliação ótima e alternou entre bom e regular. A qualidade da arquitetura como um jogador, eficiência no ataque, gerenciamento de estratégias e produção de recursos foram os destaques nesta avaliação. O grupo *jogadoresT* sentiu-se ameaçado com a mudança de estratégia em diferentes partidas e com as iniciativas de ataque da arquitetura. A quantidade de unidades nos ataques também foi um diferencial segundo o grupo. Entre os pontos negativos o grupo *jogadorT* considera que a defesa não é levada em conta o suficiente nas estratégias, pois ficam poucas unidades na base para protegê-la. Quando a arquitetura está sob ataque o tempo para reunir unidades e efetuar a defesa também ficou abaixo do esperado segundo o grupo *jogadorT*.

O grupo *jogadorE* possuí muita experiência no jogo e o critério adotado por esses foi mais rígido. As avaliações tiveram notas entre bom e ruim. Os itens de gerenciamento de estratégias, produção de recursos e a qualidade como jogador mantiveram uma boa avaliação. O grupo considerou que nesses quesitos a arquitetura consegue competir com jogadores humanos de qualidade. Outro ponto destacado foi a capacidade de reagir a presença de inimigos da arquitetura. Sempre que o inimigo fazia um movimento de reconhecimento da base ou enviava algumas unidades para mais próximo dessa as mesmas eram abatidas. Em muitos casos jogadores humanos não percebem essa movimentação pois a mesma não é feita em um ataque direto e sim de forma furtiva. Entre as deficiências o grupo *jogadorE* também consideram que a defesa deve ser melhorada com uso de mais

Tabela 22 – Questionário para valiar a qualidade da arquitetura proposta do ponto de vista de jogadores humanos.

Questionário			
	JogadorI	JogadorT	JogadorE
Qualidade como jogador	Ótimo	Bom	Bom
Gerenciamento de estratégias	Bom	Bom	Bom
Gerenciamento de táticas	Bom	Regular	Regular
Gerenciamento do c. reativo	Bom	Regular	Ruim
Eficiência em atacar	Ótimo	Bom	Regular
Eficiência em defender	Bom	Regular	Ruim
Qualidade da produção de recursos	Bom	Bom	Bom
Qualidade da movimentação de unidades	Bom	Regular	Regular
Eficiência na reatividade a inimigos	Bom	Bom	Bom
Eficiência na alternância de estratégias	Bom	Bom	Regular
Eficiência na execução de múltiplas tarefas	Bom	Regular	Regular
Eficiência na execução de ações em paralelo	Bom	Bom	Regular
Qualidade do comportamento em relação a jogador humano	Bom	Regular	Regular

unidades na base do agente. Em relação aos confrontos diretos a estratégia de combate foi questionada. Nesses as unidades do agente sempre eram dividas em dois grupos que atacavam duas unidades do inimigo mas esses não eram sempre os mais fortes ou que deveriam ser abatidos primeiro. A estratégia de combate é desenvolvida com macroações de *ControleReativo*. O grupo considera que a quantidade de tarefas executadas em paralelo pode ser maior. Em alguns momentos o agente coordenava movimentação de unidades pelo mapa e a produção de novas unidades estava parada. Sempre que estava sofrendo um ataque a produção de recursos também era paralisada pelo agente.

O comportamento e aspectos funcionais da arquitetura proposta foram bem avaliados pelos grupos de jogadores. O grupo *jogadorI* considera que o POMDP *online* pode ser comparado a um jogador humano e seu comportamento é próximo desse. O grupo *jogadorT* avaliou como regular o comportamento mas ressaltou que é possível confundir o agente com um jogador humano pela alternância de estratégias durante a partida. Já o grupo *jogadorE* disse que em alguns momentos o comportamento é similar ao de um jogador humano principalmente na produção de recursos e estratégias de ataque. Mas ressaltaram que durante a partida é possível perceber no panorama geral que é um agente jogador devido as falhas de defesa e execução de algumas tarefas em conjunto.

8.4 Considerações Finais

Este capítulo apresentou os experimentos feitos junto a arquitetura de planejamento probabilístico proposta. Os testes foram divididos explorando cada módulo e seus algoritmos. Por fim a abordagem completa foi avaliada com outras propostas da literatura e com jogadores humanos no contexto de jogos RTS. O StarCraft foi utilizado como ambiente de testes.

Nos experimentos com o módulo *Classificação* foram analisados os algoritmos PrefixSpan e GSP. Esses foram comparados com outros algoritmos de classificação. A performance foi analisada dentro do problema de classificar macroações em jogos RTS. Os resultados mostram que os algoritmos propostos junto com as estratégias de mineração sequencial são superiores aos do estado da arte. A qualidade das macroações obtidas foi mensurada em relação a outros algoritmos de mineração sequencial. Os resultados apontam boa vantagem do PrefixSpan e GSP sobre os algoritmos FPGrowth e FreeSpan. Foram obtidas diversas macroações em todos os níveis de abstração do jogo com boa variação entre os tamanhos dessas.

O módulo *Decisão* foi analisado em relação a performance do POMDP com as políticas AEMS(tempo) e RTSSample. Testes foram feitos em relação a outros algoritmos de POMDP *online* da literatura. O POMDP *online* superou os algoritmos AEMS e DESPOT na obtenção de recompensas e na redução erro dos limite. As estratégias de heurística e amostragem com base no tempo do jogo mostram-se eficazes dentro do domínio de jogos

RTS. O uso de duas políticas em conjunto mostrou-se eficiente na cobertura das possíveis variações que ocorrem em uma partida do jogo. Entre essas variações destaque para a mudança no ambiente do jogo e do POMDP quando inimigos começam a ser detectados.

Nos experimentos com o estado da arte todos os principais aspectos da arquitetura foram avaliados em confrontos entre *bots* e jogadores. O módulo *Controle* da arquitetura teve suas estratégias medidas nesses experimentos também. A eficacia na execução em paralelo das macroações e cancelamento dessas foi determinante nas vitórias contra *bots* e jogadores humanos. O POMDP *online* mostrou-se apto a competir com os melhores *bots* mesmo estes utilizando técnicas de IA apenas em alguns módulos específicos. O uso de estratégias predefinidas pode ser vista como uma vantagem já que essas são garantidamente eficientes na maioria dos cenários. Contra jogadores humanos a arquitetura foi capaz de impor certa dificuldade para jogadores intermediários vencendo algumas partidas. Na avaliação da qualidade da arquitetura pelos jogadores a mesma apresentou boas notas em quesitos que são pertinentes com o objetivo desse trabalho. Entre esses ressalta-se a produção de recursos, gerenciamento de estratégias, reatividade a eventos inimigos, movimentação de unidades e eficiência em atacar. Assim a tomada de decisão o uso do conceito de macroações e planejamento reativo conseguiram desenvolver um comportamento desejável para a arquitetura proposta como um todo.

Capítulo 9

Conclusão

Este trabalho apresentou uma arquitetura de uso geral baseada em planejamento probabilístico para controle de um agente em jogos RTS. A abordagem proposta tem como principal objetivo realizar todas as tarefas de um jogo RTS dentro das restrições de tempo real. A arquitetura foi dividida em três principais módulos, cada um com seus algoritmos e técnicas modelados para atingir a melhor performance e qualidade possíveis.

Os módulos propostos são *Classificação*, *Decisão* e *Controle*. O módulo *Classificação* é responsável por minerar a base de dados com *replays* de partidas disputadas entre jogadores profissionais. Com isso, são classificadas macroações nos três níveis de abstração do jogo. Essas macroações são responsáveis pela característica de uso geral da arquitetura que as utiliza como objeto de dados e ações em todos os módulos. As macroações são colocadas em uma árvore de predição, para que sejam escolhidas aquelas complementares de mais baixo nível para execução. O módulo *Decisão* decide quais macroações de mais alto nível de abstração serão executadas. Esse módulo utiliza duas políticas de execução modeladas exclusivamente para este trabalho e para o domínio de jogos RTS. O módulo *Controle* gerencia a troca de informações entre o jogo e a arquitetura. O escalonamento das ações que serão executadas no jogo e o controle do planejamento da arquitetura de forma reativa também são feitos nesse módulo.

Os experimentos foram conduzidos para validar primeiramente cada módulo em separado e, por fim, a arquitetura completa. Os resultados mostraram que o módulo *Classificação* obteve bons resultados na identificação de macroações e predição de uso destas. Os algoritmos de classificação da literatura foram superados em todos os aspectos testados. Os resultados obtidos com o módulo *Decisão* comprovam a eficiência do POMDP *online* na tomada de decisão com as macroações durante uma partida. Os resultados obtidos superam outros algoritmos *online* com boas performances. Além disso, os resultados mostram uma evolução nas recompensas obtidas, quando existem mudanças dentro do ambiente da partida que não são levadas em conta por outras abordagens *online*. Nos resultados obtidos com a arquitetura completa, o agente mostrou capacidade de competir contra *bots* e jogadores humanos. Os algoritmos do módulo de *Controle* mostraram

eficácia ao auxiliar a arquitetura na obtenção de vitórias nos confrontos.

A abordagem atingiu todos os resultados esperados e superou as expectativas em alguns quesitos. O controle das tarefas do jogo é mantido de modo eficiente e em paralelo pela arquitetura em todas as partidas. Mesmo aquelas com duração de vários minutos e mais de um adversário no mapa são controladas com eficácia. Quanto mais tempo disponível o algoritmo consegue durante a partida para executar a tomada de decisão, melhores são os resultados. Contudo, em certos momentos, as observações chegam em intervalos de segundos, devido à presença de inimigos, mas os valores obtidos são satisfatórios, mesmo nessas condições. Os confrontos com os *bots* e jogadores humanos tiveram resultados acima do esperado. As vitórias sobre o Tscmoo e o UalbertaBot mostram a capacidade da arquitetura operando de modo integrado, junto com seus algoritmos. Isso ocorre porque esses *bots* são desenvolvidos, há vários anos, especificamente para vencer outros em competições. Suas arquiteturas utilizam poucas técnicas de IA, e grande parte de suas estratégias e planejamento são feitos com conjuntos de ações predefinidos. Em relação aos jogadores, as vitórias em algumas partidas contra aqueles de nível intermediário foram um resultado excelente, dada a capacidade desses jogadores. Em suas avaliações, a arquitetura obteve boas notas em relação à capacidade de gerenciamento das tarefas e comportamento mais próximo ao de jogadores humanos. Essas notas são expressivas, pois superam alguns dos objetivos propostos, e os resultados são balizados inclusive por jogadores humanos experientes.

9.1 Principais Contribuições

Este trabalho apresentou uma arquitetura de uso geral baseada em planejamento probabilístico para Jogos RTS. Na seção 1.2, foram apresentadas as principais contribuições por módulo proposto e com foco nos algoritmos, técnicas e modificações propostas. Nesta seção, serão listadas as principais contribuições com foco na proposta da arquitetura de forma geral e seu funcionamento como um agente integrado. As principais contribuições da proposta são:

- *Proposta de uma arquitetura AUG com base em planejamento probabilístico voltada para jogos RTS e ambientes de tempo real, em geral com tomada de decisão.* A abordagem combina o uso de mineração de padrões sequenciais, predição, tomada de decisão, planejamento reativo e escalonamento. Essa arquitetura é responsável pelo controle total do jogo quando sequências de ações nos diversos níveis de abstração são escolhidas com base em raciocínio e probabilidade.

- *Uso do conhecimento específico do domínio de jogos RTS para inferir sobre macroações em todos os níveis de abstração.* Bases de *replays* com partidas disputadas entre jogadores profissionais foram analisadas para extrair padrões de ações e estados do

jogo de forma geral. Com essas informações, uma árvore preditiva foi modelada para escolha de macroações complementares para execução com a macroação principal.

- *Modelagem de um POMDP online com duas políticas de execução considerando mudanças dentro do ambiente.* O POMDP utiliza as observações da partida para gerar estados de crença e cenários que avaliam possíveis mudanças no ambiente do jogo. Os valores de recompensas obtidos são maximizados com o uso de duas políticas e dentro das restrições de tempo real.

- *Uso de planejamento reativo e escalonamento de ações na arquitetura para Jogos RTS.* O planejamento reativo permitiu que ações em execução fossem canceladas e outras novas consideradas. Isso contribuiu para garantir a troca da estratégia proposta pelo POMDP, quando o ambiente do jogo é alterado. O escalonamento garantiu a execução em paralelo de macroações de todos níveis de abstração e consequentemente a execução de diferentes tarefas em paralelo.

- *Especificação dos módulos necessários e a comunicação de dados para a arquitetura.* Os três módulos principais, a definição dos dados utilizados e como eles são compartilhados estabelecem a arquitetura proposta nesta tese. Cada módulo tem seus algoritmos e técnicas definidos para melhor performance das tarefas que ele executa. Com essa padronização, a arquitetura pode ser incrementada com novas técnicas, mediante a adição de novos módulos ou complementos dos já existentes. Além disso, a arquitetura pode ser replicada e utilizada em qualquer outro jogo RTS ou ambiente de execução que possua ações com restrições de tempo real.

9.2 Limitações

Algumas das limitações deste trabalho são descritas a seguir.

- ***Replays*:** o objeto de dados mais importante da arquitetura é a macroação. Para operar com estratégia, tática e controle reativo de modo semelhante a um jogador humano, é preciso que as macroações sejam classificadas. Essa classificação é feita com base nos *replays* de partidas, e a arquitetura fica dependente da existência dessa base para operar. Encontrar um padrão de dados que possa ser estendido a qualquer tipo de jogo RTS ou problema similar e utilizado para mineração de dados pode aumentar o alcance do trabalho proposto.

- **Controle reativo:** dentre os níveis de abstração presentes em um jogo RTS e utilizados, o controle reativo é o mais baixo. Ele refere-se a ações direcionadas para uma única unidade. As macroações de *ControleReativo* são complexas de replicar para mais de uma unidade. Durante um confronto, torna-se inviável utilizar o

POMDP para decidir sob uma macroação para cada uma das unidades utilizadas em batalha. É preciso construir uma função para extensão das macroações de *ControleReativo*, para que todas as unidades em confronto possam ser orientadas de acordo com um conjunto de ações apropriado.

- **Tomada de Decisão:** sempre que novas observações são recebidas, o processo de tomada de decisão é reiniciado para escolha de novas macroações. Contudo, não existe garantia de que uma nova macroação considerada seja melhor para o atual momento da partida do que aquela já em execução. Muitas vezes, é melhor não decidir por nenhuma nova macroação e manter a execução da atual. No POMDP *online*, isso é feito sempre que a nova macroação não possui recompensa maior do que aquela atual. No entanto, esse critério não cobre todos os casos, como quando uma nova macroação possui maior recompensa, mas vai deslocar unidades que ainda não foram produzidas ou estão sendo utilizadas em um combate, e não devem deixar suas posições. Um critério para determinar se uma nova macroação deve substituir a atual levando em conta a estratégia em execução e contexto geral da partida no momento seria melhor para a arquitetura.

- **Escalonamento:** o algoritmo ESCALONA foi adaptado para as macroações e níveis de abstração propostos neste trabalho, contudo, existem ações em macroações que são escalonadas em separado, mas deveriam ficar próximas umas das outras na execução na partida. Essa proximidade pode trazer recursos que irão acelerar a execução de todas as macroações escalonadas em determinado intervalo de tempo. Seria interessante analisar essas situações e definir uma estrutura de prioridades. Nessa estrutura, ações que estão em macroações diferentes poderiam ser escalonadas em intervalos próximos, que beneficiariam a execução mútua dessas macroações. O escalonamento seria condicionado aos valores de prioridade definidos para cada ação.

9.3 Trabalhos Futuros

Durante o desenvolvimento deste trabalho, algumas questões foram levantadas para melhorar o desempenho e validação da metodologia proposta. Entretanto, essas questões não foram investigadas no escopo deste trabalho, constituindo assim assunto para trabalhos futuros. A seguir, são listadas as questões levantadas que proporcionariam trabalhos futuros:

- Explorar o uso de multiagentes com múltiplas macroações. Essa proposta poderia aumentar a capacidade de tomada de decisão da arquitetura. Macroações seriam escolhidas a partir do cruzamento de informação dos diversos agentes, que captariam mais informações do ambiente e as utilizariam para aumentar o *lookahead* do POMDP, melhorando inclusive a predição da CHAID.

- ❑ Explorar o uso de técnicas de raciocínio espacial e temporal. Essas técnicas realizam análises do mapa e pontos importantes dentro do ambiente do jogo. Seu uso poderia expandir a forma de classificar e utilizar as macroações de *ControleReativo*, que são difíceis de gerenciar. Algoritmos utilizados com frequência no campo da robótica, como Decomposição de Voronoi (FU; BANDYOPADHYAY; ANG, 2009), Mapas de Influência (URIARTE; ONTANÓN, 2012) e Campos Potenciais (MIELNICZUK, 2017), poderiam aumentar a capacidade da arquitetura.

- ❑ Explorar diferentes algoritmos para tomada de decisão. Novos algoritmos de tomada de decisão em ambientes não determinísticos poderiam ser estudados. Algoritmos com foco em restrições de tempo e mudanças no modelo de ações e do ambiente poderiam ampliar os resultados.

9.4 Contribuições em Produção Bibliográfica e Software

No que tange a produção científica relacionada com esta tese foram publicados artigos em eventos internacionais e nacionais listados a seguir.

1. Naves, T. F.; Lopes, C. R. (2015, November). One Approach to Determine Goals in RTS Games Using Maximization of Resource Production with Local Search and Scheduling. In Tools with Artificial Intelligence (ICTAI), 2015 IEEE 27th International Conference on (pp. 469-477). IEEE. (Ciência da Computação: Qualis A2)

2. Naves, T. F.; Lopes, C. R. (2017, July). Resource Production in StarCraft Based on Local Search and Planning. In International Conference on Computational Science and Its Applications (pp. 87-102). Springer, Cham. (Ciência da Computação: Qualis B1)

3. Naves, T. F.; Lopes, C. R. (2017, September). Use Of Online POMDP with Heuristic Search Applied to Decision Making in Real Time Strategy Games. In SBGames - Simpósio Brasileiro de Games e Entretenimento Digital. (Ciência da Computação: Qualis B2)

4. Naves, T. F.; Lopes, C. R. (2017, September). Online POMDP with Heuristic Search and Sampling Applied to Real Time Strategy Games. In Tools with Artificial Intelligence (ICTAI), 2017 IEEE 29th International Conference on. IEEE. (Ciência da Computação: Qualis A2)

O artigo listado abaixo foi submetido a um periódico e está em avaliação:

1. Naves, T. F.; Lopes, C. R. (2017, August). Decision Making in Real Time Strategy Games through POMDP with Heuristic Search and State Sampling. International Journal Of Computer Games Technology, 2017. ISSN:1012-2443 (Ciência da Computação: Qualis B1)

A arquitetura proposta nesta tese e o *bot* criado para uso no StarCraft está disponível no seguinte sítio na internet:

- http://www.mediafire.com/file/qi64v1m40vj1re1/POMDPbot.zip

Referências

AAMODT, A.; PLAZA, E. Case-based reasoning: Foundational issues, methodological variations, and system approaches. **Artificial Intelligence Communications,**, v. 7, p. 39 a 59, 1994.

AGRAWAL, R.; SRIKANT, R. Mining sequential patterns. In: IEEE. **Data Engineering, 1995. Proceedings of the Eleventh International Conference on**. [S.l.], 1995. p. 3–14. <https://doi.org/10.1109/ICDE.1995.380415>.

AHA, D.; MATTHEW, M.; PONSEN, M. Learning to win: Case-based plan selection in a real-time strategy game. In: **Proceedings of ICCBR**. [S.l.]: Springer, 2005. p. 5 – 20.

ALMEIDA, P. J. de; KAJIN, M.; VIEIRA, M. Equilíbrio de nash e estratégias evolutivamente estáveis: a teoria dos jogos na ecologia de populações. **Oecologia australis**, v. 16, n. 1, p. 127–140, 2012. <https://doi.org/10.4257/oeco.2012.1601.11>.

AMATO, C. et al. Probabilistic planning for decentralized multi-robot systems. In: **2015 AAAI Fall Symposium Series, North America, sep.** [s.n.], 2015. Disponível em: <https://www.aaai.org/ocs/index.php/FSS/FSS15/paper/view/11655>.

ANAGNOSTOPOULOS, A.; BRODER, A.; CARMEL, D. Sampling search-engine results. In: ACM. **Proceedings of the 14th international conference on World Wide Web**. [S.l.], 2005. p. 245–256. <https://doi.org/10.1145/1060745.1060784>.

AUER, P.; BIANCHI, N. C.; FISCHER, P. Finite-time analysis of the multiarmed bandit problem. **Machine learning**, Springer, v. 47, n. 2-3, p. 235–256, 2002.

AVERY, P.; LOUIS, S.; AVERY, B. Evolving coordinated spatial tactics for autonomous entities using influence maps. IEEE Press, Piscataway, NJ, USA, p. 341–348, 2009. Disponível em: <http://dl.acm.org/citation.cfm?id=1719293.1719350>.

BALLA, R.; FERN, A. Uct for tactical assault planning in real time strategy games. **International Joint Conference of Artificial Intelligence, IJCAI. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc**, p. 70 a 75, 2009.

BARRIGA, N. A.; STANESCU, M.; BURO, M. Game tree search based on non-deterministic action scripts in real-time strategy games. **IEEE Transactions on Computational Intelligence and AI in Games**, IEEE, 2017.

BREMAUD, P. **Markov Chains: Gibbs Fields, Monte Carlo Simulation, and Queues**. [S.l.]: Springer, 1999. v. 31. <https://doi.org/10.1007/978-1-4757-3124-8>.

BURO, M. Call for ai research in rts games. AAAI Press, p. 139–141, 2004.

BURO, M.; CHURCHILL, D. Real-time strategy game competitions. **AI Magazine, vol. 33, no. 3, pp. 106 a 108**, 2012.

BURO, M.; FURTAK, T. Rts games as test-bed for real-time research. **Workshop on Game AI, JCIS, 481-484**, 2003.

CABRERA, R.; COTTA, C.; LEIVA, A. A review of computational intelligence in rts games. **IEEE Symposium on Foundations of Computational Intelligence (FOCI)**, p. 114–121, 04 2013. <https://doi.org/10.1109/FOCI.2013.6602463>.

CADENA, P.; GARRIDO, L. Fuzzy case-based reasoning for managing strategic and tactical reasoning in starcraft. **MICAI, ser. Lecture Notes in Computer Science**, Springer Berlin Heidelberg, p. 113–124, 2011. <https://doi.org/10.1007/978-3-642-25324-9_10>.

CASSANDRA, A. R. A survey of pomdp applications. In: **Working notes of AAAI 1998 fall symposium on planning with partially observable Markov decision processes**. [S.l.: s.n.], 1998. v. 1724.

CERTICKY, M. Case-based reasoning for army compositions in real-time strategy games. **Conference of Young Researchers**, p. 70–73, 01 2013.

CHAWLA, N. C4. 5 and imbalanced data sets: investigating the effect of sampling method, probabilistic estimate, and decision tree structure. In: **Proceedings of the ICML**. [S.l.: s.n.], 2003. v. 3.

CHEUNG, M. Z. B. K. C. Y. D. A gsp-based efficient algorithm for mining frequent sequences. **International Conference on Artificial Intelligence (IC-AI 2001)**, 2002. Disponível em: <http://hdl.handle.net/10722/93116>.

CHOW, S. Testing software design modeled by finite-state machines. **IEEE transactions on software engineering**, IEEE, n. 3, p. 178–187, 1978.

CHUNG, M.; BURO, M.; SCHAEFFER, J. Monte carlo planning in rts games. **IEEE Symposium on Computational Intelligence and Games**, 2005.

CHURCHIL, D.; BURO, M. Build order optimization in starcraft. **AIIDE 2011: AI and Interactive Digital Entertainment Conference**, 01 2011.

CHURCHILL, D.; SAFFIDINE, A.; BURO, M. Fast heuristic search for rts game combat scenarios. **Proc. 8th AAAI Conf. Artif. Intell. Interactive Digit. Entertain**, p. 112–117, 2012.

CUNHA, R. F.; MACHADO, M. C.; CHAIMOWICZ, L. Rtsmate: Towards an advice system for rts games. **Comput. Entertain.**, ACM, New York, NY, USA, v. 12, n. 1, p. 1–20, 2015. ISSN 1544-3574.

CUNNINGHAM, H.; WILKS, Y.; GAIZAUSKAS, R. Gate: a general architecture for text engineering. In: ASSOCIATION FOR COMPUTATIONAL LINGUISTICS. **Proceedings of the 16th conference on Computational linguistics - Volume 2**. [S.l.], 1996. p. 1057–1060. <https://doi.org/10.3115/993268.993365>.

DERESZYNSKI, E. et al. Learning probabilistic behavior models in real-time strategy games. **Artificial Intelligence and Interactive Digital Entertainment (AIIDE)**, AAAI Press, p. 20–25, 2011. Disponível em: <http://dl.acm.org/citation.cfm?id=3014589.3014594>.

DOSHI, P.; GMYTRASIEWICZ, P. J. Monte carlo sampling methods for approximating interactive pomdps. **Journal of Artificial Intelligence Research**, v. 34, p. 297–337, 2009.

E, B.; RON, K. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. **Machine learning**, Springer, v. 36, n. 1, p. 105–139, 1999.

EDELKAMP, S.; HOFFMANN, J. **PDDL2.2: The language for the Classical Part of the 4th International Planning Competition**. [S.l.], 2004.

FERREIRA, C. **Gene expression programming: mathematical modeling by an artificial intelligence**. [S.l.]: Springer, 2006. v. 21.

FIRBY, R. An investigation into reactive planning in complex domains. In: **International Joint Conference on Artificial Intelligence**. [S.l.: s.n.], 1987. v. 87, p. 202–206.

FU, J. G.; BANDYOPADHYAY, T.; ANG, M. Local voronoi decomposition for multi-agent task allocation. In: IEEE. **Robotics and Automation, 2009. ICRA'09. IEEE International Conference on**. [S.l.], 2009. p. 1935–1940. <https://doi.org/10.1109/ROBOT.2009.5152829>.

HAGELBACK, J. **Multi-Agent Potential Field Based Architectures for Real-Time Strategy Game Bots**. Tese (Doutorado) — Blekinge Institute of Technology, Sweden, 2012.

HAGENAARS, J. A.; MCCUTCHEON, A. L. **Applied latent class analysis**. [S.l.]: Cambridge University Press, 2002. <https://doi.org/10.1017/CBO9780511499531>.

HALE, D.; YOUNGBLOOD, G.; DIXIT, P. Automatically-generated convex region decomposition for real-time spatial agent navigation in virtual worlds. **Artificial Intelligence and Interactive Digital Entertainment AIIDE**, 2008.

HAN, J. et al. Freespan: frequent pattern-projected sequential pattern mining. In: ACM. **Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining**. [S.l.], 2000. p. 355–359.

_____. Prefixspan: mining sequential patterns efficiently by prefix-projected pattern growth. In: **proceedings of the 17th international conference on data engineering**. [S.l.: s.n.], 2001. p. 215–224.

HAN, J.; PEI, J.; YIN, Y. Mining frequent patterns without candidate generation. In: ACM. **ACM Sigmod Record**. [S.l.], 2000. v. 29, n. 2, p. 1–12. <https://doi.org/10.1145/342009.335372>.

HESS, R.; MODJTAHEDZADEH, A. A control theoretic model of driver steering behavior. **IEEE Control Systems Magazine**, IEEE, v. 10, n. 5, p. 3–8, 1990. <https://doi.org/10.1109/37.60415>.

HOANG, H.; LEE-URBAN, S.; MUNOZ-AVILA, H. Hierarchical plan representations for encoding strategic game ai. **AIIDE**, p. 63 a 68, 2005.

HSIEH, J.; SUN, C. Building a player strategy model by analyzing replays of real-time strategy games. **IJCNN**, p. 3106 a 3111, 2008. <https://doi.org/10.1109/IJCNN.2008.4634237>.

JAIDEE, U.; MUNOZ, H. Classq-l: A q-learning algorithm for adversarial real-time strategy games. **Eighth Artificial Intelligence and Interactive Digital Entertainment Conference**, 2012.

JAIDEE, U.; MUNOZ, H.; AHA, D. Case-based learning in goal-driven autonomy agents for real-time strategy combat tasks. **ICCBR Workshop on Computer Games**, 2011.

JAIN, K. Data clustering: 50 years beyond k-means. **Pattern recognition letters**, Elsevier, v. 31, n. 8, p. 651–666, 2010.

KABANZA, F. et al. Opponent behaviour recognition for real-time strategy games. **AAAI Workshops**, AAAI Press, p. 29–36, 2010. Disponível em: <http://dl.acm.org/citation.cfm?id=2908558.2908563>.

KAELBLING M. LITTMAN, A. C. P. Planning and acting in partially observable stochastic domains. **Artif. Intell.**, Elsevier Science Publishers Ltd., Essex, UK, v. 101, n. 1-2, p. 99–134, may 1998. ISSN 0004-3702. <https://doi.org/10.1016/S0004-3702(98)00023-X>.

KASS, G. An exploratory technique for investigating large quantities of categorical data. **Applied statistics**, JSTOR, p. 119–127, 1980.

KOCSIS, L.; SZEPESVARI, C. **Bandit Based Monte-carlo Planning**. Berlin, Heidelberg, 2006. 282–293 p. (ECML'06). <http://dx.doi.org/10.1007/11871842_29>. ISBN 3-540-45375-X, 978-3-540-45375-8.

KUMAR, V.; KANAL, L. N. Search and heuristics a general branch and bound formulation for understanding and synthesizing and/or tree search procedures. **Artificial Intelligence**, v. 21, n. 1, p. 179 – 198, 1983. ISSN 0004-3702. <http://dx.doi.org/10.1016/S0004-3702(83)80009-5>. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0004370283800095>.

KURNIAWATI, H.; HSU, D.; LEE, W. S. Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces. In: ZURICH, SWITZERLAND. **Robotics: Science and systems**. [S.l.], 2008. v. 2008.

KUTLUHAN, E.; HENDLER, J.; NAU, D. S. Umcp: A sound and complete procedure for hierarchical task-network planning. **Proceedings of Second Conference on artificial intelligence planning system**, 2003.

LITTMAN, L.; MOORE, W. Reinforcement learning: A survey. **J. Artif. Int. Res.**, AI Access Foundation, USA, v. 4, n. 1, p. 237–285, 5 1996. ISSN 1076-9757. Disponível em: <http://dl.acm.org/citation.cfm?id=1622737.1622748>.

LIU, H.; SETIONO, R. Chi2: Feature selection and discretization of numeric attributes. In: IEEE. **Tools with artificial intelligence, seventh international conference**. [S.l.], 1995. p. 388–391.

LIU, L.; LI, L. **Regional cooperative multi-agent q-learning based on potential field**. 2008. 535 a 539 p.

LOVE, G. M. N.; PELL, B. General game playing: Overview of the aaai competition. **AI Magazine**, v. 26, p. 62–72, 2005.

MARESCHAL, D.; SHULTZ, R. Generative connectionist networks and constructivist cognitive development. **Cognitive Development**, Elsevier, v. 11, n. 4, p. 571–603, 1996.

MARIA., F.; LONG, D. Pddl2. 1: An extension to pddl for expressing temporal planning domains. **Journal of Artificial Intelligence Research (JAIR)**, p. 61–124, 2003.

MARTHI, B. et al. Concurrent hierarquical reinforcement learning. **International Joint Conference of Artificial Intelligence**, p. 779 a 785, 2005.

MARTIN, B. **Instance-based learning: nearest neighbour with generalisation-alisation**. Dissertação (Mestrado) — University of Waikato, Hamilton, New Zealand, 1995.

MATEAS, M. **Interactive Drama, Art and Artificial Intelligence.** Tese (Doutorado) — Carnegie Mellon University, 2002.

MATEAS, M.; STERN, A. A behavior language for story-based believable agents. In: **IEE Intelligent Systems (17)**. [S.l.: s.n.], 2014. p. 39 a 47.

MENDEL, M. **Uncertain rule-based fuzzy logic systems: introduction and new directions**. [S.l.]: Prentice Hall PTR Upper Saddle River, 2001.

MIELNICZUK, S. A method of artificial potential fields for the determination of ship's safe trajectory. **51 Scientific Journals of the Maritime University of Szczecin**, Scientific Journals Maritime University of Szczecin, Zeszyty Naukowe Akademia Morska w Szczecinie, n. 51, p. 45–49, 2017.

MILES, C.; LOUIS, S. Co-evolving real-time strategy game playing influence map trees with genetic algorithms. **in Proceedings of the International Congress on Evolutionary Computation**, 2006.

MISHRA, K.; ONTANON, S.; RAM, A. Situation assessment for plan retrieval in real-time strategy games. **ECCBR**, p. 355 a 369., 2008. <https://doi.org/10.1007/978-3-540-85502-6_24>.

MONAHAN, G. E. A survey of partially observable markov decision processes: Theory, models, and algorithms. **Management Science**, v. 28, p. 1–16, 1982.

MOON, T. The expectation-maximization algorithm. **IEEE Signal Processing Magazine**, 1996. ISSN 1053-5888.

NAVES, T. F. **Uma Abordagem para Maximização da Produção de Recursos em Jogos RTS**. Dissertação (Mestrado) — Universidade Fedral de Uberlândia, 2012.

NAVES, T. F.; LOPES, C. R. Maximization of the resource production in rts games through stochastic search and planning. **IEEE international conference on systems, man and cybernetics - SMC**, Fac. of Comput., Fed. Univ. of Uberlandia, Uberlandia, Brazil, p. 2241–2246, 2012.

_____. Stochastic search and planning for maximization of resource production in rts games. **International Conference on Artificial Intelligence - ICAI**, Fac. of Comput., Fed. Univ. of Uberlandia, Uberlandia, Brazil, p. 2241– 2246, 2012.

NETER, J.; KUTNER, M.; NACHTSHEIM, C. **Applied linear statistical models**. [S.l.]: Irwin Chicago, 1996. v. 4.

ONTAÑÓN, S. Informed monte carlo tree search for real-time strategy games. In: IEEE. **Computational Intelligence and Games (CIG), 2016 IEEE Conference on**. [S.l.], 2016. p. 1–8.

_____. Combinatorial multi-armed bandits for real-time strategy games. **Journal of Artificial Intelligence Research**, v. 58, p. 665–702, 2017.

ONTANON, S. et al. Learning from demonstration and case-based planning for real-time strategy games. In: **Studies in Fuzziness and Soft Computing**. [S.l.: s.n.], 2008. v. 226, p. 293 310.

_____. A survey of real-time strategy game ai research and competition in starcraft. **IEEE Transactions on Computational Intelligence and AI in Games**, 2013.

PAQUET, S.; TOBIN, L.; CHAIB-DRAA, B. An online pomdp algorithm for complex multiagent environments. In: ACM. **Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems**. [S.l.], 2005. p. 970–977.

PELLEGRINI, J.; WAINER, J. Processos de decisao de markov: um tutorial. **Revista de Informatica Teórica e Aplicada (RITA)**, v. 14, n. 2, p. 133–179, 2007.

PENG, S. A generalized dynamic programming principle and hamilton-jacobi-bellman equation. **Stochastics: An International Journal of Probability and Stochastic Processes**, Taylor & Francis, v. 38, n. 2, p. 119–134, 1992.

PRADHAN, B. A comparative study on the predictive ability of the decision tree, support vector machine and neuro-fuzzy models in landslide susceptibility mapping using gis. **Computers & Geosciences**, Elsevier, v. 51, p. 350–365, 2013.

PREUSS, M. et al. Towards intelligent team composition and maneuvering in real-time strategy games. **Computational Intelligence and AI in Games (TCIAIG)**, 2010.

QUINLAN, J. Induction of decision trees. **Machine learning**, Springer, v. 1, n. 1, p. 81–106, 1986.

RABINER, L.; JUANG, B. An introduction to hidden markov models. **Speech, Signal Processing Magazine**, IEEE, v. 3, n. 1, p. 4–16, 1986.

ROKACH, L.; MAIMON, O. Top-down induction of decision trees classifiers-a survey. **IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)**, IEEE, v. 35, n. 4, p. 476–487, 2005.

ROSS, S.; CHAIB-DRAA, B. Aems: An anytime online search algorithm for approximate policy refinement in large pomdps. In: **IJCAI**. [S.l.: s.n.], 2007. p. 2592–2598.

ROSS, S. et al. Online planning algorithms for pomdps. **Journal of Artificial Intelligence Research**, v. 32, p. 663–704, 2008.

RUSSELL, S. J.; NORVIG. **Artificial Intelligence: A Modern Approach (Second Edition)**. [S.l.]: Prentice Hall, 2003.

SAFAVIAN, S.; LANDGREBE, D. A survey of decision tree classifier methodology. **IEEE transactions on systems, man, and cybernetics**, IEEE, v. 21, n. 3, p. 660–674, 1991.

SILVA, G. B. da; COSTA, H. G. Mapeamento de um núcleo de partida de referências em data mining a partir de periódicos publicados no brasil. **Gestão & Produção**, v. 22, n. 1, p. 107–118, 2015.

SMITH, T.; SIMMONS, R. Point-based pomdp algorithms: Improved analysis and implementation. **21th Conference on Uncertainty in Artificial Intelligence**, 2012.

SOMANI, A. et al. Despot: Online pomdp planning with regularization. In: **Advances in neural information processing systems**. [S.l.: s.n.], 2013. p. 1772–1780.

SRIKANT, R.; AGRAWAL, R. Mining sequential patterns: Generalizations and performance improvements. In: **Proceedings of the 5th International Conference on Extending Database Technology: Advances in Database Technology**. London, UK, UK: Springer-Verlag, 1996. (EDBT '96), p. 3–17. ISBN 3-540-61057-X. Disponível em: <http://dl.acm.org/citation.cfm?id=645337.650382>.

STEINBERG, D.; COLLA, P. Cart: classification and regression trees. **The top ten algorithms in data mining**, CRC Press, v. 9, p. 179, 2009. <https://doi.org/10.1201/9781420089653.ch10>.

SYNNAEVE, G. **Bayesian Programming and Learning for Multi-Player Video Games**. Tese (Doutorado) — Doctorale de Mathamatiques, Sciences et Technologies de Information, L UNIVERSITY DE GRENOBLE, 2012.

SYNNAEVE, G.; BESSIERE, P. A bayesian model for opening prediction in rts games with application to starcraft. **in Proceedings of 2011 IEEE CIG, Seoul, p. 000**, 2011.

_____. A bayesian model for plan prediction in rts games applied to starcraft. **in Proceedings of the Seventh Artificial Intelligence and Interactive Digital Entertainment Conference**, p. 79 a 84, 2011.

_____. Special tactics: a bayesian approach to tactical decision-making. **Conference on Computational Intelligence and Games**, 2012. <https://doi.org/10.1109/CIG.2012.6374184>.

SYNNAEVE, G.; BESSIÈRE, P. Multiscale bayesian modeling for rts games: An application to starcraft ai. **IEEE Transactions on Computational intelligence and AI in Games**, IEEE, v. 8, n. 4, p. 338–350, 2016.

TAVARES, A. et al. Rock, paper, starcraft: Strategy selection in real-time strategy games. In: **12th Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE)**. [S.l.: s.n.], 2016. p. 93–99.

URIARTE, A.; ONTANON, S. Kiting in rts games using influence maps. **Artificial Intelligence and Interactive Digital Entertainment Conference**, 2011.

URIARTE, A.; ONTANÓN, S. Kiting in rts games using influence maps. In: **Eighth Artificial Intelligence and Interactive Digital Entertainment Conference**. [S.l.: s.n.], 2012.

URIARTE, A.; ONTAÑÓN, S. Game-tree search over high-level game states in rts games. In: **Tenth Artificial Intelligence and Interactive Digital Entertainment Conference**. [S.l.: s.n.], 2014.

_____. Improving monte carlo tree search policies in starcraft via probabilistic models learned from replay data. In: **Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference**. [S.l.: s.n.], 2016.

WATKINS, C.; DAYAN, P. **Q-learning**. v. 8, Issue 3-4, 1992. 279-292 p.

WEBER, B. G.; MATEAS, M.; JHALA, A. A particle model for state estimation in real-time strategy games. **in Proceedings of AIIDE, AAAI Press. Stanford, Palo Alto, California: AAAI Press**, p. 103 a 108, 2011.

WEBER, B. G. et al. Reactive planning idioms for multi-scale game ai. **In Proceedings of IEEE CIG**, 2010. <https://doi.org/10.1109/ITW.2010.5593363>.

WENDER, S.; WATSON, I. Applying reinforcement learning to small scale combat in the real-time strategy game starcraft:broodwar. **IEEE Conference on Computational Intelligence and Games (CIG)**, 2012. <https://doi.org/10.1109/CIG.2012.6374183>.

WILKINSON, L. Tree structured data analysis: Aid, chaid and cart. **Retrieved February**, v. 1, p. 2008, 1992.

WITTEN, I.; FRANK, E. Data mining: Practical machine learning tools and techniques. **San Francisco and California: Morgan Kaufmann**, 2005.

WOOLDRIDGE, M.; JENNINGS, N. Intelligent agents: Theory and practice. **The knowledge engineering review**, Cambridge Univ Press, v. 10, n. 02, p. 115–152, 1995.

YOUNG, J.; HAWES, N. Evolutionary learning of goal priorities in a real-time strategy game. **Interactive Digital Entertainment Conference, AIIDE**, 2012.

YOUNG, J. et al. Scail: An integrated starcraft ai system. **CIG (IEEE)**, 2012.

ZANTEMA, H.; BODLAENDER, H. Finding small equivalent decision trees is hard. **International Journal of Foundations of Computer Science**, World Scientific, v. 11, n. 02, p. 343–354, 2000.

ZHANG, W.; DIETTERICH, T. A reinforcement learning approach to job-shop scheduling. In: **IJCAI**. [S.l.: s.n.], 1995. v. 95, p. 1114–1120.